DESARROLLO DEL SOFTWARE EMPRES360 PRO PARA EL DIAGNÓSTICO EMPRESARIAL EN LA UNIVERSIDAD DE CUNDINAMARCA, SEDE FUSAGASUGÁ

Manual Técnico

AUTORES
CRISTIAN STIVEN GUERRERO

Director:
ing. Alexander Mendoza Monaga

TRABAJO PARA OBTENER TITULO (TECNOLOGO EN DESARROLLO DE SOFTWARE)

Universidad de Cundinamarca
Facultad de ingeniería
Programa Tecnología en Desarrollo de Software
Soacha (Cundinamarca)
Septiembre 2024

Tabla de contenido

Tabla de ilustraciones

Introducción

Este manual técnico está destinado a los responsables del mantenimiento del software y a quienes buscan información sobre la aplicación para asegurar un uso adecuado del sistema. Incluye detalles sobre el diseño y el código del software, así como una descripción de sus componentes.

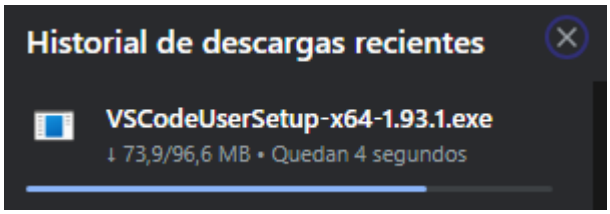
Tecnólogas del desarrollo

Durante el proyecto, se emplearán herramientas avanzadas propias de la programación moderna, a diferencia de las herramientas tradicionales. Es crucial comprender qué son estas herramientas, cómo funcionan y cómo se integrarán en el proyecto. Estas tecnologías juegan un papel esencial en el desarrollo, ya que se utilizarán de manera efectiva para mejorar el proceso y los resultados del software.

Visual Studio Code

Se ha optado por utilizar Visual Studio Code como el entorno de desarrollo integrado (IDE) para este proyecto. Esta herramienta destaca por su versatilidad en el desarrollo de software, proporcionando soporte para una gran variedad de lenguajes y tecnologías modernas. Visual Studio Code ofrece una amplia gama de extensiones y plugins que mejoran la productividad y facilitan la integración con diferentes herramientas de desarrollo. Su interfaz ligera y personalizable permite gestionar de manera efectiva los directorios y archivos del proyecto, promoviendo un flujo de trabajo ordenado y eficiente. La capacidad de adaptar el entorno de trabajo a las necesidades específicas del proyecto convierte a Visual Studio Code en una elección excelente para un desarrollo ágil y eficaz.

Ilustración 1

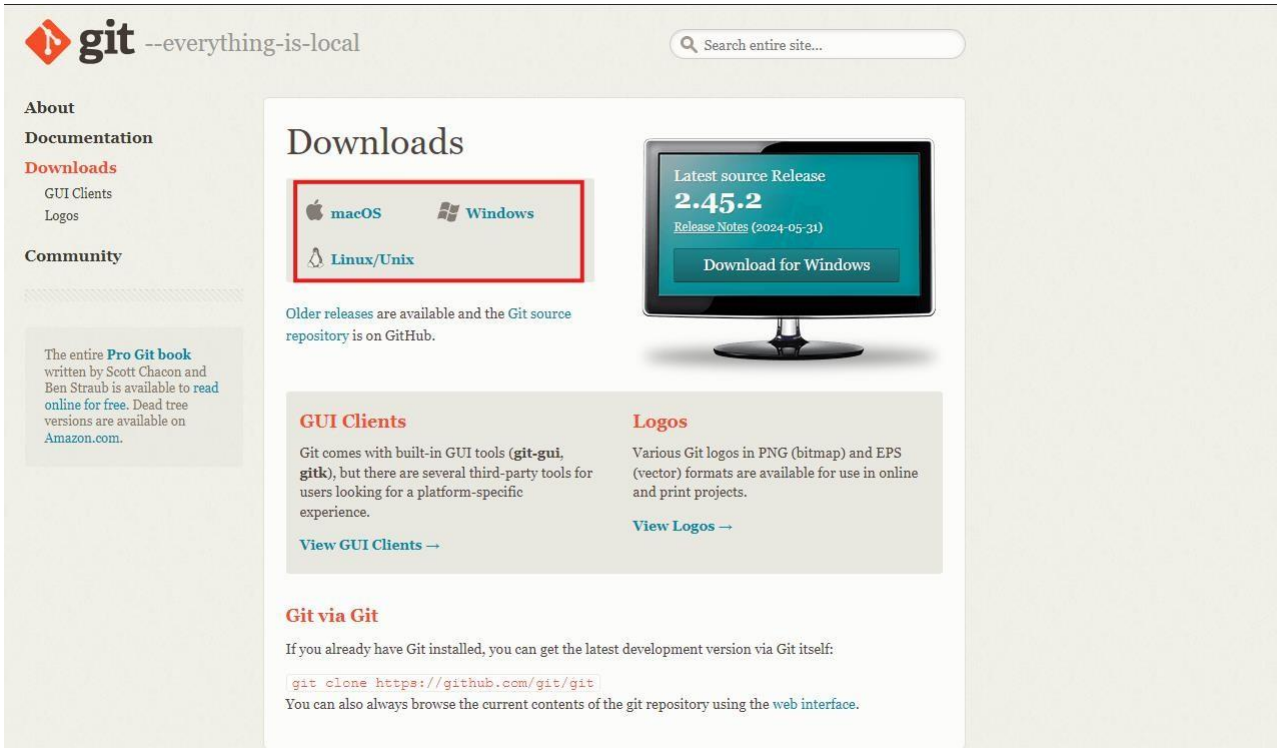


Fuente: Autoría propia.

Git

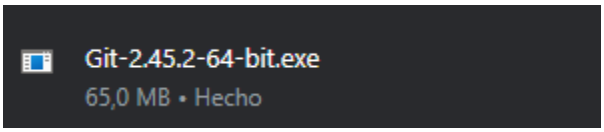
He decidido utilizar Git como el sistema de control de versiones para este proyecto. Aunque el desarrollo se realizó de manera individual, Git ha sido una herramienta fundamental para la gestión del código, permitiendo un control organizado de las versiones y ramas del proyecto. Esta organización ha sido crucial para integrar eficazmente la siguiente herramienta, Vercel. La capacidad de gestionar el proyecto en Git facilitó la implementación de Vercel, mostrando cómo la gestión del código puede potenciar el uso de otras herramientas y mejorar el flujo de trabajo general. La compatibilidad de Git con múltiples plataformas refuerza su papel como una herramienta esencial para un desarrollo sólido y efectivo.

Ilustración 2



Fuente: Autoría propia.

Ilustración 3



Fuente: Autoría propia.

Ilustración 4

Fuente: Autoría propia.

