Universidad Politécnica
de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Grado en Ingeniería Informática

Trabajo Fin de Grado

Diseño e Implementación de una Plataforma en Formato de Aplicación Web para la Gestión de Recursos Humanos y Empleados

Autor: Luis Jacinto Donoso Balmaseda

Tutor: Vicente Martínez Orga

Madrid, Junio 2024

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado
Doble Grado en Ingeniería Informática y Administración y Dirección de Empresas
Título: Diseño e Implementación de una Plataforma en Formato de Aplicación Web para la Gestión de Recursos Humanos y EmpleadosJunio 2024
Autor: Luis Jacinto Donoso Balmaseda

Tutor:

Vicente Martínez Orga
Departamento de Inteligencia Artificial
ETSI Informáticos
Universidad Politécnica de Madrid

Agradecimientos:

Antes de nada, quiero dar las gracias a mi tutor por su gran paciencia y confianza depositadas en mí estos meses.

A mi familia, por siempre estar cuando hacía falta, aunque no lo pidiera sabíais cuando ayudarme.

Gracias también especialmente a Miguel, Jorge, Pedro y a todo mi círculo de amigos cercanos por su constante apoyo, consejo, ayuda y por acompañarme siempre en este camino.

Por último, gracias a todos mis amigos de la ETSIINF, saber que todos pasábamos por prácticamente lo mismo para lo bueno y para lo malo, además de habernos apoyado mutuamente y haber disfrutado del camino juntos ha hecho de esto una experiencia inolvidable.

Resumen

En el presente Trabajo de Fin de Grado (TFG), se aborda el desarrollo de una plataforma de gestión de recursos humanos (RRHH) diseñada para satisfacer las necesidades de pequeñas y medianas empresas (PYMEs). La gestión eficaz de los RRHH es un componente fundamental para el éxito de cualquier empresa en el actual panorama empresarial, caracterizado por una rápida transformación digital y una alta competitividad. La capacidad de una empresa para reclutar, retener y desarrollar talento es crucial para su crecimiento y competitividad a largo plazo.

Este proyecto se sitúa en el contexto de la creciente demanda de soluciones tecnológicas innovadoras para la gestión de RRHH. Se propone el desarrollo de una plataforma integral que aproveche las últimas tecnologías y metodologías para ofrecer una solución eficiente y escalable, adaptada a las necesidades específicas de cada empresa. La plataforma está diseñada para ser una herramienta completa que permita a las empresas gestionar de manera eficaz sus procesos de RRHH, incluyendo el reclutamiento, la gestión del desempeño, la formación y el desarrollo, la gestión de ausencias y licencias, entre otros.

Para el desarrollo de la plataforma, se ha optado por un enfoque basado en microservicios. Este modelo arquitectónico proporciona una mayor modularidad y flexibilidad, permitiendo el desarrollo y despliegue independiente de cada componente. En el backend, se ha utilizado Spring Boot, un framework de aplicación Java que facilita el desarrollo de aplicaciones empresariales robustas y escalables. En el frontend, se ha empleado ReactJS, una biblioteca de JavaScript ampliamente utilizada para la construcción de interfaces de usuario interactivas y dinámicas. La elección de ReactJS garantiza una experiencia de usuario fluida y receptiva, así como una fácil integración con el backend a través de API RESTful.

La plataforma incluye varias funcionalidades básicas que son escalables y ampliables en el futuro. Estas funcionalidades no solo facilitan la administración del capital humano de una empresa, sino que también contribuyen significativamente a mejorar la productividad, la satisfacción laboral y el cumplimiento normativo. Entre las funcionalidades destacadas se encuentran la gestión de perfiles de empleados, el registro de horas laborales, la generación de informes, la gestión de reclutamiento y selección, la automatización de tareas administrativas, la capacitación y desarrollo, y la gestión de ausencias y licencias.

La metodología de desarrollo empleada ha sido ágil, lo que ha permitido una entrega incremental y continua de funcionalidades. Este enfoque ha facilitado la adaptación a los cambios y la incorporación de mejoras continuas en la plataforma.

En resumen, este TFG presenta el diseño y desarrollo de una plataforma de gestión de RRHH que se posiciona como una respuesta a las demandas del mercado. A través de la implementación de tecnologías y metodologías modernas, se ha logrado crear una solución completa, eficiente y escalable que puede adaptarse a las necesidades cambiantes de las empresas, ofreciendo una herramienta que mejora significativamente la gestión del talento humano

Abstract

In this Final Degree Project (TFG), the development of a human resources (HR) management platform is addressed, designed to meet the needs of small and medium-sized enterprises (SMEs). Effective HR management is a fundamental component for the success of any company in the current business landscape, characterized by rapid digital transformation and high competitiveness. A company's ability to recruit, retain, and develop talent is crucial for its long-term growth and competitiveness.

This project is situated in the context of the growing demand for innovative technological solutions for HR management. The development of a comprehensive platform is proposed, leveraging the latest technologies and methodologies to offer an efficient and scalable solution tailored to the specific needs of each company. The platform is designed to be a complete tool that allows companies to effectively manage their HR processes, including recruitment, performance management, training and development, leave and absence management, among others.

For the development of the platform, a microservices-based approach has been adopted. This architectural model provides greater modularity and flexibility, allowing for the independent development and deployment of each component. In the backend, Spring Boot has been used, a Java application framework that facilitates the development of robust and scalable enterprise applications. In the frontend, ReactJS has been employed, a widely-used JavaScript library for building interactive and dynamic user interfaces. The choice of ReactJS ensures a smooth and responsive user experience, as well as easy integration with the backend through RESTful APIs.

The platform includes several basic functionalities that are scalable and extendable in the future. These functionalities not only facilitate the management of a company's human capital but also significantly contribute to improving productivity, job satisfaction, and regulatory compliance. Key functionalities include employee profile management, work hours tracking, report generation, recruitment and selection management, administrative task automation, training and development, and leave and absence management.

The development methodology employed has been agile, which has allowed for incremental and continuous delivery of functionalities. This approach has facilitated adaptation to changes and the incorporation of continuous improvements into the platform.

In summary, this TFG presents the design and development of an HR management platform positioned as a response to market demands. By implementing modern technologies and methodologies, a comprehensive, efficient, and scalable solution has been created that can adapt to the

changing needs of companies, offering a tool that significantly improves human talent management.

Tabla de contenido

1 Introducción

1.1 Contexto

La gestión eficaz de los Recursos Humanos (RRHH) es un componente fundamental para el éxito de cualquier empresa en el actual panorama empresarial altamente competitivo y en constante evolución. En este contexto, el presente Trabajo de Fin de Grado (TFG) se centra en el diseño y desarrollo de una plataforma de gestión de RRHH, con el objetivo de proporcionar a las empresas una herramienta integral y eficiente para administrar su talento humano.

En un entorno empresarial caracterizado por la rápida transformación digital, la demanda de soluciones tecnológicas innovadoras para la gestión de RRHH está en constante aumento. La capacidad de una empresa para reclutar, retener y desarrollar talento es crucial para su crecimiento y competitividad a largo plazo. Por lo tanto, la necesidad de sistemas de gestión de RRHH ágiles, personalizables y escalables se ha convertido en una prioridad para muchas organizaciones.

En este contexto, este trabajo se posiciona como una respuesta a estas demandas del mercado. Se propone el desarrollo de una plataforma de gestión de RRHH que aproveche las últimas tecnologías y metodologías para ofrecer una solución completa y adaptada a las necesidades específicas de cada empresa.

Para lograr este objetivo, se han seleccionado cuidadosamente las tecnologías más adecuadas para cada componente de la plataforma.

En el backend, se utilizará Spring Boot, un framework de aplicación Java que facilita el desarrollo de aplicaciones empresariales robustas y escalables. Este framework proporciona un entorno de desarrollo ágil y eficiente, permitiendo la creación de microservicios independientes y altamente interoperables.

En el frontend, se empleará ReactJS, una biblioteca de JavaScript ampliamente utilizada para la construcción de interfaces de usuario interactivas y dinámicas. La elección de ReactJS garantiza una experiencia de usuario fluida y receptiva, así como una fácil integración con el backend a través de API RESTful.

Además, se ha adoptado un enfoque basado en microservicios para la arquitectura de la plataforma. Este modelo arquitectónico proporciona una mayor modularidad y flexibilidad, permitiendo el desarrollo y despliegue independiente de cada componente. Esto facilita la escalabilidad y el mantenimiento a largo plazo de la plataforma, así como la integración con sistemas existentes en el entorno empresarial.

En cuanto a los métodos empleados, se seguirá una metodología de desarrollo ágil, como Scrum o Kanban, que permite una entrega incremental y continua de funcionalidades.

En resumen, este trabajo se sitúa en el contexto de la creciente demanda de soluciones tecnológicas innovadoras para la gestión de RRHH en un entorno empresarial altamente competitivo. Se propone el desarrollo de una plataforma integral de gestión de RRHH que aproveche las últimas tecnologías y metodologías para ofrecer una solución eficiente y escalable a las necesidades cambiantes de las empresas.

1.2 Objetivos

El principal objetivo de este TFG es diseñar y desarrollar una aplicación web con el propósito de que se una herramienta que las empresas puedan usar para agilizar su gestión de recursos humanos, ahorrando tiempo y recursos. Para lograrlo se plantean los siguientes objetivos:

2 Estado del arte

En el contexto actual, la gestión de recursos humanos y empleados en empresas ha evolucionado significativamente gracias a la implementación de plataformas tecnológicas avanzadas. Estas herramientas no solo facilitan la administración de personal sino que también optimizan procesos y mejoran la eficiencia operativa. Este capítulo presenta una revisión crítica de las principales plataformas de gestión de recursos humanos y empleados en el mercado, destacando sus características, funcionalidades, ventajas y limitaciones.

2.1 Workday

Características y Funcionalidades Clave:

Workday es una plataforma integral que ofrece soluciones para la gestión de capital humano, finanzas y planificación. Sus principales funcionalidades incluyen:

- Gestión del talento: seguimiento de candidatos, gestión de rendimiento y desarrollo de carrera.

- Nómina y beneficios: administración de nóminas, compensaciones y beneficios.

- Análisis y reportes: herramientas avanzadas de análisis para la toma de decisiones estratégicas.

Puntos Fuertes:

- Integración Completa: Ofrece una solución unificada para la gestión de recursos humanos y finanzas, lo que permite una mejor coordinación y visibilidad en toda la empresa.

- Escalabilidad: Adecuada tanto para medianas como grandes empresas, ofreciendo funcionalidades avanzadas que se adaptan al crecimiento empresarial.

- Innovación Continua: Constantemente actualiza sus funcionalidades para incluir las últimas tendencias y tecnologías del sector.

Limitaciones:

- Coste Elevado: El alto coste de implementación y mantenimiento puede ser prohibitivo para pequeñas empresas.

- Curva de Aprendizaje: La complejidad de la plataforma puede requerir una formación considerable para los usuarios.

Evaluación Comparativa:

Workday se destaca por su robustez y capacidad de integración, pero su coste y complejidad limitan su accesibilidad para empresas más pequeñas. Nuestra plataforma, en contraste, se orientará inicialmente a PYMEs ofreciendo funcionalidades básicas a un coste accesible, facilitando una adopción más rápida y menos costosa. [1]

2.2 SAP SuccessFactors

Características y Funcionalidades Clave:

SAP SuccessFactors es una solución en la nube que proporciona herramientas para la gestión del talento y el desempeño, incluyendo:

- Reclutamiento y selección: automatización del proceso de contratación.

- Gestión del desempeño y objetivos: seguimiento del rendimiento y establecimiento de metas.

- Aprendizaje y desarrollo: módulos de formación y desarrollo profesional.

Puntos Fuertes:

- Profundidad Funcional: Ofrece un conjunto completo de herramientas para todas las etapas del ciclo de vida del empleado.

- Integración con SAP: Se integra fácilmente con otros productos de SAP, beneficiando a empresas que ya utilizan soluciones SAP.

Limitaciones:

- Coste y Complejidad: Similar a Workday, su alto coste y complejidad técnica pueden ser una barrera para pequeñas empresas.

- Personalización: La personalización puede ser limitada sin incurrir en costes adicionales significativos.

Evaluación Comparativa:

SAP SuccessFactors es ideal para grandes empresas con necesidades complejas y recursos suficientes. Para PYMEs, nuestra plataforma ofrecerá una solución más accesible y fácil de implementar, priorizando la usabilidad y la reducción de costes. [2]

2.3 Oracle HCM Cloud

Características y Funcionalidades Clave:

Oracle HCM Cloud proporciona una solución completa para la gestión de capital humano con funcionalidades como:

- Gestión del talento: desarrollo del talento y sucesión.

- Gestión de la fuerza laboral: planificación y optimización de la fuerza laboral.

- Nómina y beneficios: administración de nóminas globales.

Puntos Fuertes:

- Funcionalidades Avanzadas: Ofrece capacidades avanzadas y personalizables para la gestión de RRHH.

- Seguridad y Cumplimiento: Alta seguridad y cumplimiento con las normativas internacionales.

Limitaciones:

- Alto Coste: Elevados costes de implementación y suscripción.

- Requiere Capacitación: Los usuarios necesitan capacitación extensa debido a su complejidad.

Evaluación Comparativa:

Oracle HCM Cloud es una opción potente para empresas grandes que requieren soluciones avanzadas y personalizables. Nuestra plataforma, dirigida a PYMEs, se enfocará en ofrecer funcionalidades esenciales a un precio más asequible y con menor complejidad. [3]

2.4 BambooHR

Características y Funcionalidades Clave:

BambooHR es una plataforma de gestión de RRHH diseñada para pequeñas y medianas empresas, ofreciendo:

- Gestión de datos de empleados: almacenamiento centralizado de información.

- Reclutamiento: seguimiento y gestión de candidatos.

- Gestión del desempeño: evaluación del rendimiento y feedback continuo.

Puntos Fuertes:

- Facilidad de Uso: Interfaz intuitiva y fácil de usar, adecuada para empresas con recursos tecnológicos limitados.

- Coste Asequible: Precios competitivos que hacen accesible la tecnología de gestión de RRHH a pequeñas empresas.

Limitaciones o Deficiencias:

- Funcionalidades Limitadas: Menos funcionalidades avanzadas en comparación con soluciones más grandes como Workday o SAP.

- Integraciones Limitadas: Menor capacidad de integración con otras herramientas empresariales.

Evaluación Comparativa:

BambooHR es una solución excelente para PYMEs que buscan una herramienta de gestión de RRHH accesible y fácil de usar. Nuestra plataforma se posicionará de manera similar, ofreciendo un enfoque inicial en funcionalidades esenciales con posibilidad de expansión futura. [4]

2.5 ADP Workforce Now

Características y Funcionalidades Clave:

ADP Workforce Now es una solución integral de gestión de la fuerza laboral para medianas empresas, que incluye:

- Gestión de nómina y beneficios: procesamiento de nóminas y administración de beneficios.

- Gestión del talento: contratación, desarrollo y retención de talento.

- Cumplimiento y análisis: herramientas para asegurar el cumplimiento normativo y realizar análisis de datos.

Puntos Fuertes:

- Solución Integral: Cobertura completa de todas las necesidades de gestión de RRHH.

- Experiencia de Mercado: Amplia experiencia y confiabilidad en el sector.

Limitaciones o Deficiencias:

- Coste: Puede ser costoso para pequeñas empresas.

- Personalización: Limitada capacidad de personalización sin incurrir en costes adicionales.

Evaluación Comparativa:

ADP Workforce Now es adecuado para medianas empresas que necesitan una solución integral y confiable. Nuestra plataforma, en cambio, se centrará en ofrecer una alternativa asequible y fácil de implementar para PYMEs, con funcionalidades básicas y escalables. [5]

2.6 Evaluación y posicionamiento:

Nuestra plataforma de gestión de recursos humanos se orientará inicialmente a PYMEs, ofreciendo funcionalidades básicas esenciales para la gestión eficiente de recursos humanos. Aunque no competirá directamente con gigantes como SAP y Oracle en el corto plazo, nuestra propuesta de valor se centrará en la simplicidad, accesibilidad y coste-efectividad, con la posibilidad de expansión y personalización futura según las necesidades del cliente.

3 Selección de Funcionalidades Básicas a nivel de RRHH para la plataforma

De cara a la implementación, las funcionalidades que se escojan y puedan proporcionar la plataforma pueden resultar cruciales. Estas funcionalidades no solo facilitan la administración del capital humano de una empresa, sino que también contribuyen significativamente a mejorar la productividad, la satisfacción laboral y el cumplimiento normativo.

Es por ello qué, analizando lo visto en el estado del arte, se han definido una serie de funcionalidades básicas que pueden ser escalables y ampliadas en un futuro.

A continuación, se detalla la importancia de cada una de las funcionalidades mencionadas:

3.1 Gestión de perfiles de empleados:

La capacidad de mantener perfiles detallados de los empleados es fundamental para tener una visión completa y actualizada de la fuerza laboral de la empresa. Esto permite un seguimiento eficaz del historial laboral, las habilidades y las certificaciones de cada empleado, lo que a su vez facilita la asignación de tareas, la identificación de talento y el desarrollo profesional.

3.2 Registro de horas laborales:

El registro preciso de las horas laborales no solo es crucial para el cálculo preciso de la nómina, sino que también proporciona datos importantes sobre la productividad y el rendimiento de los empleados. Facilita la identificación de patrones de trabajo, la gestión del tiempo y la detección de posibles problemas de cumplimiento laboral.

En esta sección además se podrán introducir las horas “reales” que se considere que se han trabajado de forma manual o dejar este campo en blanco y dejar que la plataforma haga el informe automáticamente calculando las horas con la hora de entrada y la de salida.

3.3 Generación de informes:

La generación de informes permite a los gerentes y profesionales de RRHH analizar datos clave relacionados con la fuerza laboral y tomar decisiones fundamentadas. Los informes sobre el desempeño, la productividad, la rotación de personal y otros aspectos proporcionan una visión completa de la situación

laboral de la empresa, lo que permite identificar áreas de mejora y oportunidades de crecimiento.

Estos informes constarán de varios campos interesantes a comentar:

Week Hours: Sumará las horas trabajadas en total los últimos 7 días desde la generación del informe. Esto permite parametrizar el trabajo y exportarlo a Excel o con el propio SQL server usarlo para gestiones de Business Intelligence y Business Analysis.

Month Hours: Mismo caso, pero con el último mes.

Evaluación: Además de datos numéricos permite incorporar a la base de datos una evaluación del desempeño final .

3.4 Gestión de reclutamiento y selección:

Un proceso de reclutamiento eficiente es crucial para atraer y retener talento de alta calidad. La automatización de tareas como la publicación de vacantes, la evaluación de candidatos y la gestión de entrevistas agiliza el proceso y garantiza que se seleccionen los candidatos más adecuados para cada puesto.

3.5 Automatización de tareas administrativas:

La automatización de tareas administrativas libera tiempo y recursos que pueden dedicarse a actividades más estratégicas y de mayor valor añadido. Reducir la carga administrativa también minimiza el riesgo de errores y mejora la eficiencia operativa en general.

3.6 Capacitación y desarrollo:

La capacitación y el desarrollo profesional son fundamentales para el crecimiento y la retención del talento en una empresa. Proporcionar herramientas para planificar, administrar y hacer seguimiento de programas de capacitación garantiza que los empleados estén constantemente actualizados y preparados para enfrentar los desafíos del mercado.

3.7 Gestión de ausencias y licencias:

La gestión eficiente de ausencias y licencias es vital para garantizar la continuidad operativa y el cumplimiento de las políticas de la empresa. Automatizar este proceso facilita la solicitud, aprobación y seguimiento de ausencias, lo que minimiza las interrupciones y asegura la cobertura adecuada

de las funciones laborales. Esto lo hará la plataforma de forma automática en la base de datos cuando se registra que un usuario no ha ido a trabajar.

3.8 Seguridad y cumplimiento normativo:

La seguridad y el cumplimiento normativo son aspectos críticos en la gestión de recursos humanos, especialmente en lo que respecta a la privacidad de los datos de los empleados y el cumplimiento de las regulaciones laborales. Garantizar la seguridad de los datos y cumplir con las normativas vigentes es esencial para proteger la reputación y la integridad de la empresa.

3.9 Interfaz intuitiva y fácil de usar:

Una interfaz intuitiva y fácil de usar es clave para asegurar la adopción y el uso efectivo de la aplicación por parte de los usuarios. Una interfaz amigable facilita la navegación y el acceso a las funcionalidades de la plataforma, lo que aumenta la satisfacción del usuario y mejora la eficiencia en general.

4 Selección de tecnologías y diseño a utilizar.

A raíz de las funcionalidades seleccionadas, para desarrollar la aplicación web de gestión de Recursos Humanos se han empleado una serie de tecnologías y herramientas que abarcan diferentes aspectos del proceso de desarrollo de software. Desde el backend hasta el frontend, así como la gestión de bases de datos y la coordinación del proyecto, cada tecnología desempeña un papel crucial en la construcción de una aplicación robusta y eficiente. A continuación, se detallarán con mayor profundidad las tecnologías utilizadas y su función en el desarrollo de la plataforma.

4.1 Backend:

Java:

Java, un lenguaje de programación orientado a objetos ampliamente utilizado en el desarrollo de aplicaciones empresariales, ha sido la columna vertebral del backend de la aplicación. Su robustez, portabilidad y escalabilidad lo convierten en una elección natural para entornos empresariales exigentes. Java ha sido empleado para implementar la lógica de negocio, así como para desarrollar los servicios RESTful que gestionan las operaciones de la aplicación.

Spring Boot:

Spring Boot, un framework de desarrollo de aplicaciones Java, ha sido utilizado para simplificar la configuración y el despliegue de la aplicación. Este framework proporciona un entorno de desarrollo ágil que permite crear microservicios independientes y altamente escalables. Spring Boot facilita la creación de aplicaciones web y servicios RESTful, y ofrece características como la gestión de dependencias, la configuración automática y la integración con otras tecnologías de Spring, como Spring Security para la gestión de la seguridad.

Maven:

Maven, una herramienta de gestión de proyectos ampliamente utilizada en el ecosistema Java, ha sido empleada para gestionar las dependencias del proyecto y automatizar el proceso de compilación y despliegue. Maven simplifica la gestión de las bibliotecas y frameworks utilizados en el proyecto, garantizando la coherencia y la eficiencia en el manejo de las dependencias.

Postman:

Postman, una herramienta de colaboración para el desarrollo de APIs, ha sido utilizada para probar y documentar los servicios RESTful desarrollados con Spring Boot. Postman permite enviar solicitudes HTTP a los endpoints de la API, validar las respuestas y automatizar las pruebas para garantizar el correcto funcionamiento de los servicios. Además, ofrece características como la generación de documentación API y la colaboración entre equipos de desarrollo.

4.2 Frontend:

JavaScript:

JavaScript, un lenguaje de programación ampliamente utilizado en el desarrollo web, ha sido la base del frontend de la aplicación. JavaScript se ha utilizado en conjunto con ReactJS para desarrollar interfaces de usuario interactivas y dinámicas. La versatilidad y la compatibilidad de JavaScript con los navegadores web modernos garantizan una experiencia de usuario fluida y receptiva en la aplicación.

