



Ingeniería en Computadores

CE3101

Bases de Datos

Proyecto 2 – Documentación Técnica

Profesor:

Marco Rivera Meneses

Estudiantes:

Brayan Alpízar

Christian Navarro

Jorge Gutiérrez

Dario Garro

Giancarlo Vega Marín

I Semestre 2025

Fecha de entrega: 23/06/2025

Documentación técnica

Diagrama Conceptual

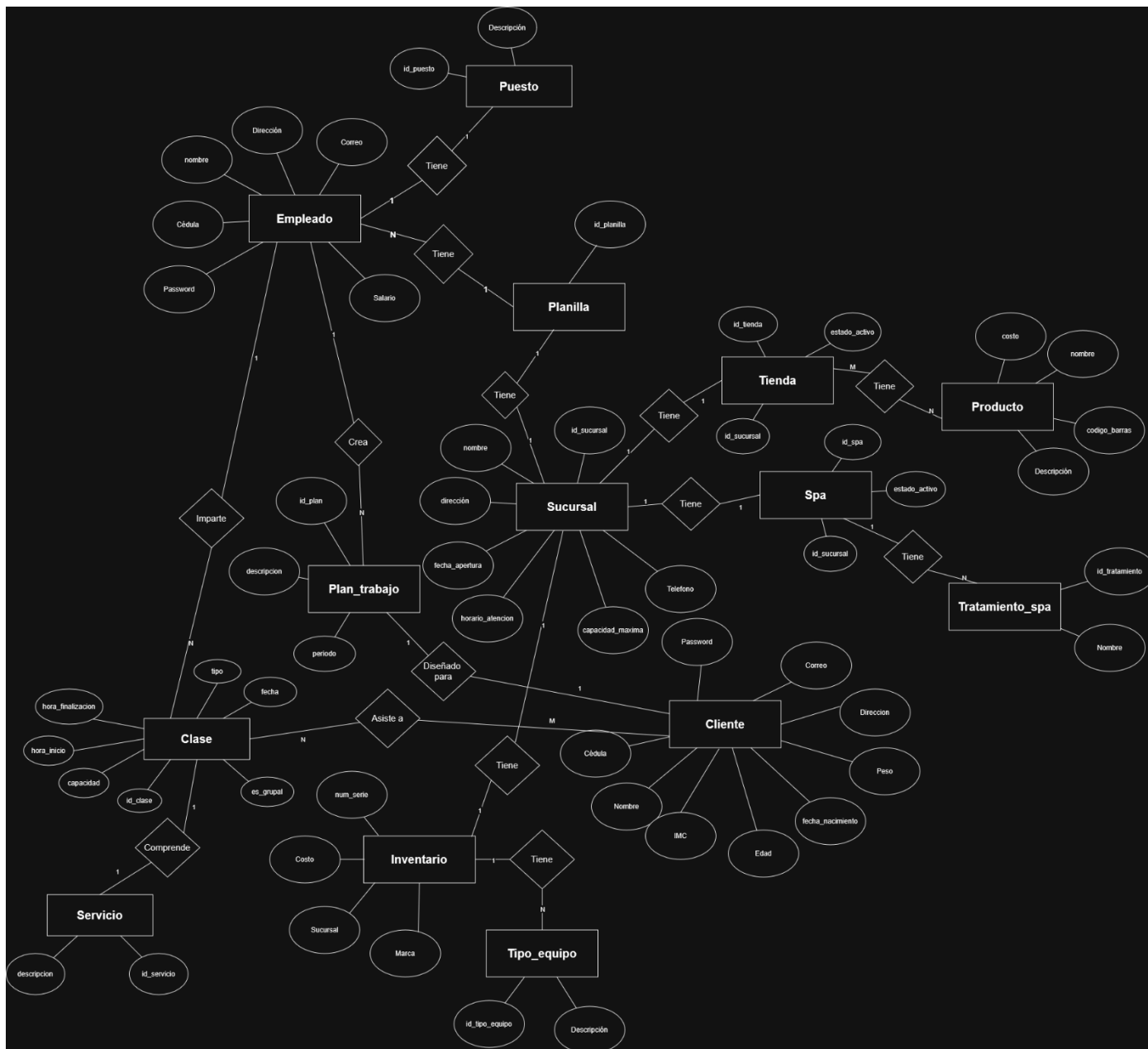
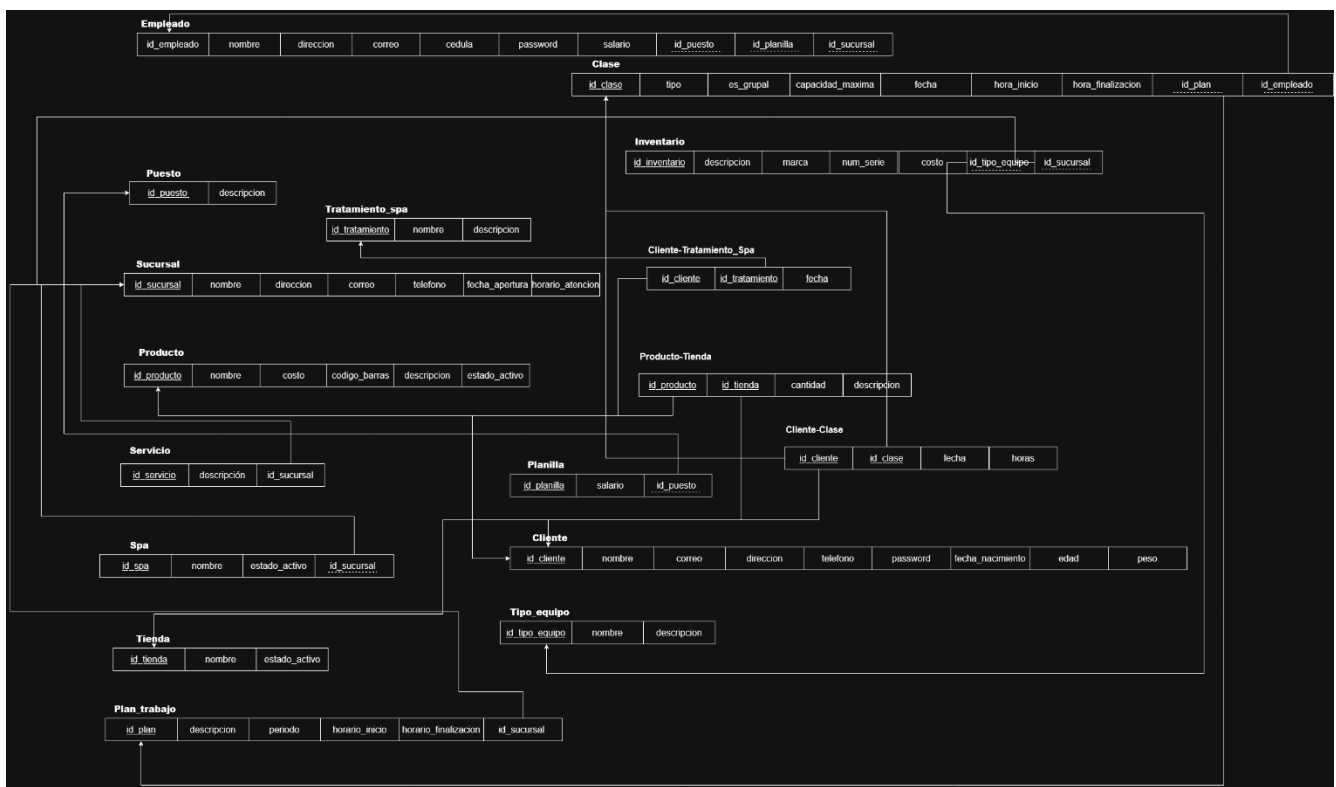


Diagrama de base de datos (Relacional)



Métodos implementados, estructuras de datos y algoritmos (Koki, edite esto)

El sistema GymTec se construye sobre una base de datos relacional que organiza y gestiona la información operativa, administrativa y de servicios del gimnasio. Las tablas están interrelacionadas para permitir el control detallado de empleados, clientes, sucursales, clases, tiendas y servicios.

