### **ProjectSync - Backend (Go)**

#### Objetivo:

Desarrollar un servidor backend robusto utilizando Go y PostgreSQL para manejar la autenticación de usuarios, gestión de proyectos, gestión de tareas y sincronización de datos para la aplicación de gestión de proyectos "ProjectSync".

#### Requerimientos:

1. **Autenticación de Usuarios**:
   * Implementar puntos finales de autenticación de usuarios para registro, inicio de sesión, cierre de sesión y gestión de tokens.
   * Utilizar métodos de autenticación seguros como JWT (JSON Web Tokens) para la gestión de sesiones.
2. **Gestión de Proyectos**:
   * Configurar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) para proyectos, incluyendo puntos finales para crear, actualizar y eliminar proyectos.
   * Implementar comprobaciones de autorización para asegurar que solo los usuarios autorizados puedan acceder o modificar proyectos.
3. **Gestión de Tareas**:
   * Crear puntos finales para gestionar tareas dentro de proyectos, incluyendo operaciones como crear, actualizar y eliminar tareas.
   * Implementar lógica para asignar tareas a usuarios y gestionar plazos de entrega de tareas.
4. **Sincronización de Datos**:
   * Desarrollar mecanismos para la sincronización de datos entre el frontend y el backend para garantizar la consistencia en todos los clientes.
   * Implementar puntos finales RESTful o GraphQL para la recuperación y manipulación eficientes de datos.
5. **Colaboración en Tiempo Real**:
   * Integrar funcionalidad de WebSocket para habilitar características de colaboración en tiempo real como actualizaciones en vivo, notificaciones y mensajería.
6. **Integración de Base de Datos**:
   * Conectar a una base de datos PostgreSQL para almacenar información de usuarios, detalles de proyectos, datos de tareas y otra información relevante.
   * Utilizar esquemas y modelos de base de datos apropiados para representar la estructura de datos.

#### Tablas de la Base de Datos:

1. **Tabla de Usuarios**:
   * Campos: user\_id, username, email, password\_hash, role, avatar\_image.

CREATE TABLE IF NOT EXISTS users (

user\_id SERIAL PRIMARY KEY,

username VARCHAR(50) NOT NULL,

email VARCHAR(100) NOT NULL UNIQUE,

password\_hash VARCHAR(100) NOT NULL,

role VARCHAR(20) NOT NULL,

avatar\_image TEXT

);

avatar\_image TEXT -- Almacenar la imagen de avatar como una cadena Base64 ó Link a la ruta de la imagen.

En esta tabla:

* role\_id: Se utiliza el tipo SERIAL para crear un campo autoincremental que sirva como clave primaria de la tabla.
* role\_name: Es un campo de texto para almacenar el nombre del rol de usuario. Se establece como NOT NULL y UNIQUE para garantizar que cada nombre de rol sea único en la tabla.

1. **Tabla User Role:**
   * Campos:role\_id, role, role\_description.

CREATE TABLE IF NOT EXISTS user\_roles (

role\_id SERIAL PRIMARY KEY,

role\_name VARCHAR(50) NOT NULL UNIQUE

);

INSERT INTO user\_roles (role\_name, role\_description) VALUES

('Project Lead', 'Responsible for leading and managing projects.'),

('Project Manager', 'Responsible for overseeing project planning, execution, and delivery.'),

('Team Member', 'Member of the project team responsible for user interface design and implementation.'),

('Designer', 'Responsible for graphic design and user interface (UI) design.'),

('Main Developer', 'Lead developer responsible for overall software architecture and design.'),

('Developer', 'Member of the development team responsible for coding and implementation.'),

('Junior Developer', 'Entry-level developer learning and contributing to development tasks.'),

('Guest', 'Limited access user with read-only permissions.');

En esta tabla:

* role\_id: Se utiliza el tipo SERIAL para crear un campo autoincremental que sirva como clave primaria de la tabla.
* role\_name: Es un campo de texto para almacenar el nombre del rol de usuario. Se establece como NOT NULL y UNIQUE para garantizar que cada nombre de rol sea único en la tabla.
* role\_description: Es un campo de texto largo (TEXT) para almacenar una descripción opcional del rol de usuario.

1. **Tabla de Proyectos**:
   * Campos: project\_id, project\_name, description, start\_date, end\_date, project\_lead\_id.

CREATE TABLE IF NOT EXISTS projects (

project\_id SERIAL PRIMARY KEY,

project\_name VARCHAR(100) NOT NULL,

description TEXT,

start\_date DATE,

end\_date DATE,

project\_lead\_id INT

);

En esta tabla:

* project\_id: Se utiliza el tipo SERIAL para crear un campo autoincremental que sirva como clave primaria de la tabla.
* project\_name: Es un campo de texto para almacenar el nombre del proyecto. Se establece como NOT NULL para asegurarse de que siempre se proporcione un nombre de proyecto.
* description: Es un campo de texto largo (TEXT) para almacenar una descripción opcional del proyecto.
* start\_date y end\_date: Son campos de tipo DATE para almacenar la fecha de inicio y fin del proyecto, respectivamente.
* project\_lead\_id: Es un campo de tipo INT que servirá como clave externa (foreign key) que hace referencia al user\_id del usuario que lidera el proyecto.

1. **Tabla de Tareas**:
   * Campos: task\_id, project\_id, task\_name, description, status, priority, assigned\_user, deadline.

