

DOCUMENTACION

Gestor de Biblioteca

Universidad Tecnológica Santa Catrina Ext. Montemorelos

Los Kemoñitos de la Verdad

Néstor Sebastián Ruiz Pérez

Edson Alexis Rodriguez Ibarra

Alan Gabriel Puente Gutiérrez

Contenido

Introducción.....	3
Configuracion del Editor de Codigo.....	4
Instalación de Visual Studio Code.....	4
Instalación de Extensiones Recomendadas.....	5
BACKEND	7
Instalación	10
Typescript.....	9
Dependencias en Git.....	10 y 11
Backend + Git.....	12
Archivo Readme.....	13
FrontEnd.....	13
Base de Datos	19
Postgre SQL.....	20
Docker	21
Instalación de Git...	22
Owasp Zap Instalacion.....	24
Postman Instalacion...	25
Gestion Reporte de Avance	26
Esquema de la Base de Datos	27
Diseño de Login...	28

Introducción

LitClub es una innovadora aplicación web destinada a facilitar la gestión de una biblioteca digital, permitiendo a los usuarios acceder, organizar y compartir libros de manera eficiente. En esta presentación, abordaremos dos aspectos clave de la aplicación: el diseño de la base de datos y el diseño de la aplicación en sí. El diseño de la base de datos ha sido concebido para garantizar un almacenamiento óptimo y una estructura que permita gestionar de manera flexible los libros, los usuarios y sus interacciones dentro de la plataforma. Por otro lado, el diseño de la aplicación se enfoca en ofrecer una interfaz intuitiva y atractiva, proporcionando una experiencia de usuario fluida para facilitar la navegación entre los distintos servicios de la biblioteca, desde la búsqueda de libros hasta la gestión de préstamos y recomendaciones personalizadas. Este enfoque integral busca ofrecer tanto un rendimiento robusto como una experiencia agradable para los usuarios jóvenes interesados en la lectura y el préstamo de libros.

Configuracion del Editor de Código

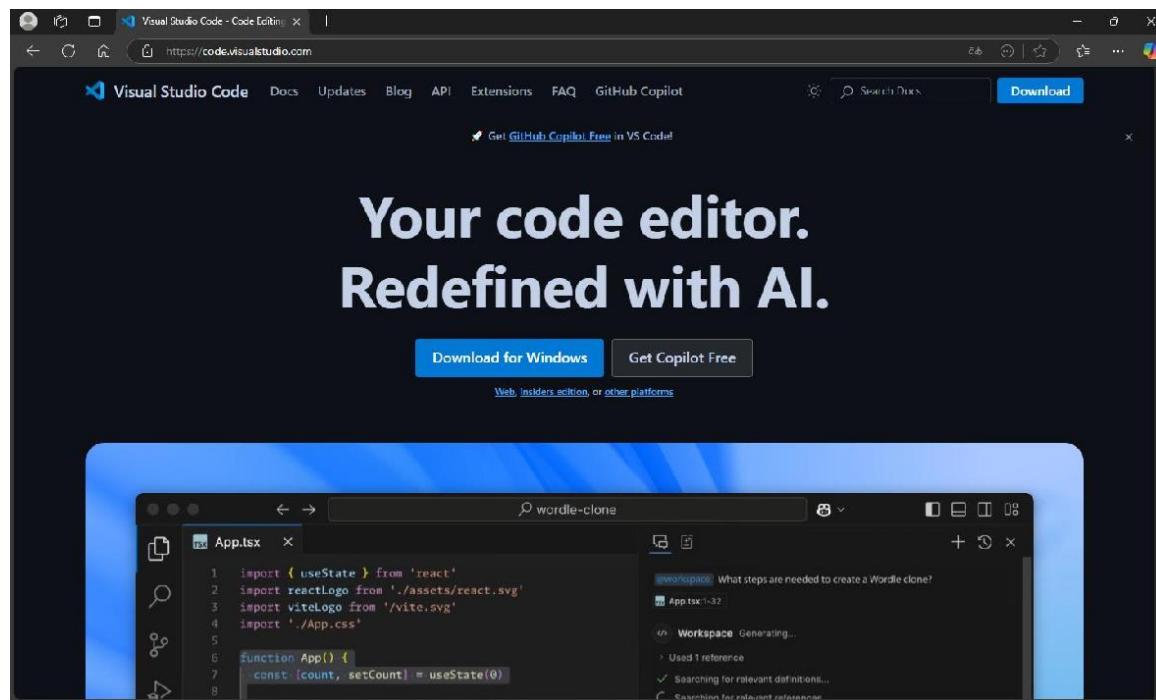
Visual Studio Code (VS Code) es un editor de código muy utilizado por programadores debido a su facilidad de uso y compatibilidad con muchas tecnologías.

Instalación de Visual Studio Code

Descarga el instalador desde la página oficial: <https://code.visualstudio.com/>

Si usas Windows, abre el archivo descargado y sigue las instrucciones del asistente de instalación. En Mac, abre el archivo .dmg y arrastra VS Code a la carpeta de Aplicaciones.

Para Ubuntu, puedes instalarlo ejecutando el siguiente comando en la terminal.



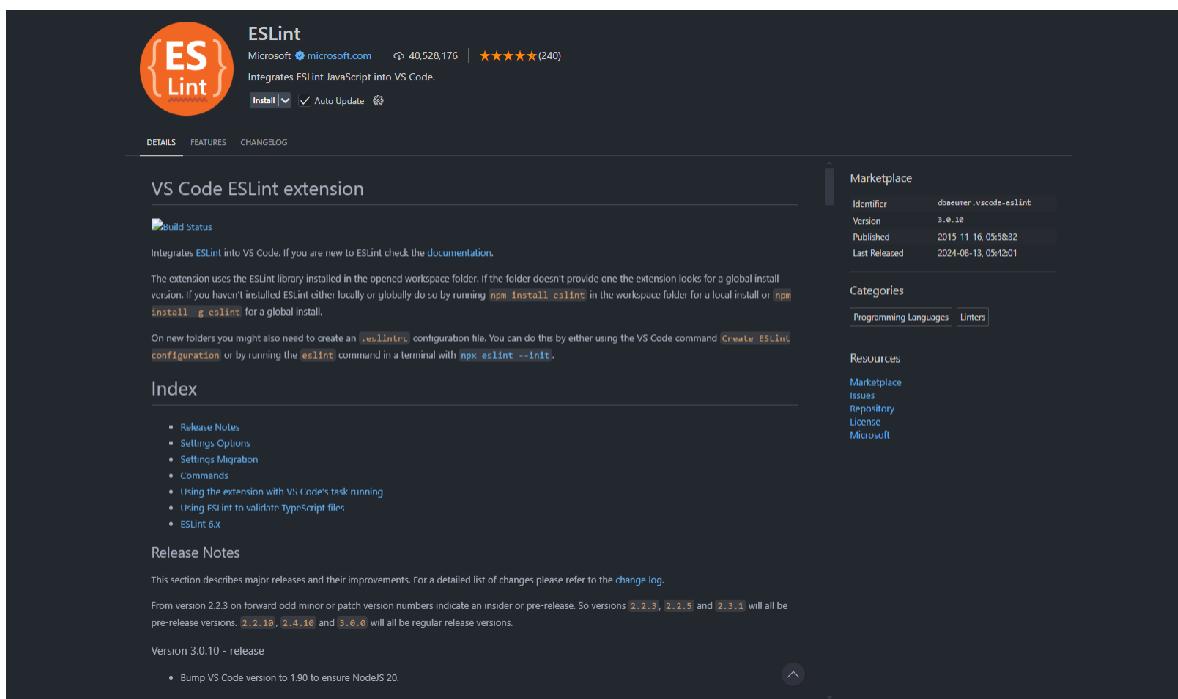
Instalación de Extensiones Recomendadas.

Para instalar estas extensiones:

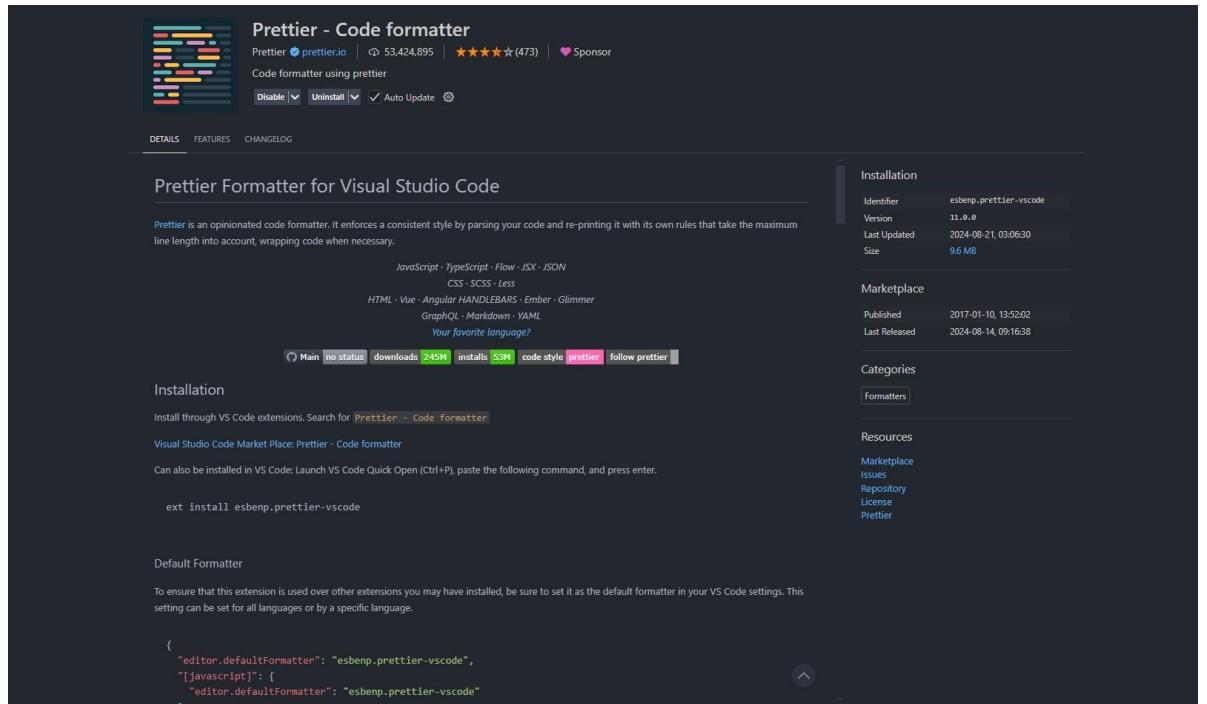
- I. Abre VS Code.
- II. Ve a la pestaña de Extensiones (ícono de cuadrados en la barra lateral izquierda).
- III. Busca cada extensión por su nombre e instálala.