La tabla Empleado almacena los datos del personal, incluyendo identificador único `id_empleado`, información personal, salario, y sus asociaciones con una Sucursal, un Puesto y una Planilla. Puesto describe el cargo desempeñado. Planilla define el salario relacionado al puesto.

Sucursal representa las distintas ubicaciones del gimnasio, registrando dirección, contacto, fecha de apertura y horarios. En cada sucursal se relacionan otros elementos como clases, inventario, servicios, tiendas y spas.

Cliente guarda la información personal y de acceso de los usuarios del gimnasio. Se relacionan con las tablas Cliente-Tratamiento_Spa y Cliente-Clase, que registran la participación del cliente en tratamientos de spa y clases respectivamente, junto con fechas y horas.

Clase almacena las sesiones de entrenamiento ofrecidas, con detalles como tipo, capacidad, si es grupal, horarios, plan asociado y el empleado que la imparte. El Plan_trabajo especifica turnos y horarios vinculados a la sucursal.

Inventario administra los equipos con su descripción, marca, número de serie y costo. Se asocian a la sucursal y a un Tipo_equipo, el cual describe el tipo y uso del equipo.

Producto representa los artículos disponibles, con costo, código de barras y estado de activación. Estos se enlazan a la tabla Producto-Tienda, que detalla la distribución del producto entre tiendas (`id_tienda`), cantidad y descripción adicional.

Tienda, Servicio y Spa son recursos localizados en una sucursal, cada uno con un estado de activación para su gestión. Los servicios se registran por sucursal, mientras que los tratamientos de spa se describen en Tratamiento_spa y su uso se registra en Cliente-Tratamiento_Spa.

Esta estructura de base de datos garantiza integridad referencial mediante claves foráneas, y permite realizar consultas complejas relacionadas con el personal, clientes, programación de clases, ventas, tratamientos y administración de recursos en distintas sucursales del gimnasio.

Descripción de los store procedures.

La descripción de los store procedures, se encuentra en un documento aparte, llamado Anexo de Documentación Técnica - Store Procedures”, debido a que esta sección era muy grande, causando desorden en el documento.

Descripción de la arquitectura

La arquitectura del sistema se organiza en capas con responsabilidades claras, facilitando mantenimiento y escalabilidad. El sistema está distribuido utilizando dos bases de datos: una base de datos relacional en PostgreSQL, y una base de datos empotrada en SQLite. La primera base de datos se conecta a un backend desarrollado en C#, que funciona como una API REST; la segunda base empotrada se conecta igual al backend, pero se debe sincronizar con el sistema centralizado (PostgreSQL). Esta API REST provee los servicios y puntos finales necesarios para que una aplicación Frontend desarrollada en React pueda consumir y manipular los datos. La capa de presentación (Frontend) está en un proyecto separado y ofrece la interfaz gráfica para estudiantes, profesores y administradores, comunicándose con el backend a través de esta API REST para enviar y recibir datos. La capa de lógica de negocio (Backend) está organizada en carpetas para controladores, modelos y utilidades. Los controladores gestionan las solicitudes del Frontend y coordinan el flujo de datos hacia las bases de datos. Los modelos representan las entidades que reflejan las tablas de PostgreSQL, facilitando el mapeo objeto-relacional y objeto-documento respectivamente.

Los modelos de entrada y salida (Data_input_models y Data_output_models) permiten transformar y validar datos para proteger la integridad y optimizar la comunicación. La carpeta Utilities contiene funciones auxiliares reutilizables, incluyendo la gestión de conexiones a PostgreSQL, y operaciones comunes. La capa de acceso a datos gestiona la conexión con ambas bases de datos, realiza consultas y operaciones CRUD usando los modelos, asegurando integridad referencial y manejando transacciones cuando es necesario en PostgreSQL. La organización del proyecto incluye carpetas separadas para archivos binarios generados (bin, obj) y código fuente, manteniendo una estructura modular y limpia. Los proyectos Frontend y Backend están separados para facilitar desarrollo y despliegue independientes. Este diseño modular permite que el sistema sea extensible, seguro y

mantenible, facilitando la integración de nuevas funcionalidades y adaptación a futuros requerimientos.