Ilustración 5

Fuente: Autoría propia.

En la (ilustración 2) se presenta la página de descarga de Git, donde se debe hacer clic en "Descargas". La (ilustración 3) muestra el proceso de instalación del instalador de Git y su tamaño de archivo. Desde la (ilustración 4) hasta la (ilustración 5), se detalla el proceso de instalación del gestor, resaltando cada paso requerido para finalizar la configuración de Git en el sistema.

React

Se utilizó React como la principal tecnología para desarrollar la lógica y funcionalidad de la aplicación. React es una biblioteca de JavaScript versátil y eficiente que se destaca por su capacidad para construir interfaces de usuario interactivas y reactivas. Esta tecnología permite la creación de componentes reutilizables, facilitando la gestión del estado y la actualización dinámica de la interfaz. React.js es ampliamente adoptado en el desarrollo de aplicaciones web modernas debido a su rendimiento, flexibilidad y gran comunidad de desarrolladores.

Taildind Css

Se utilizó Tailwind CSS para definir el diseño y estilo de la interfaz de usuario (UI). Tailwind CSS es un marco de trabajo de utilidades que permite aplicar estilos de manera rápida y eficiente mediante clases predefinidas. Con Tailwind, es posible construir y personalizar elementos visuales como botones, campos de texto, imágenes, y listas de manera flexible y modular. Este enfoque facilita una separación clara entre el diseño y el contenido, mejorando la legibilidad del código y simplificando el mantenimiento y la actualización del proyecto.

Node.js

Se utilizó Node.js como entorno de ejecución para la gestión de dependencias y la automatización de tareas. Node.js es una plataforma de JavaScript basada en el motor V8 de Google Chrome, que permite la ejecución de código JavaScript en el servidor. Su capacidad para manejar paquetes y dependencias a través de npm (Node Package Manager) facilita la configuración y optimización del proceso de desarrollo. Node.js es conocido por su eficiencia en la construcción de aplicaciones escalables y de alto rendimiento. Su uso en el proyecto

proporcionó una base sólida para automatizar tareas y gestionar de manera efectiva los componentes del software.

Next.js

Se utilizó Next.js como marco de trabajo para el desarrollo de la aplicación. Next.js es un framework de React que facilita la creación de aplicaciones web escalables y optimizadas, ofreciendo características como renderizado del lado del servidor (SSR), generación de sitios estáticos (SSG) y enrutamiento automático. Con Next.js, se logró una configuración eficiente del proyecto y una gestión fluida de las dependencias y tareas de desarrollo. Su capacidad para combinar el renderizado en el servidor y en el cliente permite crear aplicaciones rápidas y con una excelente experiencia de usuario. La integración de Next.js proporcionó una base sólida para desarrollar una aplicación robusta y de alto rendimiento.

Prisma

Se utilizó Prisma como el ORM (Object-Relational Mapping) para la gestión de la base de datos. Prisma es una herramienta moderna y eficiente que simplifica la interacción con bases de datos al proporcionar una capa de abstracción que permite trabajar con datos de manera intuitiva y segura. Con Prisma, se definieron los modelos de datos y se realizaron consultas de manera estructurada, facilitando la gestión de esquemas y la sincronización con la base de datos. Esta integración optimizó el desarrollo y mantenimiento de la base de datos, proporcionando una experiencia de desarrollo más fluida y eficiente. Prisma se destacó por su capacidad para generar consultas SQL seguras y su compatibilidad con diversas bases de datos, lo que contribuyó a un desarrollo más ágil y efectivo.

PostCss

Se utilizó PostCSS para el procesamiento y optimización de hojas de estilo CSS. PostCSS es una herramienta flexible que permite transformar CSS con plugins, lo que facilita la adición de nuevas funcionalidades y la mejora del código de estilos. Con PostCSS, se aplicaron transformaciones como la autoprefixación de propiedades, la minificación del CSS y la

integración de características avanzadas de CSS mediante plugins personalizados. Esta configuración optimizó el rendimiento de las hojas de estilo y garantizó una compatibilidad más amplia con diferentes navegadores. PostCSS proporcionó una solución robusta y extensible para manejar el diseño y la presentación de la aplicación de manera eficiente.

PostgreSQL

Se utilizó PostgreSQL como la base de datos relacional para la gestión de datos. PostgreSQL es un sistema de gestión de bases de datos avanzado y de código abierto, conocido por su robustez, escalabilidad y compatibilidad con una amplia gama de tipos de datos. Su capacidad para manejar consultas complejas y garantizar la integridad de los datos lo hizo ideal para almacenar y gestionar la información de manera eficiente. La integración de PostgreSQL permitió una estructura de base de datos bien organizada y un rendimiento óptimo en la aplicación, asegurando que los datos se gestionaran de forma segura y efectiva.

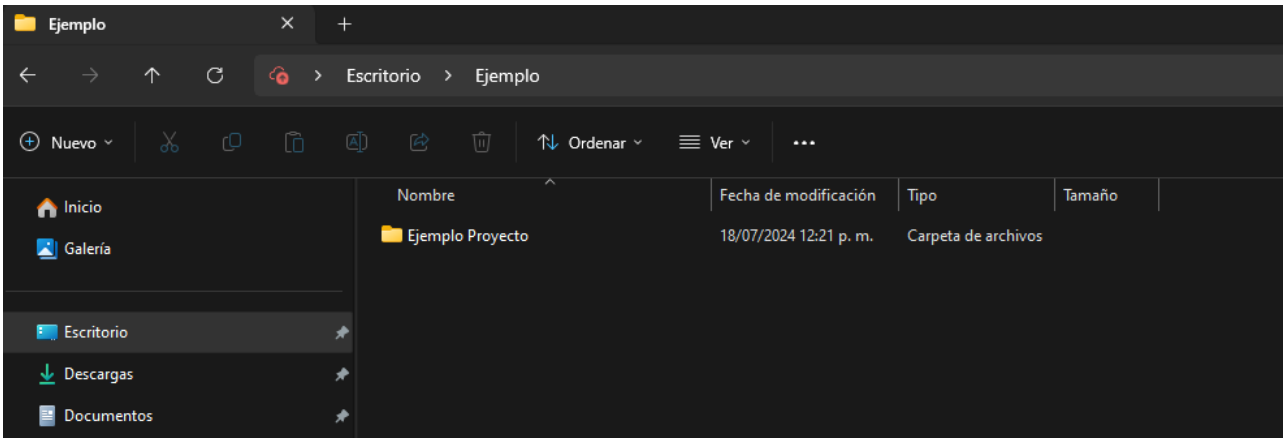
Instalación del software

Para desarrollar el software, es esencial seguir un procedimiento específico para transferir el proyecto desde Git a la máquina local. Este proceso consta de una serie de pasos organizados que aseguran la correcta obtención y configuración inicial del código fuente desde el repositorio remoto en Git al entorno de desarrollo local.