Las extensiones son complementos que mejoran las funciones del editor:

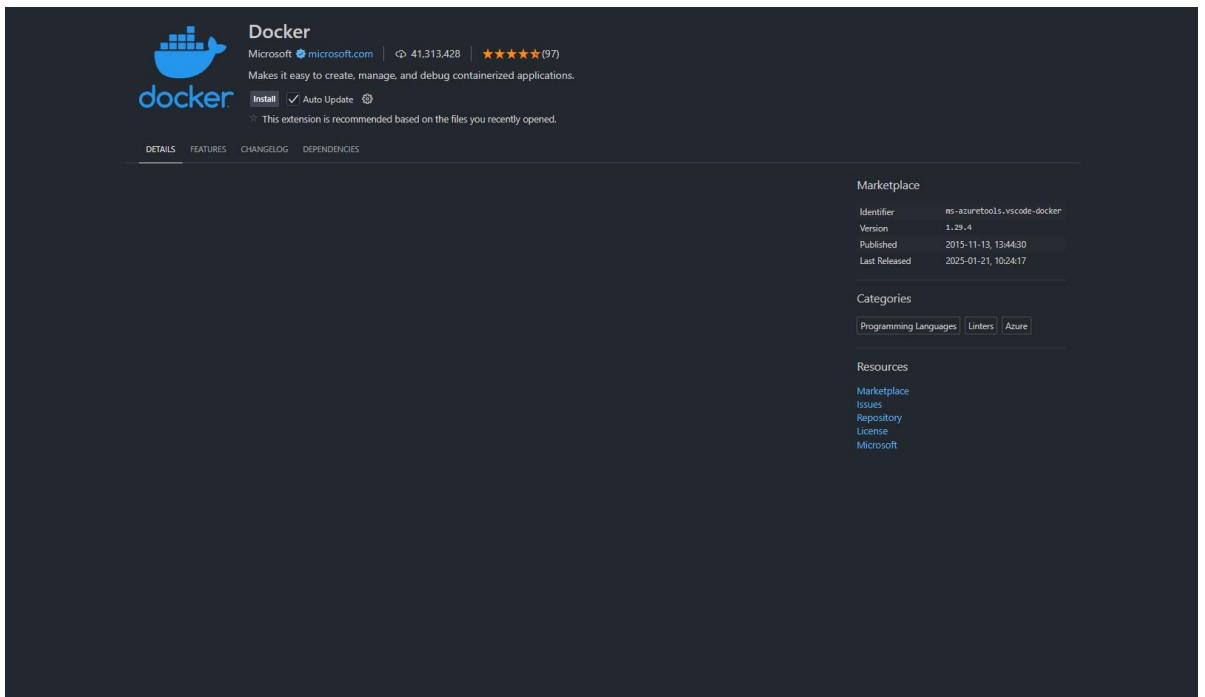
- *ESLint*: Ayuda a detectar errores en el código JavaScript



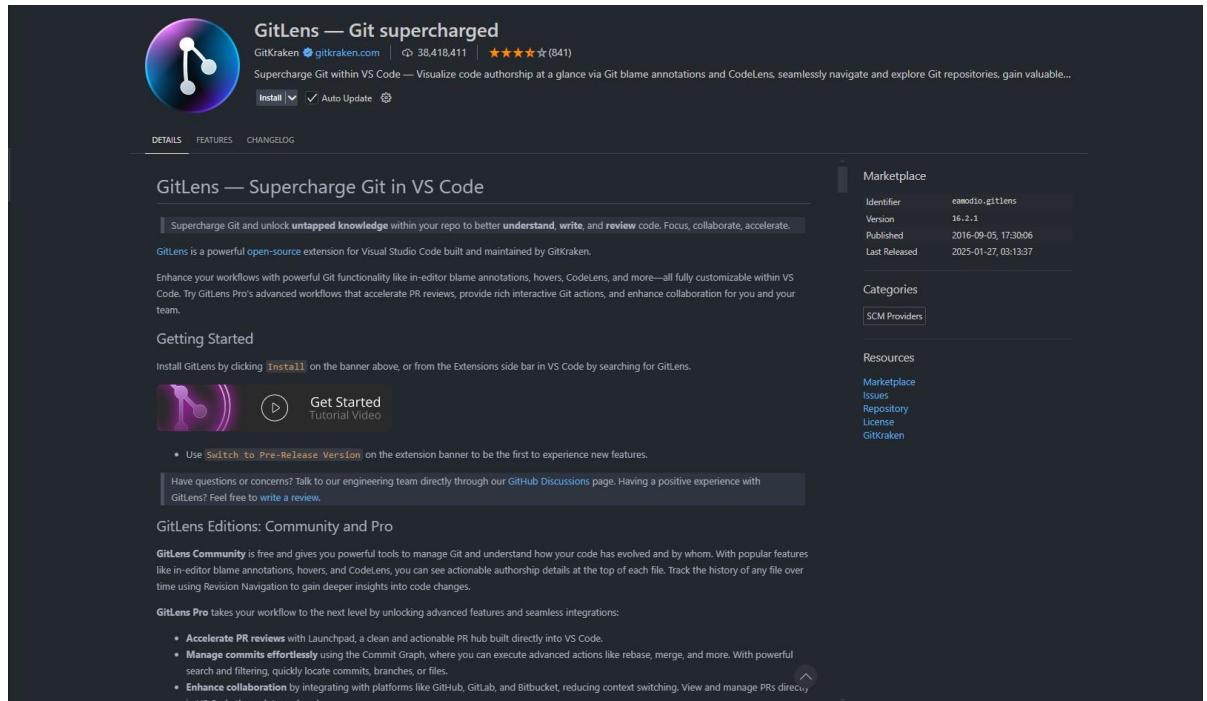
- *Prettier*: Formatea el código automáticamente para mejorar su legibilidad.



- Docker: Facilita el trabajo con contenedores Docker dentro de VS Code.



- GitLens: Proporciona herramientas avanzadas para trabajar con Git y ver el historial de cambios en el código.



BACKEND

El backend de LitClub está desarrollado utilizando tecnologías modernas y robustas para garantizar un rendimiento eficiente y una escalabilidad óptima. Node.js proporciona una plataforma rápida y flexible para manejar las operaciones del servidor, permitiendo que la aplicación responda de manera ágil a las solicitudes de los usuarios. Utilizamos TypeScript para aprovechar la tipificación estática y mejorar la mantenibilidad y la seguridad del código, reduciendo errores durante el desarrollo y mejorando la experiencia general del equipo de trabajo.

La base de datos está gestionada con PostgreSQL, un sistema de gestión de bases de datos relacional potente y confiable, que nos permite almacenar y consultar grandes volúmenes de datos de manera eficiente. La combinación de estas tecnologías asegura que LitClub no solo sea una

aplicación rápida y segura, sino que también sea fácil de mantener y expandir a medida que crecen las necesidades de los usuarios.

Instalación de Node.js y NPM

Node.js es un entorno de ejecución para JavaScript que permite ejecutar código fuera del navegador.

Instalación en Windows y Mac

- Ve al sitio oficial de Node.js: <https://nodejs.org/>
- Descarga la versión 18.x LTS (Long Term Support) porque es la más estable.
- Abre el archivo descargado (.msi en Windows, .pkg en Mac) y sigue las instrucciones del instalador.
- Una vez instalado, abre la terminal (Mac/Linux) o el Símbolo del sistema (Windows) y ejecuta:

```
node -v  
npm -v
```

Esto mostrará los números de versión instalados.

Instalación de Express.js

Express.js es un marco de trabajo para Node.js que facilita la creación de servidores y APIs. Instalación

Crea una carpeta para tu proyecto y entra en ella

The screenshot shows the Express.js homepage. At the top, there's a navigation bar with links for Home, Getting started, Guide, API reference, Advanced topics, Resources, Support, and Blog. Below the navigation is a search bar. The main content area features the Express logo and the tagline "Fast, unopinionated, minimalist web framework for Node.js". A code snippet is shown in a box:

```
const express = require('express')
const app = express()
const port = 3000

app.get('/', (req, res) => {
  res.send('Hello world!')
})

app.listen(port, () => {
  console.log(`Example app listening on port ${port}`)
})
```

Below the code, there's a terminal-like box containing the command `$ npm install express --save`. A note at the bottom says "Express 5.0 documentation is now available. The [API documentation](#) is a work in progress. For information on what's in the release, see the Express [release history](#)".

Below the main content, there are four sections: Web Applications, APIs, Performance, and Middleware, each with a brief description.

Instalación de TypeScript

TypeScript es un lenguaje basado en JavaScript que agrega tipos estáticos, lo que ayuda a reducir errores en el código.

The screenshot shows the TypeScript homepage. The header includes a search bar and links for Download, Docs, Handbook, Community, Playground, and Tools. The main content area features the tagline "TypeScript is JavaScript with syntax for types." A "Try TypeScript Now" button is visible. On the right, there's a code editor window showing a TypeScript file with an error message:

```
const user = {
  firstName: "Angela",
  lastName: "Mertz",
  role: "Professor"
}

console.log(user.name)

Property 'name' does not exist on type '{ firstName: string; lastName: string; role: string; }'.
```

Below the code editor, a message says "TypeScript 5.2 is now available". The footer contains sections for "What is TypeScript?", "JavaScript and More", "A Result You Can Trust", and "Safety at Scale".

Requisitos

Antes de comenzar, asegúrate de tener lo siguiente:

- [Node.js](<https://nodejs.org/>) instalado.
- [PostgreSQL](<https://www.postgresql.org/>) instalado y configurado.
- [Docker](<https://www.docker.com/products/docker-desktop>) instalado y en ejecución.

Instalación

1. Clona este repositorio en tu máquina local:

```
git init  
git clone <URL_DE_TU_REPO>
```

2. Navega a la carpeta del proyecto:

```
cd App LITCLUB
```

Ejecuta el siguiente comando para inicializar el proyecto y crear el archivo package.json:

```
npm init -y
```

3. Instala las dependencias necesarias para el proyecto:

```
npm install express pg bcryptjs jsonwebtoken dotenv
```

Instala las dependencias de desarrollo necesarias para TypeScript:

```
npm install --save-dev typescript @types/express  
@types/node
```

```
@types/bcryptjs @types/jsonwebtoken
```

Inicializa la configuración de TypeScript con:

```
npx tsc --init
```

Asegúrate de tener el archivo .env con las siguientes configuraciones de tu base de datos y aplicación:

```
DB_USER=postgres      # Usuario de la base de  
datos DB_PASSWORD=123456          #  
Contraseña del usuario de la base de datos  
DB_HOST=localhost      # Dirección del servidor  
PostgreSQL DB_PORT=5432 # Puerto de  
PostgreSQL  
DB_NAME=LITCLUB      #  
Nombre de la base de datos  
PORT=3000          # Puerto de  
tu aplicación Express  
JWT_SECRET=your_jwt_secret # Secreto para JWT (cámbialo por algo  
más seguro)
```

- Bibliotecas adicionales
- Instalar las siguientes bibliotecas:
 - JWT: npm install [jsonwebtoken@9.0.0](#).
 - dotenv: npm install [dotenv@16.1.4](#).
 - bcryptjs: npm install [bcryptjs@2.4.3](#).
 - pg: npm install [pg@8.11.0](#).
- Configura cada una según la documentación oficial de sus respectivos paquetes.

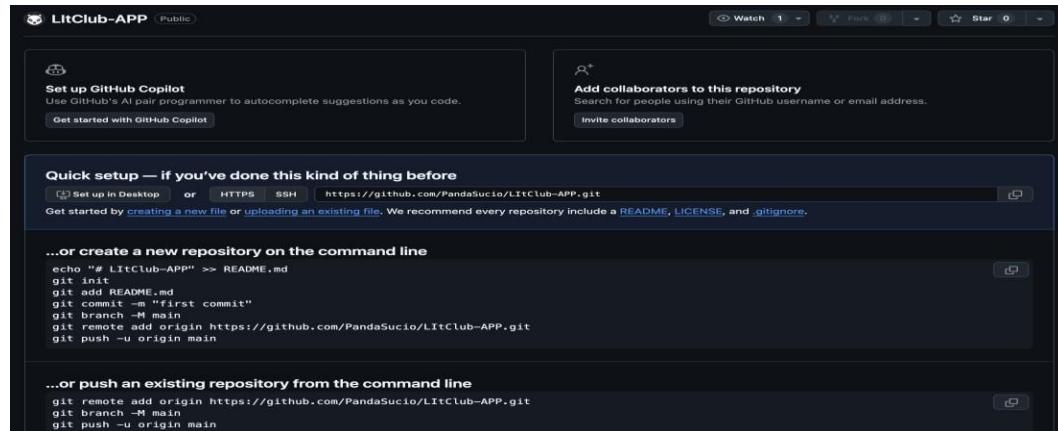
Backend + Git

Para la gestión de la base de datos, LitClub utiliza PostgreSQL, un sistema de gestión de bases de datos relacional robusto y confiable, ideal para almacenar y consultar los datos de los usuarios, libros y transacciones de manera eficiente. La estructura de la base de datos ha sido diseñada para ofrecer escalabilidad y facilitar la gestión de grandes volúmenes de información, mientras mantenemos un alto nivel de rendimiento.

El entorno de desarrollo y producción de la base de datos, así como de toda la aplicación, está configurado con Docker y Docker Compose. Docker nos permite contenerizar las aplicaciones y sus dependencias, asegurando que el entorno sea consistente en todas las etapas del ciclo de vida del proyecto. Docker Compose facilita la orquestación de múltiples contenedores, como el de la base de datos PostgreSQL y el backend, permitiendo un despliegue rápido y eficiente tanto en desarrollo como en producción.

En cuanto al control de versiones y colaboración en el código, utilizamos Git para gestionar el historial de cambios y facilitar la colaboración entre los miembros del equipo. Todo el código fuente se almacena en GitHub, una plataforma de control de versiones en la nube que ofrece herramientas para la colaboración, revisión de código y despliegue continuo, asegurando

que el desarrollo de LitClub sea transparente, eficiente y organizado.



Archivo README:

App LitClub

Este es el backend para la aplicación llamada LitClub, diseñada para gestionar usuarios, autentificación y roles utilizando Express y PostgreSQL.

Requisitos

Antes de comenzar, asegúrate de tener lo siguiente:

- [Node.js](<https://nodejs.org/>) instalado.
- [PostgreSQL](<https://www.postgresql.org/>) instalado y configurado.
- [Docker](<https://www.docker.com/products/docker-desktop>) instalado y en ejecución.