Adicionalmente, se presenta una aplicación móvil desarrollada en React Native. Dicha aplicación busca utilizar las bases de datos empotradas en SQLite previamente mencionadas. El proceso de sincronización tiene como objetivo guardar los datos enviados por la app, y recibir los datos actualizados y sincronizados del sistema centralizado.

Problemas Encontrados

- Cuando se realizaron los endpoints en el Backend, existieron algunos conflictos para definir qué cosas se envían al Frontend. Por ende, se solucionó definiendo un formato estándar para todas las consultas que contengan un estado y el mensaje.
- Al hacer los store procedures, a la hora de recuperar las tablas resultado, fue necesario la creación de modelos específicos para cada query. A diferencia del proyecto pasado, en la cual existía un modelo "Espejo" de cada tabla de la base de datos
- Un problema fue que PostgreSQL solo permite 4 conexiones a la vez, entonces cuando trabajamos todo el grupo colapsaba. Llegó un punto en donde por limitaciones del servicio, había desconexiones y se debía desconectar la base durante unos minutos para poder seguir.

Problemas conocidos

- Hubo problemas con la conexión del frontend, el backend y la base de datos. Ciertas funciones como editar sucursales, consulta de clientes, y algunas inserciones causaron errores, donde la base de datos no procesó los datos enviados, y por ende el frontend no recibió nada.
- En la creación de la base de datos SQLite, no se lograron replicar los queries utilizados en la base hecha en PostgreSQL, debido a que muchas funciones como store

procedures y triggers no existen en SQLite, forzando a la creación de nuevos queries desde cero.

- No se logró la sincronización de datos entre la base de datos SQLite, y la base de datos PostgreSQL.