El primer paso para traer un proyecto desde Git a la máquina local es situarse en la carpeta del sistema donde se desea almacenar y trabajar con el código. Este paso es fundamental, ya que define el directorio principal en el que se establecerá la conexión con el repositorio remoto y se descargará el código fuente.

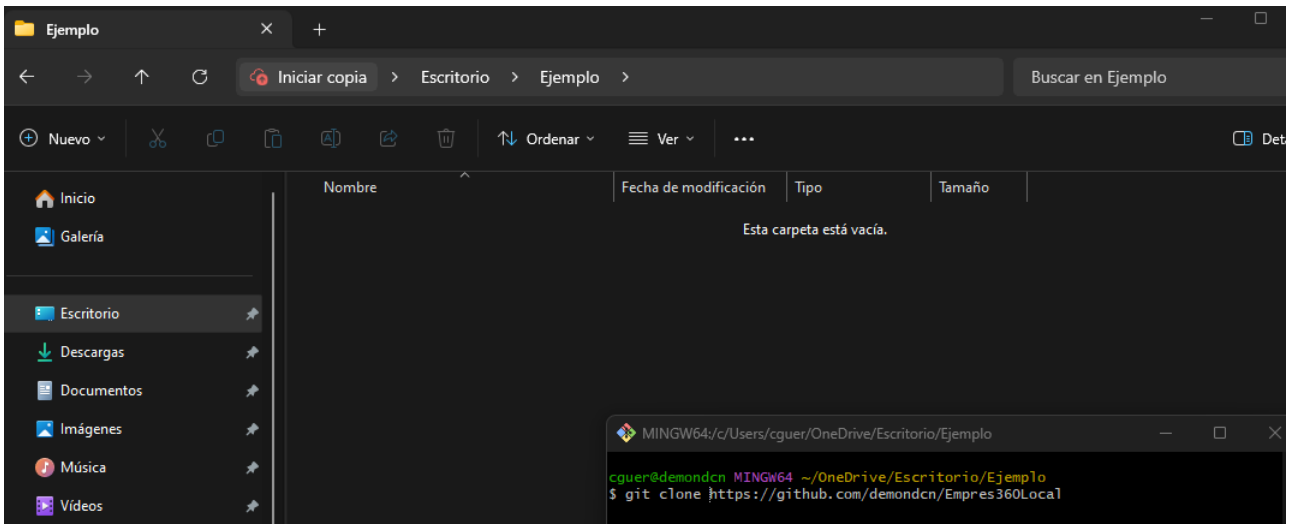
1. **Clonar el Repositorio desde Git:**

Ilustración 6



Fuente: Autoría propia.

Ilustración 7

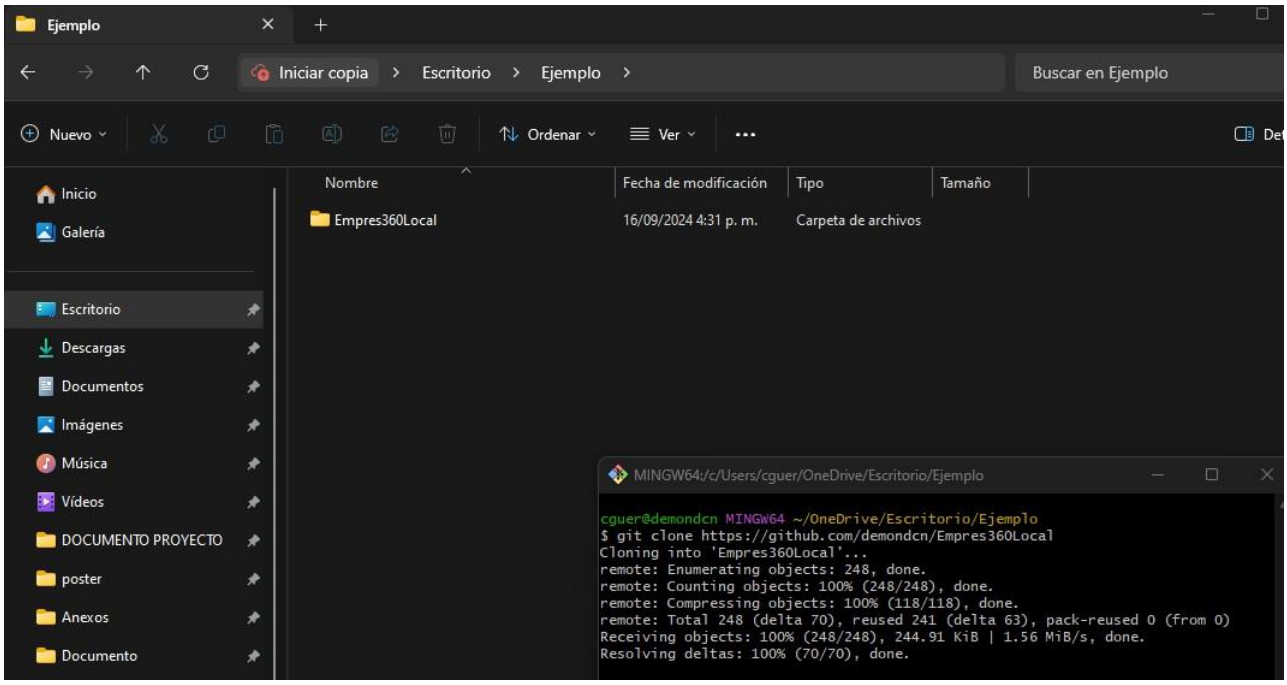


Fuente: Autoría propia.

En la (ilustración 6) y (ilustración 7), se ubica en la carpeta y se accede al repositorio en Git utilizando el comando git clone https://github.com/demondcn/Empres360Local. Esto descarga todos los archivos del repositorio remoto a la máquina local

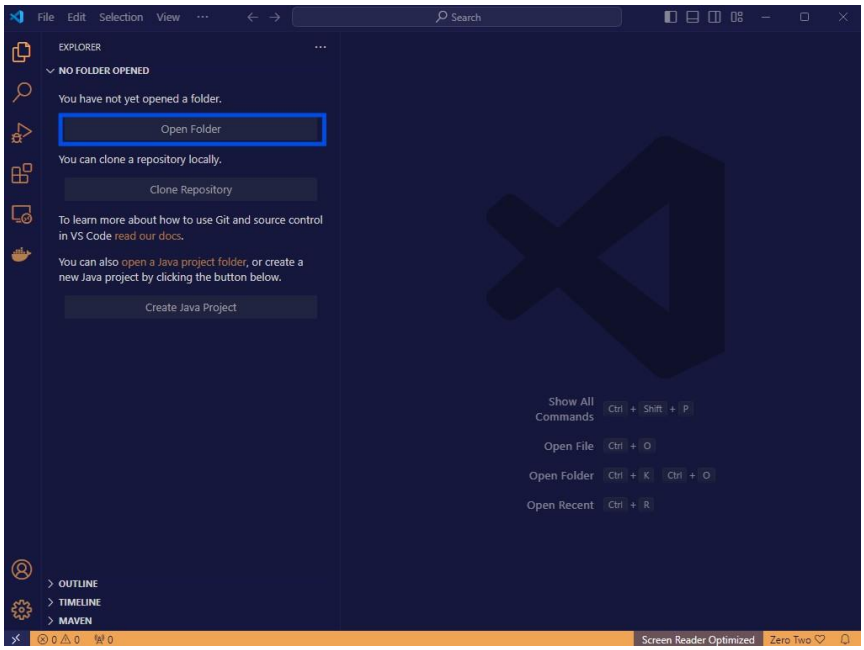
2. **Acceder a la Carpeta del Proyecto por Visual Studio:**