ReactJS:

ReactJS, una biblioteca de JavaScript desarrollada por Facebook, ha sido utilizada en el frontend de la aplicación para la construcción de componentes reutilizables y la gestión eficiente del estado de la interfaz de usuario. ReactJS utiliza un enfoque basado en componentes que facilita la modularidad y el mantenimiento de la aplicación, permitiendo un desarrollo rápido y escalable de la interfaz de usuario.

DevExtreme:

DevExtreme, un conjunto de componentes de desarrollo de interfaz de usuario (UI) desarrollado por DevExpress, ha sido utilizado en el frontend de la aplicación para mejorar la experiencia del usuario y facilitar la creación de interfaces de usuario dinámicas y receptivas. DevExtreme proporciona una amplia variedad de widgets y herramientas que permiten a los desarrolladores construir aplicaciones web avanzadas con funcionalidades ricas y un alto rendimiento.

HTML y CSS:

HTML y CSS han sido utilizados para gestionar la estructura y apariencia de la plataforma aplicando estilos propios y gestionando las apariencias y la compatibilidad de las vistas con los diferentes navegadores

4.3 Base de datos:

Microsoft SQL Server:

Microsoft SQL Server, un sistema de gestión de bases de datos relacional, ha sido utilizado para almacenar y gestionar los datos de la aplicación. SQL Server ofrece un rendimiento escalable, seguridad robusta y herramientas de administración avanzadas que facilitan el desarrollo y el despliegue de aplicaciones empresariales. Se ha elegido SQL Server por su amplia adopción en el entorno empresarial y su compatibilidad con las tecnologías utilizadas en el proyecto.

4.3.1.1 Aspecto a destacar del alumno:

Este sistema de gestión de bases de datos no fue mi elección inicial, yo estaba más familiarizado con MySQL, pero debido a muchos fallos de compatibilidad con Spring tuve que hacer el traslado a SQL server.

Pero, SQL server ha resultado no solo muy útil para realizar análisis y pruebas sino que además, realizando el TFG de ADE me topé con que en PowerBi existe la posibilidad de cargar datos desde un servidor SQL Server, lo cual da muchísimo valor a la plataforma final, ya que a parte de las funcionalidades, se genera una base de datos con posibilidades infinitas de Business Analytics y Business Intelligence.

Microsoft SQL Management Studio:

Microsoft SQL Management Studio ha sido utilizada como herramienta de administración y desarrollo para interactuar con la base de datos en SQL Server. Esta herramienta proporciona una interfaz gráfica intuitiva para ejecutar consultas SQL, realizar tareas de administración y optimizar el rendimiento de la base de datos. SQL Management Studio facilita la gestión de bases de datos en entornos de desarrollo y producción, permitiendo a los desarrolladores y administradores de bases de datos colaborar de manera eficiente en el desarrollo y mantenimiento de la aplicación.

4.4 Herramientas de desarrollo:

Visual Studio Code:

Visual Studio Code, un editor de código fuente ligero y potente, ha sido utilizado en el desarrollo de la aplicación. Visual Studio Code proporciona características avanzadas de edición de código, como resaltado de sintaxis, completado automático y depuración integrada, que mejoran la productividad del desarrollador. Además, es altamente personalizable y cuenta con una amplia variedad de extensiones que amplían su funcionalidad para el desarrollo de aplicaciones web para adaptarse a las necesidades específicas del proyecto.

5 Desarrollo

El desarrollo de la plataforma de gestión de recursos humanos se ha llevado a cabo con el objetivo de ofrecer una solución integral y accesible para pequeñas y medianas empresas (PYMEs). Desde el inicio, se ha centrado en identificar y satisfacer las necesidades específicas de estas empresas, proporcionando funcionalidades esenciales que optimizan la gestión del personal y mejoran la eficiencia operativa.

El proceso comenzó con un análisis de las funcionalidades necesarias que se establecieron anteriormente. Esta etapa incluyó la identificación de los diferentes componentes necesarios para la implementación correcta de las funcionalidades que se querían introducir.

La base y estructura de los datos fueron planificadas para garantizar que la plataforma pueda manejar de manera eficiente grandes volúmenes de información. En las secciones siguientes, se detallará la arquitectura de la base de datos, explicando la lógica detrás de la configuración de las tablas, los campos y las relaciones. Este diseño asegura que los datos se almacenen y gestionen de manera óptima, facilitando un acceso rápido y seguro a la información.

Posteriormente, se desarrollaron las funcionalidades principales de la plataforma. Cada característica fue diseñada para ser intuitiva y fácil de usar, permitiendo a los usuarios realizar sus tareas de manera eficiente. Desde la gestión del talento y la evaluación del desempeño hasta la administración de nóminas y beneficios, cada módulo de la plataforma está alineado con las mejores prácticas de la industria y las necesidades específicas de las PYMEs.

Además, se eligieron cuidadosamente las tecnologías y herramientas utilizadas en el desarrollo de la plataforma para asegurar su rendimiento, escalabilidad y facilidad de mantenimiento. Este capítulo proporcionará una visión detallada de las decisiones técnicas tomadas durante el desarrollo, incluyendo la selección de tecnologías de front-end y back-end, y cómo estas decisiones han impactado el resultado final del proyecto.

En resumen, este capítulo tiene como objetivo proporcionar una comprensión completa de la lógica y las decisiones técnicas que se llevaron a cabo para implementar la plataforma de gestión de recursos humanos. A través de un análisis detallado de su arquitectura y funcionalidades, se demostrará cómo la plataforma está diseñada para satisfacer las necesidades específicas de las PYMEs, ofreciendo una solución eficiente y efectiva para la gestión del personal.

5.1 Desarrollo de la Base de datos

Inicialmente, se consideró MySQL como la elección principal para el desarrollo de la base de datos de nuestra plataforma debido a su simplicidad y popularidad. Sin embargo, por problemas de compatibilidad, a medida que el proyecto avanzaba, se decidió utilizar SQL Server debido a sus robustas capacidades de integración con herramientas de business intelligence como Power BI. Esta decisión ha resultado ser más útil a largo plazo, proporcionando una base de datos escalable, segura y altamente eficiente que apoya la visión estratégica de la plataforma.

En la plataforma, las entidades clave incluyen:

- **Users** (Usuarios): Corresponde a los empleados de la empresa.
- **Entries** (Entradas): Corresponde a los registros de horas trabajadas por los empleados.
- **Roles:** Corresponde con el rol jerárquico dentro de la empresa (Puede o no coincidir con la posición específica) Ej: Senior Managers y Senior Manager del departamento de transformación financiera
- **Reports** (Informes): Corresponde a los informes de trabajo y evaluaciones.
- **Candidates** (Candidatos): Corresponde a los candidatos que aplican a posiciones dentro de la empresa.

5.1.1 Estructura de la Base de Datos

La base de datos está compuesta por varias entidades clave, cada una de las cuales tiene un conjunto específico de atributos. A continuación, se detallan las principales entidades y sus atributos:

5.1.1.1 Entidad “users”

name: Nombre del usuario (empleado).

email: Correo electrónico del usuario.

password: Contraseña del usuario, almacenada de manera segura.

position: Posición o título del usuario dentro de la empresa.

status: Estado del usuario (activo, inactivo, etc.).

username: Nombre de usuario para el inicio de sesión.

role_id: Referencia al rol del usuario.

description: Descripción adicional del usuario.

solicited_position: Posición solicitada por el usuario, si aplica.

5.1.1.2 Entidad “roles”

role_id: Identificador único del rol.

description: Descripción del rol.

5.1.1.3 Entidad “entries”

entry_id: Identificador único del registro de la jornada.
datetime_end: Fecha y hora de finalización del registro de jornada.
datetime_start: Fecha y hora de inicio del registro de jornada.
hours: Número de horas trabajadas en la jornada.
user_name: Nombre del usuario asociado con el objeto en específico.

5.1.1.4 Entidad “report”

report_id: Identificador único del informe.
datetime: Fecha y hora del informe.
description: Descripción del empleado.
evaluation: Evaluación o comentario en el informe.
month_hours_worked: Horas trabajadas en el mes.
user_name: Nombre del usuario asociado con el informe.
week_hours: Horas a la semana por contrato.
week_hours_worked: Horas totales trabajadas en la semana.

5.1.1.5 Entidad “candidates”

name: Nombre del candidato.
description: Descripción del candidato.
email: Correo electrónico del candidato.
solicited_position: Posición solicitada por el candidato.

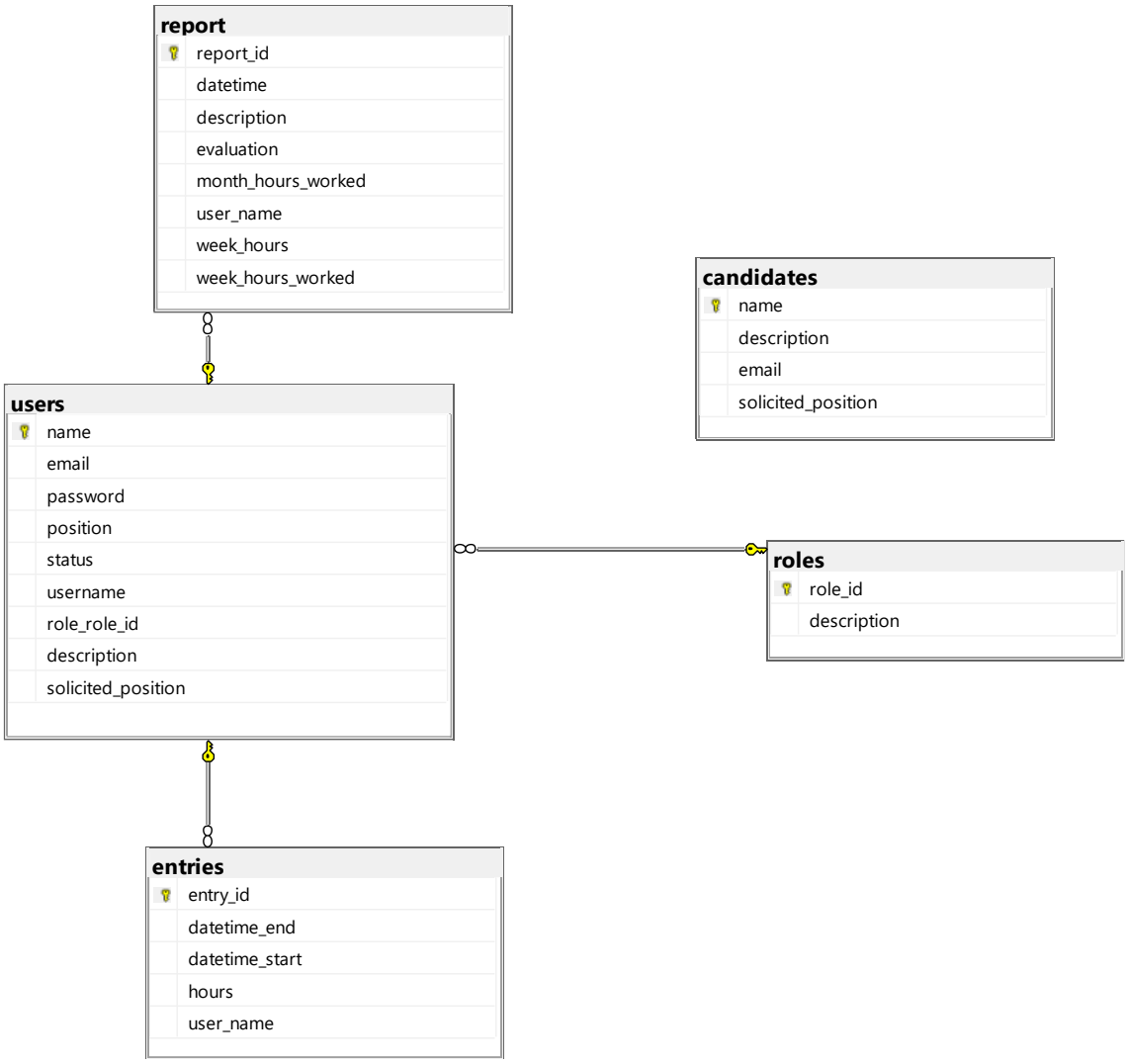


Ilustración 1: Diagrama entidad-relación de la base de datos

5.1.2 Diseño de la Base de Datos

La base de datos está diseñada para garantizar la integridad referencial y la eficiencia en el acceso a los datos. Las relaciones entre las entidades están definidas mediante claves foráneas que aseguran la coherencia de los datos.

5.1.2.1 Relaciones Clave:

Relación entre “users” y “roles”:

Descripción:

Un usuario puede tener un rol específico dentro de la empresa.

Implementación:

La entidad “users” contiene el campo “role_id”, que es una clave foránea que referencia al campo “role_id” en la entidad “roles”.

Relación entre “users” y “entries”:

Descripción: Un usuario puede tener múltiples registros de horas trabajadas para distintas jornadas.

Implementación: La entidad “entries” incluye el campo “user_name”, que referencia al campo name” en “users”, estableciendo así una relación uno a muchos.

Relación entre “users” y “report”:

Descripción:

Un usuario puede estar asociado a múltiples informes de trabajo y evaluaciones.

Implementación:

La entidad “report” utiliza el campo “user_name” para asociar cada informe con un usuario específico, también en una relación uno a muchos.

“candidates”:

Descripción:

Un candidato puede ser evaluado y luego convertirse en usuario (empleado) en la plataforma.

Implementación:

Aunque no hay una relación directa entre las entidades “candidates” y las demás, en el modelo actual, esta relación se puede gestionar a nivel de aplicación para facilitar la transición de candidatos a empleados.

5.1.3 Ventajas del Uso de SQL Server

Desarrollar la base de datos en SQL Server ofrece numerosas ventajas, especialmente en el contexto de business intelligence (BI):

Escalabilidad y Rendimiento:

SQL Server es conocido por su capacidad para manejar grandes volúmenes de datos y ofrecer un rendimiento consistente. Esto es crucial para nuestra plataforma, que necesita almacenar y procesar grandes cantidades de información de usuarios y registros de trabajo.

Seguridad:

SQL Server proporciona robustas características de seguridad, incluyendo autenticación y autorización avanzadas, cifrado de datos y auditoría, garantizando la protección de información sensible.

Alta Disponibilidad y Recuperación ante Desastres:

Con características como Always On Availability Groups y backups automáticos, SQL Server asegura la alta disponibilidad y recuperación rápida en caso de fallos. [6]

Integración con Power BI:

Una de las mayores ventajas de usar SQL Server es su integración nativa con Power BI, una potente herramienta de BI que permite transformar los datos en visualizaciones interactivas y dashboards intuitivos.

5.1.3.1 Integración con Power BI

La capacidad de integrar SQL Server con Power BI permite aprovechar al máximo los datos almacenados en la base de datos. Algunas de las posibilidades que ofrece esta integración incluyen:

Dashboards en Tiempo Real:

Los datos de SQL Server pueden ser actualizados en tiempo real en Power BI, proporcionando a los gestores una visión instantánea del rendimiento y la eficiencia operativa.

Análisis Avanzado: Power BI permite realizar análisis complejos y crear visualizaciones sofisticadas que pueden identificar tendencias, patrones y áreas de mejora.

Acceso a Información Centralizada: Con Power BI, es posible consolidar información de múltiples fuentes en un único dashboard, facilitando la toma de decisiones informadas.

5.1.4 Recapitulación

El desarrollo de la base de datos de nuestra plataforma en SQL Server no solo asegura una gestión eficiente y segura de la información, sino que también abre la puerta a avanzadas capacidades de business intelligence mediante la integración con Power BI. Esta combinación permite transformar los datos operativos en valiosas percepciones estratégicas, apoyando la toma de decisiones y mejorando la eficiencia general de la gestión de recursos humanos. La decisión de optar por SQL Server en lugar de MySQL ha resultado en una plataforma más robusta y adaptable a las necesidades de nuestras empresas clientes, garantizando un futuro de crecimiento y éxito.

5.2 Back End

El backend de la plataforma ha sido implementado utilizando una arquitectura de microservicios, diseñada para proporcionar una mayor flexibilidad, escalabilidad y facilidad de mantenimiento. Esta arquitectura modular nos permite separar las distintas funcionalidades del sistema en servicios independientes, cada uno responsable de una tarea específica. Esto no solo mejora la eficiencia operativa, sino que también facilita la integración de nuevas características y la resolución de problemas sin afectar a todo el sistema.

En la arquitectura de microservicios de nuestra plataforma, se han definido cuatro microservicios principales, cada uno con responsabilidades claramente delineadas:

- **Reader:** Este microservicio se encarga de gestionar todas las peticiones de lectura (GET) a la base de datos. Actúa como el punto de acceso para cualquier consulta de datos, garantizando respuestas rápidas y eficientes a las solicitudes de los usuarios.

- **Writter**: Este microservicio maneja las peticiones de modificación y escritura (POST, PUT, DELETE) en la base de datos. Se ocupa de la inserción, actualización y eliminación de datos, asegurando que todas las operaciones de escritura se realicen de manera segura y coherente.

- **Gateway**: El Gateway actúa como un broker central que unifica todas las peticiones definidas en los diferentes microservicios. Es el punto de entrada principal para todas las solicitudes de la plataforma, distribuyendo las peticiones al microservicio adecuado y simplificando la comunicación entre los componentes del sistema.

- **Auth**: Este microservicio gestiona la autorización de acceso a todos los microservicios cuando se realizan peticiones de login desde la plataforma. Utiliza un sistema de tokens para renovar las autorizaciones de manera segura, asegurando que solo los usuarios autenticados puedan acceder a los recursos y funcionalidades del sistema.

Cada uno de estos microservicios ha sido diseñado para operar de manera autónoma, comunicándose entre sí mediante APIs RESTful. Esta independencia permite que cada microservicio pueda ser desarrollado, desplegado y escalado de forma independiente, lo que reduce la complejidad y mejora la capacidad de respuesta del sistema a los cambios y demandas de los usuarios.

En este apartado, se explorará en detalle la implementación de cada uno de estos microservicios, incluyendo las tecnologías utilizadas, las decisiones de diseño, y cómo se integran para formar un backend cohesivo y eficiente. También se discutirá el papel crucial del Gateway en la orquestación de peticiones y la importancia del Auth en la gestión de la seguridad y la autenticación en la plataforma.

Al final de este apartado, se habrá proporcionado una visión completa de cómo la arquitectura de microservicios ha sido implementada para soportar la funcionalidad robusta y escalable de la plataforma de gestión de recursos humanos.

5.2.1 Ventajas de la arquitectura basada en microservicios frente a la arquitectura monolítica clásica

La elección de una arquitectura basada en microservicios en lugar de una arquitectura monolítica ofrece numerosas ventajas que son esenciales para el desarrollo y la operación eficiente de nuestra plataforma de gestión de recursos humanos y empleados. A continuación, se detallan algunas de las principales ventajas de adoptar una arquitectura de microservicios:

5.2.1.1 Escalabilidad

Microservicios:

- Escalabilidad Independiente**:** Cada microservicio puede escalarse de manera independiente en función de su demanda. Por ejemplo, si el microservicio de lectura (Reader) experimenta un alto volumen de solicitudes, puede escalarse sin afectar a los microservicios de escritura (Writer) o autenticación (Auth).

- Uso Eficiente de Recursos: Permite una utilización más eficiente de los recursos del servidor, ya que solo los componentes que necesitan escalarse lo hacen, evitando el escalado innecesario de todo el sistema.

Monolítica:

- Escalabilidad Global: La escalabilidad debe aplicarse a toda la aplicación, lo que puede resultar en un uso ineficiente de los recursos y costos adicionales.

- Limitaciones de Rendimiento: A medida que la aplicación crece, se hace más difícil y costoso escalar de manera efectiva, lo que puede limitar el rendimiento general.

5.2.1.2 Flexibilidad y Mantenimiento

Microservicios:

- Despliegue Independiente: Los microservicios pueden desarrollarse, desplegarse y actualizarse de forma independiente. Esto permite una mayor agilidad en el desarrollo y la implementación de nuevas funcionalidades o correcciones.

- Reducción del Tiempo de Inactividad: Las actualizaciones o cambios en un microservicio específico no requieren el despliegue completo de la aplicación, lo que reduce el tiempo de inactividad y mejora la continuidad del servicio.

- Fácil Mantenimiento: Los equipos de desarrollo pueden trabajar en diferentes microservicios simultáneamente sin interferencias, mejorando la productividad y facilitando el mantenimiento.

Monolítica:

- Despliegue Completo: Cualquier cambio o actualización requiere desplegar toda la aplicación, lo que puede ser costoso y propenso a errores.

- Mayor Complejidad: A medida que la aplicación crece, se vuelve más compleja y difícil de mantener, con un riesgo aumentado de introducir errores al modificar o añadir funcionalidades.

5.2.1.3 Tolerancia a Fallos

Microservicios:

- Aislamiento de Fallos: Si un microservicio falla, no afecta necesariamente al resto del sistema. Esto permite una mayor resiliencia, ya que los fallos pueden ser contenidos y gestionados más fácilmente.

- Manejo de Errores: Los microservicios pueden diseñarse para manejar errores y recuperarse de fallos sin interrumpir el funcionamiento global de la plataforma.

Monolítica:

- Propagación de Fallos: Un fallo en cualquier parte de la aplicación monolítica puede potencialmente derribar todo el sistema, lo que resulta en una menor resiliencia.

- Difícil Diagnóstico: Identificar y aislar fallos es más difícil en aplicaciones monolíticas debido a su naturaleza interdependiente y compleja.

Tecnologías y Lenguajes

Microservicios:

- Diversidad Tecnológica: Los microservicios pueden ser desarrollados utilizando diferentes tecnologías y lenguajes de programación que sean más adecuados para cada tarea específica. Esto permite una mayor flexibilidad y la posibilidad de utilizar las herramientas más avanzadas y apropiadas para cada componente.

- Evolución Tecnológica: La arquitectura de microservicios facilita la adopción de nuevas tecnologías sin necesidad de reescribir toda la aplicación, permitiendo una evolución continua y más rápida.

Monolítica:

- Tecnología Uniforme: Está restringida a un único stack tecnológico para toda la aplicación, lo que puede limitar la flexibilidad y la capacidad de adoptar nuevas tecnologías.

- Actualizaciones Complejas: La actualización o el cambio de tecnología en una aplicación monolítica suele ser un proceso complejo y arriesgado.

5.2.1.4 Desarrollo y Productividad

Microservicios:

- Equipos Autónomos: Permite que equipos de desarrollo más pequeños y autónomos trabajen en diferentes microservicios de manera simultánea, aumentando la velocidad de desarrollo y la innovación.

- Ciclo de Vida Rápido: Facilita ciclos de desarrollo, prueba y despliegue más rápidos, lo que se traduce en una mejora continua y ágil del producto.

Monolítica:

- Dependencias Interconectadas: Los equipos de desarrollo deben coordinarse estrechamente debido a la interdependencia de los componentes, lo que puede ralentizar el desarrollo.