Actividades Planeadas

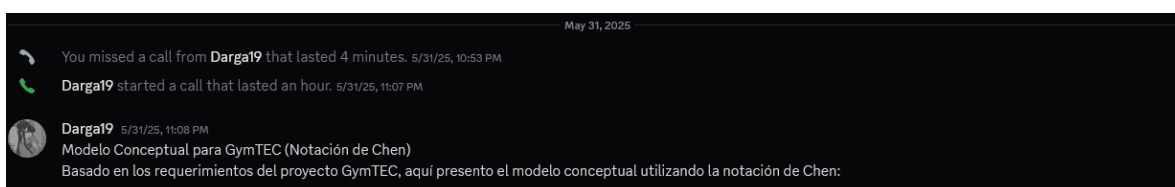
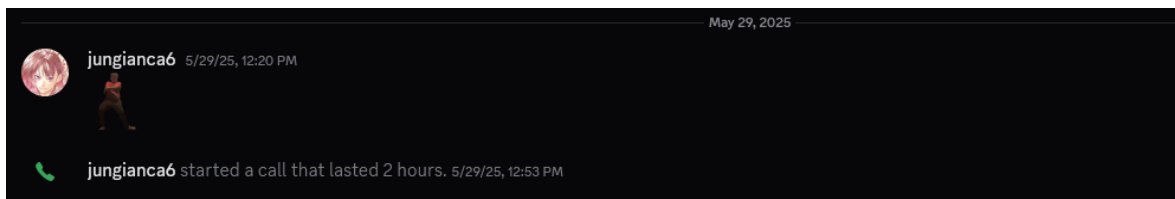
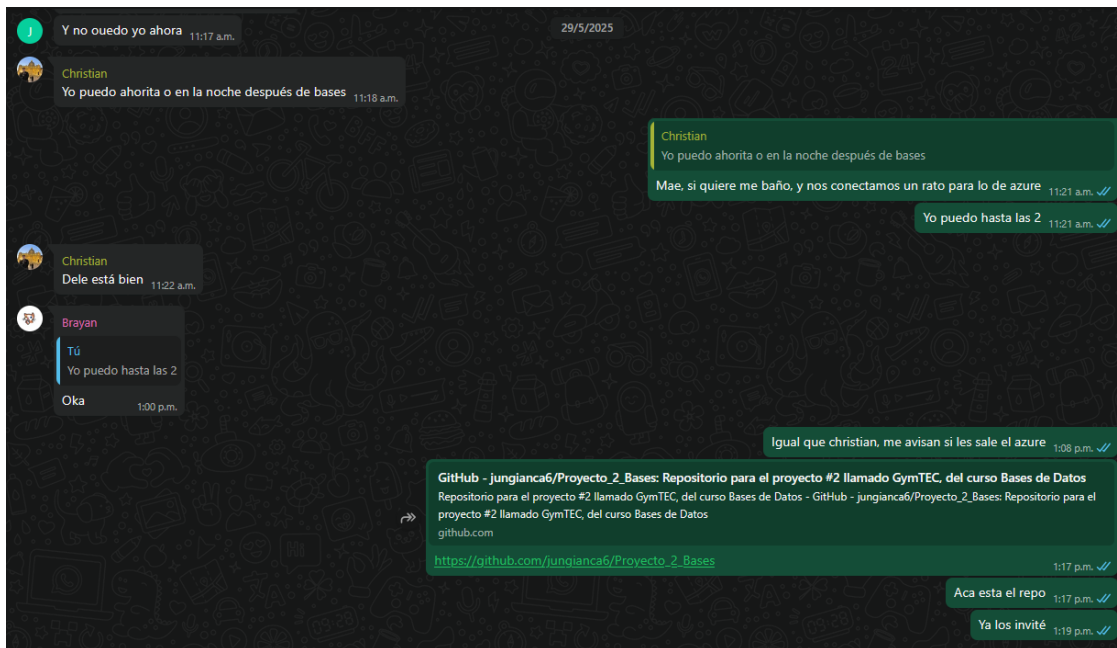
Para trabajar bien en equipo, se harán reuniones por Discord donde todos puedan participar y resolver dudas en tiempo real. Además, se usará WhatsApp para comunicarnos rápido y mantenernos al tanto del progreso.