Ilustración 8



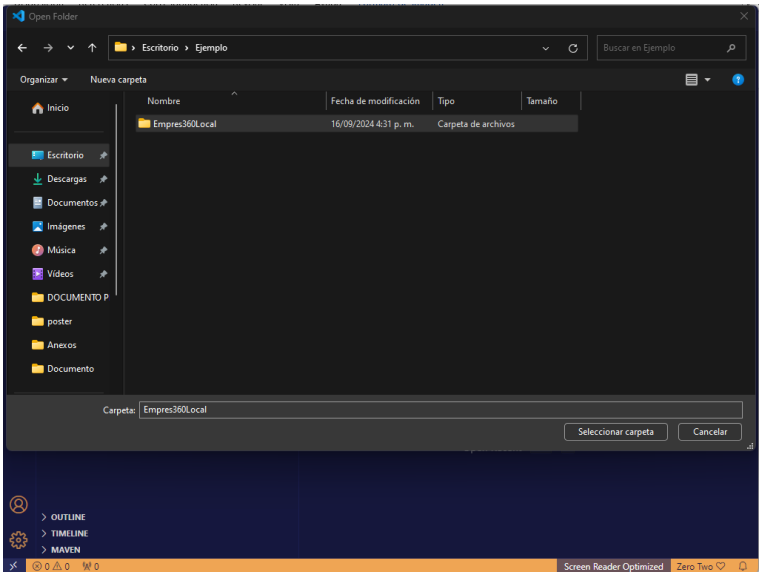
Fuente: Autoría propia.

Ilustración 9



Fuente: Autoría propia.

Ilustración 10

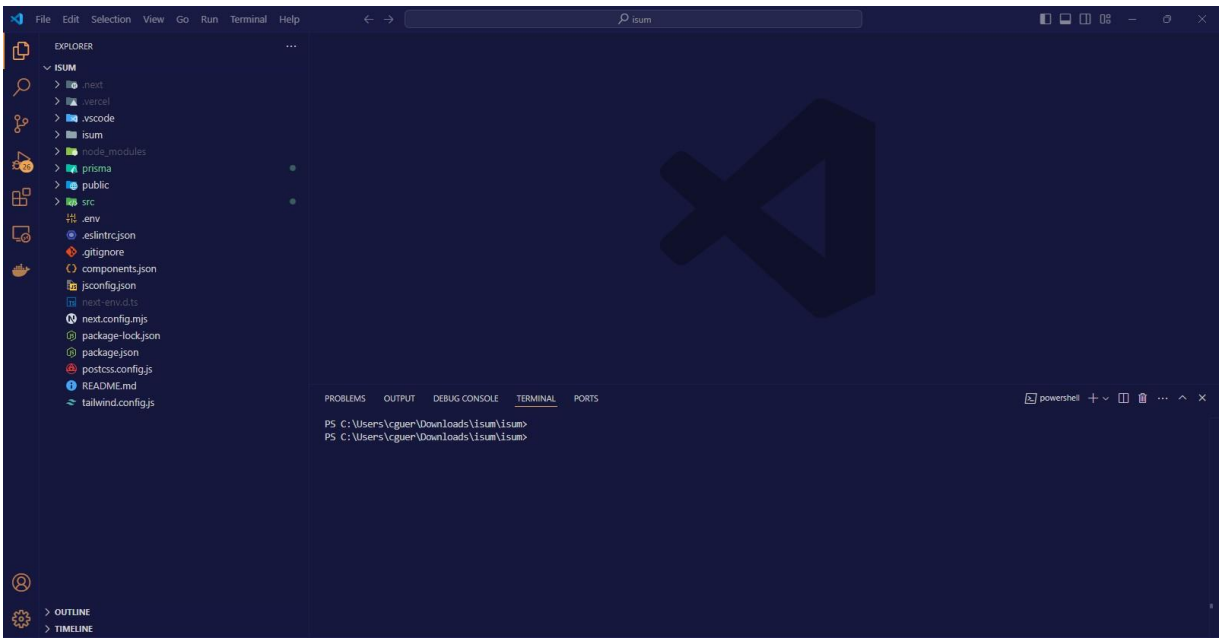


Fuente: Autoría propia.

En la (ilustración 8), verificamos que la carpeta Empres360Local haya sido descargada correctamente. En la (ilustración 9), hacemos clic en el botón "Open Folder" en Visual Studio y navegamos hasta la carpeta Empres360Local. La (ilustración 10) muestra detalladamente cómo se lleva a cabo este proceso.