- Mayor Complejidad: La complejidad y el tamaño del código pueden dificultar la gestión y el desarrollo rápido de nuevas funcionalidades.

5.2.2 Evaluación y recapitulación

La adopción de una arquitectura de microservicios para el backend de nuestra plataforma de gestión de recursos humanos y empleados proporciona una serie de ventajas significativas en términos de escalabilidad, flexibilidad, resiliencia, tecnología y productividad. Estas ventajas son cruciales para soportar el crecimiento y la evolución de la plataforma, asegurando que pueda adaptarse rápidamente a las necesidades cambiantes de las empresas y proporcionar un servicio robusto y eficiente. Comparada con una arquitectura monolítica, la arquitectura de microservicios permite una mejor gestión de los recursos y una mayor capacidad de respuesta, lo que se traduce en una plataforma más ágil y competitiva.

5.3 Estructura general de los microservicios

En la implementación del backend de nuestra plataforma, se ha utilizado Spring Boot como el framework principal para desarrollar los microservicios. Spring Boot proporciona una estructura robusta y modular que facilita la creación de microservicios independientes y escalables. Cada microservicio en el sistema sigue una estructura consistente que incluye tres carpetas principales: "external", "model" y "repo". Esta organización asegura que el código sea mantenible y fácilmente comprensible, permitiendo una rápida evolución y expansión de la plataforma. Más tarde, en cada apartado individual de cada microservicio se hablará sobre especificaciones particulares de cada uno.

5.3.1 Estructura de Carpetas de los Microservicios

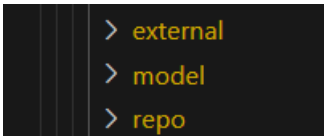


Ilustración 2: Carpetas principales de los microservicios

1. "external":

- Descripción: Esta carpeta contiene las clases que gestionan las peticiones REST. Aquí es donde se definen los controladores REST que exponen los endpoints de la API. Los controladores manejan las solicitudes HTTP (GET, POST, PUT, DELETE) y se comunican con las capas de servicio y repositorio para cumplir con las operaciones solicitadas.

2. "model":

- Descripción: Esta carpeta contiene las clases que representan los objetos del modelo de datos. Estos objetos están mapeados a las tablas de la base de datos y definen los atributos y relaciones entre ellos. Cada clase de modelo corresponde a una entidad en la base de datos y define los campos y relaciones necesarios para gestionar los datos de manera estructurada.

3. "repo":

- Descripción: Esta carpeta contiene las interfaces de los repositorios que extienden de "JpaRepository" o "CrudRepository". Estas interfaces definen métodos para realizar operaciones CRUD (crear, leer, actualizar, eliminar) y consultas personalizadas. Los repositorios son responsables de interactuar directamente con la base de datos, proporcionando una capa de abstracción para el acceso y manipulación de datos.

5.3.2 Funcionamiento de Spring Boot

Spring Boot es un framework que simplifica el desarrollo de aplicaciones Java basadas en Spring, proporcionando configuraciones predeterminadas y herramientas que facilitan la creación de aplicaciones robustas y escalables. Algunas características clave de Spring Boot incluyen:

- Configuración Automática: Spring Boot ofrece una configuración automática que detecta las dependencias en el classpath y configura automáticamente los componentes del framework.

- Servidor Embebido: Permite ejecutar la aplicación en un servidor embebido como Tomcat eliminando la necesidad de desplegar la aplicación en un servidor externo.

- Dependencias Gestionadas: A través de "Spring Boot Starters", se gestionan las dependencias necesarias para desarrollar la aplicación, simplificando la inclusión de librerías comunes.

- Monitorización y Métricas: Ofrece soporte para la monitorización y recopilación de métricas mediante Actuator, permitiendo el seguimiento del rendimiento y la salud de la aplicación.

Cada microservicio en la plataforma tiene un archivo "application.properties" que se utiliza para definir la configuración específica del microservicio. Este archivo incluye parámetros críticos como:

- Puerto del Servidor: El puerto en el que el microservicio escucha las peticiones HTTP. Esto permite ejecutar múltiples microservicios en diferentes puertos sin conflictos.

Ej:

```
server.port=8080
```

- URL de la Base de Datos: La conexión a la base de datos, incluyendo el host, el puerto, y el nombre de la base de datos.

```
spring.datasource.url=jdbc:sqlserver://localhost:8080/TFG
```

- Credenciales de la Base de Datos: El nombre de usuario y la contraseña necesarios para conectarse a la base de datos.

```
spring.datasource.username=root
spring.datasource.password=secret
```

- Otras Configuraciones: Pueden incluir configuraciones de seguridad, gestión de logs, propiedades específicas de los microservicios, entre otros.

```
logging.level.org.springframework=DEBUG
spring.security.user.name=admin
spring.security.user.password=admin
```

5.3.3 Recapitulación y Evaluación

La estructura modular y consistente de los microservicios basada en Spring Boot facilita el desarrollo y mantenimiento de la plataforma. Al separar las responsabilidades en diferentes componentes (“external”, “model”, “repo”), se mejora la claridad del código y se permite una escalabilidad más eficiente. La configuración centralizada mediante el archivo “application.properties” asegura que cada microservicio pueda ser configurado de manera precisa y flexible, adaptándose a las necesidades específicas del entorno de despliegue y mejorando la capacidad de gestión y monitoreo del sistema.

5.4 Reader y Writter

En el desarrollo de la arquitectura de microservicios de nuestra plataforma, se ha adoptado un enfoque modular que facilita la escalabilidad y el mantenimiento del sistema. Dos de los componentes clave en esta arquitectura son el microservicio Reader y el microservicio Writer. Aunque estos microservicios tienen responsabilidades distintas, comparten una base común en términos de repositorios y modelos de datos, lo que asegura la coherencia y la reutilización del código.

Ambos microservicios utilizan los mismos repositorios definidos en la carpeta "repo" y los mismos objetos de datos en la carpeta "model". La diferencia principal entre ellos radica en su funcionalidad específica, reflejada en las clases definidas en la carpeta "external".

Reader: Este microservicio se encarga exclusivamente de las operaciones de lectura (GET) sobre la base de datos. Gestiona todas las consultas y devoluciones de datos, proporcionando una interfaz para que otras partes del sistema obtengan la información necesaria.

Ilustración 3: Banner de ejecución del Reader

Writer: Este microservicio, por otro lado, maneja todas las operaciones de escritura, eliminación y modificación de datos (POST, PUT, DELETE). Es responsable de insertar nuevos registros, actualizar los existentes y eliminar aquellos que ya no son necesarios.

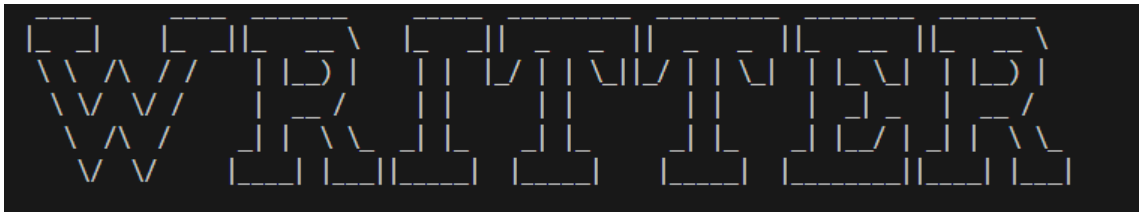


Ilustración 4: Banner de ejecución del Writter

La separación de responsabilidades entre el Reader y el Writer no solo mejora la organización del código, sino que también permite optimizar el rendimiento y la seguridad del sistema. Al aislar las operaciones de lectura de las de escritura, podemos escalar cada microservicio de manera independiente según la carga de

trabajo específica, y aplicar políticas de seguridad más precisas para cada tipo de operación.

En este apartado, se explorará en detalle cómo se estructuran y operan estos microservicios, enfatizando su uso compartido de repositorios y modelos, así como sus implementaciones específicas en las clases de la carpeta “external”. También se discutirá cómo esta separación de responsabilidades contribuye a la robustez y eficiencia de nuestra plataforma, permitiendo un desarrollo más ágil y una gestión más eficaz de los recursos.

5.4.1 Recursos compartidos

Como se ha mencionado antes ambos microservicios comparten los recursos de la carpeta “repo” y la carpeta “model”, esto se debe principalmente a que ambos microservicios han de mantener para repositorios y entidades una consistencia férrea ya que de lo contrario esto da lugar a incompatibilidades, errores de base de datos, fallos de ejecución y fallos de funcionalidades. A continuación, se muestran los recursos que se comparten en el “model” y en el “repo”:

5.4.1.1 Model

En ambas carpetas “model” existen 5 objetos para crear entidades de bases de datos: User.java, Entries.java, Report.java, Candidate.java y Role.java. Por el nombre se sabe qué objeto se encarga de crear cada entidad pero vamos a echar un vistazo más a fondo a cada una de ellas:

5.4.1.1.1 User.java

```java
@Entity
@Table(name = "users")
@JsonIdentityInfo(generator = ObjectIdGenerators.PropertyGenerator.class, resolver = EntityIdResolver.class, property = "name", scope = User.class)
public class User {




    @JsonIdentityReference(alwaysAsId = true)
    @ManyToOne(fetch = FetchType.LAZY, cascade = CascadeType.ALL)
    @JdbcTypeCode(java.sql.Types.VARCHAR)
    private Role role;

    private String username;

    @Id
    private String name;

    private String position;


    private String password;
    private String email;
    @JdbcTypeCode(java.sql.Types.INTEGER)
    private Status status;

    public User() {

    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
```

Ilustración 5: Contenidos del objeto User

La imagen proporcionada muestra la definición de la clase “User” en Java. Esta clase representa la entidad “User” en la base de datos y contiene varios atributos y anotaciones que definen su estructura y comportamiento. A continuación, se describe en detalle cada parte del código:

Anotaciones de Clase

@Entity: Indica que esta clase es una entidad JPA (Java Persistence API) que se mapeará a una tabla en la base de datos.

@Table(name = "users"): Especifica el nombre de la tabla en la base de datos a la que se mapeará esta entidad. En este caso, la tabla se llama “users”.

@JsonIdentityInfo: Utilizada para manejar la serialización y deserialización de objetos con referencias cíclicas. Define un generador y un resolvedor de identidad para la propiedad “name” en el contexto de la clase “User”.

Atributos de Clase

- **role:** Define una relación de muchos a uno (ManyToOne) con la entidad "Role". Las anotaciones adicionales especifican que la relación se carga de manera perezosa (lazy) y que todas las operaciones en cascada se aplican a esta relación.
  @JsonIdentityReference(alwaysAsId = true): Indica que se debe usar la identidad del objeto "Role" en la serialización JSON.
  @ManyToOne(fetch = FetchType.LAZY, cascade = CascadeType.ALL): Define la relación de muchos a uno con carga perezosa y operaciones en cascada.
  @JdbcTypeCode(java.sql.Types.VARCHAR): Especifica el tipo de dato SQL para la columna en la base de datos.

- **username:** Cadena de texto que representa el nombre de usuario. Este campo se utiliza para el inicio de sesión y otras operaciones relacionadas con el usuario.

- **name:** Cadena de texto que representa el nombre del usuario. Este campo está anotado con "@Id", lo que indica que es la clave primaria de la entidad "User".

- **position:** Cadena de texto que representa la posición o título del usuario dentro de la empresa.

- **password:** Cadena de texto que almacena la contraseña del usuario. Es crucial que este campo se maneje con seguridad para proteger la información sensible.

- **email:** Cadena de texto que almacena la dirección de correo electrónico del usuario.

- **status:** Enum o tipo de dato específico que representa el estado del usuario. La anotación "@JdbcTypeCode(java.sql.Types.INTEGER)" especifica que este campo se almacena como un entero en la base de datos (1=IN, 0=OUT).

Métodos de Clase

Constructor "User": Constructor por defecto de la clase "User".

Métodos Getters y Setters: Métodos públicos que permiten acceder y modificar los atributos de la clase. Por ejemplo, "getUsername" y "setUsername" permiten acceder y modificar el atributo "username".

La clase “User” define una entidad clave dentro del modelo de datos de la plataforma. Al compartir esta definición entre los microservicios Reader y Writer, se garantiza la coherencia de los datos y la reutilización del código. Las anotaciones de JPA y JSON utilizadas en esta clase facilitan la integración con la base de datos y la serialización/deserialización de objetos, respectivamente.

5.4.1.1.2 Role.java

```java
@Entity
@Table(name = "roles")
@JsonIdentityInfo(generator = ObjectIdGenerators.PropertyGenerator.class, resolver = EntityIdResolver.class, property = "roleId", scope = Role.class)
public class Role {

    @Id
    private String roleId;

    @Column(name = "description")
    private String description;

    @ElementCollection
    private List<Privilege> privileges;


    public String getDescription() {
        return description;
    }
```

Ilustración 6: Contenidos del objeto Rol

La imagen proporcionada muestra la definición de la clase “Role” en Java, que se encuentra dentro de las carpetas “model” de los microservicios Reader y Writer.

Anotaciones de Clase: Vamos a obviar esta parte ya que las anotaciones de Role.java están explicadas también en User.java

Atributos de Clase

- **roleId:** Este campo es la clave primaria de la entidad “Role”. Está anotado con “@Id” para indicar que es el identificador único de cada instancia de “Role”
- **description:** Cadena de texto que proporciona una descripción del rol. Está anotado con “@Column(name = "description")” para especificar el nombre de la columna correspondiente en la base de datos.
- **privileges:** Lista de privilegios asociados con el rol. La anotación “@ElementCollection” indica que esta es una colección de elementos, que se almacenan en una tabla separada pero están relacionados con la entidad “Role”.

Métodos de Clase: constructor de objeto, getters y setters.

5.4.1.1.3 Entries.java

```java
@Entity
public class Entries {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "entry_id")
    private int entryId;

    @JsonIdentityReference(alwaysAsId = true)
    @ManyToOne(fetch = FetchType.LAZY, cascade = CascadeType.ALL)
    @JdbcTypeCode(java.sql.Types.VARCHAR)
    private User user;



    @Column(name = "hours")
    private String hours;

    @Column(name = "datetime_start")
    private Timestamp datetimeStart;

    @Column(name = "datetime_end")
    private Timestamp datetimeEnd;

    public Entries() {
    }
```

Ilustración 7: Contenidos del objeto Entries

La imagen proporcionada muestra la definición de la clase “Entries” en Java, que se encuentra dentro de las carpetas “model” de los microservicios Reader y Writer.

Anotaciones de Clase: Vamos a obviar esta parte ya que las anotaciones de Role.java están explicadas también en User.java

Atributos de Clase

- **entryId:** Este campo es la clave primaria de la entidad “Entries”. Está anotado con “@Id” para indicar que es el identificador único de cada instancia de “Entries”. La anotación “@GeneratedValue(strategy = GenerationType.AUTO)” especifica que el valor de este campo será generado automáticamente.

- **user**: Este atributo representa una relación de muchos a uno (ManyToOne) con la entidad “User”. La anotación “@JdbcTypeCode(java.sql.Types.VARCHAR)” especifica el tipo de dato SQL para la columna en la base de datos.

- **hours:** Cadena de texto que representa las horas registradas en esta entrada. Está anotado con “@Column(name = "hours")” para especificar el nombre de la columna correspondiente en la base de datos.

- **datetimeStart**: Marca de tiempo que representa el inicio del período registrado en esta entrada. Está anotado con “@Column(name = "datetime_start")” para especificar el nombre de la columna correspondiente en la base de datos.

- **datetimeEnd**: Marca de tiempo que representa el final del período registrado en esta entrada. Está anotado con “@Column(name = "datetime_end")” para especificar el nombre de la columna correspondiente en la base de datos.

Métodos de Clase: Constructor de objeto, getters y setters.

5.4.1.1.4 Report.java

```java
@Entity
public class Report {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "report_id")
    private int reportId;

    @JsonIdentityReference(alwaysAsId = true)
    @ManyToOne(fetch = FetchType.LAZY, cascade = CascadeType.ALL)
    @JdbcTypeCode(java.sql.Types.VARCHAR)
    private User user;

    @Column(name = "monthHoursWorked")
    private String monthHoursWorked;

    @Column(name = "description")
    private String description;

    @Column(name = "datetime")
    private Timestamp datetime;

    @Column(name = "evaluation")
    private String evaluation;

    @Column(name = "weekHoursWorked")
    private String weekHoursWorked;

    @Column(name = "weekHours")
    private String weekHours;
```

Ilustración 8: Contenidos del objeto Entries

La imagen proporcionada muestra la definición de la clase “Report” en Java, que se encuentra dentro de las carpetas “model” de los microservicios Reader y Writer.

Anotaciones de Clase: Vamos a obviar esta parte ya que las anotaciones de “Role.java” están explicadas también en “User.java”.

Atributos de Clase:

- **reportId:** Este campo es la clave primaria de la entidad “Report”. Está anotado con “@Id” para indicar que es el identificador único de cada instancia de “Report”.

- **user:** Este atributo representa una relación de muchos a uno (ManyToOne) con la entidad “User”.

- **monthHoursWorked:** Cadena de texto que representa las horas trabajadas en el mes. Está anotado con “@Column(name = "monthHoursWorked")” para especificar el nombre de la columna correspondiente en la base de datos.

- **description:** Cadena de texto que proporciona una descripción del reporte. Está anotado con “@Column(name = "description")” para especificar el nombre de la columna correspondiente en la base de datos.

- **datetime:** Marca de tiempo que representa el momento en que se generó el reporte. Está anotado con “@Column(name = "datetime")” para especificar el nombre de la columna correspondiente en la base de datos.

- **evaluation:** Cadena de texto que contiene la evaluación para el empleado asociada al reporte. Está anotado con “@Column(name = "evaluation")” para especificar el nombre de la columna correspondiente en la base de datos.

- **weekHoursWorked:** Cadena de texto que representa las horas trabajadas en la semana. Está anotado con “@Column(name = "weekHoursWorked")” para especificar el nombre de la columna correspondiente en la base de datos.

- **weekHours:** Cadena de texto que representa las horas por contrato por semana. Está anotado con “@Column(name = "weekHours")” para especificar el nombre de la columna correspondiente en la base de datos.

Métodos de Clase: Constructor de objeto, getters y setters.

5.4.1.1.5 Candidate.java

```java
@Entity
@Table(name = "candidates")
@JsonIdentityInfo(generator = ObjectIdGenerators.PropertyGenerator.class, resolver = EntityIdResolver.class, property = "name", scope = Candidate.class)
public class Candidate {

    @Id
    private String name;

    private String solicitedPosition;

    private String description;



    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    private String email;

    public Candidate() {

    }
```

Ilustración 9: Contenidos del objeto Candidate

La imagen proporcionada muestra la definición de la clase “Candidate” en Java, que se encuentra dentro de las carpetas “model” de los microservicios Reader y Writer.

Anotaciones de Clase: Vamos a obviar esta parte ya que las anotaciones de “Role.java” están explicadas también en “User.java”.

Atributos de Clase

- **name:** Este campo es la clave primaria de la entidad “Candidate”. Está anotado con “@Id” para indicar que es el identificador único de cada instancia de “Candidate”.

- **solicitedPosition:** Cadena de texto que representa la posición solicitada por el candidato. Está anotado con “@Column(name = "solicitedPosition")”

para especificar el nombre de la columna correspondiente en la base de datos.

- **description:** Cadena de texto que proporciona una descripción del candidato o de la posición solicitada. Está anotado con "@Column(name = "description")" para especificar el nombre de la columna correspondiente en la base de datos.

- **email:** Cadena de texto que representa el correo electrónico del candidato.

Métodos de Clase: Constructor de objeto, getters y setters

5.4.1.2 Repo

En esta sección, se describen las interfaces de repositorio que se encuentran en la carpeta "repo" de los microservicios Reader y Writer. Estas interfaces son esenciales para la interacción con la base de datos, ya que proporcionan métodos para realizar operaciones CRUD (Create, Read, Update, Delete) sobre las entidades. A continuación, se detallan las interfaces "IUserRepo", "ICandidateRepo", "IEntryRepo", "IReportRepo" e "IRolRepo" junto con sus métodos y funcionalidades específicas.

5.4.1.2.1 IUserRepo

```
public interface IUserRepo extends JpaRepository<User, String> {
    @Query(value = """
            SELECT password,status,username,name,position,role_role_id,email
            FROM users order by name asc""", nativeQuery = true)
    public List<Object[]> findUsers();

    public User findByUsername(String username);

    public User findByName(String name);


    public boolean existsByUsername(String username);

    public boolean existsByName(String name);

}
```

Ilustración 10: Contenidos de la interfaz IUserRepo

La interfaz "IUserRepo" extiende "JpaRepository" y proporciona métodos para interactuar con la entidad "User". Además de los métodos heredados de "JpaRepository", esta interfaz define métodos personalizados para encontrar

usuarios por nombre de usuario y nombre, verificar la existencia de usuarios por nombre de usuario y nombre, y ejecutar una consulta nativa para listar usuarios.

Métodos:

- "**findUsers()":** Ejecuta una consulta nativa para listar usuarios con atributos específicos.
- **"findByUsername(String username**)": Encuentra un usuario por nombre de usuario.
- **"findByName(String name)":** Encuentra un usuario por nombre.
- **"existsByUsername(String username)":** Verifica si un usuario existe por nombre de usuario.
- **-"existsByName(String name)":** Verifica si un usuario existe por nombre.

5.4.1.2.2 ICandidateRepo

```
public interface ICandidateRepo extends JpaRepository<Candidate, String> {
    public Candidate findByName(String name);

    boolean existsByName(String name);

    public Integer deleteByName(String name);
}
```

Ilustración 11: Contenidos de la interfaz ICandidateRepo

La interfaz "ICandidateRepo" extiende "JpaRepository" y proporciona métodos para interactuar con la entidad "Candidate". Además de los métodos heredados de "JpaRepository", esta interfaz define métodos personalizados para encontrar, verificar y eliminar candidatos por nombre.

Métodos

- **"findByName(String name)":** Encuentra un candidato por nombre.
- **"existsByName(String name)":** Verifica si un candidato existe por nombre.
- **"deleteByName(String name)":** Elimina un candidato por nombre.

5.4.1.2.3 IEntryRepo

```java
public interface IEntryRepo extends JpaRepository<Entries, Integer> {
    public List<Entries> findByUser(User user);

    boolean existsByUser(User User);

    public Integer deleteByUser(User user);
}
```

Ilustración 12: Contenidos de la interfaz IEntryRepo

La interfaz "IEntryRepo" extiende "JpaRepository" y proporciona métodos para interactuar con la entidad "Entries". Además de los métodos heredados de "JpaRepository", esta interfaz define métodos personalizados para encontrar, verificar y eliminar entradas por usuario.

Métodos

- **"findByUser(User user)":** Encuentra entradas por usuario.
- **"existsByUser(User user)":** Verifica si existen entradas para un usuario específico.
- **"deleteByUser(User user)":** Elimina entradas por usuario.