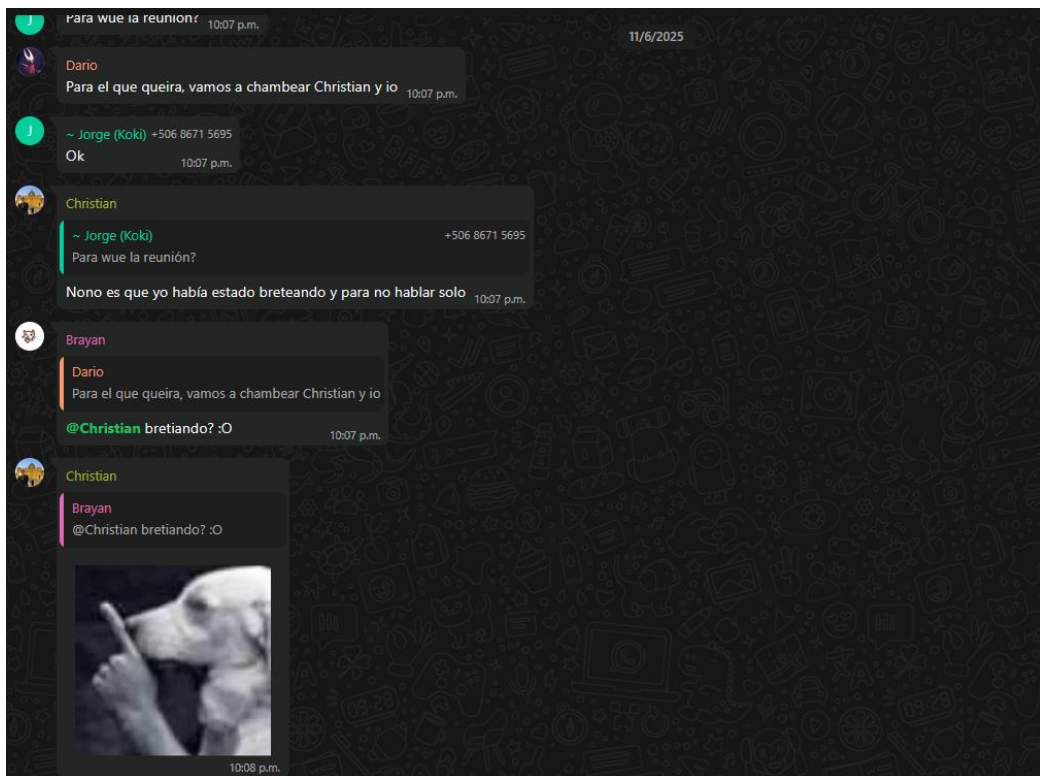
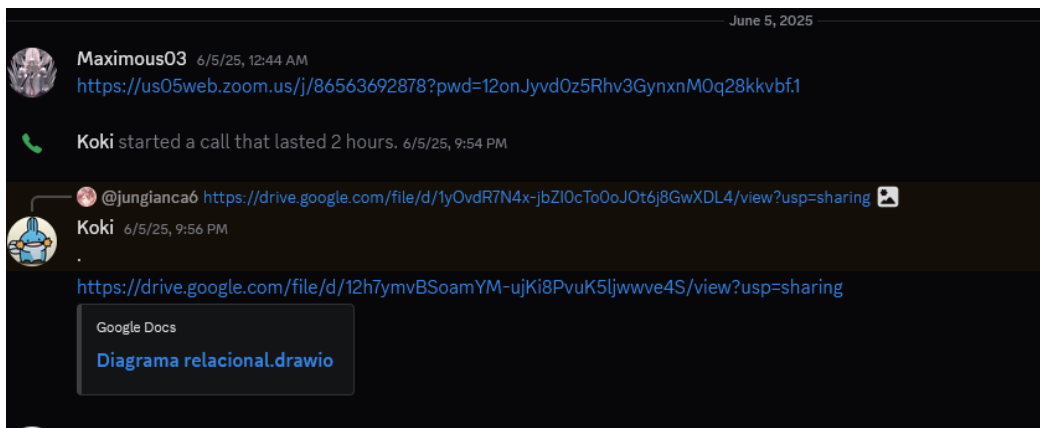
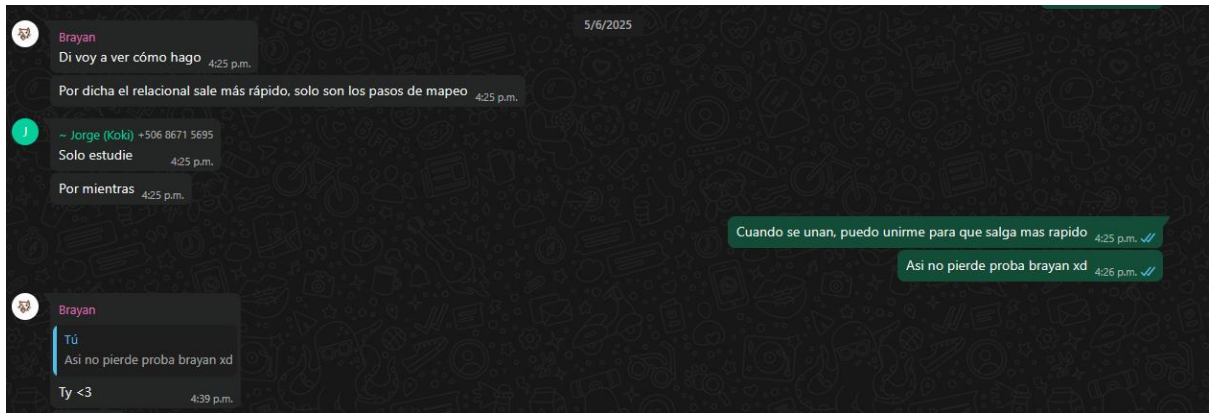
Al inicio del proyecto, se hicieron varias actividades importantes. Se creó el diagrama relacional y el diagrama conceptual para definir cómo se estructurará la base de datos. Esto ayudó a organizar las tablas y sus relaciones.