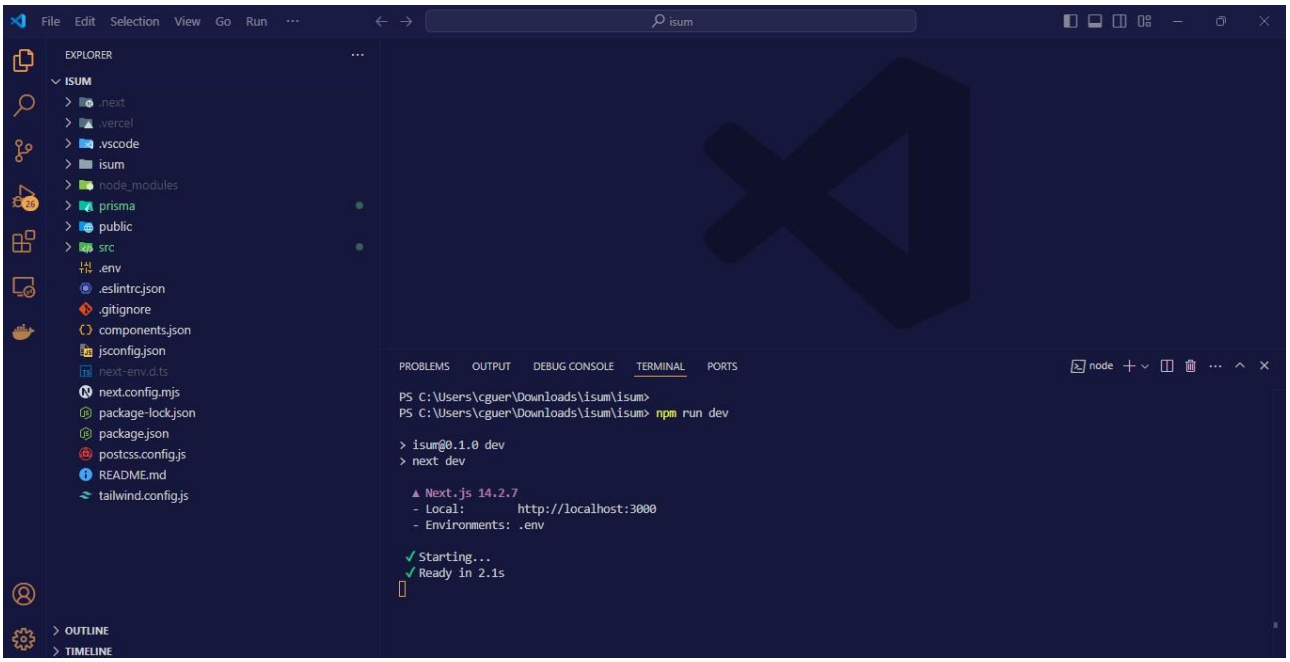
3. **Compilar:**

Ilustración 11



Fuente: Autoría propia.

Ilustración 12



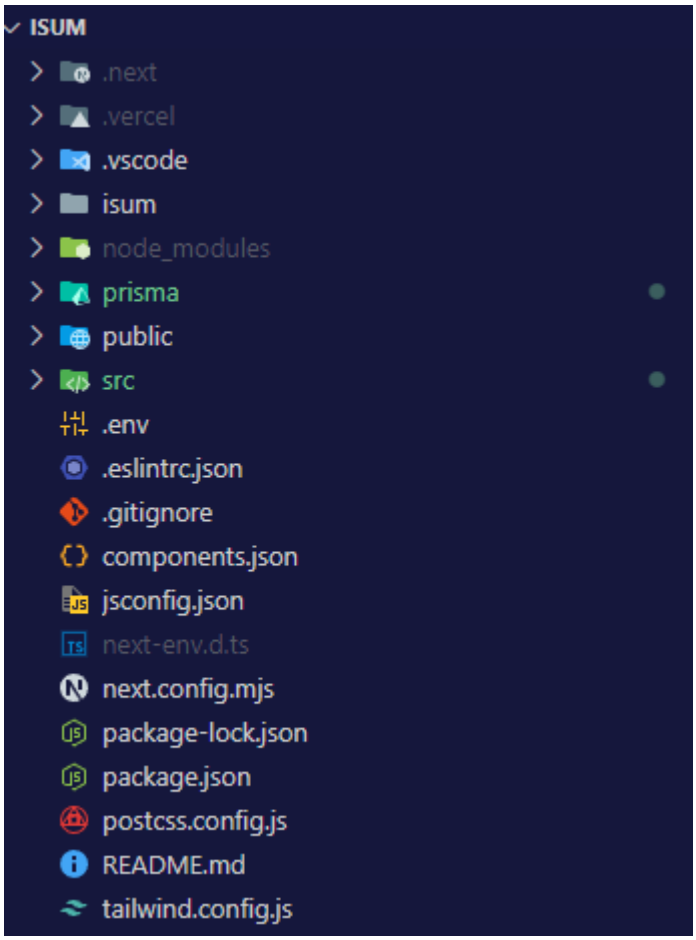
Fuente: Autoría propia.

En la (ilustración 11), verificamos que haya abierto el proyecto y abrimos la terminal de este y ponemos el siguiente comando: npm run dev. En la (ilustración 12),Podemos ver que se ejecuta el programa y nos da una url para ingresar al mismo.

Estructura Del Proyecto

Para el desarrollo del software en Next.js, se adoptó una arquitectura modular y bien estructurada, que optimiza el desarrollo, mantenimiento y escalabilidad de la aplicación. Esta organización permite una separación clara de responsabilidades, facilita la integración de componentes reutilizables y asegura un flujo de trabajo eficiente.

Ilustración 13



Fuente: Autoría propia.

En la (ilustración 13) se puede visualizar la estructura del proyecto. A continuación, detallaremos cada una de sus partes de manera específica.

Carpeta Prisma

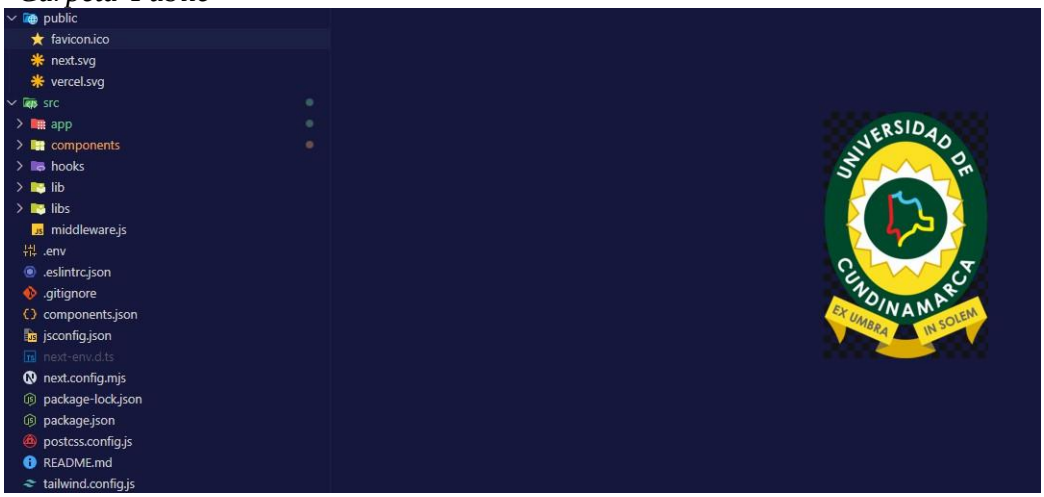
Una parte crucial de esta explicación es la carpeta Prisma, que juega un papel esencial en la configuración y gestión de la conexión a la base de datos. Prisma es una herramienta de mapeo objeto-relacional (ORM) que facilita la interacción entre el software y la base de datos, permitiendo una comunicación eficiente y estructurada. En la carpeta Prisma, se encuentran los archivos de esquema que definen la estructura de la base de datos, así como la configuración necesaria para establecer la conexión con el sistema de gestión de bases de datos.