Instalación

1. Clona este repositorio en tu máquina local:

```
```bash
git init
git clone <URL_DE_TU_REPOITORIO>
````
```

2. Navega a la carpeta del proyecto:

```
```bash
cd App LITCLUB
````
```

3. Ejecuta el siguiente comando para inicializar el proyecto y crear el archivo `package.json`:

```
```bash
npm init -y
````
```

4. Instala las dependencias necesarias para el proyecto:

```
```bash
npm install express pg bcryptjs jsonwebtoken dotenv
````
```

5. Instala las dependencias de desarrollo necesarias para TypeScript:

```
```bash
npm install --save-dev typescript @types/express @types/node @types/bcryptjs @types/jsonwebtoken
````
```

6. Inicializa la configuración de TypeScript con:

```
```bash
npx tsc --init
````
```

7. Asegúrate de tener el archivo `.env` con las siguientes configuraciones de tu base de datos y aplicación:

```
```env
DB_USER=postgres # Usuario de la base de datos
DB_PASSWORD=123456 # Contraseña del usuario de la base de datos
DB_HOST=localhost # Dirección del servidor PostgreSQL
DB_PORT=5432 # Puerto de PostgreSQL
DB_NAME=LITCLUB # Nombre de la base de datos
PORT=3000 # Puerto de tu aplicación Express
JWT_SECRET=your_jwt_secret # Secreto para JWT (cámbialo por algo más seguro)
````
```

Bakend

2. Variables de Entorno Utilizadas

| Variable | Descripción |
|-----------------------------------|--|
| PORT=5001 | Puerto en el que se ejecuta el backend. |
| DB_USER=a
lexisrodrigu
ez | Usuario de la base de datos PostgreSQL. |
| DB_HOST=l
ocalhost | Host de la base de datos (en este caso, local). |
| DB_NAME=
LitClub-1.0 | Nombre de la base de datos usada por la aplicación. |
| DB_PASSW
ORD=Edson
ibarra12 | Contraseña del usuario de la base de datos. (Debe mantenerse segura y fuera del código público) . |
| DB_PORT=5 | Puerto en el que PostgreSQL está escuchando conexiones. |

432

| | |
|-----------------------|--|
| JWT_SECRET
T=admin | Clave secreta utilizada para la generación de tokens JWT en el sistema de autenticación. (Debe ser una clave segura y robusta para evitar ataques). |
|-----------------------|--|

Este documento explica cómo configurar y manejar una conexión a una base de datos **PostgreSQL** en un proyecto de **Node.js** utilizando la librería pg y el uso de variables de entorno con dotenv.

2. Instalación de Dependencias

Antes de usar esta configuración, asegúrate de tener instaladas las dependencias necesarias:

```
bash
CopiarEditar
npm install pg dotenv
```

- pg: Cliente para interactuar con bases de datos PostgreSQL.
- dotenv: Permite cargar variables de entorno desde un archivo .env.

3. Código de Conexión a PostgreSQL

A continuación, se muestra la configuración para establecer una conexión segura a PostgreSQL:

TS database.ts ×

```
LitClub-1.0 > backend > src > config > TS database.ts > [o] pool

1 import { Pool } from "pg";
2 import dotenv from "dotenv";
3
4 dotenv.config();
5
6 const pool = new Pool({
7   user: process.env.DB_USER,
8   host: process.env.DB_HOST,
9   database: process.env.DB_NAME,
10  password: process.env.DB_PASSWORD,
11  port: Number(process.env.DB_PORT),
12 });
13
14 export default pool;
15
```

Este código maneja la autenticación de usuarios en una API utilizando Node.js con Express, PostgreSQL, bcryptjs y jsonwebtoken (JWT).

La función register permite registrar un usuario en la base de datos. Primero, toma los datos de la solicitud HTTP (nombre, email y contraseña), luego cifra la contraseña con bcrypt y almacena el usuario en la base de datos con un rol predeterminado de "usuario". Finalmente, responde con un mensaje de éxito y los datos del usuario registrado.

La función login permite a los usuarios iniciar sesión. Primero, busca en la base de datos un usuario con el correo electrónico proporcionado. Si no se encuentra, devuelve un error. Luego, compara la contraseña ingresada con la almacenada en la base de datos utilizando bcrypt. Si coincide, genera un token JWT con la información del usuario (id y rol), estableciendo un tiempo de expiración de una hora. Finalmente, devuelve el token y los datos del usuario.

Ambas funciones incluyen manejo de errores para garantizar estabilidad y seguridad en el sistema. Además, se recomienda almacenar las credenciales y claves sensibles en variables de entorno utilizando dotenv.

```
ts authController.ts ×
LitClub-1.0 > backend > src > controllers > ts authController.ts > ...
1 import { Request, Response } from "express";
2 import bcrypt from "bcryptjs";
3 import jwt from "jsonwebtoken";
4 import pool from "../config/database";
5 import { User } from "../models/userModel";
6
7 export const register = async (req: Request, res: Response) => {
8   try {
9     const { nombre, email, contraseña } = req.body;
10    const hashedPassword = await bcrypt.hash(contraseña, 10);
11
12    const result = await pool.query(
13      "INSERT INTO usuarios (nombre, email, contraseña, rol, fecha_ingreso) VALUES ($1, $2, $3, $4, NOW()) R
14      [nombre, email, hashedPassword, "usuario"]
15    );
16
17    res.status(201).json({ message: "Usuario registrado", user: result.rows[0] });
18    console.log(hashedPassword);
19  } catch (error) {
20    console.error("Error en el registro:", error);
21    res.status(500).json({ error: "Error al registrar usuario" });
22  }
23};
24
25 export const login = async (req: Request, res: Response): Promise<void> => {
26   try {
27     const { email, contraseña } = req.body;
28     const result = await pool.query("SELECT * FROM usuarios WHERE email = $1", [email]);
29
30     if (result.rows.length === 0) {
31       res.status(400).json({ error: "Usuario no encontrado" });
32       return;
33     }
34
35     const user: User = result.rows[0];
36     const isMatch = await bcrypt.compare(contraseña, user.contraseña);
37
38     if (!isMatch) {
39       res.status(400).json({ error: "Contraseña incorrecta" });
40       return;
41     }
42
43     const token = jwt.sign({ id: user.id, rol: user.rol }, process.env.JWT_SECRET as string, { expiresIn: "1h" });
44
45     res.json({ message: "Inicio de sesión exitoso", token, user });
46   } catch (error) {
47     console.error("Error en login:", error);
48     res.status(500).json({ error: "Error en el inicio de sesión" });
49   }
50 }
51
```

Este código implementa un conjunto de funciones para gestionar libros en una base de datos PostgreSQL dentro de una API desarrollada con Node.js y Express.

La función getAllBooks recupera todos los libros almacenados en la base de datos, devolviendo una lista con información relevante como el nombre, descripción, autor, número de páginas, imagen y stock disponible.

La función addBook permite agregar un nuevo libro. Recibe datos como nombre, frase, descripción, autor, páginas, URL de imagen y stock, los inserta en la base de datos y devuelve el libro recién agregado.

La función updateBook actualiza los datos de un libro existente en la base de datos según el ID proporcionado. Modifica los valores según la información enviada en la

solicitud y devuelve el libro actualizado.

La función `deleteBook` elimina un libro de la base de datos según el ID proporcionado. Si la operación es exitosa, responde con un mensaje de confirmación.

Cada función maneja posibles errores para asegurar el correcto funcionamiento de la API, devolviendo un código de error 500 en caso de fallos en la consulta a la base de datos.

```
TS bookController.ts × | LitClub-1.0 > backend > src > controllers > TS bookController.ts > ⚡ deleteBook
1 import { Request, Response } from "express";
2 import pool from "../config/database";
3 // Obtener todos los libros
4 export const getAllBooks = async (req: Request, res: Response) => {
5   try {
6     const result = await pool.query("SELECT id, nombre, frase, descripcion, autor, paginas, imagen_url, stock FROM libros");
7     res.json(result.rows);
8   } catch (error) {
9     console.error("Error al obtener libros:", error);
10    res.status(500).json({ error: "Error al obtener la lista de libros" });
11  }
12};
13 // Agregar un nuevo libro
14 export const addBook = async (req: Request, res: Response) => {
15  try {
16    const { nombre, frase, descripcion, autor, paginas, imagen_url, stock } = req.body;
17
18    const result = await pool.query(
19      "INSERT INTO libros (nombre, frase, descripcion, autor, paginas, imagen_url, stock) VALUES ($1, $2, $3, $4, $5, $6, $7) RETURNING [nombre, frase, descripcion, autor, paginas, imagen_url, stock]"
20    );
21
22    res.status(201).json(result.rows[0]);
23  } catch (error) {
24    console.error("Error al agregar libro:", error);
25    res.status(500).json({ error: "Error al agregar libro" });
26  }
27};
28
29
30 export const updateBook = async (req: Request, res: Response) => {
31  try {
32    const { id } = req.params;
33    const { nombre, frase, descripcion, autor, paginas, imagen_url, stock } = req.body;
34
35    const result = await pool.query(
36      "UPDATE libros SET nombre = $1, frase = $2, descripcion = $3, autor = $4, paginas = $5, imagen_url = $6, stock = $7 WHERE id = $8"
37      [nombre, frase, descripcion, autor, paginas, imagen_url, stock, id]
38    );
39
40    res.json(result.rows[0]);
41  } catch (error) {
42    console.error("Error al actualizar libro:", error);
43    res.status(500).json({ error: "Error al actualizar libro" });
44  }
45};
46
47 // Eliminar un libro
48 export const deleteBook = async (req: Request, res: Response) => {
49  try {
50    const { id } = req.params;
51    await pool.query("DELETE FROM libros WHERE id = $1", [id]);
52    res.json({ message: "Libro eliminado correctamente" });
53  } catch (error) {
54    console.error("Error al eliminar libro:", error);
55    res.status(500).json({ error: "Error al eliminar libro" });
56  }
57};
58
```

Este código maneja la gestión de transacciones de libros en un sistema de compra y renta, asegurando control de stock y fechas de vencimiento.

La función `createTransaction` permite a los usuarios comprar o rentar un libro. Antes de procesar la transacción, verifica que el libro exista y tenga stock disponible. Si es una compra o renta, reduce el stock en la base de datos. En el caso de una renta, calcula la fecha de vencimiento y almacena la transacción en la base de datos.

La función `getUserBooks` recupera los libros comprados o rentados por un usuario. Busca en la base de datos todas las transacciones activas asociadas al usuario y devuelve detalles como el nombre del libro, autor, imagen, tipo de transacción y fecha de vencimiento.

La función `removeExpiredRentals` actualiza el estado de las rentas vencidas, marcándolas como no disponibles cuando la fecha de vencimiento ha pasado. Esto ayuda a mantener un control sobre los libros rentados.

La función `returnBook` permite a los usuarios devolver un libro rentado. Verifica que la transacción esté activa y, si es una renta, incrementa nuevamente el stock del libro en la base de datos. Luego, marca la transacción como finalizada.

