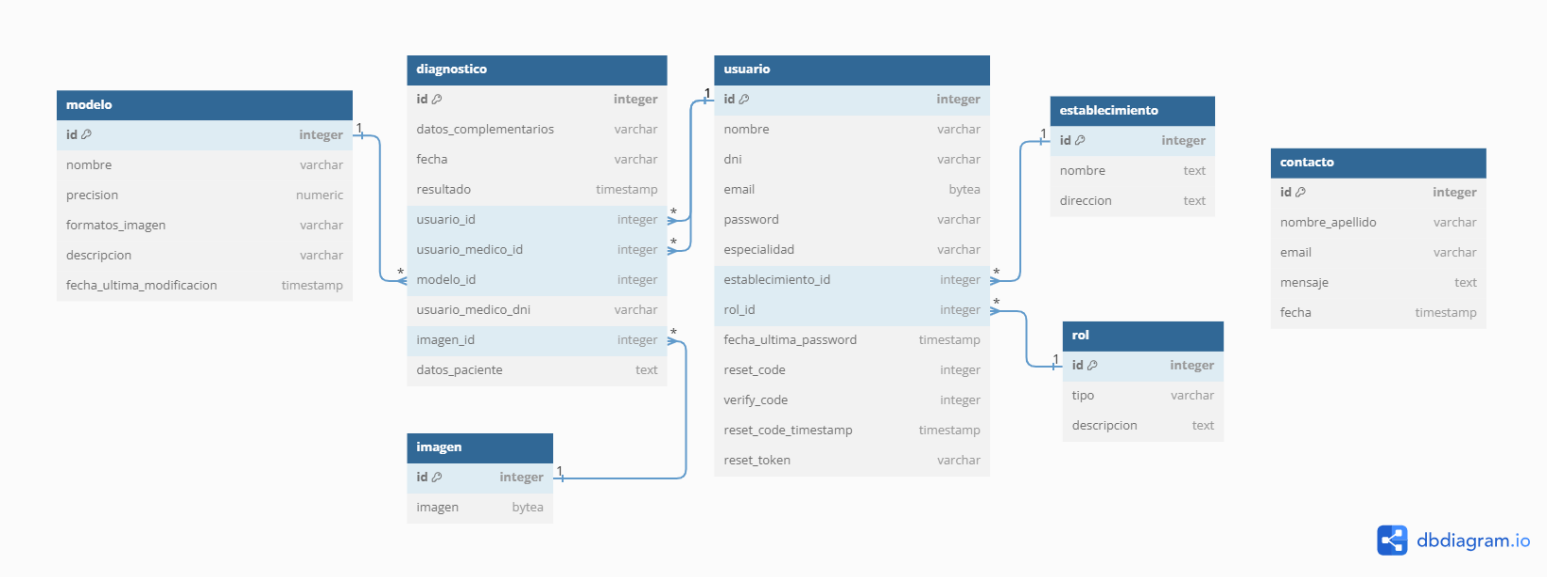
# Diseño técnico y de integraciones

## Introducción

Para cumplir con los objetivos de nuestro equipo de gestionar y almacenar los datos del sistema, ya sea de diagnósticos o usuarios llevamos a cabo el siguiente el diseño técnico que se explicará en este documento, para luego especificar cómo implementamos las integraciones con los demás componentes del sistema.

## Herramientas utilizadas

En primer lugar, implementamos una base de datos relacional para gestionar los datos de los diagnósticos y usuarios del sistema. Esta base de datos la implementamos con el motor de **PostgreSQL** en un servidor de donde fue expuesta a través de **Ngrok**.
A continuación podemos ver el diagrama de entidades y relaciones:

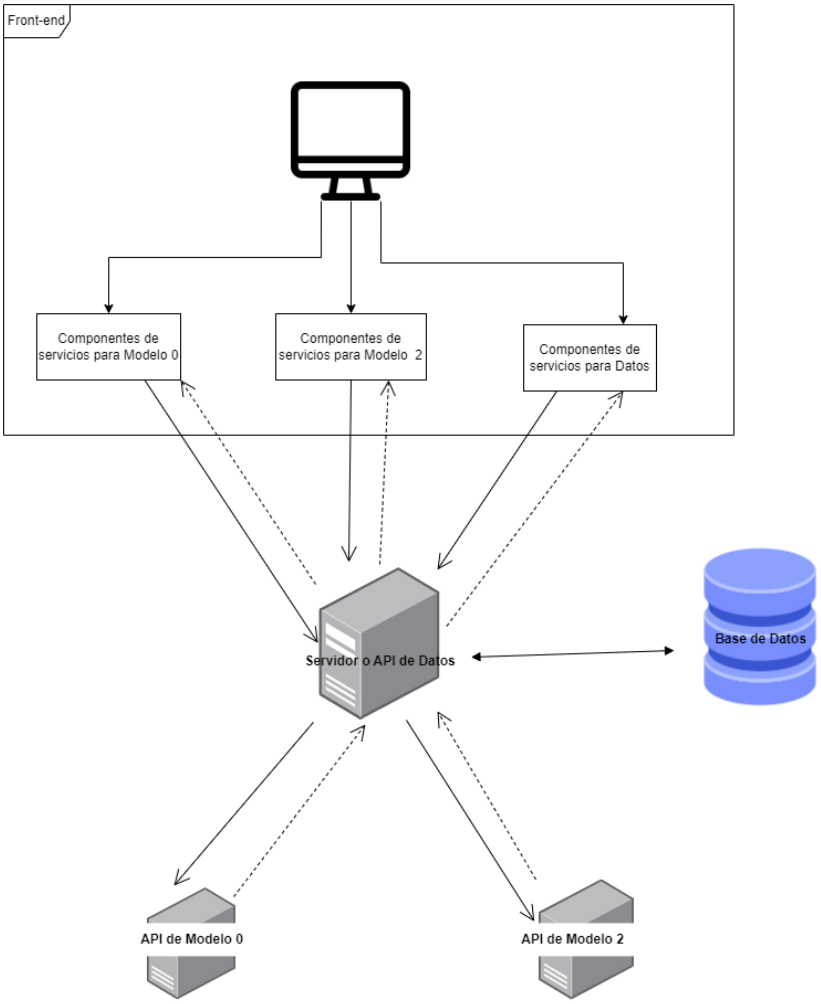



Una documentación extensa y detallada de la base de datos se puede ver en el siguiente enlace: Documentación BBDD

Luego, como nuestro equipo se encargó de realizar el desarrollo backend del sistema, desarrollamos una API Rest tanto para que desde el frontend se pueda acceder y/o guardar nuevos datos, como para realizar la comunicación con los modelos de inteligencia artificial que realizan las predicciones para los diagnósticos.
Las herramientas que utilizamos para esta API fueron, en primer lugar el lenguaje de programación **Python** debido a que empleamos el framework **Flask-RestX** de este

lenguaje. Este framework nos permitió desarrollar la API facilitandonos tareas debido a sus funcionalidades incluidas. Una de estas funcionalidades fue **Swagger.** Esta herramienta nos permite generar una documentación interactiva y gracias al framework que se realice automáticamente al desarrollar los diferentes endpoints, simplemente agregando algunos decoradores y sin tener que realizar configuraciones complejas.
Para el manejo del repositorio, versión y documentación complementaria utilizamos **GitHub**.
Por último, para hostear esta API y que sea accesible a través de internet utilizamos el sitio **Render.** Las características más notables de este sitio que nos sirvieron para el proyecto fueron su versión gratuita, conexión con repositorio de github lo que permite un deploy en cuestión de segundos, autodeploy de actualizaciones cada vez que se actualiza el repositorio de GitHub, dashboard para ver logs y eventos, configuración de variables de entorno lo que nos permitió darle más seguridad a nuestras claves, y métricas de uso de recursos.
La API con su interfaz Swagger se puede ver en el siguiente enlace:
https://api-resultados.onrender.com
El repositorio de GitHub con una documentación más detallada de los endpoints y sus usos se encuentra en el siguiente enlace:https://github.com/LucasPrieto30/ApiResultados#readme

## Integraciones



Como podemos ver en el gráfico de componentes del sistema, nuestro equipo realizó una función de “mediador” entre todos los demás componentes. Es decir, todas las solicitudes y acciones que se realizan en el frontend se comunican con nuestro servicio de API y base de datos, y así también la información de los modelos pasa por nosotros antes de mostrarse en el frontend.
El uso de APIs facilitó mucho esta comunicación con los demás componentes, en especial con los modelos, lo que nos permitió transmitir la información necesaria utilizando simples métodos **HTTP** con determinados parámetros.