Carpeta Public

La carpeta public desempeña un papel fundamental en la gestión de recursos estáticos, albergando los iconos y elementos gráficos utilizados en la página web. Esta carpeta está destinada a contener todos los archivos estáticos que deben ser accesibles públicamente en la web, facilitando su integración en la interfaz del usuario. En la siguiente ilustración, la (ilustración 14), se muestra un ejemplo de un icono almacenado en esta carpeta.

Ilustración 14

Carpeta Public



Fuente: Autoría propia.

Carpeta Src

La carpeta src (source) es el corazón del desarrollo de la aplicación, albergando la mayor parte del código fuente y la lógica del software. Está organizada para gestionar eficientemente los diferentes componentes y funcionalidades de la aplicación. A continuación, se describen las subcarpetas y archivos clave que forman parte de esta estructura.

middleware.js

Este archivo es esencial para la gestión de la seguridad y el enrutamiento en la aplicación. El middleware se encarga de implementar la protección de rutas, asegurando que solo los usuarios autenticados o autorizados puedan acceder a determinadas secciones del

software como se muestra en la ilustración 15.

Ilustración 15
Middleware.js

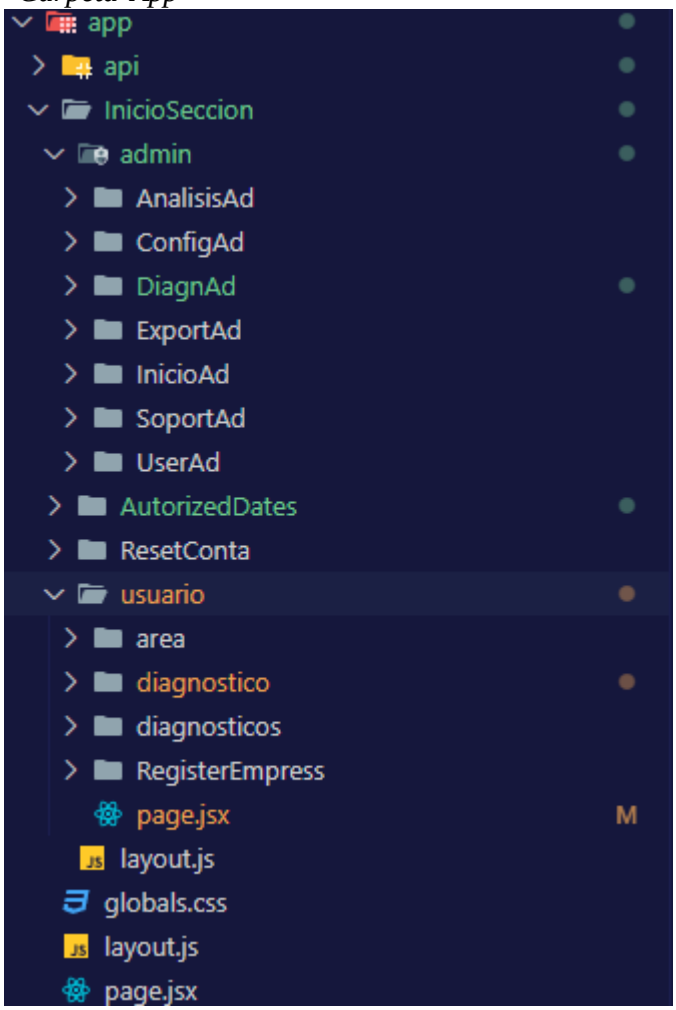
Fuente: Autoría propia.

Carpeta app

Esta carpeta alberga los archivos principales que configuran y definen la estructura de la aplicación. Incluye los archivos responsables de gestionar la integración de componentes y de establecer la configuración general del entorno de desarrollo como se puede evidenciar en la Ilustración 16.

Ilustración 16

Carpeta App



Fuente: Autoría propia.

Carpeta InicioSección y Subcarpetas

La carpeta Inicio de Sección organiza las páginas y vistas que los usuarios ven en la aplicación. Esta carpeta, junto con sus subcarpetas, está estructurada para facilitar el desarrollo y mantenimiento del software, albergando los archivos correspondientes a cada sección de la aplicación. Por ejemplo, el archivo usuario/page.jsx se encuentra en esta carpeta, como se muestra en la (ilustración 17).

Ilustración 17
usuario/page.jsx

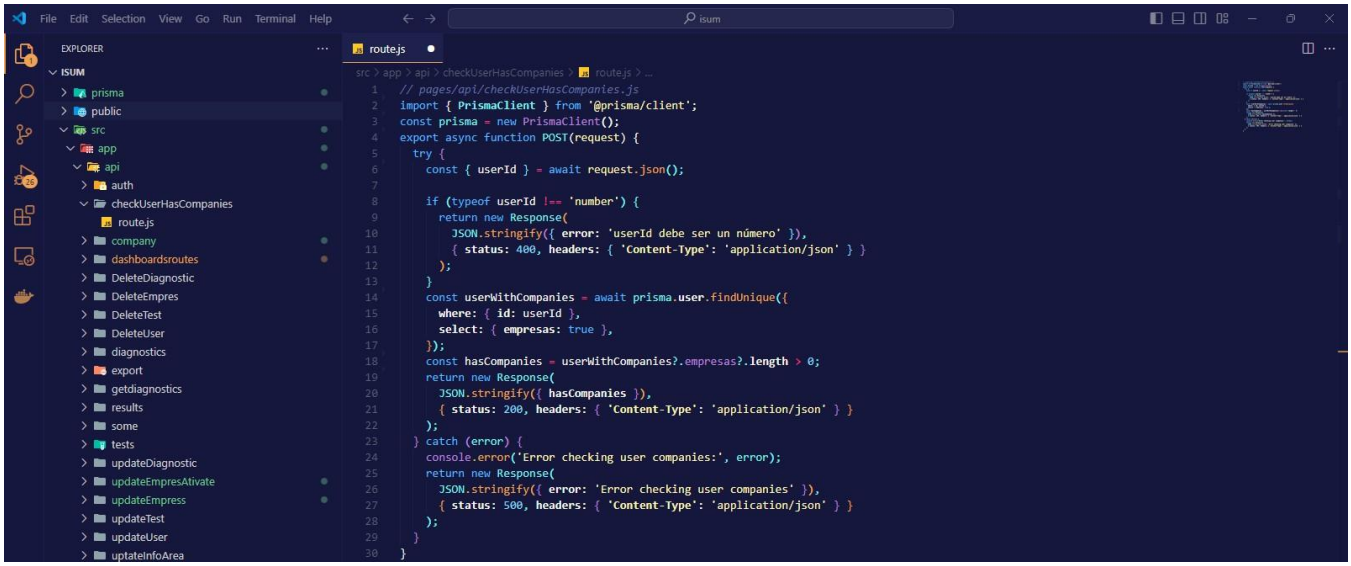
Fuente: Autoría propia.