Este sistema garantiza un control eficiente sobre la disponibilidad de libros, evitando que los usuarios renten libros sin stock y asegurando que las rentas vencidas sean deshabilitadas automáticamente.

```

ts bookController.ts ts transactionController.ts ×
LitClub-1.0 > backend > src > controllers > ts transactionController.ts > [o] returnBook
1 import { Request, Response } from "express";
2 import pool from "../config/database";
3 import { UserBook } from "../types/UserBook";
4
5 // ✅ Comprar o rentar un libro
6 > export const createTransaction = async (req: Request, res: Response) => {
55   };
56
57
58
59 // ✅ Obtener libros comprados o rentados por un usuario
60 > export const getUserBooks = async (req: Request, res: Response) => {
83   };
84
85 // ✅ Eliminar libros rentados cuando la fecha de vencimiento haya pasado
86 > export const removeExpiredRentals = async () => {
95   };
96
97 // ✅ Función para devolver un libro rentado
98 export const returnBook = async (req: Request, res: Response) => {
99   try {
100     const { transaction_id } = req.params;
101
102     if (!transaction_id) {
103       res.status(400).json({ error: "El transaction_id es requerido" });
104       return;
105     }
106
107     // Obtener los datos de la transacción
108     const transaction = await pool.query(
109       "SELECT libro_id, tipo FROM transacciones WHERE id = $1 AND disponible = TRUE",
110       [transaction_id]
111     );
112
113     if (transaction.rows.length === 0) {
114       res.status(400).json({ error: "No se encontró la renta activa." });
115       return;
116     }
117
118     const { libro_id, tipo } = transaction.rows[0];
119
120     // Marcar la transacción como finalizada
121     await pool.query("UPDATE transacciones SET disponible = FALSE WHERE id = $1", [transaction_id]);
122
123     // Si es una renta, devolver el libro al stock
124     if (tipo === "renta") {
125       await pool.query("UPDATE libros SET stock = stock + 1 WHERE id = $1", [libro_id]);
126     }
127
128     res.json({ message: "Libro devuelto con éxito" });
129   } catch (error) {
130     console.error("Error al devolver libro:", error);
131     res.status(500).json({ error: "Error al devolver el libro" });
132   }
133 }
134
135
136 // ✅ Verifica que 'returnBook' está en la lista de exportaciones
137

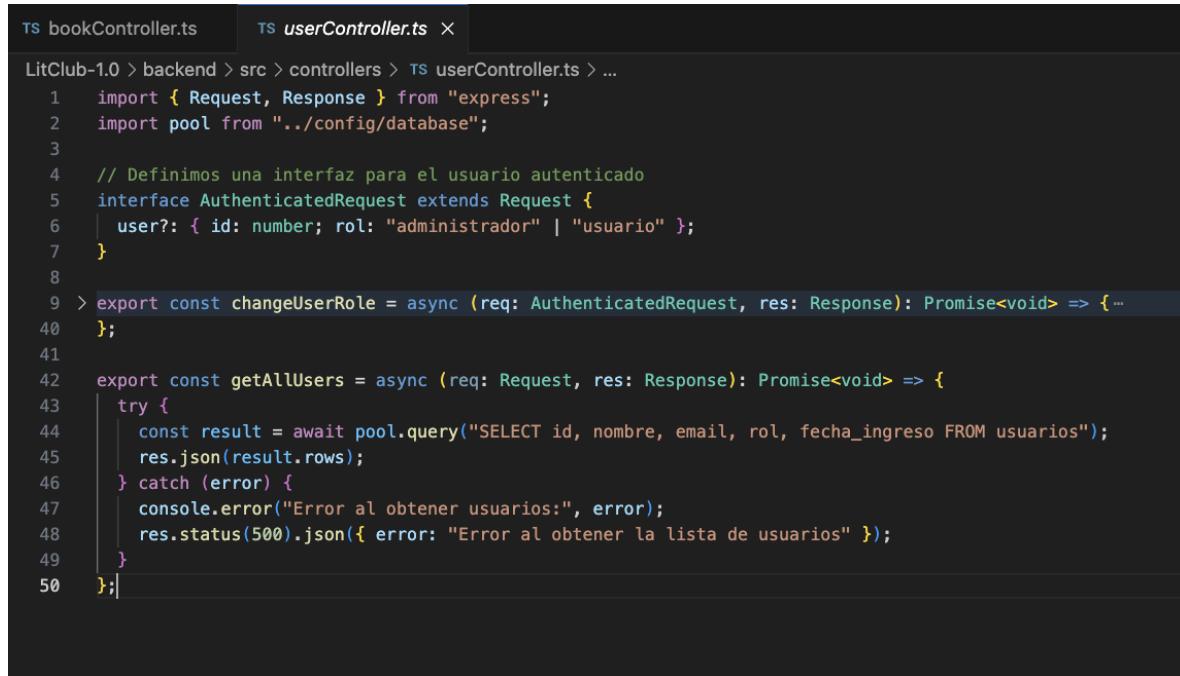
```

Este código maneja la gestión de usuarios en un sistema con roles diferenciados de **administrador** y **usuario**.

La función changeUserRole permite cambiar el rol de un usuario en la base de datos. Primero, verifica que el nuevo rol sea válido. Luego, comprueba que el usuario autenticado tenga permisos de **administrador** para realizar la modificación. Si el usuario a modificar no existe, devuelve un error. Finalmente, actualiza el rol del usuario y envía una respuesta confirmando el cambio.

La función getAllUsers obtiene la lista de todos los usuarios registrados en el sistema, incluyendo información como **ID**, **nombre**, **email**, **rol** y **fecha de ingreso**. Si ocurre un error en la consulta, devuelve un mensaje indicando la falla.

Este sistema garantiza que solo los administradores puedan modificar los roles de los usuarios y proporciona un mecanismo para visualizar los usuarios registrados.



The screenshot shows a code editor with two tabs: 'bookController.ts' and 'userController.ts'. The 'userController.ts' tab is active, displaying the following TypeScript code:

```
1 import { Request, Response } from "express";
2 import pool from "../config/database";
3
4 // Definimos una interfaz para el usuario autenticado
5 interface AuthenticatedRequest extends Request {
6   user?: { id: number; rol: "administrador" | "usuario" };
7 }
8
9 > export const changeUserRole = async (req: AuthenticatedRequest, res: Response): Promise<void> => {
10   ...
11
12   export const getAllUsers = async (req: Request, res: Response): Promise<void> => {
13     try {
14       const result = await pool.query("SELECT id, nombre, email, rol, fecha_ingreso FROM usuarios");
15       res.json(result.rows);
16     } catch (error) {
17       console.error("Error al obtener usuarios:", error);
18       res.status(500).json({ error: "Error al obtener la lista de usuarios" });
19     }
20   };
21 }
```

Este código implementa la autenticación y autorización en una API mediante **JSON Web Tokens (JWT)**.

La función `verifyToken` verifica si un usuario está autenticado. Primero, extrae el token de la cabecera de autorización. Si no hay token, devuelve un error de acceso denegado. Luego, intenta decodificar el token usando la clave secreta almacenada en las variables de entorno. Si el token es válido, extrae la información del usuario (ID y rol) y la almacena en la solicitud para su uso en rutas protegidas. Si el token es inválido, devuelve un error.

La función `isAdmin` se usa para restringir el acceso a rutas exclusivas de administradores. Verifica si el usuario autenticado tiene el rol de **administrador**. Si no es así, devuelve un error de acceso denegado. Si el usuario tiene el rol adecuado, permite continuar con la ejecución de la solicitud.

Este sistema garantiza que solo los usuarios autenticados puedan acceder a ciertas funcionalidades y que las acciones administrativas solo sean realizadas por usuarios con permisos adecuados.

```

TS authMiddleware.ts ×

LitClub-1.0 > backend > src > middlewares > ts authMiddleware.ts > ...
1 import { Request, Response, NextFunction } from "express";
2 import jwt from "jsonwebtoken";
3
4 interface DecodedToken {
5   id: number;
6   rol: "administrador" | "usuario";
7 }
8
9 export const verifyToken = (req: Request, res: Response, next: NextFunction): void => {
10   const token = req.headers.authorization?.split(" ")[1];
11
12   if (!token) {
13     res.status(401).json({ error: "Acceso denegado. Token no proporcionado" });
14     return;
15   }
16
17   try {
18     const decoded = jwt.verify(token, process.env.JWT_SECRET as string) as DecodedToken;
19     (req as any).user = decoded;
20     next();
21   } catch (error) {
22     res.status(401).json({ error: "Token inválido" });
23     return;
24   }
25 };
26
27 export const isAdmin = (req: Request, res: Response, next: NextFunction): void => {
28   if (!(req as any).user || (req as any).user.rol !== "administrador") {
29     res.status(403).json({ error: "Acceso denegado. Se requiere rol de administrador" });
30     return;
31   }
32   next();
33 };
34

```

Esta interfaz User define la estructura de los datos de un usuario en el sistema.

Cada usuario tiene un **ID único**, un **nombre** y un **correo electrónico**. Además, posee un **rol**, que puede ser "administrador" o "usuario", determinando sus permisos dentro de la aplicación. También incluye la **fecha de ingreso**, que registra el momento en que el usuario fue creado en la base de datos.

Por seguridad, la **contraseña** está almacenada, pero debe manejarse de forma cifrada antes de guardarla en la base de datos. Esta interfaz es utilizada para tipar objetos de usuario en el código, asegurando que siempre contengan los datos necesarios con el formato correcto.

```
TS userModel.ts ×  
LitClub-1.0 > backend > src > models > TS userModel.ts > ...  
1  export interface User {  
2    id: number;  
3    nombre: string;  
4    email: string;  
5    rol: "administrador" | "usuario";  
6    fecha_ingreso: Date;  
7    contraseña: string;  
8  }  
9
```

Este código define las rutas para la autenticación de usuarios en la aplicación mediante **Express**.

Se crea un **router** que gestiona las solicitudes relacionadas con el registro e inicio de sesión de usuarios.

- La ruta POST /register permite a los nuevos usuarios registrarse en el sistema. Llama a la función register, que maneja la validación de datos, el cifrado de la contraseña y el almacenamiento en la base de datos.
- La ruta POST /login permite a los usuarios autenticarse. Llama a la función login, que verifica las credenciales ingresadas y, si son correctas, genera un **token JWT** para la sesión del usuario.

Este módulo facilita la organización de las rutas de autenticación, permitiendo que sean utilizadas en el servidor principal de la aplicación.

```
TS authRoutes.ts ×  
LitClub-1.0 > backend > src > routes > TS authRoutes.ts > ...  
1  import express from "express";  
2  import { register, login } from "../controllers/authController";  
3  
4  const router = express.Router();  
5  
6  router.post("/register", register);  
7  router.post("/login", login);  
8  
9  export default router;  
10 |
```

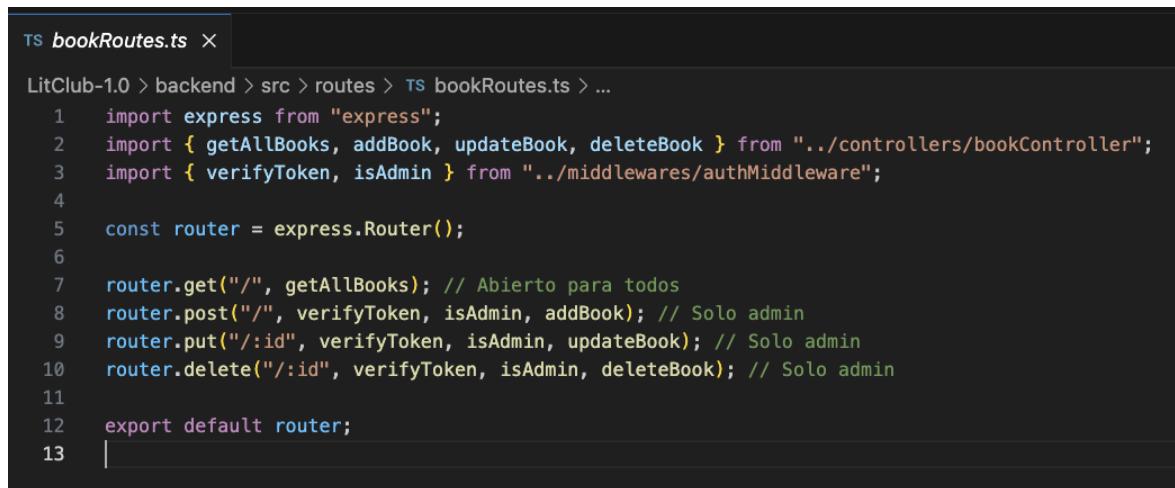
Este código define las rutas para la gestión de libros en la aplicación utilizando **Express**, asegurando control de acceso mediante autenticación y autorización.

- **GET /** → Obtiene la lista de todos los libros. Esta ruta está disponible para

cualquier usuario, sin necesidad de autenticación.

- **POST /** → Permite agregar un nuevo libro. Está protegida con los **middlewares** `verifyToken` y `isAdmin`, lo que significa que solo los **administradores** autenticados pueden realizar esta acción.
- **PUT /:id** → Permite actualizar un libro existente, identificándolo por su ID. Al igual que la anterior, solo los **administradores** autenticados pueden acceder a esta función.
- **DELETE /:id** → Permite eliminar un libro mediante su ID. También requiere autenticación y permisos de **administrador**.

El uso de los **middlewares** `verifyToken` y `isAdmin` garantiza que solo los usuarios autorizados puedan realizar cambios en la colección de libros, protegiendo la integridad del sistema.



The screenshot shows a code editor window with the file `bookRoutes.ts` open. The code defines a router for managing books, using the `express` module. It imports necessary functions from `bookController` and `authMiddleware`. The router handles four methods: `get()` for all books, `post()` for adding books (requiring admin), `put()` for updating books (requiring admin), and `delete()` for deleting books (requiring admin). The code is numbered from 1 to 13.