5.4.1.2.4 IReportRepo

```java
public interface IReportRepo extends JpaRepository<Report, Integer> {
    public Report findByUser(User user);

    boolean existsByUser(User User);

    public Integer deleteByUser(User user);
}
```

Ilustración 13: Contenidos de la interfaz IReportRepo

La interfaz "IReportRepo" extiende "JpaRepository" y proporciona métodos para interactuar con la entidad "Report". Además de los métodos heredados de "JpaRepository", esta interfaz define métodos personalizados para encontrar, verificar y eliminar reportes por usuario.

Métodos

- **"findByUser(User user)":** Encuentra un reporte por usuario.
- **"existsByUser(User user)":** Verifica si existen reportes para un usuario específico.
- **"deleteByUser(User user)":** Elimina reportes por usuario.

5.4.1.2.5 IRolRepo

```
public interface IRolRepo extends JpaRepository<Role, String> {
    @Query(value = """
        SELECT role_id,description FROM roles order by role_id asc""", nativeQuery = true)
    public List<Object[]> findRoles();

}
```

Ilustración 12: Contenidos de la interfaz IRolRepo

La interfaz "IRolRepo" extiende "JpaRepository" y proporciona métodos para interactuar con la entidad "Role". Además de los métodos heredados de "JpaRepository", esta interfaz define un método personalizado para listar roles mediante una consulta nativa.

Métodos

- **"findRoles()":** Ejecuta una consulta nativa para listar roles ordenados por ID de rol.

Estas interfaces de repositorio son fundamentales para el acceso y manipulación de los datos almacenados en la base de datos. Proporcionan una capa de abstracción que facilita las operaciones CRUD y mejora la mantenibilidad del código, permitiendo a los desarrolladores centrarse en la lógica de negocio sin preocuparse por los detalles de la interacción con la base de datos.

5.4.2 Peticiones HTTP del Reader

Este apartado se enfoca en la implementación de las peticiones HTTP que han programado en el Reader, específicamente en los archivos "CandidateREST.java", "EntriesREST.java", "ReportREST.java", "RolREST.java" y "UserREST.java" que se encuentran dentro de la carpeta "external". Estos archivos gestionan las operaciones de lectura (GET) en la base de datos. A continuación, se describen en detalle las peticiones HTTP y su funcionalidad en cada uno de estos archivos.

5.4.2.1 CandidateREST.java

El archivo "CandidateREST.java" es responsable de gestionar las operaciones de lectura relacionadas con la entidad "Candidate". Este archivo contiene los métodos para manejar las solicitudes HTTP GET que permiten recuperar la información de los candidatos desde la base de datos.

GET /candidates: Este endpoint devuelve una lista de todos los candidatos en la base de datos. Utiliza el método "findAll" del repositorio "ICandidateRepo" para obtener todos los registros de la entidad "Candidate".

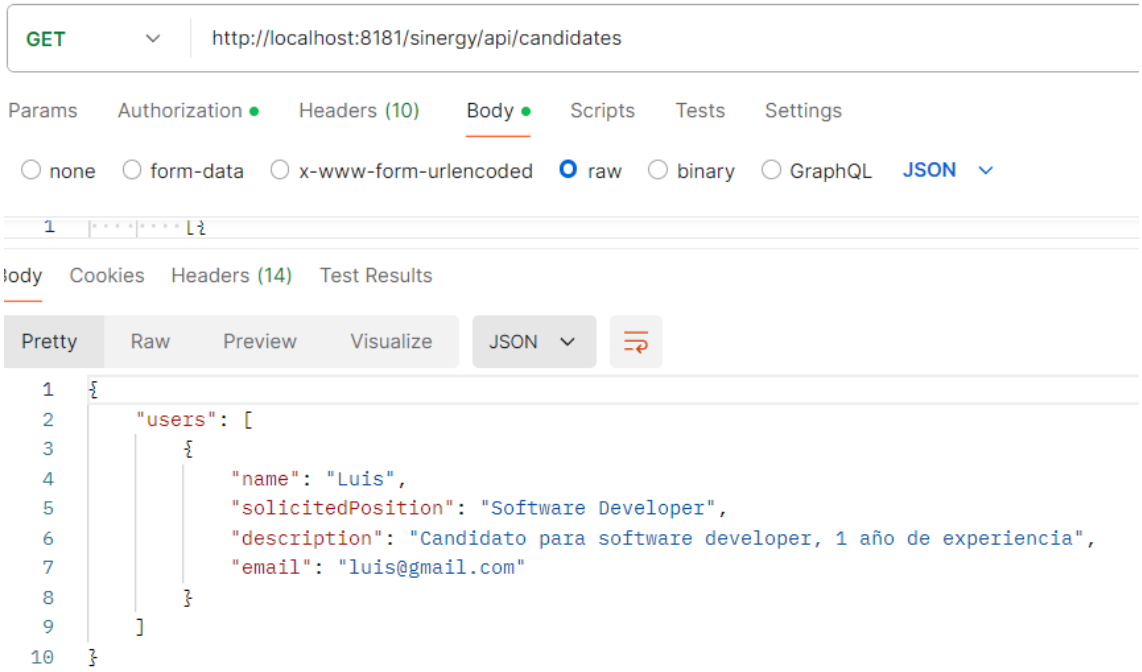


Ilustración 13: Petición exitosa de PostMan para GET candidates

GET /candidates/{name}: Este endpoint devuelve los detalles de un candidato específico basado en su nombre. Utiliza el método "findByName" del repositorio "ICandidateRepo" para buscar y devolver un candidato por su nombre.

5.4.2.2 EntriesREST.java

El archivo "EntriesREST.java" se encarga de las operaciones de lectura para la entidad "Entries". Aquí se implementan los métodos que manejan las solicitudes HTTP GET relacionadas con los registros de entradas de horas trabajadas.

GET /entries: Este endpoint devuelve una lista de todas las entradas de horas trabajadas registradas en la base de datos. Utiliza el método "findAll" del repositorio "IEntryRepo" para obtener todos los registros de la entidad "Entries".

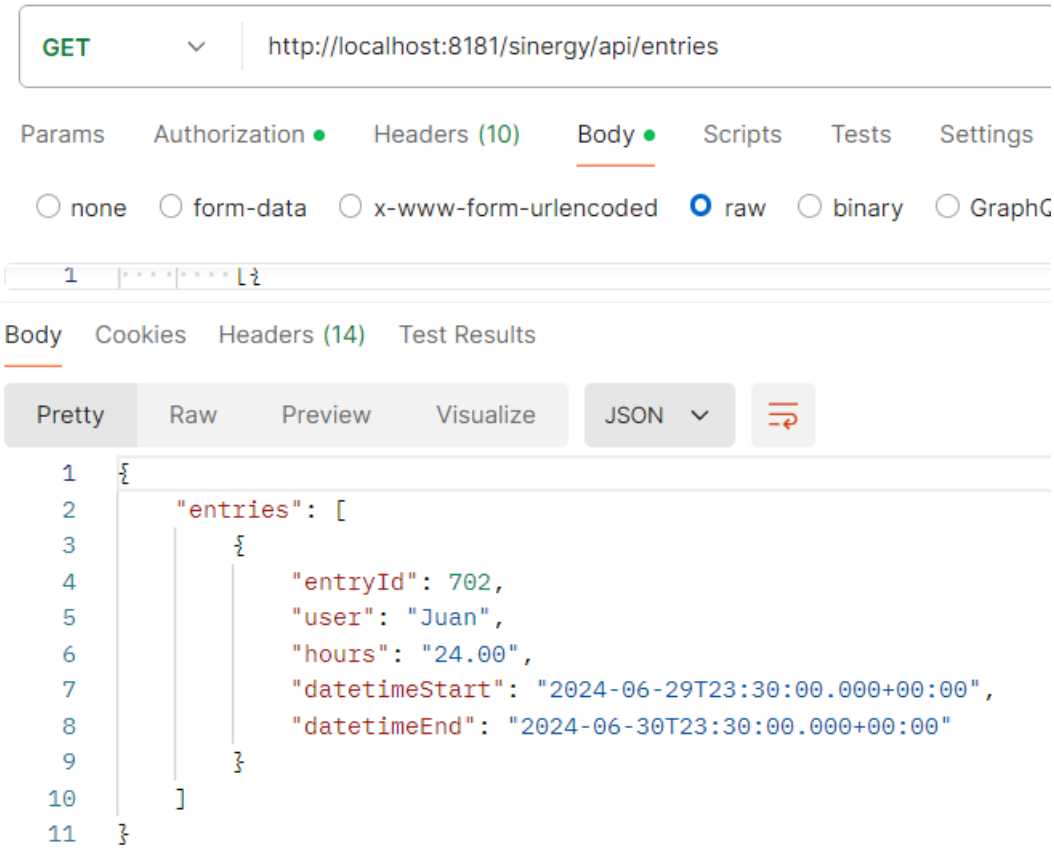


Ilustración 14: Petición exitosa de PostMan para GET entries

GET /entries/user/{user}: Este endpoint devuelve las entradas de horas trabajadas de un usuario específico. Utiliza el método "findByUser" del repositorio "IEntryRepo" para buscar y devolver las entradas asociadas a un usuario específico.

5.4.2.3 ReportREST.java

El archivo "ReportREST.java" gestiona las operaciones de lectura para la entidad "Report". Incluye métodos para manejar las solicitudes HTTP GET que permiten recuperar la información de los reportes generados.

GET /reports: Este endpoint devuelve una lista de todos los reportes en la base de datos. Utiliza el método "findAll" del repositorio "IReportRepo" para obtener todos los registros de la entidad "Report".

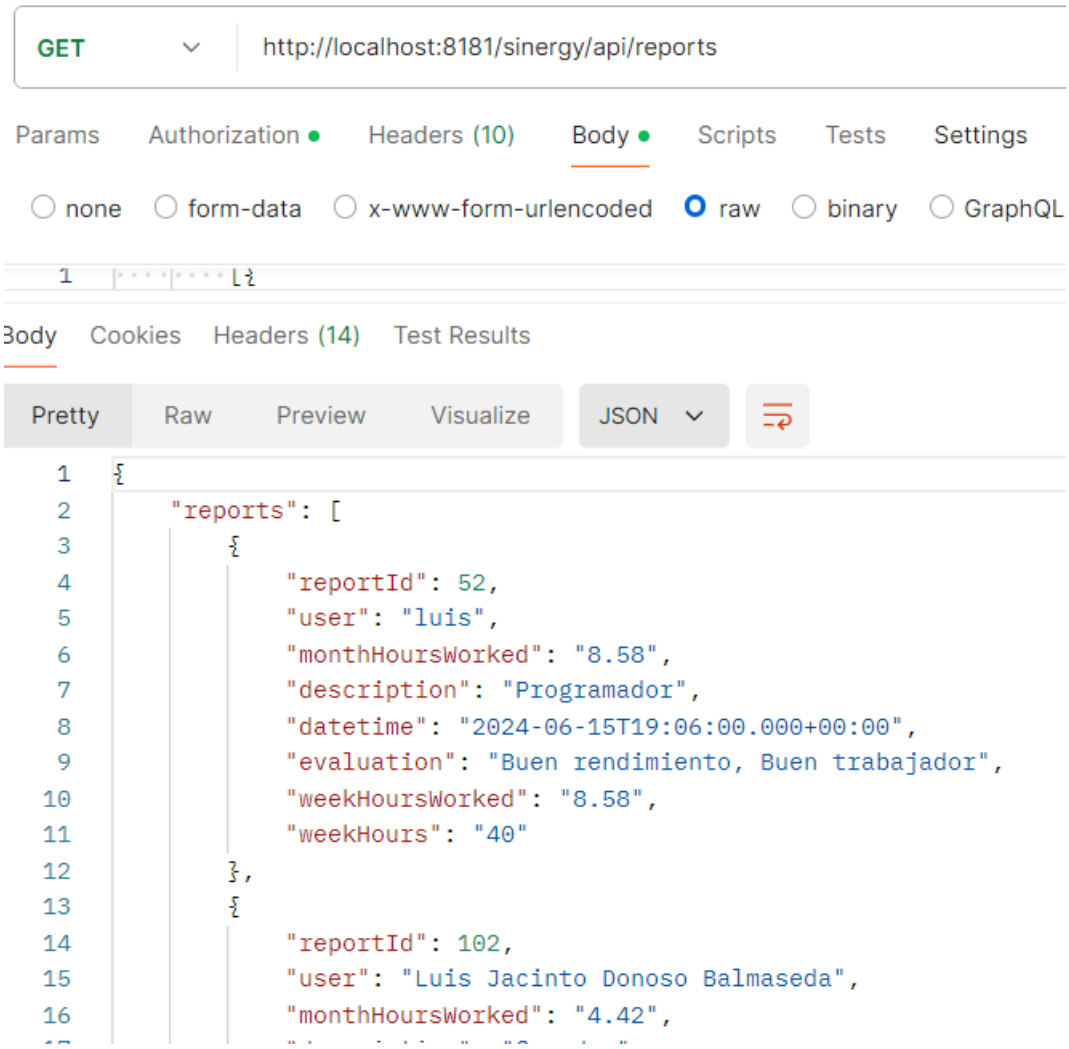


Ilustración 15: Petición exitosa de PostMan para GET reports

GET /reports/user/{user}: Este endpoint devuelve los reportes de un usuario específico. Utiliza el método "findByUser" del repositorio "IReportRepo" para buscar y devolver los reportes asociados a un usuario en particular.

5.4.2.4 RolREST.java

El archivo "RolREST.java" se ocupa de las operaciones de lectura relacionadas con la entidad "Role". Este archivo contiene los métodos para manejar las solicitudes HTTP GET que permiten recuperar la información de los roles desde la base de datos.

GET /roles: Este endpoint devuelve una lista de todos los roles en la base de datos. Utiliza el método "findRoles" del repositorio "IRolRepo" para obtener todos los registros de la entidad "Role".

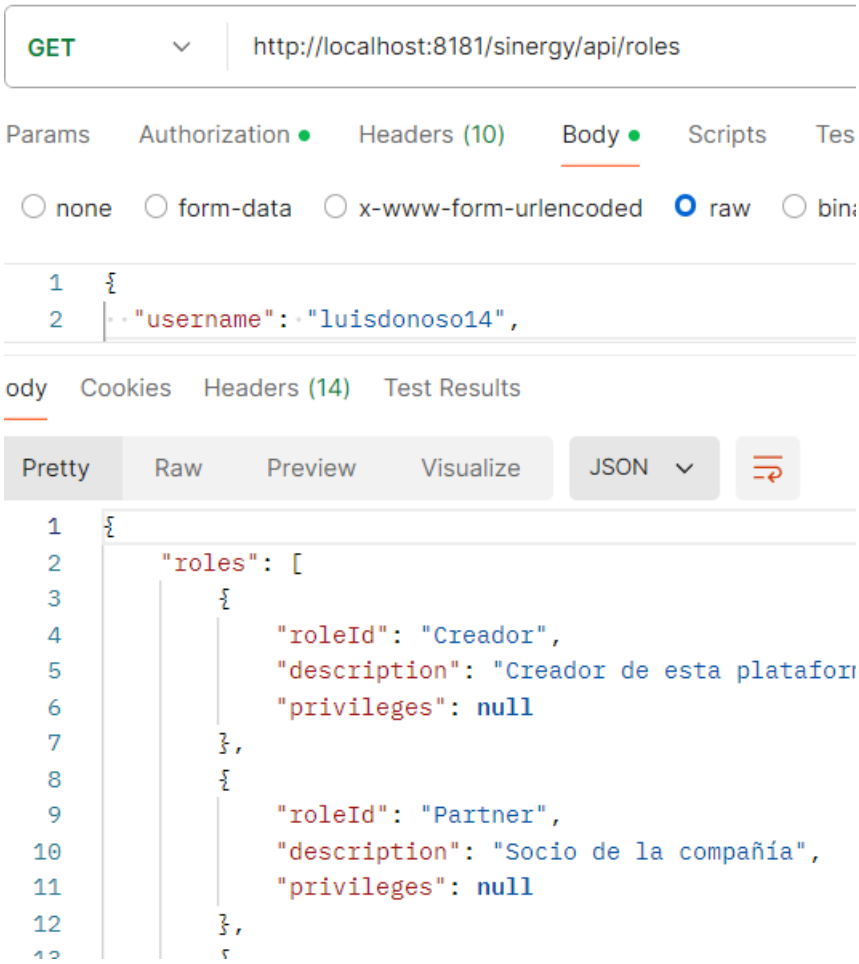


Ilustración 16: Petición exitosa de PostMan para GET roles

5.4.2.5 UserREST.java

El archivo "UserREST.java" es responsable de las operaciones de lectura para la entidad "User". Aquí se implementan los métodos que manejan las solicitudes HTTP GET relacionadas con los usuarios de la plataforma.

GET /users: Este endpoint devuelve una lista de todos los usuarios en la base de datos. Utiliza el método "findUsers" del repositorio "IUserRepo" para obtener todos los registros de la entidad "User".

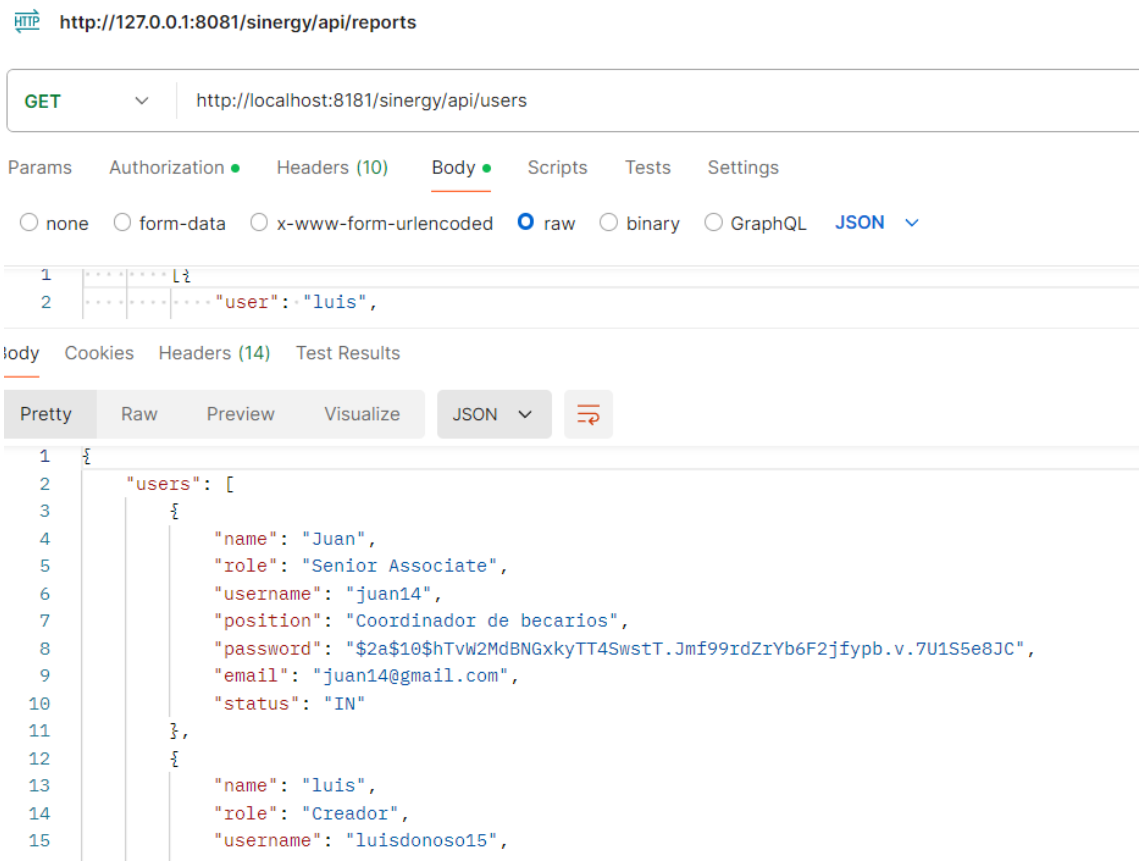


Ilustración 17: Petición exitosa de Get Users usando PostMan

GET /users/{username}: Este endpoint devuelve los detalles de un usuario específico basado en su nombre de usuario. Utiliza el método "findByUsername" del repositorio "IUserRepo" para buscar y devolver un usuario por su nombre de usuario.

GET /users/{name}: Este endpoint devuelve los detalles de un usuario específico basado en su nombre. Utiliza el método "findByName" del repositorio "IUserRepo" para buscar y devolver un usuario por su nombre.

POST /login y GET /renew?user={username}:

El endpoint POST /login y el GET /renew en UserREST son responsables de gestionar las solicitudes de inicio de sesión de los usuarios. Estos endpoint verifican las credenciales proporcionadas (nombre de usuario y contraseña) y, si son correctas, genera un token de autenticación que permite al usuario acceder a otros recursos protegidos en la plataforma. En el caso de renew se comprueba el token anterior en el body y si es correcto se renueva. El POST /login tiene algo más de complejidad. A continuación, se describe en detalle el funcionamiento de este endpoint:

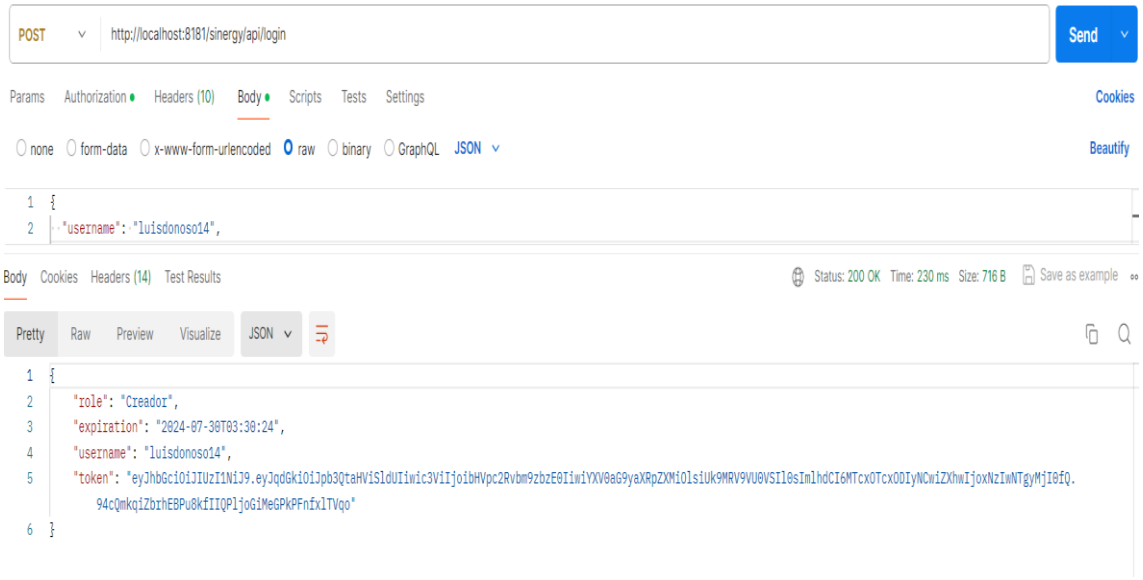


Ilustración 18: Petición exitosa de PostMan para POST Login

- **Proceso de Autenticación**

Recepción de Credenciales:

El endpoint recibe una solicitud HTTP POST que incluye el nombre de usuario y la contraseña del usuario en el cuerpo de la solicitud. Estos datos generalmente se envían en formato JSON.