Carpeta Api y Subcarpetas

La carpeta Api es esencial para gestionar la comunicación entre el frontend y el backend del software. Esta carpeta y sus subcarpetas están diseñadas para manejar las solicitudes HTTP, procesar la información recibida y enviar respuestas en formato JSON a la base de datos. Un ejemplo de esto es el archivo /api/checkUserHasCompanies/route.js, que se muestra en la (ilustración 18).

Ilustración 18

/api/checkUserHasCompanies/route.js

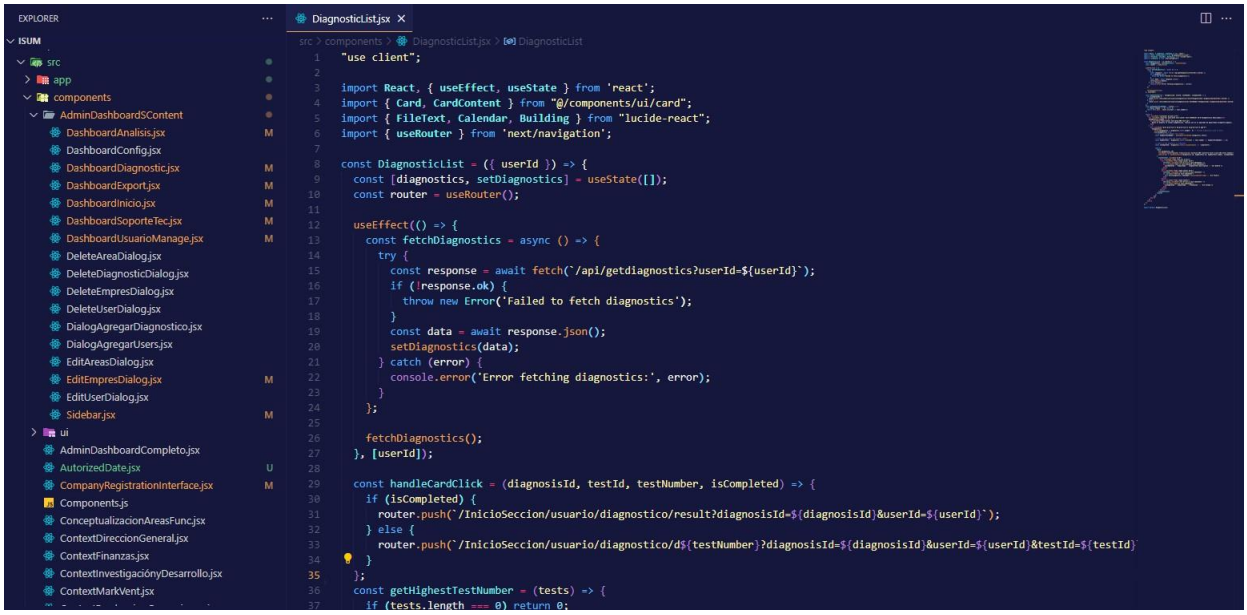


Fuente: Autoría propia.

Carpeta Components

La carpeta components y sus subcarpetas están dedicadas a la gestión de los componentes de React utilizados en la aplicación. Esta estructura es clave para la modularización y reutilización de los elementos de la interfaz de usuario (UI), permitiendo que los componentes se implementen fácilmente en diversas páginas y vistas del software. Un ejemplo destacado dentro de la carpeta components es DiagnosticList.jsx, como se muestra en la ilustración 19.

Ilustración 19
DiagnosticList.jsx



Fuente: Autoría propia.

Carpeta Lib y Libs

La carpeta lib se encarga de la importación y configuración de Tailwind CSS, una herramienta de diseño que facilita la creación de interfaces de usuario estilizadas de manera eficiente. Tailwind CSS proporciona una serie de clases utilitarias que se aplican directamente a los elementos HTML para diseñar componentes y páginas.

Además, la carpeta libs gestiona la conexión de Prisma a nivel global en todo el proyecto, como se muestra en la ilustración 20.

Ilustración 20
Carpetas Libs y Lib

Fuente: Autoría propia.

Modulo de Base de Datos Del Proyecto

Para el desarrollo del software en Next.js, se adoptó un modelo de base de datos relacional utilizando PostgreSQL, lo cual optimiza la gestión y manipulación de datos. Este enfoque permite una estructura de datos bien organizada, garantiza la integridad y consistencia de la información, y facilita la realización de consultas complejas. La elección de PostgreSQL proporciona un sólido marco para manejar relaciones entre datos y asegura un rendimiento eficiente en el acceso y procesamiento de la información.

Modelo Sql Utilizado

El esquema utilizado en el proyecto está diseñado para gestionar y organizar los datos de manera eficiente utilizando Prisma y una base de datos PosgresSql. Este esquema define las estructuras y relaciones entre diferentes modelos de datos, proporcionando una base sólida para el almacenamiento y manipulación de la información.

Descripción del Esquema:

- **User**: Este modelo representa a los usuarios del sistema, incluyendo campos como id, name, email, password, y relaciones con otros modelos como Diagnosis, Empresa, y PasswordReset. Se asegura de que cada usuario tenga una identidad única y se mantenga actualizado con marcas de tiempo (createdAt, updatedAt).