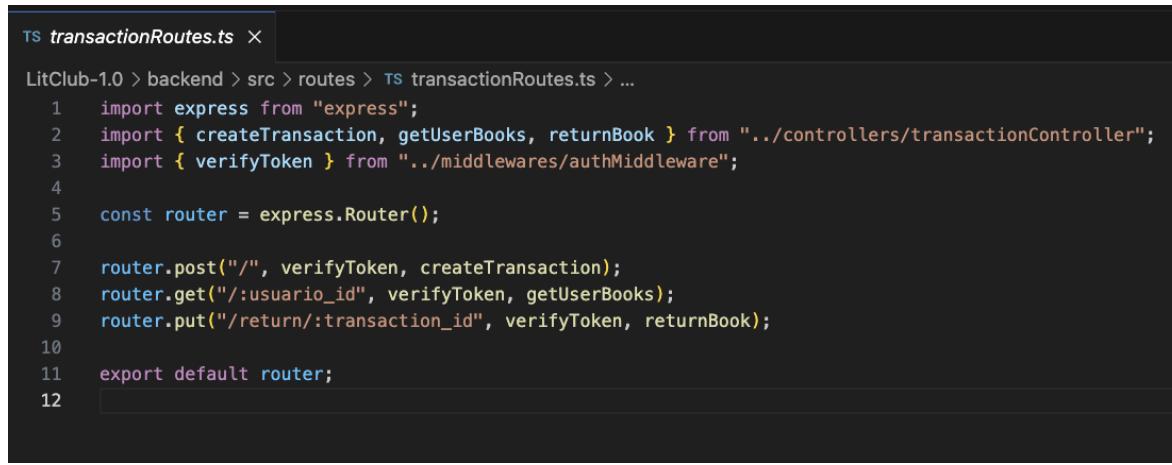
```
TS bookRoutes.ts ×  
LitClub-1.0 > backend > src > routes > TS bookRoutes.ts > ...  
1 import express from "express";  
2 import { getAllBooks, addBook, updateBook, deleteBook } from "../controllers/bookController";  
3 import { verifyToken, isAdmin } from "../middlewares/authMiddleware";  
4  
5 const router = express.Router();  
6  
7 router.get("/", getAllBooks); // Abierto para todos  
8 router.post("/", verifyToken, isAdmin, addBook); // Solo admin  
9 router.put("/:id", verifyToken, isAdmin, updateBook); // Solo admin  
10 router.delete("/:id", verifyToken, isAdmin, deleteBook); // Solo admin  
11  
12 export default router;  
13
```

Este código define las rutas para la gestión de transacciones de compra y renta de libros en la aplicación, asegurando que solo los usuarios autenticados puedan acceder a ellas mediante el **middleware** `verifyToken`.

- **POST /** → Crea una nueva transacción de compra o renta de un libro. Requiere autenticación, lo que significa que solo los usuarios con una sesión activa pueden realizar una transacción.
- **GET /:usuario_id** → Obtiene la lista de libros comprados o rentados por un usuario específico, identificado por su ID. Solo los usuarios autenticados pueden acceder a su historial de transacciones.
- **PUT /return/:transaction_id** → Permite a un usuario devolver un libro rentado, identificando la transacción por su ID. Esta acción también requiere autenticación para garantizar que solo el propietario de la transacción pueda

realizar la devolución.

Este enrutador protege la integridad de las transacciones, asegurando que solo los usuarios autenticados puedan comprar, rentar o devolver libros dentro del sistema.



TS transactionRoutes.ts X

LitClub-1.0 > backend > src > routes > **transactionRoutes.ts** > ...

```
1 import express from "express";
2 import { createTransaction, getUserBooks, returnBook } from "../controllers/transactionController";
3 import { verifyToken } from "../middlewares/authMiddleware";
4
5 const router = express.Router();
6
7 router.post("/", verifyToken, createTransaction);
8 router.get("/:usuario_id", verifyToken, getUserBooks);
9 router.put("/return/:transaction_id", verifyToken, returnBook);
10
11 export default router;
12
```

Este código configura el servidor de la aplicación utilizando **Express**, asegurando seguridad, manejo de CORS y definición de rutas principales.

- **Carga de variables de entorno:** dotenv.config() permite el uso de variables definidas en un archivo .env.
- **Seguridad con helmet:** Añade cabeceras HTTP para proteger la aplicación contra ataques como **Cross-Site Scripting (XSS)** y **Clickjacking**.
- **Configuración de CORS:** Restringe el acceso solo al frontend alojado en <http://localhost:5173>, permitiendo métodos HTTP específicos y el uso de credenciales como tokens de autenticación.
- **Middleware express.json():** Habilita la lectura de datos en formato JSON en las solicitudes.
- **Definición de rutas:**
 - o **/api/auth** → Maneja la autenticación (register y login).
 - o **/api/users** → Gestiona usuarios y roles.
 - o **/api/books** → Controla la administración de libros.
 - o **/api/transactions** → Permite comprar, rentar y devolver libros.

Este servidor está diseñado para ofrecer una API segura y bien estructurada, facilitando la comunicación con el frontend.

```
TS app.ts X
LitClub-1.0 > backend > src > TS app.ts > ...
1 import express from "express";
2 import cors from "cors";
3 import dotenv from "dotenv";
4 import helmet from "helmet";
5 import authRoutes from "./routes/authRoutes";
6 import userRoutes from "./routes/userRoutes";
7 import bookRoutes from "./routes/bookRoutes";
8 import transactionRoutes from "./routes/transactionRoutes";
9
10 dotenv.config();
11
12 const app = express();
13
14 // 🔔 Configurar seguridad con Helmet
15 app.use(helmet());
16
17 // 🔔 Configurar CORS correctamente
18 app.use(cors({
19   origin: "http://localhost:5173", // 🌐 Permitir solo el frontend local
20   methods: ["GET", "POST", "PUT", "DELETE"],
21   allowedHeaders: ["Content-Type", "Authorization"],
22   credentials: true // Habilitar credenciales si usas autenticación con cookies o tokens
23 }));
24
25 app.use(express.json());
26
27
28 // Definir rutas
29 app.use("/api/auth", authRoutes);
30 app.use("/api/users", userRoutes);
31 app.use("/api/books", bookRoutes);
32 app.use("/api/transactions", transactionRoutes);
33
34 export default app;
```

Este código inicia el servidor de la aplicación y configura una tarea programada para gestionar rentas vencidas.

- **Importación de app:** Se carga la configuración del servidor desde el archivo principal de la aplicación.
- **Uso de node-cron:** Se puede utilizar para ejecutar tareas periódicas, como la eliminación de rentas vencidas automáticamente.
- **Definición del puerto:** La aplicación escucha en el puerto definido en las variables de entorno (PORT) o, por defecto, en 5001.
- **Inicio del servidor:** Al ejecutar app.listen(PORT), el servidor queda activo y listo para recibir solicitudes en <http://localhost:5001>.

Este código garantiza que la API esté operativa y lista para gestionar solicitudes de clientes, facilitando la administración de libros y transacciones.

Control de Versiones en LitClub

Se configuró Git en el proyecto para gestionar versiones y se vinculó con un repositorio en GitHub. La primera versión del código se subió a la rama principal (main) y se creó una rama adicional (v1.0) para mantener un historial estructurado.

Para futuras actualizaciones, se estableció un flujo de trabajo basado en ramas, donde cada versión nueva tiene su propia rama (v1.1, v2.0, etc.), asegurando organización y trazabilidad. Se definió un proceso para fusionar cambios con main cuando las versiones sean estables.

También se documentaron los pasos para eliminar o desvincular el repositorio en caso de ser necesario.

FrontEnd

El frontend de LitClub está construido utilizando React, una biblioteca de JavaScript que nos permite crear una interfaz de usuario dinámica y altamente interactiva. Con React, podemos gestionar de manera eficiente el estado de la aplicación, lo que se traduce en una experiencia fluida para el usuario. Utilizamos Axios para realizar las solicitudes HTTP de manera sencilla y eficiente, garantizando que los datos se sincronicen correctamente entre el cliente y el servidor.

Para el diseño de la interfaz y las diapositivas del proyecto, recurrimos a Figma, una herramienta colaborativa de diseño que nos permite crear prototipos interactivos y visualizar cómo se verá la aplicación antes de comenzar la implementación. Esto facilita la toma de decisiones sobre el diseño

y mejora la comunicación dentro del equipo. Todo el desarrollo del frontend se realiza en Visual Studio Code, un editor de código liviano y altamente configurable que nos permite trabajar de manera eficiente y productiva. La combinación de estas herramientas asegura un desarrollo frontend ágil, atractivo y de alta calidad para LitClub

LitClub.

Diseño

de la

Aplicaci

ón

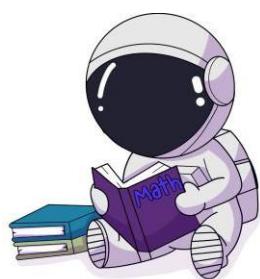
Tipograf

ía:

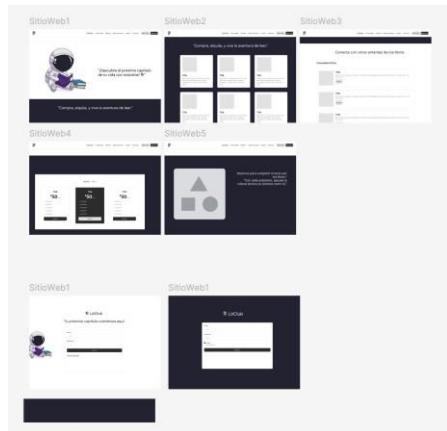
Varela

Rounde

d Logo:



Diapositivas:



Catalogo Comunidad Ofertas Sobre Nosotros Ayuda Contacto Iniciar Sesión Registrarse



"¡Descubre el próximo capítulo
de tu vida con nosotros! 📚"

The wireframe shows a dark-themed website layout. At the top is a header bar with a logo icon, navigation links (Catalogo, Comunidad, Ofertas, Sobre Nosotros, Ayuda, Contacto), and two buttons (Iniciar Sesión, Registrarse). Below the header is a dark banner containing the slogan "Compra, alquila, y vive la aventura de leer.". The main content area features a 2x3 grid of six placeholder cards. Each card has a small image placeholder at the top, followed by a title section and a body text area. The body text includes a placeholder for a short story or quote.

"Compra, alquila, y vive la aventura de leer."

Title
Body text for whatever you'd like to say.
Add main takeaway points, quotes, anecdotes, or even a very very short story.

Title
Body text for whatever you'd like to say.
Add main takeaway points, quotes, anecdotes, or even a very very short story.

Title
Body text for whatever you'd like to say.
Add main takeaway points, quotes, anecdotes, or even a very very short story.

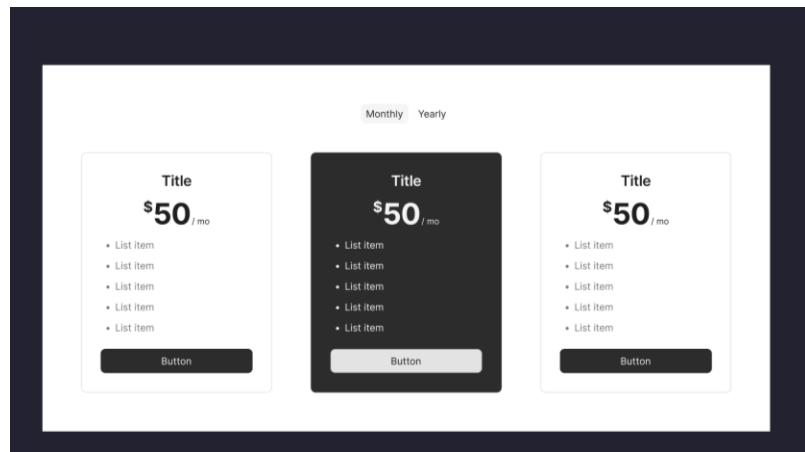
The wireframe shows a dark-themed website layout. At the top is a header bar with a logo icon, navigation links (Catalogo, Comunidad, Ofertas, Sobre Nosotros, Ayuda, Contacto), and two buttons (Iniciar Sesión, Registrarse). Below the header is a dark banner containing the slogan "Conecta con otros amantes de los libros". The main content area features a 1x2 grid of two placeholder cards. Each card has a small image placeholder at the top, followed by a title section and a body text area. The body text includes a placeholder for a short story or quote.

Conecta con otros amantes de los libros

Comunidad LitClub

Title
Body text for whatever you'd like to say. Add main takeaway points, quotes, anecdotes, or even a very very short story.
Button

Title
Body text for whatever you'd like to say. Add main takeaway points, quotes, anecdotes, or even a very very short story.
Button



Tu próximo capítulo comienza aquí



Email
Value

Password
Value

[Sign In](#)

[Forgot password?](#)



Email
Value

Password
Value

Label
Description

[Register](#)

Base de Datos

PostgreSQL (versión 15.x)

Descarga desde PostgreSQL.