CREATE TABLE IF NOT EXISTS tasks (

task\_id SERIAL PRIMARY KEY,

project\_id INT,

task\_name VARCHAR(100) NOT NULL,

description TEXT,

status VARCHAR(20),

priority VARCHAR(20),

assigned\_user INT,

deadline DATE,

FOREIGN KEY (project\_id) REFERENCES projects(project\_id),

FOREIGN KEY (assigned\_user) REFERENCES users(user\_id)

);

En esta tabla:

* task\_id: Se utiliza el tipo SERIAL para crear un campo autoincremental que sirva como clave primaria de la tabla.
* project\_id: Es un campo que indica a qué proyecto pertenece la tarea. Se establecerá como una clave externa que hace referencia al project\_id en la tabla de proyectos.
* task\_name: Es un campo de texto para almacenar el nombre de la tarea. Se establece como NOT NULL para asegurarse de que siempre se proporcione un nombre de tarea.
* description: Es un campo de texto largo (TEXT) para almacenar una descripción opcional de la tarea.
* status: Es un campo de texto para indicar el estado actual de la tarea (por ejemplo, "pendiente", "en progreso", "completada", etc.).
* priority: Es un campo de texto para indicar la prioridad de la tarea (por ejemplo, "alta", "media", "baja", etc.).
* assigned\_user: Es un campo que indica a qué usuario se le ha asignado la tarea. Se establecerá como una clave externa que hace referencia al user\_id en la tabla de usuarios.
* deadline: Es un campo de tipo DATE para indicar la fecha límite de la tarea.

1. **Tabla de Miembros del Equipo**:
   * Campos: team\_member\_id, project\_id, user\_id, role\_id.

CREATE TABLE IF NOT EXISTS team\_members (

team\_member\_id SERIAL PRIMARY KEY,

project\_id INT,

user\_id INT,

role\_id INT,

FOREIGN KEY (project\_id) REFERENCES projects(project\_id),

FOREIGN KEY (user\_id) REFERENCES users(user\_id),

FOREIGN KEY (role\_id) REFERENCES user\_roles(role\_id)

);

En esta tabla:

* team\_member\_id: Se utiliza el tipo SERIAL para crear un campo autoincremental que sirva como clave primaria de la tabla.
* project\_id: Es un campo que indica a qué proyecto pertenece el miembro del equipo. Se establecerá como una clave externa que hace referencia al project\_id en la tabla de proyectos.
* user\_id: Es un campo que indica qué usuario es miembro del equipo. Se establecerá como una clave externa que hace referencia al user\_id en la tabla de usuarios.
* role\_id: Es un campo que indica el rol del usuario en el equipo. Se establecerá como una clave externa que hace referencia al role\_id en la tabla de roles de usuario (user\_roles).

1. **Tabla de Comentarios**:
   * Campos: comment\_id, project\_id, task\_id, user\_id, timestamp, comment\_text.

CREATE TABLE IF NOT EXISTS comments (

comment\_id SERIAL PRIMARY KEY,

project\_id INT,

task\_id INT,

user\_id INT,

timestamp TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

comment\_text TEXT,

FOREIGN KEY (project\_id) REFERENCES projects(project\_id),

FOREIGN KEY (task\_id) REFERENCES tasks(task\_id),

FOREIGN KEY (user\_id) REFERENCES users(user\_id)

);

En esta tabla:

* comment\_id: Se utiliza el tipo SERIAL para crear un campo autoincremental que sirva como clave primaria de la tabla.
* project\_id: Es un campo que indica a qué proyecto pertenece el comentario. Se establecerá como una clave externa que hace referencia al project\_id en la tabla de proyectos.
* task\_id: Es un campo que indica a qué tarea está asociado el comentario. Se establecerá como una clave externa que hace referencia al task\_id en la tabla de tareas.
* user\_id: Es un campo que indica qué usuario ha realizado el comentario. Se establecerá como una clave externa que hace referencia al user\_id en la tabla de usuarios.
* timestamp: Es un campo de tipo TIMESTAMP que registra la fecha y hora en que se realizó el comentario. Se establece con un valor predeterminado para que se actualice automáticamente al momento de inserción.
* comment\_text: Es un campo de texto para almacenar el contenido del comentario.

1. **Tabla de Sesiones**:
   * Campos: session\_id, user\_id, expiration\_timestamp.

CREATE TABLE IF NOT EXISTS sessions (

session\_id SERIAL PRIMARY KEY,

user\_id INT,

expiration\_timestamp TIMESTAMP,

FOREIGN KEY (user\_id) REFERENCES users(user\_id)

);

En esta tabla:

* session\_id: Se utiliza el tipo SERIAL para crear un campo autoincremental que sirva como clave primaria de la tabla.
* user\_id: Es un campo que indica a qué usuario está asociada la sesión. Se establecerá como una clave externa que hace referencia al user\_id en la tabla de usuarios.
* expiration\_timestamp: Es un campo de tipo TIMESTAMP que registra la fecha y hora en que expira la sesión del usuario.

**Ayudas de comandos:**

**Usando Docker:**

docker container run -d --name=pg -p 5432:5432 -e POSTGRES\_PASSWORD=Calibre92\* -e PGDATA=/pgdata -v /home/krl/docker\_file\_shared/pgdata:/pgdata postgres:latest