Validación de Credenciales:

El método correspondiente en UserREST extrae el nombre de usuario y la contraseña de la solicitud y los valida contra los datos almacenados en la base de datos.

Se utiliza el repositorio IUserRepo para buscar el usuario por su nombre de usuario (findByUsername).

Verificación de Contraseña:

Si el usuario existe, se compara la contraseña proporcionada con la contraseña almacenada. Esta comparación se usa con técnicas de cifrado y descifrado para proteger la seguridad de la base de datos.

Generación de Token:

Si las credenciales son válidas, se genera un token de autenticación (por ejemplo, un JWT - JSON Web Token).

Este token contiene información sobre el usuario y tiene una fecha de expiración para asegurar que no sea reutilizado indefinidamente.

Respuesta de la Solicitud:

El endpoint responde con un código de estado HTTP 200 (OK) y el token de autenticación en el cuerpo de la respuesta si las credenciales son correctas.

Si las credenciales no son válidas, responde con un código de estado HTTP 401 (Unauthorized) y un mensaje de error.

5.4.3 Peticiones HTTP del Writter

Los microservicios Writter se encargan de gestionar las peticiones de modificación y escritura (POST, PUT, DELETE) en la base de datos. Estos microservicios tienen estructuras similares a las del Reader, pero se enfocan en cambiar los datos en lugar de simplemente leerlos. A continuación, se describen las peticiones HTTP programadas en cada uno de los archivos REST de estos microservicios.

5.4.3.1 CandidateREST.java

POST /candidates

- Descripción: Crea un nuevo candidato en la base de datos.
- Funcionalidad: Recibe los datos del candidato en el cuerpo de la solicitud y los guarda en la base de datos.
- Proceso:

  Validación de los datos del candidato (método propio checkParameters) que .

  Uso del repositorio "ICandidateRepo" para guardar el nuevo candidato.

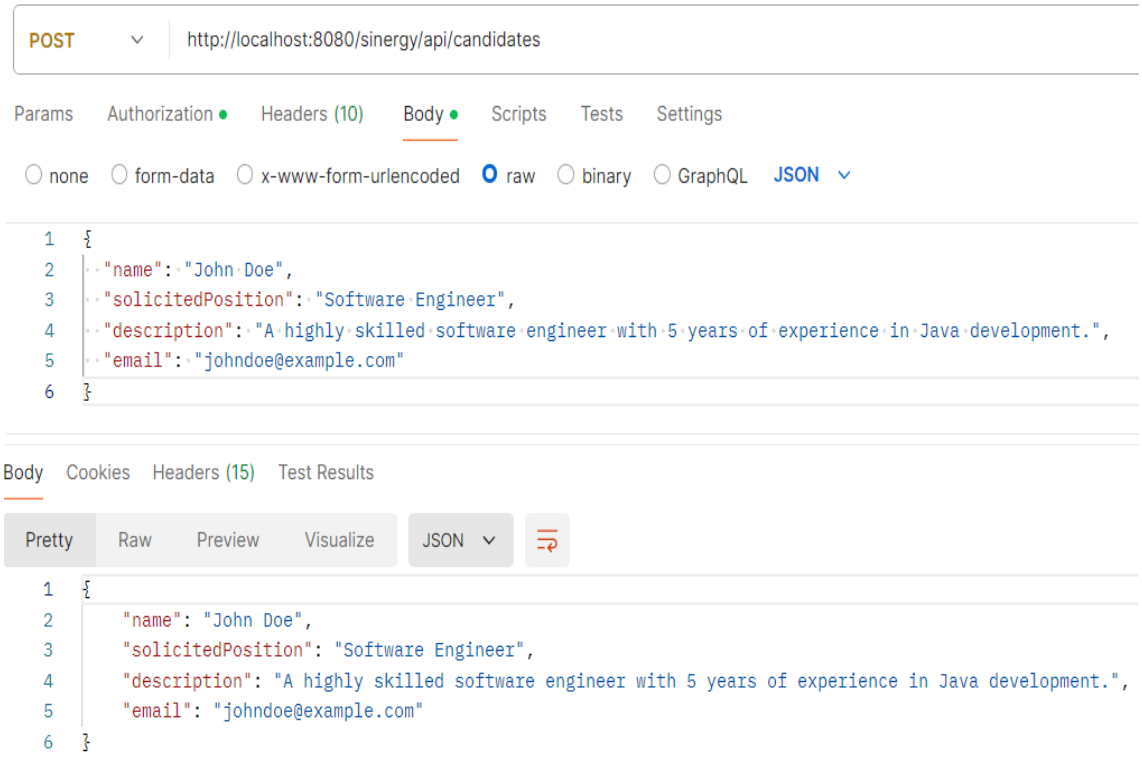


Ilustración 19: Petición exitosa de PostMan para POST Candidate

PUT /candidates/{name}

- Descripción: Actualiza los detalles de un candidato existente identificado por su nombre.
- Funcionalidad: Recibe los datos actualizados en el cuerpo de la solicitud y actualiza el registro correspondiente en la base de datos.
- Proceso:

  Validación de los datos actualizados.

  Uso del repositorio "ICandidateRepo" para buscar y actualizar el candidato.

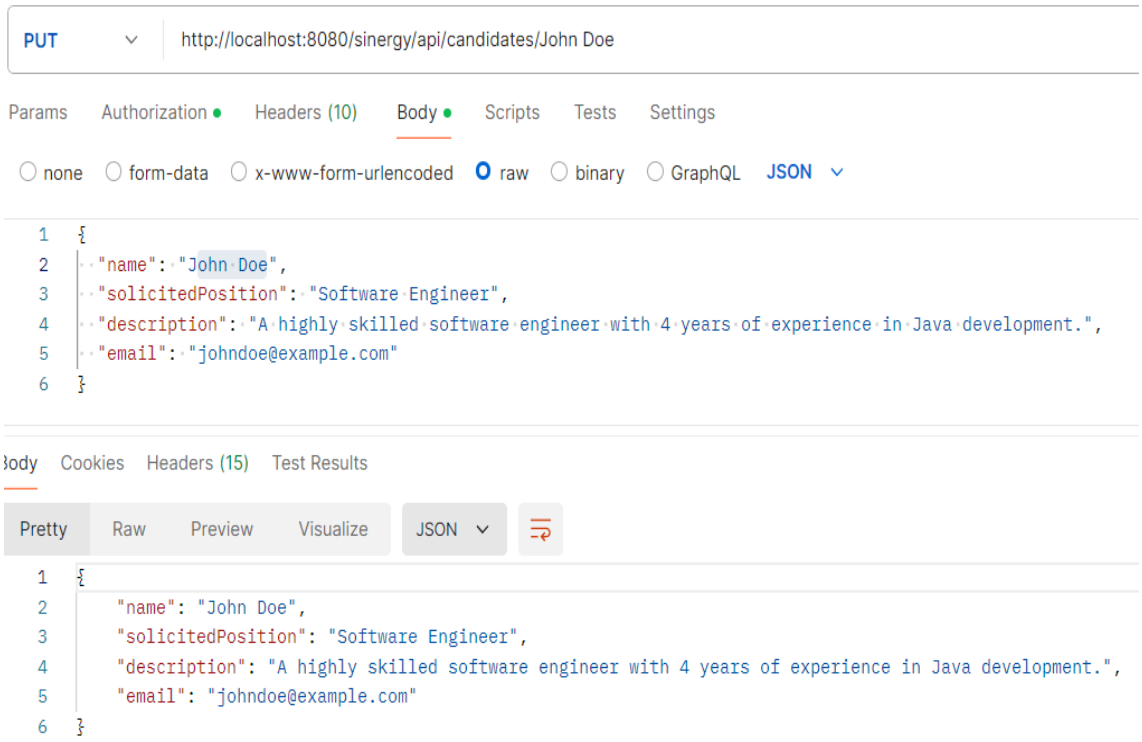


Ilustración 20: Petición exitosa de PostMan para PUT Candidate

DELETE /candidates/{name}

- Descripción: Elimina un candidato de la base de datos identificado por su nombre.
- Funcionalidad: Elimina el registro correspondiente en la base de datos.
- Proceso:

  Uso del repositorio "ICandidateRepo" para eliminar el candidato.

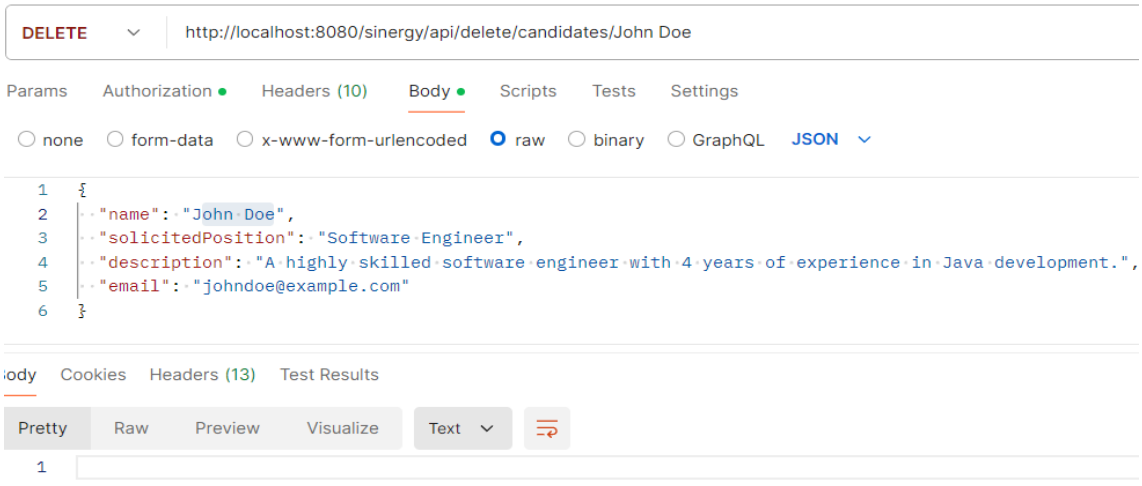


Ilustración 21: Petición exitosa de PostMan para DELETE Candidate

5.4.3.2 EntriesREST.java

POST /entries

- Descripción: Crea una nueva entrada de horas trabajadas.
- Funcionalidad: Recibe los datos de la entrada en el cuerpo de la solicitud y los guarda en la base de datos.
- Proceso:

  Validación de los datos de la entrada: En el caso de las "entries", al comprobar los parámetros, si no se ha pasado un parámetro "hours" en el cuerpo de la petición, calcula e introduce automáticamente el dato en la base de datos restando la hora de entrada y la de salida si ambas han sido proporcionadas.

  Uso del repositorio "IEntryRepo" para guardar la nueva entrada.

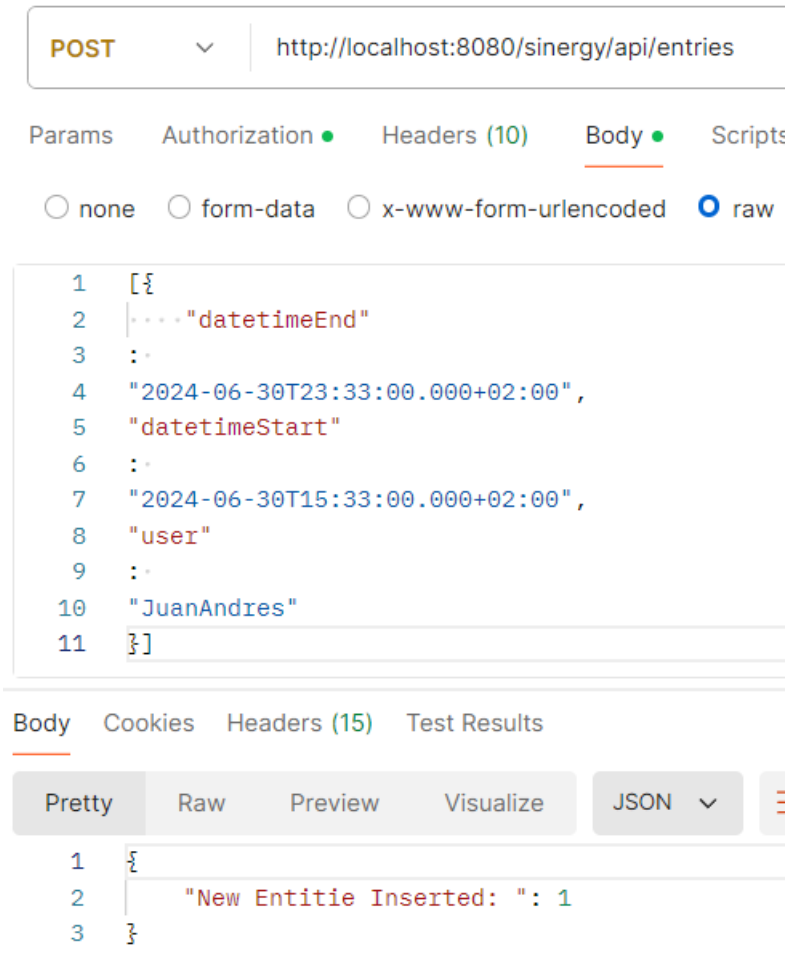


Ilustración 22: Petición exitosa de PostMan para POST Entrie

Seguidamente se comprueba que aun no habiendo pasado parámetro "hours" , la plataforma ha usado las horas de entrada y de salida y lo ha calculado:

```
17      {
18          "entryId": 753,
19          "user": "JuanAndres",
20          "hours": "8.00",
21          "datetimeStart": "2024-06-30T13:33:00.000+00:00",
22          "datetimeEnd": "2024-06-30T21:33:00.000+00:00"
23      }
```


Ilustración 23: Petición exitosa de PostMan para GET Entrie modificada

PUT /entries/{entryId}

- Descripción: Actualiza los detalles de una entrada existente identificada por su ID.
- Funcionalidad: Recibe los datos actualizados en el cuerpo de la solicitud y actualiza el registro correspondiente en la base de datos.
- Proceso:

  Validación de los datos de la entrada: En el caso de las "entries", al comprobar los parámetros, si no se ha pasado un parámetro "hours" en el cuerpo de la petición, calcula e introduce automáticamente el dato en la base de datos restando la hora de entrada y la de salida si ambas han sido proporcionadas.

  Uso del repositorio "IEntryRepo" para buscar y actualizar la entrada.

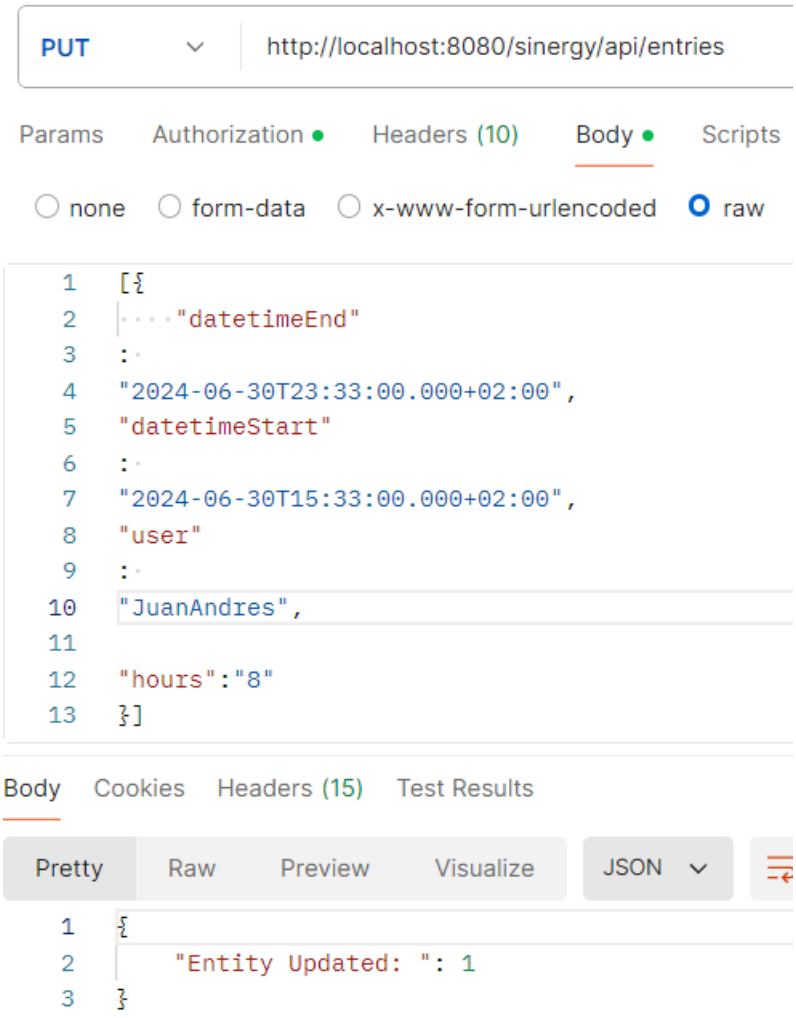


Ilustración 24: Petición exitosa de PostMan para PUT Entrie

DELETE /entries/{entryId}

- Descripción: Elimina una entrada de horas trabajadas identificada por su ID.
- Funcionalidad: Elimina el registro correspondiente en la base de datos.
- Proceso:

  Uso del repositorio "IEntryRepo" para eliminar la entrada.

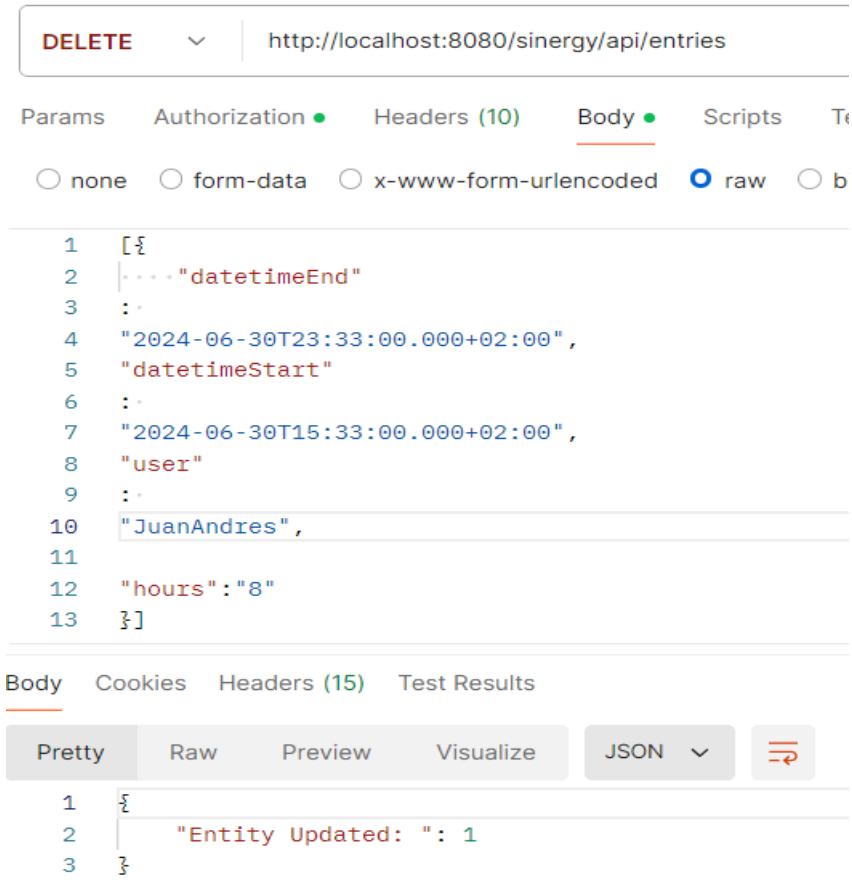


Ilustración 24: Petición exitosa de PostMan para DELETE Entrie

5.4.3.3 ReportREST.java

POST /reports

- Descripción: Crea un nuevo reporte de horas trabajadas.
- Funcionalidad: Recibe los datos del reporte en el cuerpo de la solicitud y los guarda en la base de datos.
- Proceso:

  Validación de los datos del reporte.

  Uso del repositorio "IReportRepo" para guardar el nuevo reporte.

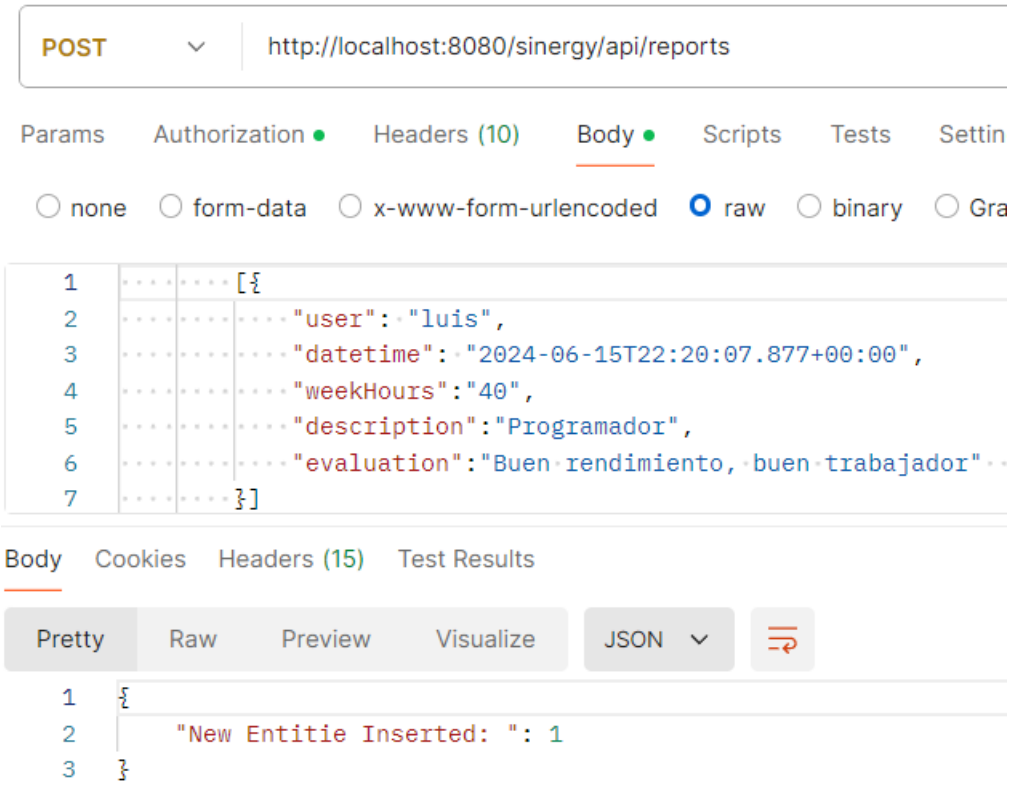


Ilustración 25: Petición exitosa de PostMan para POST Report

PUT /reports/

- Descripción: Actualiza los detalles de un reporte existente identificado por su ID.
- Funcionalidad: Recibe los datos actualizados en el cuerpo de la solicitud y actualiza el registro correspondiente en la base de datos.
- Proceso:

  Validación de los datos actualizados.