Sigue el asistente para instalar:

- Configura una contraseña para el usuario "postgres".
- Selecciona las herramientas de administración, como pgAdmin.

The screenshot shows the official PostgreSQL website at [postgresql.org](https://www.postgresql.org). At the top, there's a navigation bar with links to Home, About, Download, Documentation, Community, Developers, Support, Donate, and Your account. A search bar is also present. A banner at the top right indicates the release of PostgreSQL 17.2, 16.6, 15.10, 14.15, 13.18, and 12.22. The main title "PostgreSQL: The World's Most Advanced Open Source Relational Database" is displayed over a background image of a person working outdoors. Below the title are two buttons: "Download" and "New to PostgreSQL?". On the left, there's a section for "New to PostgreSQL?" with a brief introduction and a link to learn more. On the right, there's a "Latest Releases" section with a link to the full release notes.

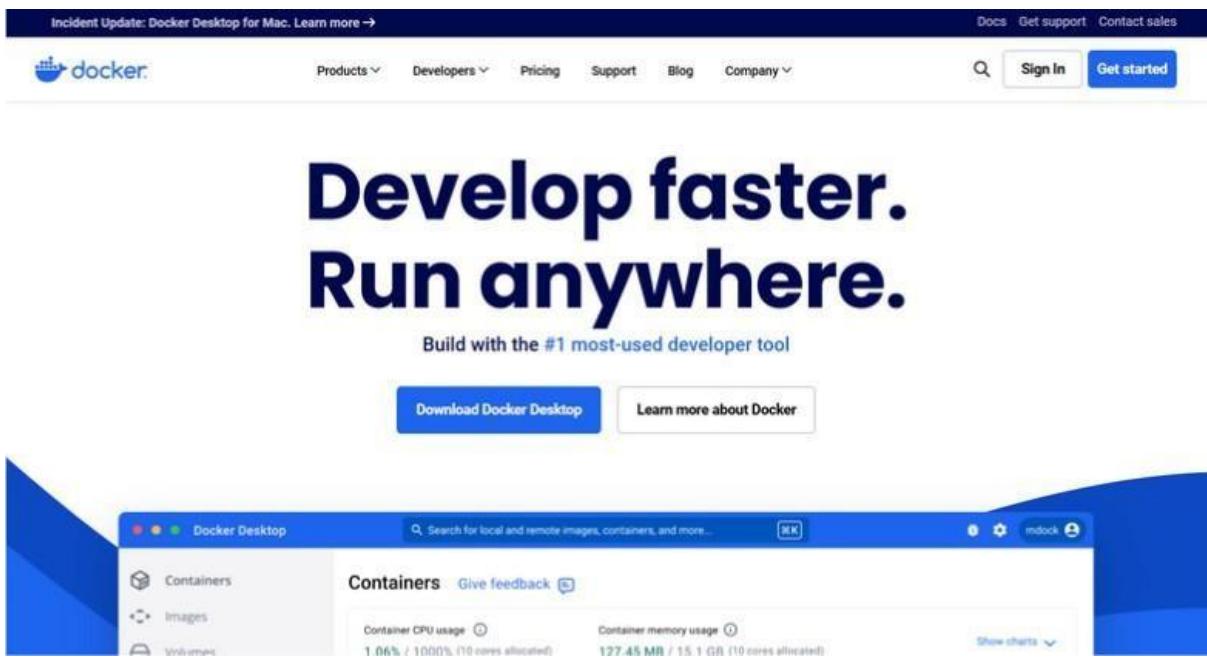
The screenshot shows the "Downloads" page of the PostgreSQL website. The main heading is "PostgreSQL Downloads". It states that PostgreSQL is available for download as ready-to-use packages or installers for various platforms, as well as a source code archive if you want to build it yourself. Below this, there's a section for "Packages and Installers" with a "Select your operating system family:" dropdown. Five options are shown: Linux (with a Linux logo), macOS (with a Mac logo), Windows (with a Windows logo), BSD (with a BSD logo), and Solaris (with a Solaris logo). There are also sections for "Source code" (with a note about finding it in the file browser or git repository) and "Beta/RC Releases and development snapshots (unstable)" (with a note about using them for testing purposes only). At the bottom, there's a section for "3rd party distributions" and a "Ready to run stacks" button.

pgAdmin 4

Incluido con PostgreSQL, puedes abrirlo desde el menú de inicio.

Conéctalo a tu base de datos PostgreSQL ingresando el usuario y contraseña configurados.

Contenedores y Gestión del Código



Docker(versión 24.x)

Descarga Docker Desktop desde Docker. Sigue el asistente para instalarlo:

- Reinicia tu computadora si es necesario
- Verifica la instalación ejecutando:

```
docker --version
```

Docker Compose (versión 2.20.0)

Ya viene integrado con Docker Desktop.

Verifica la instalación ejecutando

```
docker-compose --version
```

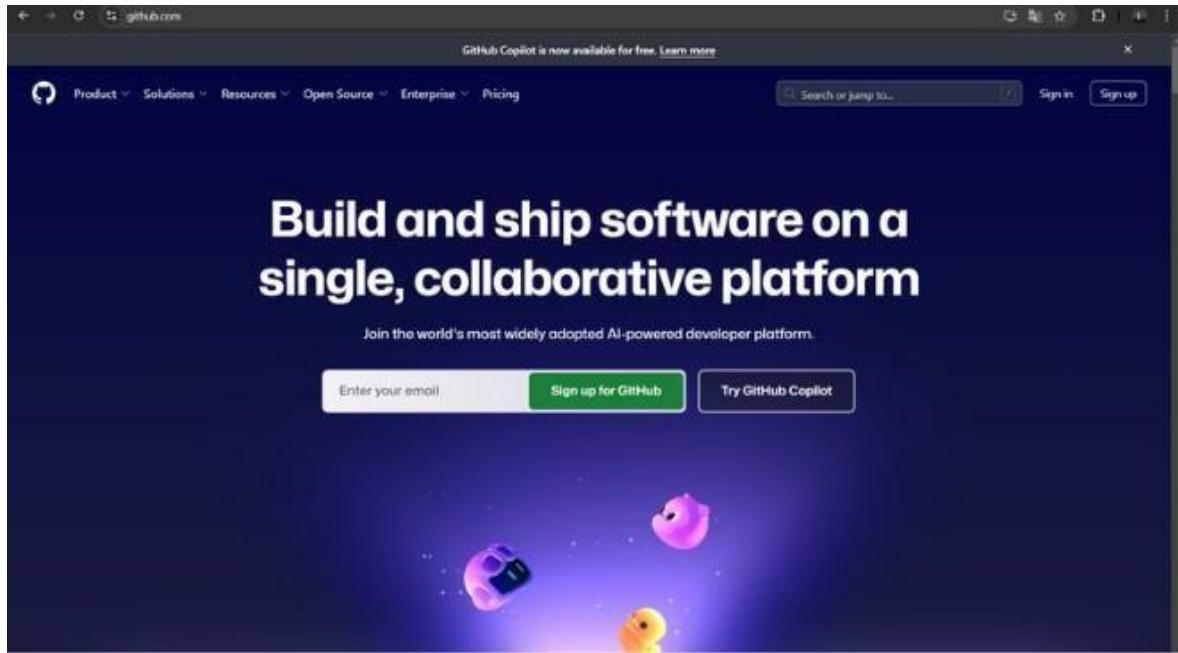
Git (versión 2.40.0 o superior)

Descarga Git desde <https://git-scm.com/>

Instálalo y configura tu nombre y correo

```
git config --global user.name "TuNombre"  
git config --global user.email "TuCorreo"
```





Cómo iniciar y detener el entorno con Docker Compose

Iniciar los contenedores

Desde la raíz del proyecto donde está ubicado docker-compose.yml, ejecuta:
docker-compose up -d

```
PS C:\Users\UsuarioPrueba\LitClub> docker-compose up -d --build
time="2025-02-07T18:47:05-06:00" level=warning msg="C:\\\\Users\\\\UsuarioPrueba\\\\LitClub\\\\LitClub\\\\docker-compose.yml: the attribute 'version' is obsolete, it will be ignored, please remove it to avoid potential confusion"
[+] Building 33.5s (19/19) FINISHED
  > [backend internal] load build definition from dockerfile
  >> [backend internal] load metadata for docker.io/library/node:18
  >> [frontend internal] load .dockerignore
  >> [backend internal] load build context
  >> [frontend 1/5] FROM docker.io/library/node:18@sha256:f12d34408955b2081f82078e8f96c3299ca0f38d11e76086cb9b1b669977e4
  >> [backend internal] load build context
  >> [transferring context: 28.37MB]
```

Esto levantará los servicios definidos en docker-compose.yml, que incluyen el backend, frontend y la base de datos PostgreSQL.

Descripción de lo que se hizo:

Ejecutaste el siguiente comando en la terminal de PowerShell dentro de tu proyecto:

docker-compose up -d

¿Qué hace este comando?

docker-compose up → Levanta los servicios definidos en docker-compose.yml.

-d (detached mode) → Ejecuta los contenedores en segundo plano.

--build → Fuerza la reconstrucción de las imágenes antes de ejecutar los contenedores.

Lo que está pasando en la terminal:

Se está procesando el archivo docker-compose.yml, pero aparece un warning que indica que la clave version es obsoleta y será ignorada (esto no impide la ejecución, pero es recomendable

actualizar el archivo).

Docker está construyendo los servicios del backend, frontend y base de datos.
Se están transfiriendo los archivos del proyecto al contenedor (transferring context).
Se están descargando las imágenes necesarias, como node:18 y postgres:latest.

Detener los contenedores

Para detener los contenedores sin eliminar los volúmenes y la red:

docker-compose down

Si deseas detener y eliminar volúmenes, usa:

docker-compose down -v

Validar Hot Reload en Desarrollo

Para asegurarte de que los cambios en el código se reflejan sin necesidad de reconstruir los contenedores, revisa lo siguiente:

1. Volúmenes en Docker Compose

Asegúrate de que en docker-compose.yml, los volúmenes están correctamente montados en el backend y frontend, por ejemplo:

```
►Run All Services
services:
  ► Run Service
    backend:
      build: ./backend
      volumes:
        - ./backend:/app
        - /app/node_modules
      ports:
        - "5001:5001"
      environment:
        - NODE_ENV=development
        - DB_USER=postgres
        - DB_PASSWORD=123456
        - DB_HOST=db
        - DB_NAME=litclub
        - DB_PORT=5432
      depends_on:
        - db
      command: npm run dev # Usa nodemon para hot reload

  ► Run Service
    frontend:
      build: ./frontend
      volumes:
        - ./frontend:/app
        - /app/node_modules
      ports:
        - "5173:5173"
      environment:
        - NODE_ENV=development
      depends_on:
        - backend
      command: npm run dev
```

Esto permite que los archivos en tu máquina se reflejen dentro del contenedor sin necesidad de reconstruirlo.