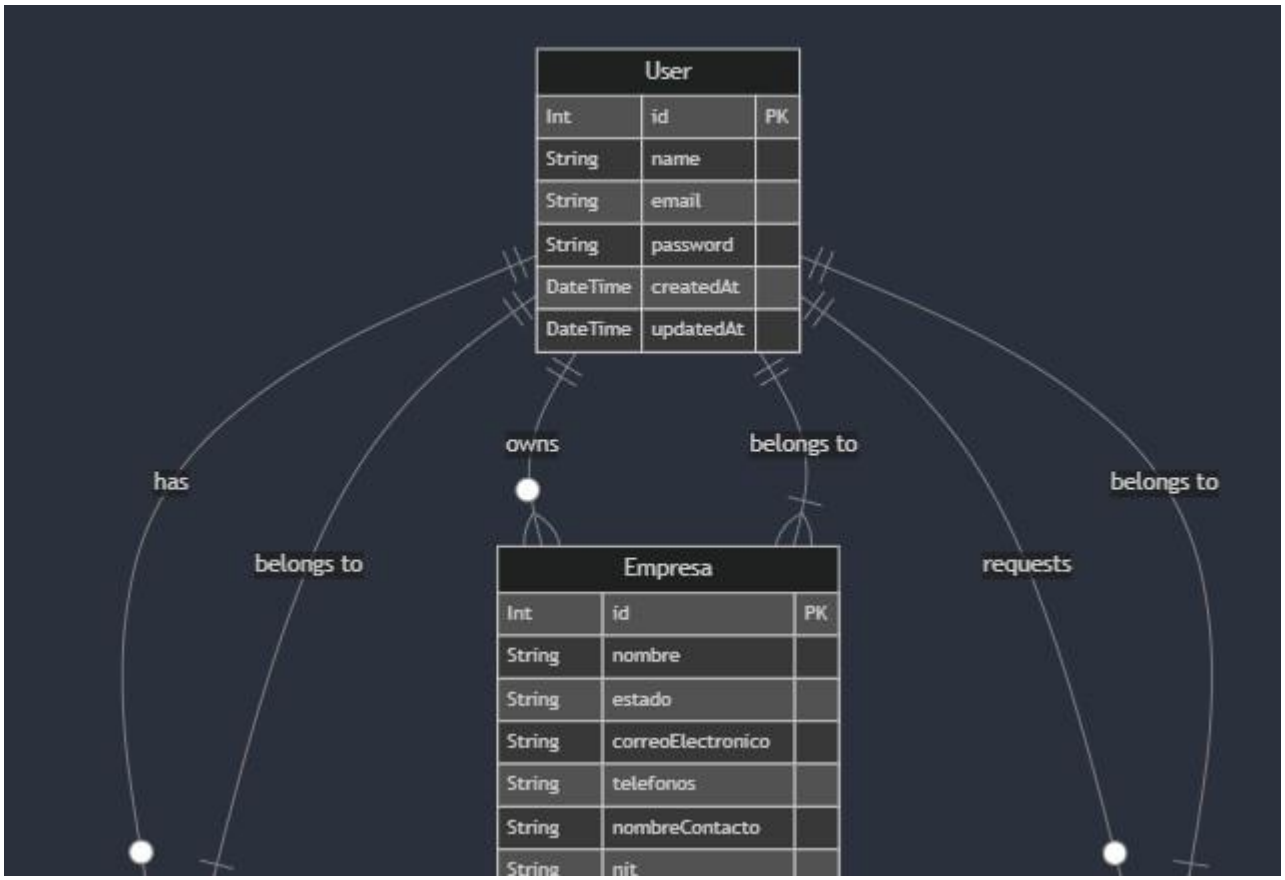
- **Diagnosis**: Representa los diagnósticos asociados a los usuarios. Incluye campos como id, userId, status, y relaciones con los modelos User y Test. Este modelo también gestiona la creación y actualización de diagnósticos.

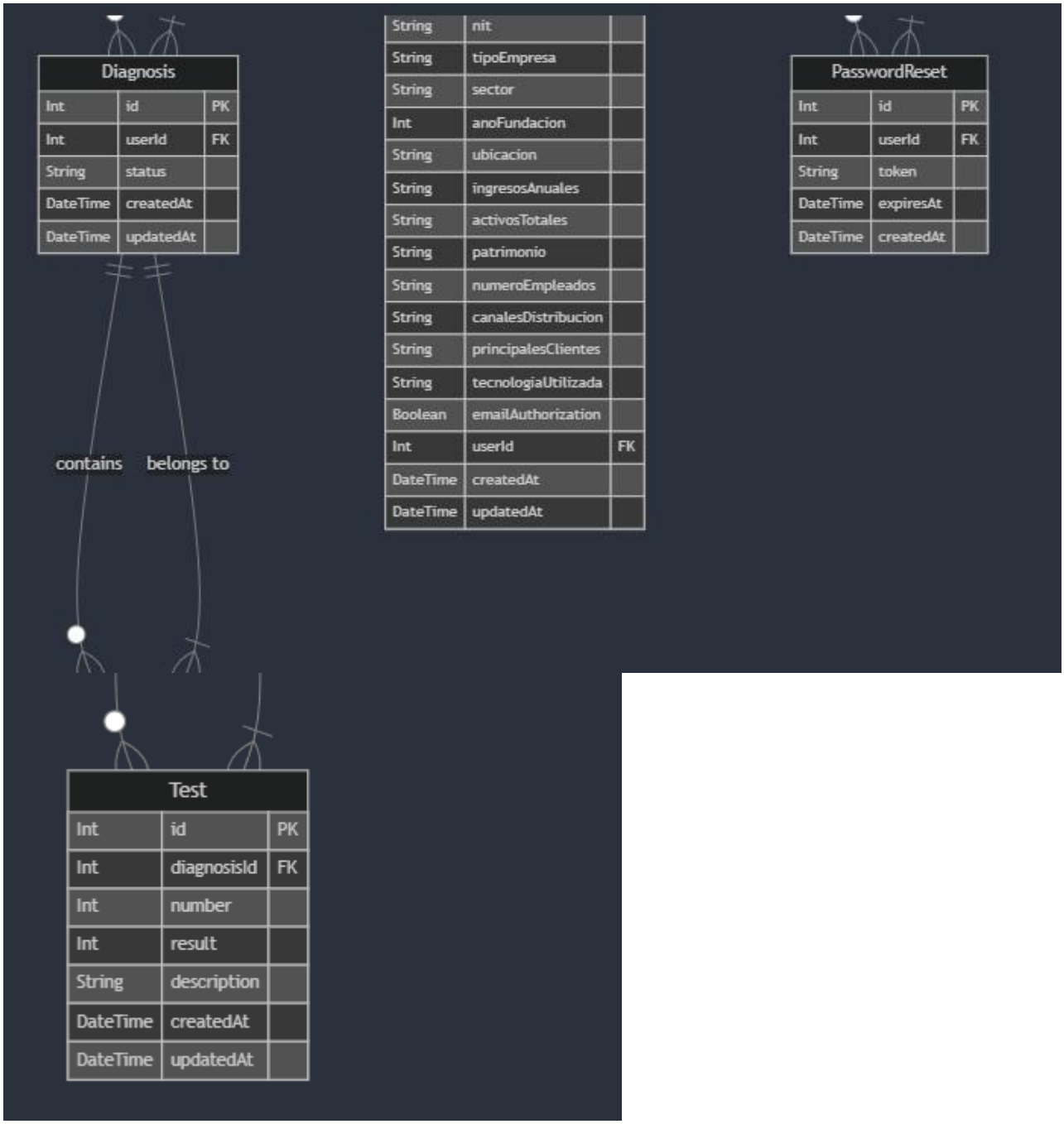
- **Test**: Este modelo almacena información sobre pruebas realizadas en los diagnósticos, incluyendo campos como id, diagnosisId, number, result, description, y relaciones con el modelo Diagnosis.

- **Empresa**: Representa a las empresas asociadas a los usuarios, con campos detallados como nombre, estado, correoElectronico, telefonos, y varios otros atributos relevantes. Este modelo también se relaciona con User y cuenta con un nombre de tabla personalizado (empresas).

- **PasswordReset**: Maneja las solicitudes de restablecimiento de contraseñas, incluyendo id, userId, token, expiresAt, y fechas de creación. Se relaciona con el modelo User para asociar cada solicitud de restablecimiento con un usuario específico.

Ilustración 21
Diagrama Relacional del Modelo





Fuente: Autoría propia.