  Uso del repositorio "IReportRepo" para buscar y actualizar el reporte.

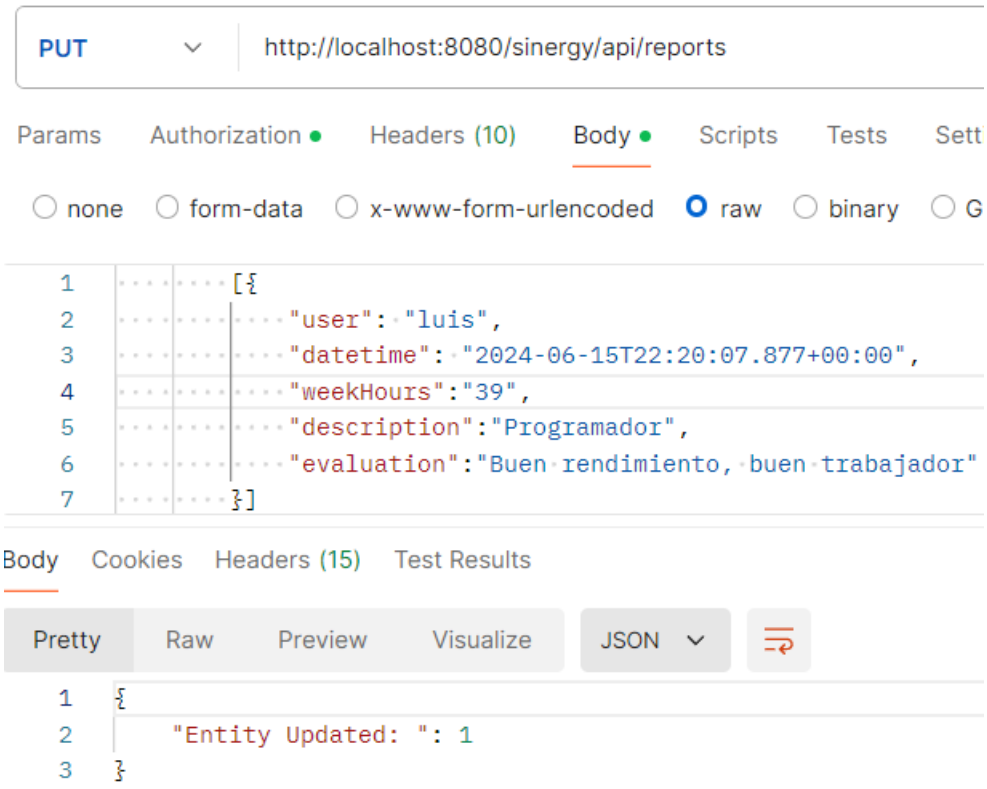


Ilustración 26: Petición exitosa de PostMan para PUT Report

DELETE /reports/?user={usuario}

- Descripción: Elimina un reporte de horas trabajadas identificado por su ID.
- Funcionalidad: Elimina el registro correspondiente en la base de datos.
- Proceso:

Uso del repositorio "IReportRepo" para eliminar el reporte.

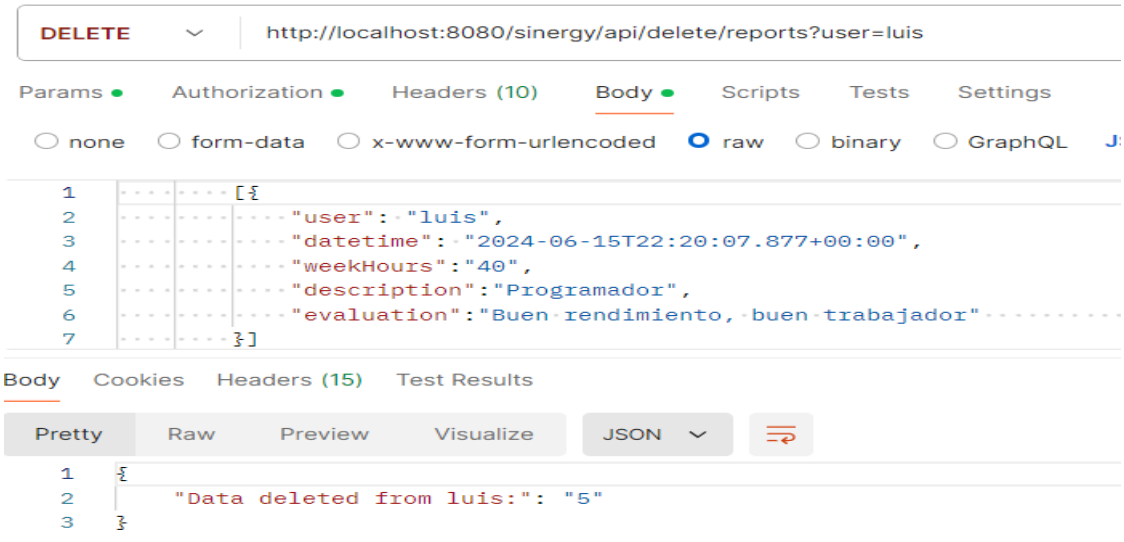


Ilustración 27: Petición exitosa de PostMan para DELETE Report

5.4.3.4 RoleREST.java

POST /roles

- Descripción: Crea un nuevo rol en la base de datos.
- Funcionalidad: Recibe los datos del rol en el cuerpo de la solicitud y los guarda en la base de datos.
- Proceso:

  Validación de los datos del rol.

  Uso del repositorio "IRolRepo" para guardar el nuevo rol.

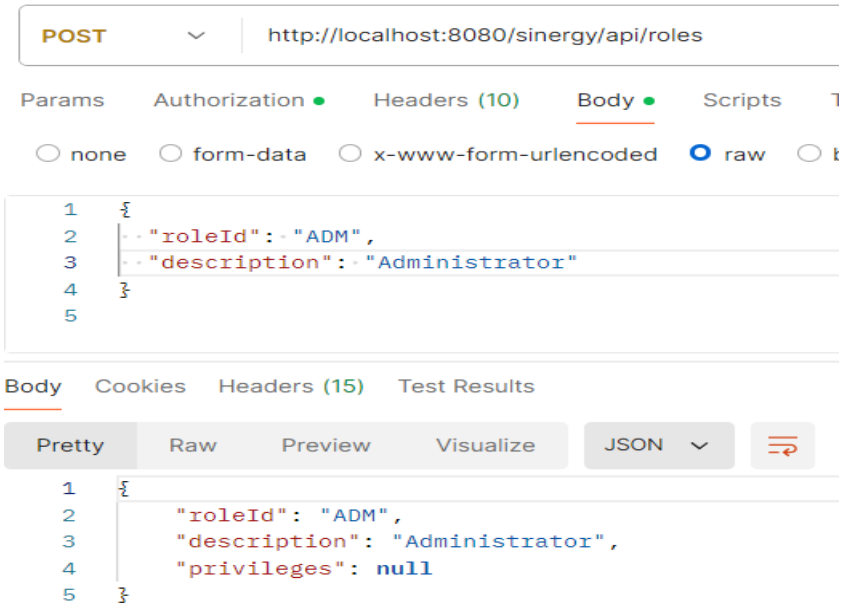


Ilustración 28: Petición exitosa de PostMan para POST Role

PUT /roles/{roleId}

- Descripción: Actualiza los detalles de un rol existente identificado por su ID.
- Funcionalidad: Recibe los datos actualizados en el cuerpo de la solicitud y actualiza el registro correspondiente en la base de datos.
- Proceso:

  Validación de los datos actualizados.

  Uso del repositorio "IRolRepo" para buscar y actualizar el rol.

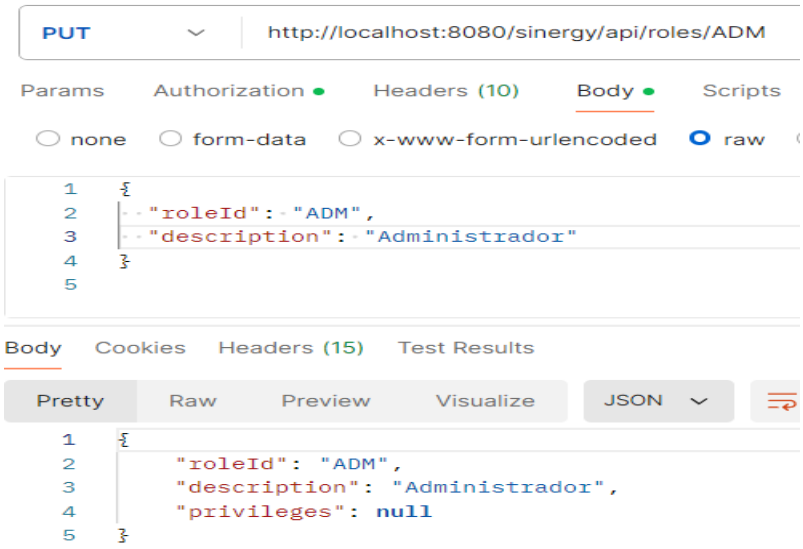


Ilustración 29: Petición exitosa de PostMan para PUT Role

DELETE /roles/{roleId}

- Descripción: Elimina un rol de la base de datos identificado por su ID.
- Funcionalidad: Elimina el registro correspondiente en la base de datos.
- Proceso:

  Uso del repositorio "IRolRepo" para eliminar el rol.

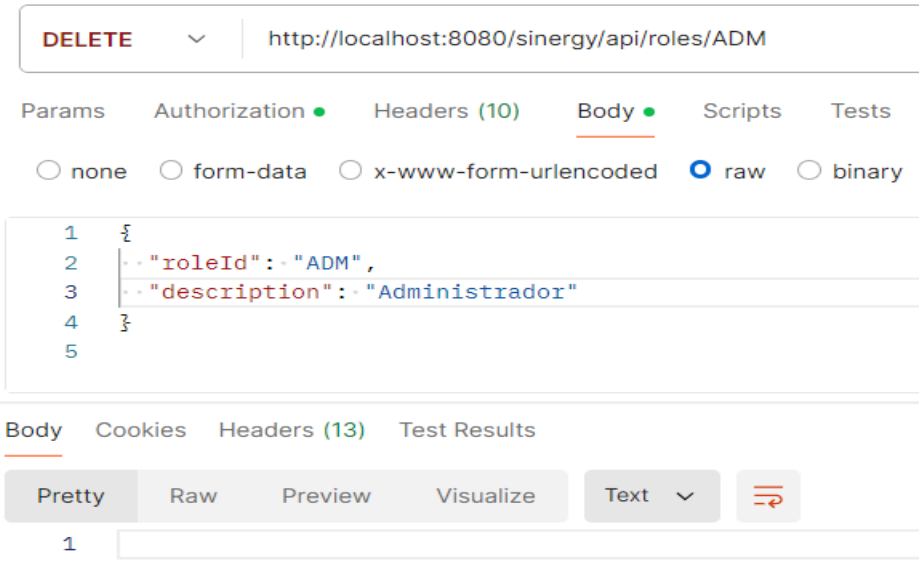


Ilustración 30: Petición exitosa de PostMan para DELETE Role

5.4.3.5 UserREST.java

POST /users

- Descripción: Crea un nuevo usuario en la base de datos.
- Funcionalidad: Recibe los datos del usuario en el cuerpo de la solicitud y los guarda en la base de datos.
- Proceso:

  Validación de los datos del usuario.

  Uso del repositorio "IUserRepo" para guardar el nuevo usuario.

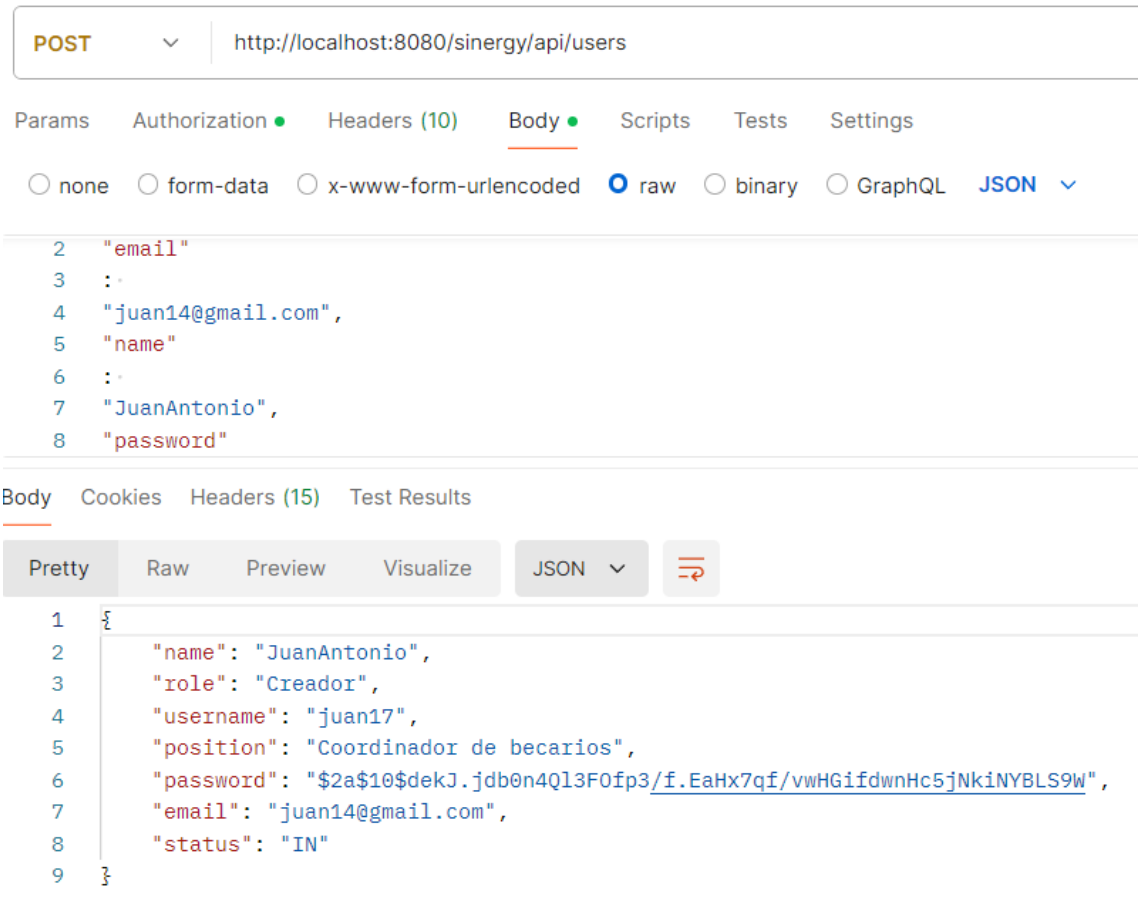


Ilustración 31: Petición exitosa de PostMan para Post User

PUT /users/{username}

- Descripción: Actualiza los detalles de un usuario existente identificado por su nombre de usuario.
- Funcionalidad: Recibe los datos actualizados en el cuerpo de la solicitud y actualiza el registro correspondiente en la base de datos.
- Proceso:

  Validación de los datos actualizados.

  Uso del repositorio "IUserRepo" para buscar y actualizar el usuario.

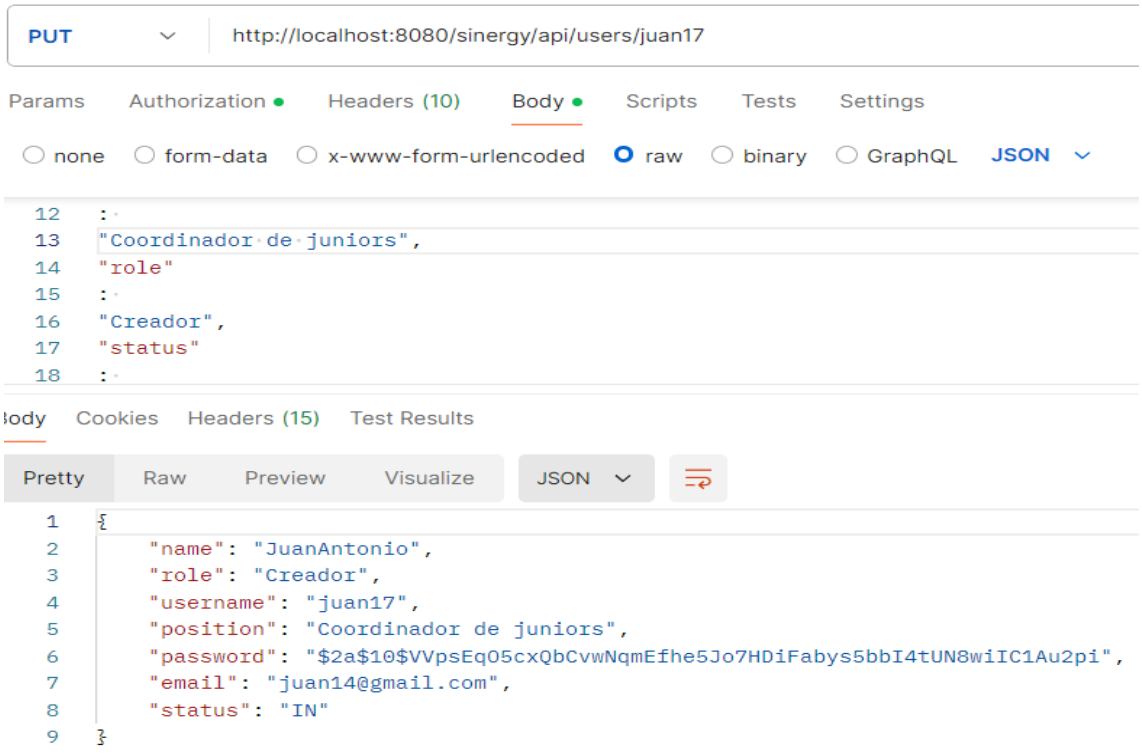


Ilustración 32: Petición exitosa de PostMan para PUT User

DELETE /users/{username}

- Descripción: Elimina un usuario de la base de datos identificado por su nombre de usuario.
- Funcionalidad: Elimina el registro correspondiente en la base de datos.
- Proceso:

  Uso del repositorio "IUserRepo" para eliminar el usuario.

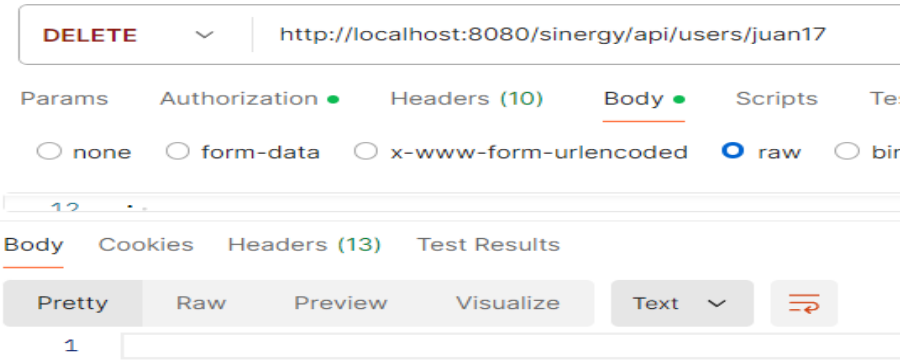


Ilustración 32: Petición exitosa de PostMan para Post User

POST /login y GET /renew?user={username}: Funcionamiento y ejecución igual que en el Reader.

5.5 Gateway

El Gateway de la plataforma juega un papel crucial en la arquitectura basada en microservicios. Sirve como el punto de entrada único para todas las peticiones de los clientes, proporcionando una capa de abstracción entre los clientes y los microservicios internos. En este capítulo, se detallará el funcionamiento del Gateway y su configuración basada en el archivo application.yml (Aunque luego no se le da un uso real, se usa yml por tema de compatibilidad y funcionalidades). También se explicará cómo se enrutan las peticiones al reader, writter y auth según lo que se desee hacer.

Ilustración 33: Banner de ejecución del Gateway

5.5.1 Funcionalidades del Gateway

Enrutamiento de Peticiones:

El Gateway se encarga de enrutar las peticiones entrantes a los microservicios adecuados. Utilizando patrones de enrutamiento definidos, redirige las peticiones basadas en la ruta y el tipo de petición a los microservicios de cada servicio correspondientes como el "UserREST", "RoleREST" o "ReportREST", entre otros.

Balanceo de Carga:

Implementa estrategias de balanceo de carga para distribuir las peticiones de manera equitativa entre las instancias disponibles de los microservicios. Esto asegura una distribución eficiente de las cargas de trabajo y mejora el rendimiento y la disponibilidad del sistema.

Seguridad y Autorización:

Actúa como un guardián de seguridad, gestionando la autenticación y autorización de las peticiones con el Auth. Utiliza tokens para validar las credenciales de los usuarios y asegurar que solo las peticiones autorizadas lleguen a los microservicios internos.

Monitorización y Registro:

Proporciona capacidades de monitorización y registro, permitiendo rastrear las peticiones, errores y métricas de rendimiento. Esto es esencial para la gestión y el mantenimiento del sistema, ayudando a identificar y solucionar problemas de manera proactiva.

5.5.2 Configuración del Gateway

El archivo “application.yml” contiene la configuración general del Gateway. Este archivo define las propiedades y comportamientos del Gateway, incluyendo las rutas y los filtros de seguridad. A continuación, se describe cómo se configuran las rutas para enrutar las peticiones a los microservicios “reader”, “writter” y “auth”.

5.5.2.1 Enrutamiento de Peticiones

El enrutamiento de peticiones se configura mediante la sección “spring.cloud.gateway.routes”. Las rutas se definen para que las peticiones sean redirigidas a los microservicios correspondientes según la URI y el puerto especificados. Por ejemplo:

```
- id: delete delete users
  uri: ${services.uriPuertoWritter}
  predicates:
  - Path=/sinergy/api/delete/users/{id}
  - Method= DELETE
  filters:
  - JwtAuthFilterFactory
#--Reader
- id: get users
  uri: ${services.uriPuertoReader}
  predicates:
  - Path=/sinergy/api/users
  - Method= GET
  filters:
  - JwtAuthFilterFactory
```

Ilustración 34: Enrutamientos de peticiones a user

Cómo se puede ver en la imagen, las peticiones del reader contienen la uri services.uriPuertoReader, y las del writter services.uriPuertoWritter; estos son dos parámetros definidos en el archivo que indican el puerto al que han de mandar cada petición para realizársela a un microservicio determinado, en este caso el Reader está alojado en el puerto localhost:8181 y el Writter en el localhost:8081.

5.5.2.2 Seguridad y Autorización

El Gateway también puede configura la seguridad y autorización de las peticiones. definen filtros de seguridad para asegurarse de que solo las peticiones autorizadas lleguen a los microservicios. Estos filtros incluyen

detalles de autenticación como OAuth2 y JWT para validar y autorizar las peticiones entrantes.

5.5.3 Uso del Gateway

El Gateway se despliega como un servicio independiente dentro de la infraestructura de microservicios con un puerto independiente que permite que mediante el envío a un solo puerto se gestionen todas las peticiones. Los clientes envían todas sus peticiones al Gateway, el cual se encarga de:

- **Recepción de Peticiones:**
  Recibe las peticiones de los clientes y aplica los filtros de seguridad de JWT y de Spring Boot para validar las credenciales.