2. Configuración en Backend

Si usas Node.js en el backend, agrega nodemon para reiniciar automáticamente el servidor cuando haya cambios:

- Instala nodemon si no está instalado:

```
npm install -g nodemon
```

```
PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS GITLENS POSTMAN CONSOLE SPELL CHECKER COMENTARIOS
PS C:\Users\UsuarioPrueba\LitClub> cd LitClub
PS C:\Users\UsuarioPrueba\LitClub> cd backend
PS C:\Users\UsuarioPrueba\LitClub\LitClub\backend> npm install nodemon --save-dev
changed 5 packages, and audited 155 packages in 2s

19 packages are looking for funding
  run 'npm fund' for details

found 0 vulnerabilities
PS C:\Users\UsuarioPrueba\LitClub\LitClub\backend>
```

- En package.json del backend, modifica el script de inicio:

```
package.json
LitClub > backend > package.json > ...
1 {
2   "name": "backend",
3   "version": "1.0.0",
4   "main": "index.js",
5   "scripts": {
6     "test": "echo \\\"Error: no test specified\\\" && exit 1",
7     "start": "node server.js",
8     "dev": "nodemon server.js"
9   },
10  "keywords": [],
11  "author": "",
12  "license": "ISC",
13  "dependencies": {
14    "bcryptjs": "^2.4.3",
15    "body-parser": "^1.20.3",
16    "cors": "^2.8.5",
17    "dotenv": "^16.4.7",
18    "express": "^4.21.2",
19    "jsonwebtoken": "9.0.2",
20    "pg": "^8.13.1"
21  },
22  "devDependencies": {
23    "@types/bcryptjs": "^2.4.6",
24    "@types/cors": "^2.8.17",
25    "@types/express": "5.0.0",
26    "@types/node": "22.13.0",
27    "@types/pg": "8.11.11",
28    "nodemon": "3.1.9",
29    "typescript": "5.7.3"
30  },
31  "description": ""
32}
33
```

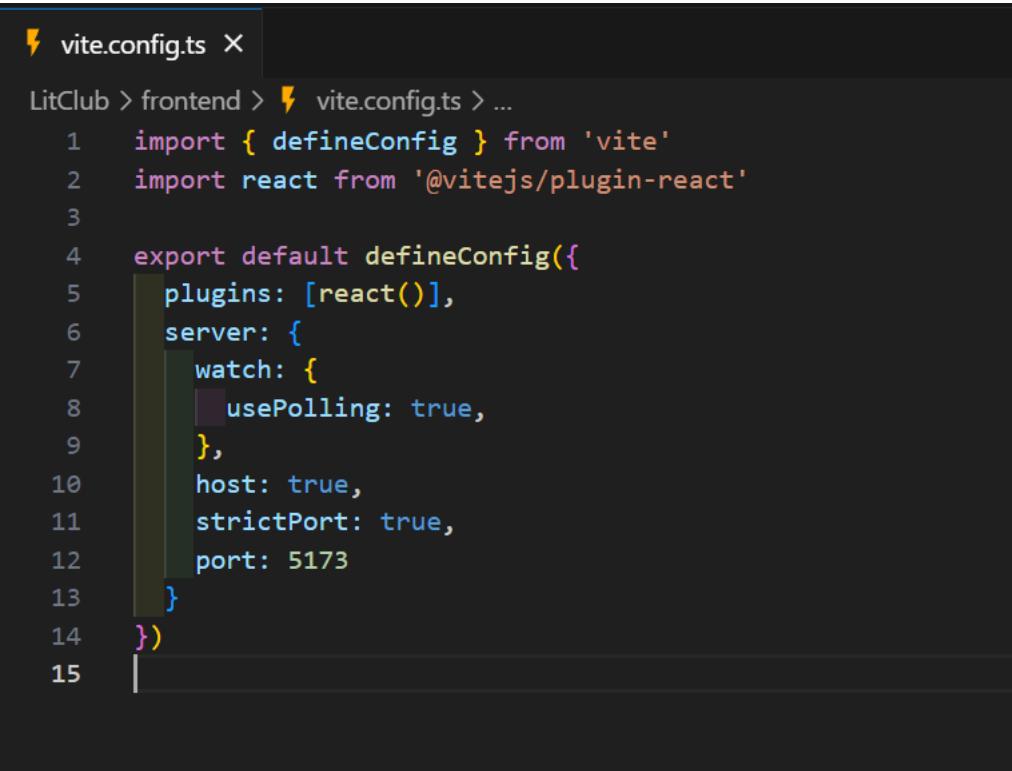
- Luego, en docker-compose.yml, inicia el backend en modo desarrollo:
comando: npm run dev

3. Configuración en Frontend

Si usas Vite en el frontend, ya tiene hot reload habilitado por defecto. Asegúrate de que en docker-compose.yml el volumen esté correctamente montado.

Si ves que no se reflejan cambios, agrega esta línea en vite.config.ts:

Explicación de los cambios:



```
vite.config.ts X
LitClub > frontend > vite.config.ts > ...
1 import { defineConfig } from 'vite'
2 import react from '@vitejs/plugin-react'
3
4 export default defineConfig({
5   plugins: [react()],
6   server: {
7     watch: {
8       usePolling: true,
9     },
10    host: true,
11    strictPort: true,
12    port: 5173
13  }
14})
15 |
```

- ♦ watch: { usePolling: true } → Permite que los archivos sean detectados correctamente en Docker.
- ♦ host: true → Permite acceder al servidor desde otras máquinas en la red.
- ♦ strictPort: true → Se asegura de que Vite solo use el puerto especificado (5173).
- ♦ port: 5173 → Define el puerto donde se ejecutará el servidor de desarrollo.

4. Reiniciar los Contenedores sin Borrar Datos

Si haces cambios en docker-compose.yml, reconstruye los servicios sin perder volúmenes:

docker-compose up -d --build

Resumen de Comandos Útiles

| Acción | Comando |
|-------------------------------|--------------------------------|
| Iniciar contenedores | docker-compose up -d |
| Detener contenedores | docker-compose down |
| Detener y eliminar volúmenes | docker-compose down -v |
| Reconstruir con cambios | docker-compose up -d --build |
| Ver logs de un servicio | docker-compose logs -f backend |
| Acceder al contenedor backend | docker exec -it backend-1 sh |

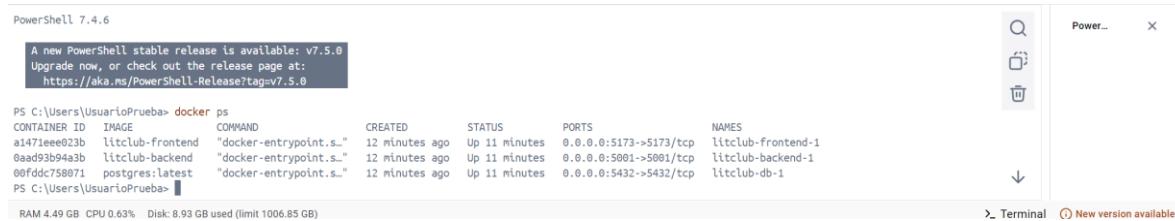
5. Cuando el proceso termine, puedes verificar que los contenedores están corriendo con:

| | litclub | | | | | | | |
|---|--------------|--------------|------------------|-------------|-------|---------------|---|---|
| □ | ● db-1 | 00fdc758071 | postgres:latest | 5432:5432 ↗ | 0.01% | 5 minutes ago | ⋮ | 🔗 |
| □ | ● backend-1 | 0aad93b94a3b | litclub-backend | 5001:5001 ↗ | 0.01% | 5 minutes ago | ⋮ | 🔗 |
| □ | ● frontend-1 | a1471eee023b | litclub-frontend | 5173:5173 ↗ | 5.82% | 5 minutes ago | ⋮ | 🔗 |

docker ps

En la imagen se puede que los contenedores de docker están corriendo

6. Si hay algún error o los servicios no inician correctamente, revisa los logs con:
docker-compose logs



```
PowerShell 7.4.6
A new PowerShell stable release is available: v7.5.0
Upgrade now, or check out the release page at:
https://aka.ms/PowerShell-Release?tag=v7.5.0

PS C:\Users\UsuarioPrueba> docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
a1471eee023b litclub-frontend "docker-entrypoint.s..." 12 minutes ago Up 11 minutes 0.0.0.0:5173->5173/tcp litclub-frontend-1
0aad93b94a3b litclub-backend "docker-entrypoint.s..." 12 minutes ago Up 11 minutes 0.0.0.0:5001->5001/tcp litclub-backend-1
00fdc758071 postgres:latest "docker-entrypoint.s..." 12 minutes ago Up 11 minutes 0.0.0.0:5432->5432/tcp litclub-db-1
PS C:\Users\UsuarioPrueba>
```

Este comando revisa que estén los contenedores estén corriendo y muestra los puertos que esta escuchando, así como muestra el nombre de cada contenedor.



```
PS C:\Users\UsuarioPrueba> docker exec -it litclub-db-1 psql -U postgres -d litclub
psql (17.2 (Debian 17.2-1.pgdg120+1))
Type "help" for help.

litclub=#
```

Este comando me permite acceder a mi base de datos litclub

Comandos para verificar la base de datos

Ver las tablas existentes en la base de datos:

SELECT * FROM pg_tables WHERE schemaname = 'public';

```
litclub=# \dt public.*
      List of relations
 Schema |   Name    | Type  | Owner
-----+-----+-----+-----
 public | usuarios | table | postgres
(1 row)
litclub=#

```

Verificar que la tabla usuarios se creó correctamente : Puedes ejecutar el

siguiente comando para asegurarte de que la tabla está en la base de datos:

\dt public.*

Este comando mostrará todas las tablas en el esquema public de la base de datos litclub, y deberías ver la tabla usuarios en la lista.

Ver los detalles de la tabla: Para ver la estructura de la tabla usuarios que acabas de crear, puedes usar:

\d usuarios

```
litclub=# \d usuarios
                                         Table "public.usuarios"
   Column |      Type       | Collation | Nullable | Default
-----+-----+-----+-----+-----+
    id   | integer        |           | not null | nextval('usuarios_id_seq'::regclass)
  nombre | character varying(100) |           |           |
  email  | character varying(100) |           |           |
     rol  | character varying(50) |           |           |
fechaingreso | timestamp without time zone |           |           | CURRENT_TIMESTAMP
  password | character varying(255) |           |           |
Indexes:
  "usuarios_pkey" PRIMARY KEY, btree (id)

RAM 4.52 GB CPU 1.06% Disk: 8.93 GB used (limit 1006.85 GB)
```

Salir de la consola de PostgreSQL: Si ya no necesitas más interacciones, puedes salir de la consola de PostgreSQL con:

\g

Comandos para verificar el backend

Ejecuta el siguiente comando para verificar que tu contenedor backend-1 está en ejecución:
docker ps

```
PS C:\Users\UsarioPrueba> docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
a1471ee023b litclub-frontend "docker-entrypoint.s..." About an hour ago Up About an hour 0.0.0.0:5173->5173/tcp litclub-frontend-1
0aa9d9b94a3b litclub-backend "docker-entrypoint.s..." About an hour ago Up About an hour 0.0.0.0:5001->5001/tcp litclub-backend-1
05ffddc758071 postgres:latest "docker-entrypoint.s..." About an hour ago Up About an hour 0.0.0.0:5432->5432/tcp litclub-db-1
PS C:\Users\UsarioPrueba>
```

Busca el contenedor backend-1 y asegúrate de que está en ejecución. Si ves algo como esto:

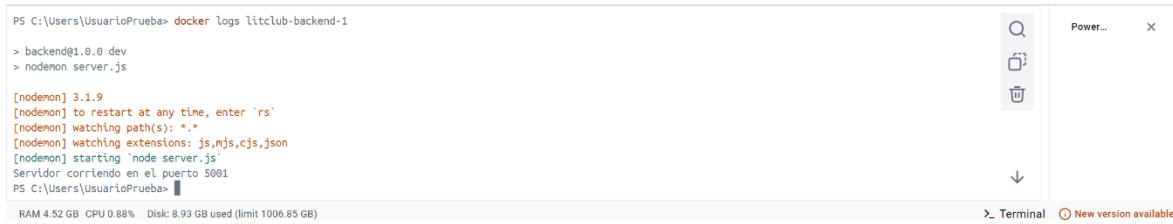
| CONTAINER ID | IMAGE | COMMAND | CREATED | STATUS | PORTS |
|--------------|-------|---------|---------|--------|-------|
| NAMES | | | | | |

```
abc123def456 my-backend  "npm start"          2 hours ago  Up 2 hours  0.0.0.0:5001->5001/tcp
backend-1
```

Este significa que tu contenedor está funcionando correctamente y el puerto 5001 está vinculado.

Verificar los logs del contenedor

Si tu contenedor está en ejecución pero necesitas más detalles, puedes revisar los logs con:
docker logs litclub-backend-1



```
PS C:\Users\UsuarioPrueba> docker logs litclub-backend-1
> backend@1.0.0 dev
> nodemon server.js

[nodemon] 3.1.9
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): "."
[nodemon] watching extensions: js,njs,cjs,json
[nodemon] starting 'node server.js'
Servidor corriendo el puerto 5001
P: C:\Users\UsuarioPrueba>
```

Busca en los logs si ves algún mensaje como:

Server is running on http://0.0.0.0:5001

Si ves este mensaje, eso confirma que el backend está corriendo en el puerto 5001. Si hay algún error, aparecerá en los logs, lo que te ayudará a solucionar problemas.

Verificar si la aplicación está escuchando en el puerto correcto

Ejecuta nuevamente el comando docker ps para asegurarte de que el puerto 5001 está correctamente expuesto. En la columna PORTS, deberías ver algo como:

0.0.0.0:5001->5001/tcp

Esto significa que el puerto 5001 dentro del contenedor está siendo redirigido al puerto 5001 de tu máquina local.

Probar la conexión al backend desde tu navegador o cliente

Si todo está correcto, abre tu navegador y navega a:

<http://localhost:5001>

Si el backend está funcionando, deberías ver la respuesta de la aplicación, como un mensaje de bienvenida o la respuesta de una API.

Si tu aplicación requiere autenticación o rutas específicas, asegúrate de probar esas rutas en el navegador o con herramientas como Postman.