También se acordaron los modelos de datos que se usarán para enviar y recibir información entre el Frontend y el Backend, asegurando que ambos lados entiendan bien qué datos manejar. Se diseñó la estructura de carpetas del proyecto para que el código y los archivos estén organizados. Esto facilita que todos trabajen de forma ordenada y que el proyecto sea más fácil de mantener.

Por último, se diseñó la estructura de carpetas del proyecto para que el código y los archivos estén organizados. Esto facilita que todos trabajen de forma ordenada y que el proyecto sea más fácil de mantener.


Sesiones





June 11, 2025

Maximous03 started a call that lasted 3 hours. 6/11/25, 10:06 PM

**jungianca6** 6/11/25, 11:51 PM

Creación de página de inventario del admin

Chris-0307 • just now

Paginas del admin de servicios y tipos de equipo hechas

Chris-0307 • 10 minutes ago

Página del admin de empleados creada

Chris-0307 • 22 minutes ago

Página de tipos de planilla en admin creada


Chris-0307 • 28 minutes ago

Página del admin de puestos creada


Chris-0307 • 1 hour ago


Página de tratamientos del admin hecha


Chris-0307 • 1 hour ago




June 18, 2025

**Darga19** 6/18/25, 12:26 AM
gymtec.postgres.database.azure.com
Darga1905
Dagamo1905.

**@Darga19** gymtec.postgres.database.azure.com

**jungianca6** 6/18/25, 12:30 AM
.

**SpaceBA** 6/18/25, 1:33 AM
salu2

jungianca6 started a call that lasted 5 hours. 6/18/25, 3:50 PM

You missed a call from Maximous03 that lasted a few seconds. 6/18/25, 8:31 PM

Jorge Gutiérrez Vindas started a call that lasted 6 hours. 6/18/25, 8:34 PM

Conclusiones

Se logra desarrollar una aplicación que permite manejar la mayoría de la descripción del caso expuesto. El backend y frontend, instalados en la nube por medio de Azure, mantienen una conexión estable y exitosa, enviándose y recibiendo datos en simultáneo, conectados a una base de datos centralizada que mantiene toda la información relevante, y permite la creación de nuevos datos por medio de este sistema.

Sin embargo, se presentan ciertas fallas en algunas peticiones y envíos de datos, lo cual causa que el sistema y la aplicación no sean completamente funcionales. A su vez, la aplicación móvil no logra sincronizarse con las bases de datos centralizado, permitiendo solamente el envío y recibimiento de datos limitados a una base de datos empotrada, previamente creada y establecida sin ninguna actualización.

Recomendaciones

Se recomienda una mejor investigación con respecto a las bases empotradas como SQLite, ya que un gran problema en el proyecto fue la falta de conocimiento sobre ellas. A su vez, investigar más a fondo sobre sincronización entre las bases

Se debe tener una mejor comunicación a la hora de crear el frontend, backend y bases de datos por separado, ya que hubo varios casos en que se crearon datos de más que era innecesarios, o faltaban datos importantes.

Bibliografía:

Elmasri, R., & Navathe, S. B. (2016). Fundamentals of database systems (7th ed.). Pearson.

Fowler, M. (2003). Patterns of enterprise application architecture. Addison-Wesley.

PostgreSQL. (s,f). PostgreSQL documentacion.
<https://www.postgresql.org/docs/17/index.html>

Microsoft. (2023). Entity Framework Core documentation.
<https://learn.microsoft.com/en-us/ef/core/>

Fielding, R. T. (2000). Architectural styles and the design of network-based software architectures (Doctoral dissertation, University of California, Irvine).
https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf

<https://www.youtube.com/watch?v=XVcyVnNWUwo>

<https://www.youtube.com/watch?v=C5rtNyOHhnM>

Link de Github:

https://github.com/jungianca6/Proyecto_2_Bases