- **Enrutamiento**

- **Envío de Respuestas:**
  Recibe las respuestas de los microservicios, aplica cualquier transformación necesaria y las envía de vuelta al cliente.

5.5.4 Beneficios del Gateway

- Seguridad Centralizada: Gestiona la autenticación y autorización en un solo punto, simplificando la seguridad de toda la arquitectura.
- Escalabilidad: Permite escalar los microservicios de manera independiente, distribuyendo la carga de trabajo de manera eficiente. Si se quieren añadir microservicios basta con añadir sus peticiones al Gateway y ya estarán incorporados a la plataforma.
- Mantenibilidad: Simplifica la gestión y el mantenimiento de las rutas y reglas de enrutamiento.
- Flexibilidad: Facilita la transformación de peticiones y respuestas, adaptándose a las necesidades de los clientes y los microservicios.

5.5.5 Recapitulación

El Gateway es un componente esencial en la arquitectura de microservicios de la plataforma, proporcionando enrutamiento, seguridad, balanceo de carga, y capacidades de transformación de peticiones. Su correcta configuración y uso aseguran que el sistema sea escalable, seguro y eficiente, facilitando la comunicación entre los clientes y los microservicios internos.

5.6 Microservicio de Autenticación (Auth)

El microservicio de autenticación, comúnmente conocido como “Auth”, es una parte integral de nuestra plataforma, encargándose de gestionar la seguridad y

la autorización de las peticiones que llegan a los demás microservicios. A continuación, se detalla el funcionamiento de este microservicio basado en el archivo “AuthREST.java”, y se explican las funciones clave que desempeña en la arquitectura del sistema.

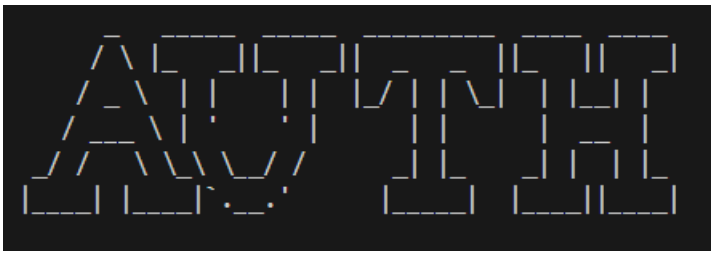


Ilustración 35: Banner de ejecución del Auth

5.6.1 Funciones del Microservicio de Autenticación (Auth)

El microservicio “Auth” tiene varias responsabilidades esenciales para asegurar la plataforma, incluyendo la validación de credenciales de usuario, la generación de tokens JWT, la validación de estos tokens, y la gestión de la autorización para acceder a diferentes recursos.

5.6.1.1 Validación de Credenciales

El microservicio “Auth” se encarga de recibir las credenciales de los usuarios (por ejemplo, nombre de usuario y contraseña), validarlas contra la base de datos de usuarios, y determinar si son correctas. Si las credenciales son válidas, se procede a generar un token JWT para el usuario.

5.6.1.2 Generación de Tokens JWT

Una vez validadas las credenciales, el microservicio “Auth” genera un token JWT (JSON Web Token). Este token contiene información sobre el usuario y sus permisos, y es firmado digitalmente para asegurar su integridad. El token se envía al cliente, que lo utilizará para autenticar futuras peticiones.

5.6.1.3 Validación de Tokens JWT

Cada vez que una petición llega a otro microservicio, el token JWT incluido en la petición es redirigido por el Gateway y es validado por el microservicio “Auth”. Esto asegura que la petición proviene de un usuario autenticado y autorizado para acceder al recurso solicitado.

5.6.1.4 Gestión de Autorización

El microservicio “Auth” también gestiona la autorización, determinando qué recursos y operaciones puede realizar un usuario basado en su rol y permisos asociados. Esta información se incluye en el token JWT y se verifica en cada petición.

5.6.1.5 AuthREST.java

El archivo AuthREST.java en la carpeta external del Auth define las rutas y los métodos para manejar las peticiones relacionadas con la autenticación y la autorización. A continuación, se describen las principales funciones que se implementan en este archivo.

Endpoint de Inicio de Sesión (Login)

El endpoint de inicio de sesión recibe las credenciales del usuario y, si son válidas, genera un token JWT.

- **Validación de Credenciales:**

  El método recibe el nombre de usuario y la contraseña, y los compara con los datos almacenados en la base de datos mediante repositorios iguales que en el Reader y el Writter.

- **Generación de Token:**

  Si las credenciales son correctas, se genera un token JWT que contiene la información del usuario y sus permisos. Este token se devuelve al cliente.

Endpoint de Renovación de Token (Renew)

Este endpoint permite a los usuarios renovar su token JWT antes de que expire.

- **Recepción del Token:**

  El método recibe el token actual del usuario y lo valida para asegurarse de que sigue siendo válido.

- **Generación de Nuevo Token:**

  Si el token es válido, se genera un nuevo token JWT con una nueva fecha de expiración y se devuelve al cliente.

Endpoint de Validación de Token (Validate)

Este endpoint permite validar un token JWT para asegurarse de que es legítimo y no ha sido manipulado.

5.6.2 Ejemplo de Uso del Microservicio “Auth”

Para ilustrar el uso del microservicio “Auth”, consideremos el siguiente flujo:

1. **Inicio de Sesión:**

Un usuario envía una petición POST al endpoint de inicio de sesión con su nombre de usuario y contraseña al Gateway que lo redirige al Auth. El microservicio “Auth” valida estas credenciales y, si son correctas, genera un token JWT y lo devuelve al usuario.

2. **Acceso a Recursos:**

El usuario incluye el token JWT en el encabezado de sus peticiones a otros microservicios (Reader o Writter). Estos microservicios envían el token al microservicio “Auth” para validación antes de permitir el acceso al recurso solicitado.

3. **Renovación del Token:**

Antes de que el token expire, el usuario puede enviar una petición GET al endpoint de renovación de token. El microservicio “Auth” valida el token actual y, si es válido, genera y devuelve un nuevo token JWT.

4. **Validación del Token:**

En cualquier momento, otros microservicios pueden enviar una petición al endpoint de validación de token para asegurarse de que un token JWT es legítimo y no ha sido manipulado.

5.6.3 Recapitulación

El microservicio de autenticación (“Auth”) es fundamental para garantizar la seguridad y la integridad de la plataforma. Gestiona la autenticación y autorización de los usuarios, generando y validando tokens JWT para asegurar que solo los usuarios legítimos puedan acceder a los recursos y realizar operaciones en la plataforma. La correcta implementación y uso de este microservicio es esencial para mantener un entorno seguro y confiable.

5.7 Front End

La implementación del frontend de la plataforma se ha realizado utilizando ReactJS y DevExtreme, dos tecnologías potentes que permiten crear interfaces de usuario modernas, interactivas y altamente funcionales. A continuación, se detalla cómo se ha llevado a cabo esta implementación y cómo se gestionan las peticiones a los microservicios reader, writter y auth a través del Gateway.

5.7.1 ReactJS y DevExtreme en el Frontend

ReactJS es una biblioteca de JavaScript desarrollada por Facebook que se utiliza para construir interfaces de usuario basadas en componentes reutilizables. ReactJS permite una gestión eficiente del estado de la interfaz de usuario, facilitando la creación de aplicaciones web dinámicas y reactivas.

DevExtreme es un conjunto de componentes de interfaz de usuario y herramientas de desarrollo para crear aplicaciones web. Ofrece una amplia gama de widgets y controles UI, como tablas, gráficos, formularios y otros elementos interactivos que mejoran la experiencia del usuario.

5.7.1.1 Ventajas de Utilizar ReactJS y DevExtreme

Modularidad y Reutilización:

La arquitectura basada en componentes de ReactJS permite construir bloques de UI reutilizables y mantenibles. Esto mejora la eficiencia del desarrollo y facilita la actualización y el mantenimiento del código.

Interactividad:

DevExtreme proporciona widgets ricos en funcionalidades que permiten crear interfaces de usuario altamente interactivas y atractivas. Los componentes de DevExtreme son fáciles de integrar con ReactJS y ofrecen una excelente experiencia de usuario.

Gestión del Estado:

ReactJS facilita la gestión del estado de la aplicación a través de su sistema de hooks y componentes funcionales, lo que permite una actualización eficiente y predecible de la UI en respuesta a las interacciones del usuario.

5.7.2 Peticiones a los Microservicios a través del Gateway

El frontend de la plataforma se comunica con los microservicios reader, writter y auth a través del Gateway. Este enfoque asegura que todas las peticiones sean

gestionadas de manera centralizada, proporcionando seguridad y eficiencia en la comunicación entre el frontend y el backend.

5.7.2.1 Flujo de Peticiones

Autenticación (Auth):

El usuario inicia sesión a través del frontend enviando una solicitud al endpoint de autenticación del Gateway.

El Gateway redirige la solicitud al microservicio auth, que valida las credenciales del usuario.

Si las credenciales son válidas, el microservicio auth genera un token JWT y lo envía de vuelta al frontend a través del Gateway.

Lecturas (Reader):

Para obtener y desplegar datos, el frontend envía solicitudes GET al Gateway.

El Gateway redirige estas solicitudes al Reader, que maneja las lecturas de datos desde la base de datos.

Los datos recuperados por el Reader son enviados de vuelta al frontend a través del Gateway.

Escrituras (Writter):

Para enviar o modificar datos, el frontend envía solicitudes POST, PUT o DELETE al Gateway.

El Gateway redirige estas solicitudes al microservicio Writter, que maneja las escrituras y modificaciones de datos en la base de datos.

Las respuestas del microservicio writter son enviadas de vuelta al frontend a través del Gateway.

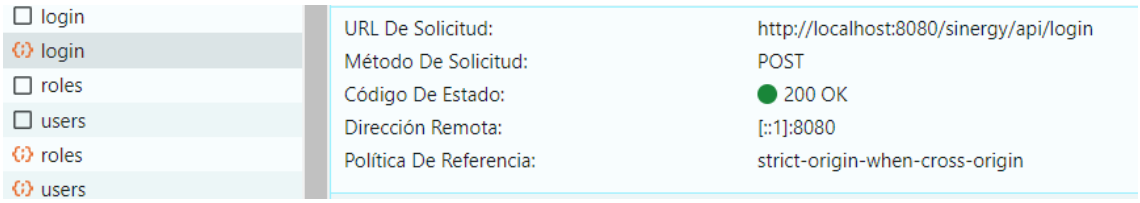


Ilustración 36: Imagen que destaca cómo es el flujo de peticiones una vez se hace un login autorizado

5.7.3 Configuración Básica a destacar

Hay varios archivos y configuraciones básicas fuera de interfaces específicas en el front-end de la plataforma, desde estilos y componentes a configuraciones de seguridad e iconos. En este apartado se van a destacar las más relevantes y que suponen el esqueleto de esta plataforma.

5.7.3.1 Config.js

```
var config = {
    apiBaseUrl: "http://localhost:8080/sinergy/api"
};
```

Ilustración 37: Contenido del archivo config.js

Este archivo define la pieza clave del despliegue de datos que es la URL del Gateway, esta variable es utilizada por todos los componentes que realicen una petición.

5.7.3.2 Estilos y elementos básicos de la plataforma:

Para el estilo básico de la plataforma se ha usado un tono gris oscuro para el fondo y una combinación de tonos blancos y azules que le otorgan una apariencia elegante y moderna. Además se ha bautizado a la plataforma como Sinergy haciendo referencia a la palabra sinergia en relación a la forma en la que tecnología y RRHH se unen en esta plataforma para optimizar un proceso crucial para toda compañía.

La navegación entre las funcionalidades se realiza a través de un menú lateral que permite acceder de manera rápida e intuitiva a las funcionalidades disponibles que se quieran consultar, modificar o eliminar.

Ilustración 38: Interfaz, estilos y elementos básicos

5.8 Configuraciones específicas de las funcionalidades del Front End

5.8.1 Implementación de la Funcionalidad de Candidatos en la Plataforma

En este capítulo se describe la implementación de la funcionalidad de gestión de candidatos en la plataforma utilizando la biblioteca DevExtreme y React. La interfaz de usuario está diseñada para ser intuitiva y funcional, permitiendo a los usuarios agregar, editar, eliminar y buscar candidatos de manera eficiente.

5.8.1.1 Descripción de la Interfaz y Funcionalidades

Elementos Principales

La interfaz de candidatos incluye varios elementos principales que facilitan la gestión de los datos:

DataGrid: El componente central de la interfaz es un DataGrid, que muestra una lista de todos los candidatos disponibles. Este grid permite la edición en línea y el acceso a funcionalidades avanzadas como filtrado, búsqueda y exportación de datos.

Columnas: Cada candidato tiene atributos que se muestran en columnas dentro del DataGrid. Las columnas incluyen:

- Nombre (name)
- Descripción (description)
- Posición solicitada (solicitedPosition)
- Correo electrónico (email)

Formularios de Edición y Creación: Para agregar o editar candidatos, se utiliza un formulario dentro de un popup. Este formulario incluye campos para cada uno de los atributos mencionados anteriormente. Los campos son:

- Nombre: Campo requerido para identificar al candidato.
- Descripción: Información adicional sobre el candidato.
- Posición solicitada: La posición que el candidato desea ocupar.
- Correo electrónico: Validado para asegurar que sigue el formato correcto.

Botones y Herramientas:

- Botón de agregar fila: Permite a los usuarios agregar un nuevo candidato.
- Botón de eliminar filas seleccionadas: Permite la eliminación de uno o varios candidatos seleccionados en el DataGrid.

-Panel de búsqueda: Facilita la búsqueda de candidatos específicos dentro del DataGrid.

- Función de exportación: Permite la exportación de los datos mostrados en el DataGrid a un archivo para uso externo.

5.8.1.2 Funcionalidades y Comportamiento

1. Inicialización de Datos: Al cargar la página, se realiza una llamada a la API para obtener la lista de candidatos y roles disponibles. Estos datos se almacenan en el estado de la aplicación para su uso posterior.

2. Agregar Candidatos: Al iniciar la creación de un nuevo candidato, se presenta un formulario en un popup. Después de ingresar los datos y guardar, se realiza una petición POST a la API para agregar el candidato a la base de datos. Si la operación es exitosa, se muestra un mensaje de éxito.

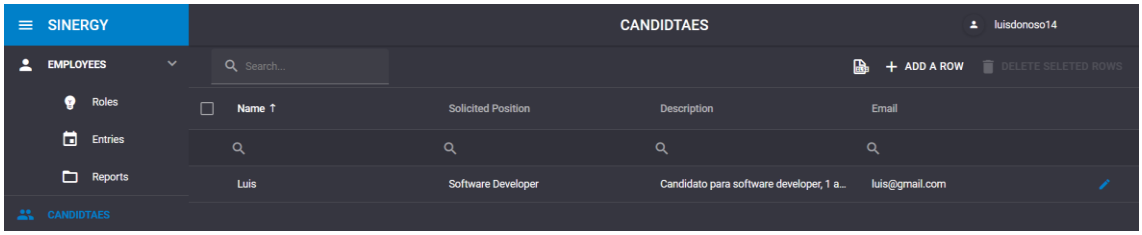
3. Editar Candidatos: Al editar un candidato existente, se abre un formulario similar en un popup con los datos actuales del candidato. Después de realizar los cambios y guardar, se realiza una petición PUT a la API para actualizar los datos del candidato en la base de datos. Se notifica al usuario del éxito o error de la operación.

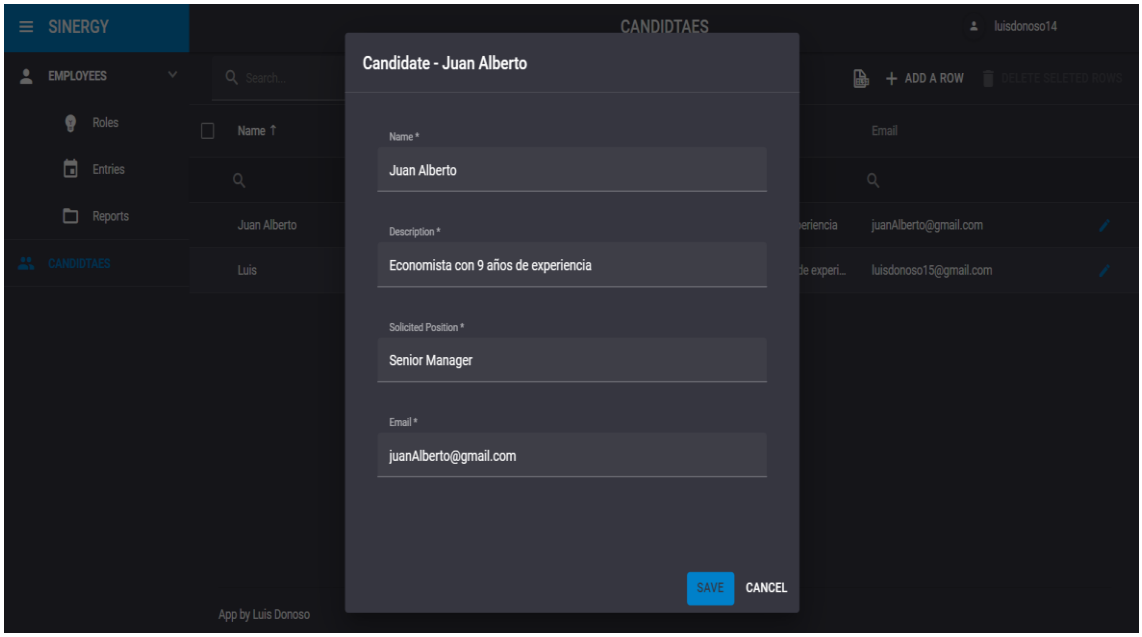
4. Eliminar Candidatos: Los usuarios pueden seleccionar uno o varios candidatos y eliminarlos utilizando el botón de eliminación. Esto dispara una petición DELETE a la API para remover los candidatos seleccionados de la base de datos.

5. Búsqueda y Filtrado: El panel de búsqueda y las opciones de filtrado permiten a los usuarios encontrar rápidamente candidatos específicos. La búsqueda es sensible a mayúsculas y minúsculas para mayor precisión.

6. Exportación de Datos: Los usuarios pueden exportar los datos del DataGrid a un archivo. Esta funcionalidad es útil para compartir información o realizar análisis externos.

7. Responsividad: La interfaz está diseñada para ser responsiva, ajustándose a diferentes tamaños de pantalla. Esto asegura que los usuarios puedan acceder a la funcionalidad completa desde dispositivos móviles y de escritorio.





Ilustraciones 39 y 40: Interfaz de la pestaña candidatos y formulario de inserción/modificación

La implementación de la funcionalidad de gestión de candidatos utilizando DevExtreme y React proporciona una interfaz rica y flexible para los usuarios. Los componentes y funcionalidades descritos aseguran que la gestión de candidatos sea eficiente y efectiva, mejorando la experiencia del usuario y facilitando la administración de datos dentro de la plataforma.

5.8.2 Implementación de las entries

En este apartado se describe la implementación de la funcionalidad de gestión de registros de horas (entries) en la plataforma utilizando la biblioteca DevExtreme y React. La interfaz de usuario está diseñada para ser intuitiva y funcional, permitiendo a los usuarios agregar, editar, eliminar y buscar registros de horas de manera eficiente.

5.8.2.1 Descripción de la Interfaz y Funcionalidades

Elementos Principales

DataGrid:

Descripción: El componente central de la interfaz es un DataGrid que muestra una lista de todos los registros de horas disponibles.

Funcionalidades: Permite visualizar, agregar, editar y eliminar registros de horas. Los datos en el grid incluyen detalles como la fecha y hora de inicio (datetimeStart), la fecha y hora de fin (datetimeEnd), el usuario asociado y las horas registradas. Además, permite la selección múltiple de filas y alternancia de filas para una mejor legibilidad.

SearchPanel:

Descripción: Un panel de búsqueda que permite a los usuarios buscar rápidamente registros específicos dentro del DataGrid.

Funcionalidades: Este panel es visible de manera predeterminada y ofrece búsqueda sensible a mayúsculas y minúsculas.

FilterRow:

Descripción: Un filtro que permite a los usuarios filtrar los datos directamente en el DataGrid.

Funcionalidades: Facilita la búsqueda de registros específicos mediante la aplicación de filtros a las columnas del DataGrid.

Toolbar:

Descripción: Una barra de herramientas que incluye botones para diversas acciones.

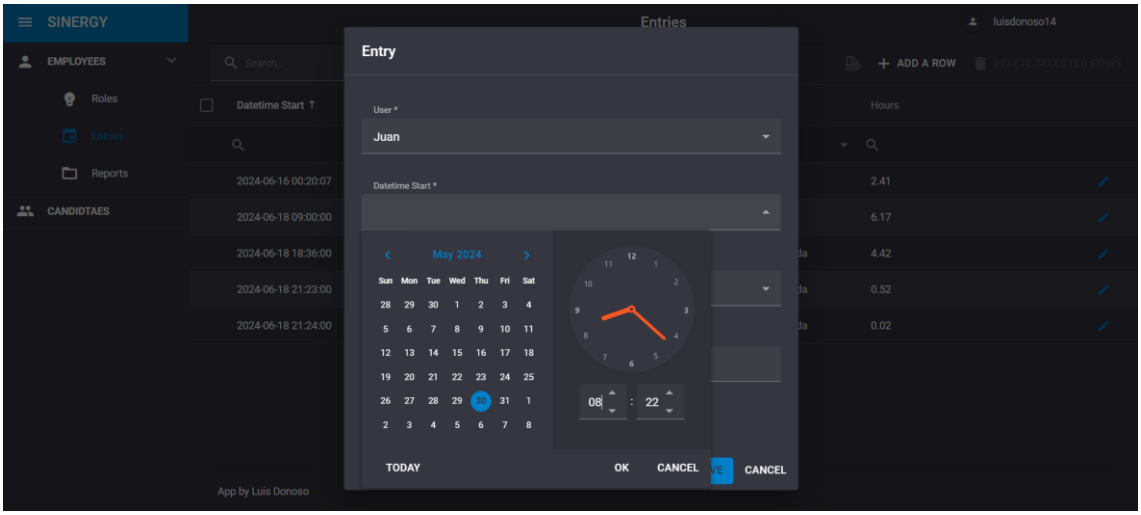
Funcionalidades: Incluye botones para la búsqueda, exportación de datos y adición de nuevas filas. También cuenta con un botón para eliminar filas seleccionadas.

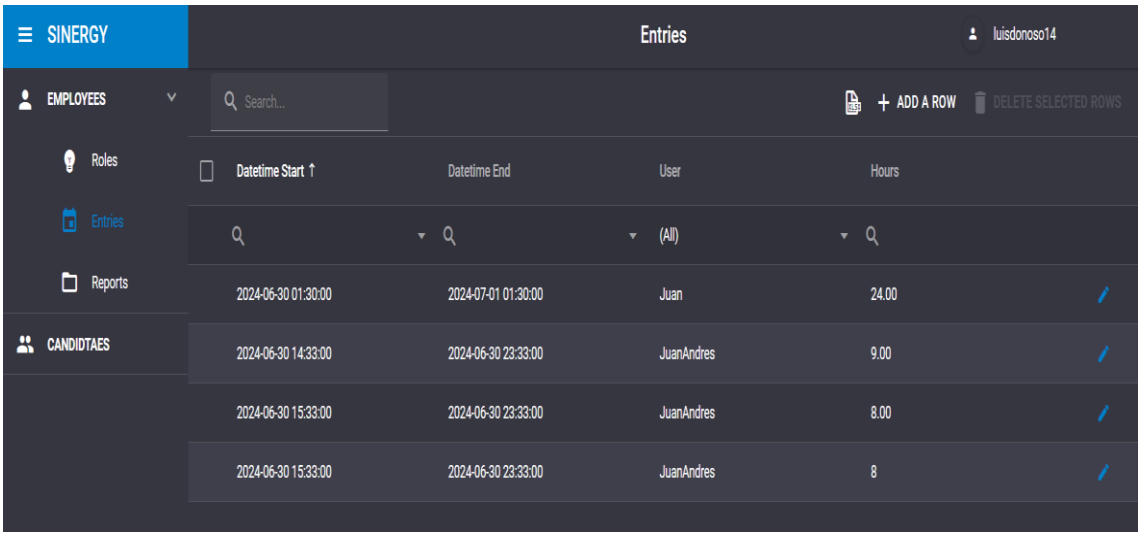
Popup de Edición/Inserción:

Descripción: Un popup que aparece al agregar o editar un registro.

Funcionalidades: Este popup incluye un formulario con los siguientes campos:

- Usuario (user): Un campo de búsqueda que permite seleccionar el usuario asociado al registro.
- Fecha y Hora de Inicio (datetimeStart): Un selector de fecha y hora para indicar el inicio del período registrado.
- Fecha y Hora de Fin (datetimeEnd): Un selector de fecha y hora para indicar el fin del período registrado.
- Horas (hours): Un campo de texto para ingresar las horas registradas, se rellena automáticamente si se deja en blanco.

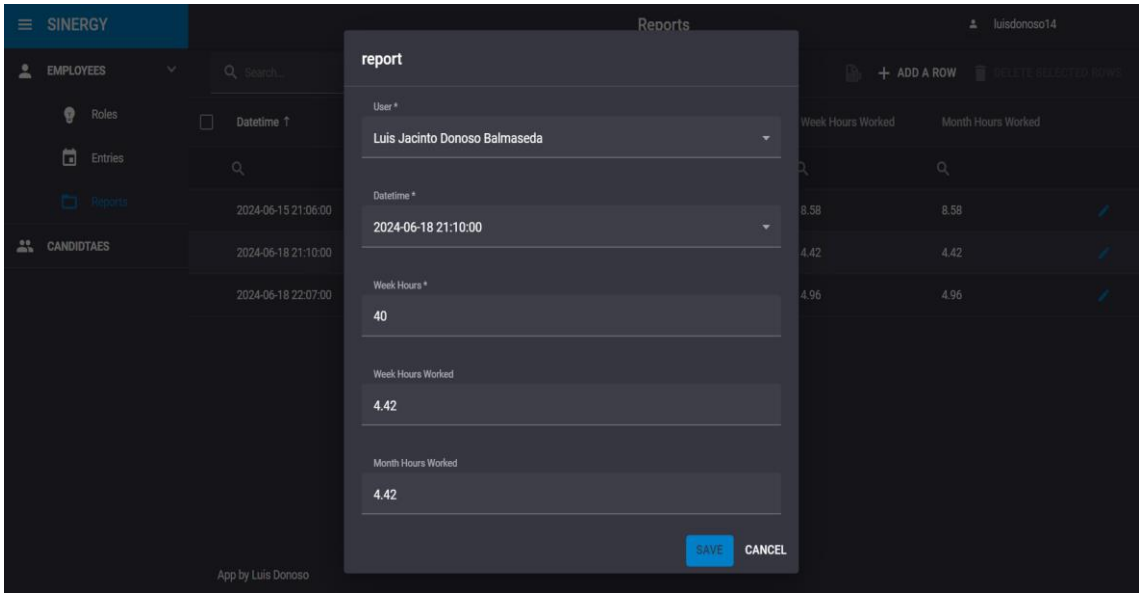


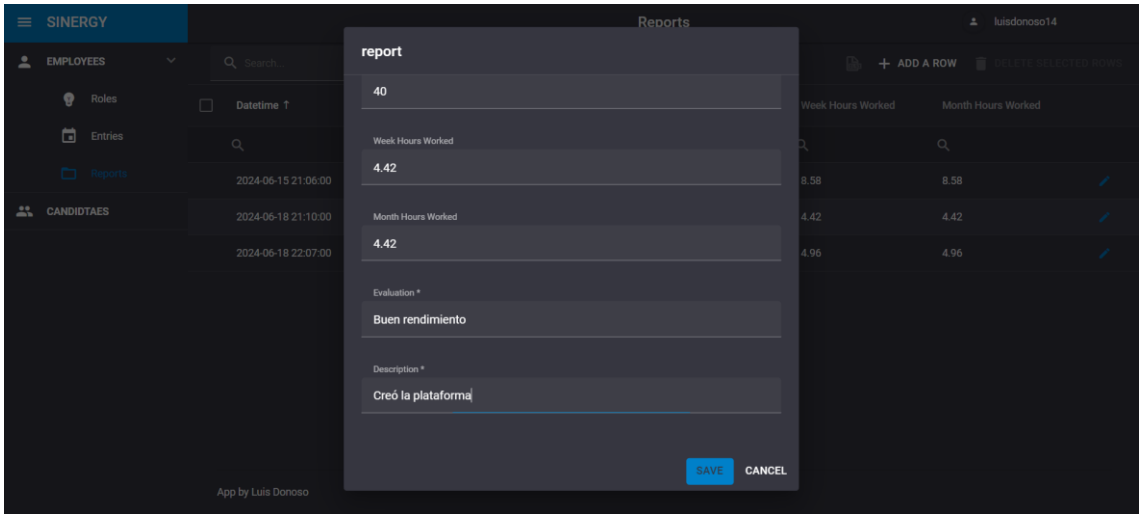


Ilustraciones 41 y 42 : Interfaz de la pestaña entriess y formulario de inserción/modificación

5.8.3 Reports

Los reports tienen una implementación muy parecida a las entries salvo que se eliminan los campos de dateTimeStart y dateTimeEnd y se sustituyen por el campo dateTime que indica la hora exacta de realización del informe. Además, se incluyen los campos week Hours, Week Hours Worked y Month Hours Worked que el primero indica las horas por contrato del trabajador y los dos segundos se rellenan automáticamente si se dejan en blanco con todas las horas de las entries de un mes y una semana antes de la fecha de creción del informe.





Ilustraciones 43 y 44 : formularios de generación/modificación de informes

5.9 Implementación de roles

Los roles solo cuentan con dos campos en el datagrid y funcionan de manera similar a los candidatos, estos son roleId con el Id abreviado del rol y description con la descripción del rol.

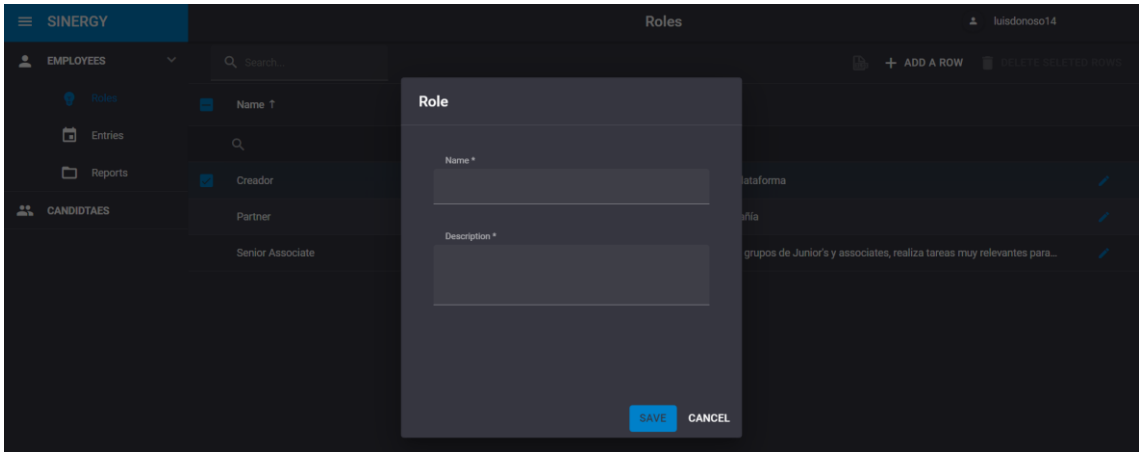


Ilustración 45: Formulario de creación generación de un Rol

5.10 Implementación de Empleados

Los empleados funcionan de manera muy parecida a los candidatos excepto que hay que relacionarlos con un Rol en el momento de su creación. Además en su datagrid se añade la posición específica que ocupan y si están IN o OUT dependiendo de las entradas.

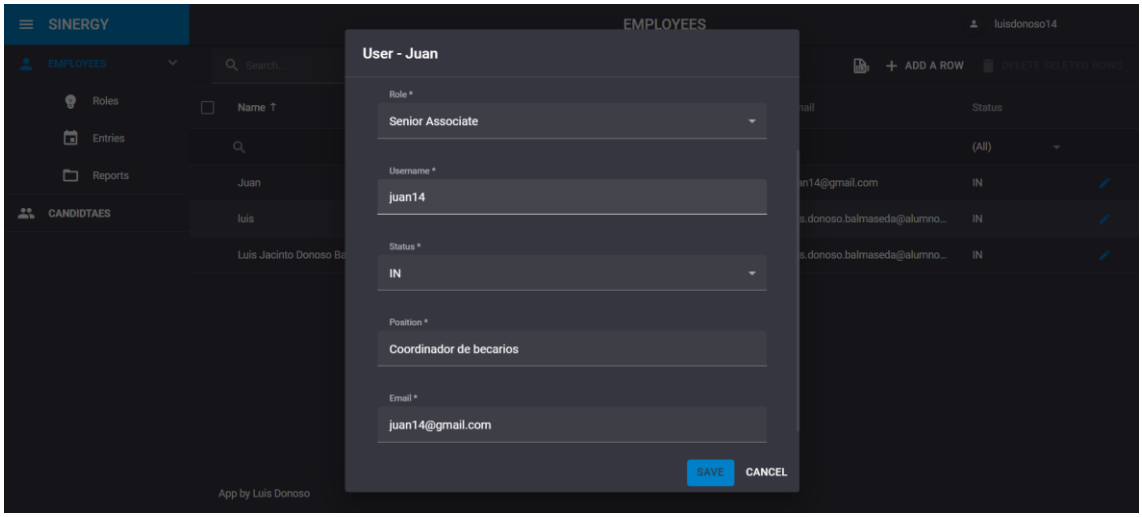


Ilustración 46: Formulario de creación/modificación de un Empleado

5.11 Implementación de Login

Probablemente la implementación más sencilla de todas. Cuenta con un formulario de username y password que manda con esas credenciales una petición al Gateway, si no se autoriza no se muestra nada más.

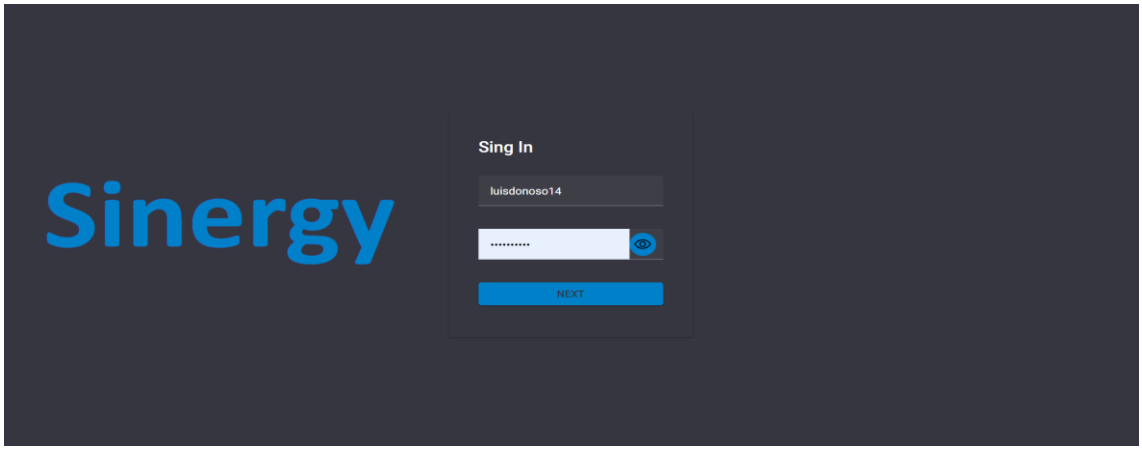


Ilustración 46: Formulario de login de un Empleado

6 Resultados y conclusiones

El desarrollo de la plataforma de gestión de RRHH ha logrado cumplir con los objetivos planteados al inicio del proyecto. Los resultados obtenidos demuestran que la plataforma es capaz de gestionar eficientemente los procesos de RRHH, proporcionando una herramienta integral para las empresas.

6.1 Resultados principales:

Funcionalidades Implementadas

Se han desarrollado e implementado todas las funcionalidades clave inicialmente definidas, incluyendo la gestión de perfiles de empleados, el registro de horas laborales, la generación de informes, la gestión de reclutamiento y selección, la automatización de tareas administrativas, la capacitación y desarrollo, y la gestión de ausencias y licencias. Estas funcionalidades permiten a las empresas manejar de manera efectiva todos los aspectos relacionados con la gestión de su personal.

Arquitectura de Microservicios

La adopción de una arquitectura de microservicios ha proporcionado una serie de ventajas significativas en términos de escalabilidad, flexibilidad, resiliencia y productividad. Cada componente de la plataforma puede desarrollarse, desplegarse y mantenerse de manera independiente, lo que facilita la integración de nuevas funcionalidades y la adaptación a los cambios.

Tecnologías Utilizadas

El uso de Spring Boot en el backend y ReactJS en el frontend ha garantizado un desarrollo ágil y eficiente, permitiendo crear una plataforma robusta y escalable. Además, la elección de Microsoft SQL Server como sistema de gestión de bases de datos ha proporcionado un rendimiento escalable y una alta seguridad, facilitando la administración de datos.

6.2 Conclusiones:

Eficiencia y Escalabilidad: La plataforma desarrollada ofrece una solución eficiente y escalable para la gestión de RRHH en PYMEs. Su arquitectura modular y flexible permite una fácil integración con otros sistemas empresariales y la adición de nuevas funcionalidades según las necesidades de cada empresa.

Mejora en la Gestión de RRHH: Las funcionalidades implementadas mejoran significativamente la gestión de RRHH, facilitando procesos como el reclutamiento, la evaluación del desempeño y la gestión de ausencias. Esto contribuye a un mejor aprovechamiento del talento humano y a una mayor eficiencia operativa.

Adopción de Tecnologías Modernas: La utilización de tecnologías modernas y metodologías ágiles ha sido clave para el éxito del proyecto. Estas herramientas no solo han facilitado el desarrollo de la plataforma, sino que también aseguran su sostenibilidad y capacidad de evolución a largo plazo.

En conclusión, el TFG ha logrado desarrollar una plataforma de gestión de RRHH que responde adecuadamente a las demandas del mercado, ofreciendo una herramienta completa y eficiente para la gestión del talento humano en las empresas. La plataforma está preparada para adaptarse a futuros cambios y mejoras, garantizando su relevancia y utilidad en un entorno empresarial en constante evolución

7 Análisis de Impacto

7.1 Impacto Personal en el Alumno

La realización de este Trabajo de Fin de Grado (TFG) ha tenido un impacto significativo en mi desarrollo personal y profesional. Desde el inicio del proyecto, he adquirido y consolidado una serie de competencias técnicas y habilidades que han sido cruciales para el éxito del trabajo y que, sin duda, marcarán mi carrera futura.

7.1.1 Competencias Técnicas:

1. **Desarrollo Full Stack:**

   El diseño y la implementación de una plataforma web completa me han permitido profundizar en el desarrollo tanto del backend como del frontend. El uso de Spring Boot para el backend y ReactJS para el frontend ha sido particularmente enriquecedor, dándome una comprensión sólida de estas tecnologías ampliamente utilizadas en la industria.

2. **Arquitectura de Microservicios**:
   La elección de una arquitectura basada en microservicios me ha permitido entender mejor las ventajas de este enfoque, como la escalabilidad, la modularidad y la resiliencia. Esta experiencia me prepara para enfrentar proyectos complejos y escalables en mi futura carrera profesional.

3. **Gestión de Bases de Datos:**
   El trabajo con Microsoft SQL Server me ha permitido adquirir habilidades avanzadas en la gestión de bases de datos, así como en la integración de herramientas de Business Intelligence como Power BI.

7.1.2 Habilidades:

1. **Gestión del Tiempo:**

   La necesidad de planificar y organizar mi tiempo de manera eficiente para cumplir con los plazos del proyecto ha mejorado significativamente mi capacidad de gestión del tiempo, aun así, he estado cerca de fallar con la hora de entrega si no llego a calcular bien así que sigue siendo un aspecto a pulir.

2. **Resolución de Problemas:**
   La identificación y resolución de problemas técnicos y de diseño a lo largo del proyecto han fortalecido mi capacidad para enfrentar y superar desafíos de manera efectiva.

7.2 Impacto del Éxito del TFG

El éxito de este TFG no solo representa un logro académico personal, sino que también tiene el potencial de generar un impacto significativo en el ámbito empresarial y en la sociedad en general. La plataforma de gestión de recursos humanos desarrollada puede ofrecer múltiples beneficios a las pequeñas y medianas empresas (PYMEs), mejorando su eficiencia operativa y contribuyendo al desarrollo sostenible.

7.2.1 Beneficios para las Empresas:

1. **Optimización de Procesos:**
   La automatización de tareas administrativas y la eficiente gestión del talento humano permiten a las empresas optimizar sus procesos internos, reduciendo costos y mejorando la productividad.

2. **Mejora en la Toma de Decisiones:**

   La generación de informes y la capacidad de análisis proporcionan a los gerentes información valiosa para la toma de decisiones estratégicas, lo que puede mejorar significativamente la competitividad de las empresas.

3. **Retención y Desarrollo del Talento:**
   La plataforma facilita la gestión del desempeño, la formación y el desarrollo profesional, contribuyendo a la retención de empleados y al desarrollo de su potencial.

7.3 Relación con los Objetivos de Desarrollo Sostenible (ODS)

El proyecto se alinea con varios de los Objetivos de Desarrollo Sostenible (ODS) de la Agenda 2030 de la ONU, destacando especialmente los siguientes:

ODS 8: Trabajo Decente y Crecimiento Económico:

- Promoción del Empleo Productivo: La plataforma contribuye a la creación de entornos de trabajo más eficientes y productivos, promoviendo el empleo productivo y el trabajo decente.

- Apoyo a las PYMEs: Al proporcionar una herramienta accesible y eficiente para la gestión de recursos humanos, el proyecto apoya el crecimiento de las

pequeñas y medianas empresas, que son motores clave del crecimiento económico y la creación de empleo.

ODS 9: Industria, Innovación e Infraestructura:

- Innovación Tecnológica: El desarrollo y la implementación de una plataforma de gestión de recursos humanos basada en tecnologías modernas representan un ejemplo de innovación en la industria de software, promoviendo una infraestructura tecnológica avanzada.

En conclusión, el impacto de este TFG va más allá del ámbito académico, contribuyendo al desarrollo profesional del alumno y ofreciendo una herramienta valiosa para las empresas. Además, su alineación con los ODS de la Agenda 2030 refuerza su relevancia y potencial para contribuir al desarrollo sostenible y a la creación de un entorno empresarial más eficiente y equitativo.

8 BIBLIOGRAFÍA

[1] Workday, «Workday,» Workday, [En línea]. Available: https://www.workday.com/. [Último acceso: 11 6 2024].

[2] SAP, «SAP-SuccesFactors,» SAP, [En línea]. Available: https://www.sap.com/spain/products/hcm/about-successfactors.html. [Último acceso: 10 6 2024].

[3] Oracle, «Oracle Human Capital Management (HCM),» Oracle, [En línea]. Available: https://www.oracle.com/human-capital-management/. [Último acceso: 10 6 2024].

[4] BambooHR, «BambooHR,» BambooHR, [En línea]. Available: https://www.bamboohr.com/g2/. [Último acceso: 11 6 2024].

[5] ADP, «ADP - España,» ADP, [En línea]. Available: https://es.adp.com/. [Último acceso: 11 6 2024].

[6] Microsoft, «Microsoft Sql Management Studio,» [En línea]. Available: https://learn.microsoft.com/es-es/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver16.

[14] Microsoft, «Vsiual Studio Marketplace,» Microsoft, [En línea]. Available: https://marketplace.visualstudio.com/subscriptions. [Último acceso: 23 6 2024].

[15] Dell, «Dell,» Dell, [En línea]. Available: https://www.dell.com/es-es. [Último acceso: 24 6 2024].

[16] HP, «HP España,» HP, [En línea]. Available: https://www.hp.com/es-es/. [Último acceso: 24 6 2024].

[17] Amazon, «AWS Backup-Pricing,» Amazon, [En línea]. Available: https://aws.amazon.com/es/backup/pricing/. [Último acceso: 24 6 2024].

[18] Microsoft, «Precios Azure Backup,» Microsoft, [En línea]. Available: https://azure.microsoft.com/es-es/pricing/details/backup/. [Último acceso: 24 6 2024].

[19] Solución Legal, «Solución legal- Servicios Legales online,» Solución Legal, [En línea]. Available: https://solucionlegal.es/servicios-legales-online/. [Último acceso: 24 6 2024].

[26] Spring Academy, «Spring Academy,» [En línea]. Available: https://spring.academy/courses.

[27] Helsinki University, «Full stack open course,» 2023. [En línea]. Available: https://fullstackopen.com/en/.

[28] Mozilla, «Structuring the web with HTML,» [En línea]. Available: https://developer.mozilla.org/en-US/docs/Learn/HTML.

[29] C. D. A. Medina, «Aplicación Web Dedicada a la coordinación de grupos de trabajo y monitorización de un proyecto,» Enero 2024. [En línea]. Available: https://oa.upm.es/80504/1/TFG_CARLOS_DANIEL_AGRAMONTE_MEDINA.pdf.

[30] Microsoft, «Microsoft PowerBI Courses,» [En línea]. Available: https://learn.microsoft.com/es-es/training/powerplatform/power-bi?WT.mc_id=powerbi_landingpage-marketing-page.

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=ETS Ingenieros Informaticos - UPM, C=ES
	Fecha/Hora	Sun Jun 30 22:03:30 CEST 2024
	Emisor del Certificado	EMAILADDRESS=camanager@etsiinf.upm.es, CN=CA ETS Ingenieros Informaticos, O=ETS Ingenieros Informaticos - UPM, C=ES
	Numero de Serie	561
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)