POST http://localhost:3000/x +

http://localhost:3000/auth/login

POST http://localhost:5001/register

Params Authorization Headers (8) Body Scripts Settings Cookies

Query Params

| Key | Value | Description | Bulk Edit |
|-----|-------|-------------|-----------|
| Key | Value | Description | ... |

Body Cookies Headers (9) Test Results

500 Internal Server Error 24 ms 358 B

```
{
  "error": "Error al registrar usuario"
}
```

Al usar post <http://localhost:5001/register> funciona y estoy conectado en mi contenedor de litclub-backend-1

POST http://localhost:5001/x + POST http://localhost:5001/i +

http://localhost:5001/login

POST http://localhost:5001/login

Params Authorization Headers (8) Body Scripts Settings Cookies

Query Params

| Key | Value | Description | Bulk Edit |
|-----|-------|-------------|-----------|
| Key | Value | Description | ... |

Body Cookies Headers (9) Test Results

500 Internal Server Error 12 ms 349 B

```
{
  "error": "Error en el login"
}
```

Al usar post <http://localhost:5001/login> funciona y estoy conectado en mi contenedor de litclub-backend-1.

Probar con curl desde la terminal

Si prefieres usar la terminal, puedes probar la conexión con curl ejecutando:

```
curl http://localhost:5001
```

Si todo está funcionando correctamente, deberías recibir una respuesta del backend. Si hay algún problema, se mostrará un mensaje de error.

Verificar la configuración de la red de Docker (si es necesario)

Si tu backend está conectado a otros servicios (como bases de datos o contenedores adicionales), asegúrate de que todos estén en la misma red de Docker. Si estás utilizando Docker Compose, revisa tu archivo docker-compose.yml para asegurarte de que todos los servicios estén conectados a la misma red.

Resumen:

1. Verificar que el contenedor esté en ejecución: docker ps.
2. Revisar los logs del contenedor: docker logs litclub-backend-1.
3. Verificar que el puerto 5001 esté expuesto correctamente: docker ps.
4. Probar la conexión con el backend en el navegador o con curl: http://localhost:5001.
5. Verificar la configuración de red de Docker (si aplica).

Comandos para verificar el frontend

Primero, verifica que el contenedor frontend-1 esté corriendo correctamente. Para esto, usa el siguiente comando en la terminal:

```
docker ps
```

Busca la columna PORTS en la salida. Debes ver algo como esto:

```
0.0.0.0:5173->5173/tcp
```

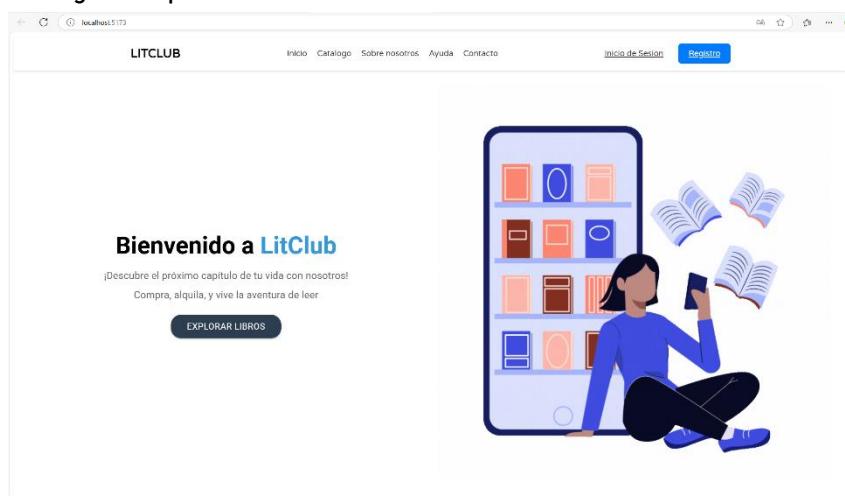
Esto indica que el puerto 5173 del contenedor está mapeado al puerto 5173 de tu máquina local.

Verificar el acceso al frontend en tu navegador

Abre un navegador web y accede a:

```
http://localhost:5173
```

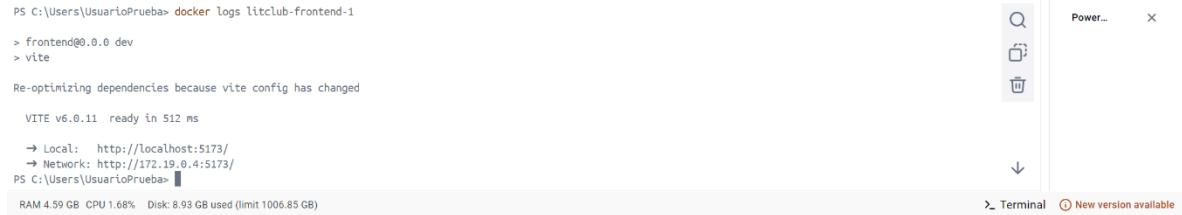
Si todo está configurado correctamente, deberías ver tu aplicación frontend cargada. Si no se carga, revisa el siguiente paso.



Revisar los logs del contenedor frontend-1

Si no se carga correctamente, verifica los logs del contenedor frontend-1 para ver si hay errores al arrancar la aplicación. Para esto, usa:

```
docker logs litclub-frontend-1
```



The screenshot shows a terminal window with the following text output:

```
PS C:\Users\UsuarioPrueba> docker logs litclub-frontend-1
> frontend@0.0.0 dev
> vite

Re-optimizing dependencies because vite config has changed

VITE v6.0.11 ready in 512 ms
→ Local: http://localhost:5173/
→ Network: http://172.19.0.4:5173/
PS C:\Users\UsuarioPrueba>
```

RAM 4.59 GB CPU 1.68% Disk: 8.93 GB used (limit 1006.85 GB)

Power... X

Terminal New version available

Esto te mostrará los mensajes del contenedor y te permitirá detectar posibles errores de configuración o de la aplicación.

Revisar las herramientas de desarrollo del navegador (DevTools)

Si la página se carga pero algo no funciona bien (por ejemplo, solicitudes HTTP que no se completan), abre las herramientas de desarrollo del navegador (F12) y ve a la pestaña Network. Esto te permitirá ver todas las solicitudes que hace tu frontend al backend.

- Si ves que las solicitudes hacia el backend están fallando (por ejemplo, 404 Not Found o 500 Internal Server Error), esto indica que hay un problema de comunicación entre el frontend y el backend.
- Asegúrate de que las rutas en las solicitudes del frontend apunten al puerto correcto de tu backend (por ejemplo, <http://localhost:5001>).

Verificar la configuración de CORS en el backend

Si estás usando un frontend en <http://localhost:5173> y un backend en <http://localhost:5001>, debes asegurarte de que tu backend permita solicitudes desde el frontend. La configuración de CORS en el backend debe ser algo como esto:

```
app.use(cors({
  origin: "http://localhost:5173", // Permitir solicitudes desde el frontend
  methods: "GET,POST,PUT,DELETE",
  allowedHeaders: "Content-Type,Authorization"
}));
```

Si el backend no permite solicitudes de este origen, recibirás un error relacionado con CORS en el navegador.

Probar la comunicación entre frontend y backend

Si la página carga pero las interacciones no funcionan (por ejemplo, el login no responde), verifica que las solicitudes del frontend estén correctamente enviadas al backend.

- Abre la consola de desarrollo del navegador (F12) y revisa los errores en la pestaña Console.
- Revisa la pestaña Network para ver si el frontend está enviando las solicitudes correctas (por ejemplo, una solicitud POST al endpoint /register del backend).

Comprobar el puerto de frontend en el contenedor Docker

Si el frontend no está accesible, pero el contenedor está en ejecución, puedes asegurarte de que el puerto 5173 esté correctamente mapeado y accesible desde tu máquina local ejecutando:

```
docker inspect litclub-frontend-1
```

En la salida, busca la sección "Ports" para asegurarte de que el puerto 5173 esté mapeado correctamente al puerto 5173 de la máquina local.

Reiniciar el contenedor

Si no has encontrado ningún problema y aún no funciona, puedes intentar reiniciar el contenedor para asegurarte de que todo se haya cargado correctamente:

```
docker restart litclub-frontend-1
```

Verificar las configuraciones de red (si aplica)

Si tu aplicación frontend y backend están en contenedores diferentes, asegúrate de que estén en la misma red de Docker. Si usas Docker Compose, el archivo docker-compose.yml debe estar configurado para que ambos contenedores compartan la misma red.

Resumen de pasos para verificar frontend-1

1. Verifica que el contenedor esté corriendo con docker ps.
2. Abre <http://localhost:5173> en tu navegador y verifica que la aplicación se cargue.
3. Revisa los logs del contenedor con docker logs frontend-1 si no se carga.
4. Usa las herramientas de desarrollo del navegador (F12) y revisa las solicitudes en la pestaña Network.
5. Asegúrate de que el backend permita solicitudes de <http://localhost:5173> (verifica la configuración de CORS).
6. Verifica que las solicitudes del frontend se envíen al backend correctamente.
7. Reinicia el contenedor con docker restart frontend-1 si es necesario.

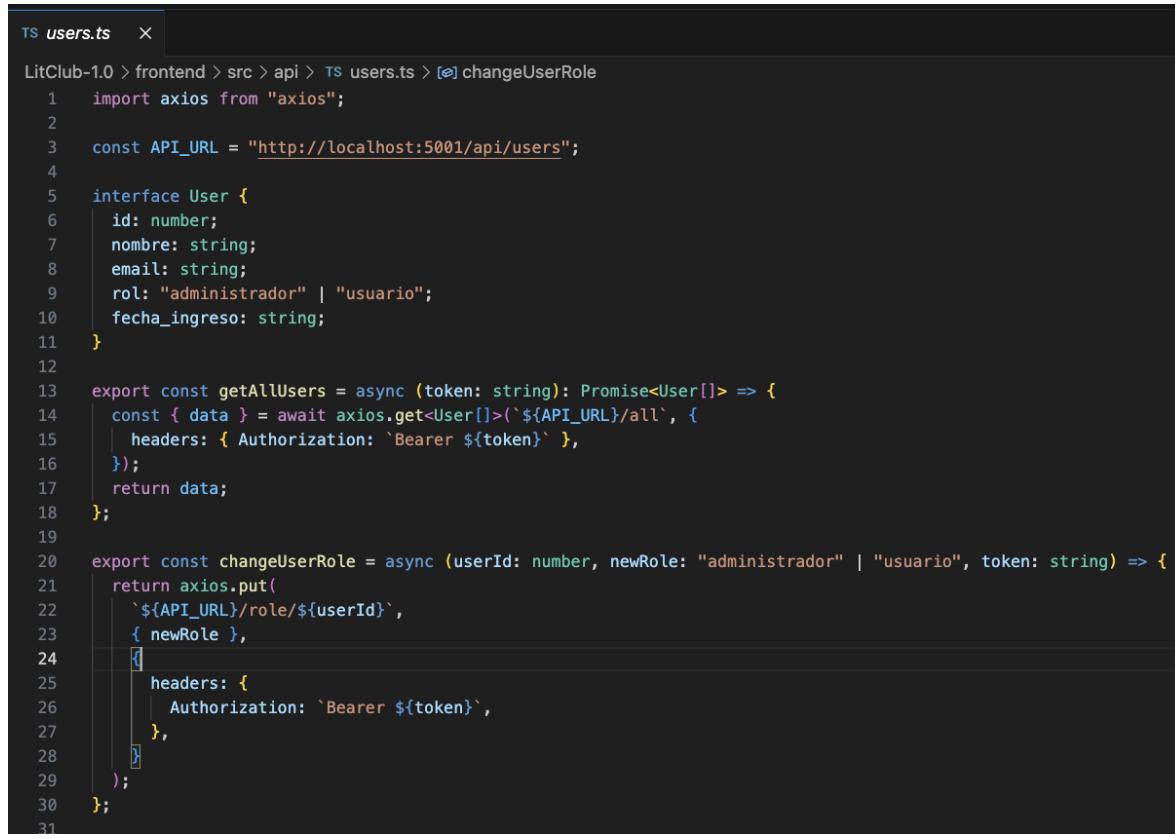
Nuevas Implementaciones

Documentos importantes del Frontend

Este código maneja la comunicación con la API del servidor para obtener y actualizar información de usuarios utilizando **Axios**.

- **Definición de la URL de la API:** API_URL apunta a <http://localhost:5001/api/users>, que es la ruta base para las solicitudes relacionadas con usuarios.
- **Interfaz User:** Define la estructura de un usuario, incluyendo **ID, nombre, email, rol y fecha de ingreso**.
- **getAllUsers:** Realiza una solicitud GET para obtener la lista de todos los usuarios. Requiere un **token de autenticación** en los encabezados para autorizar la petición.
- **changeUserRole:** Envía una solicitud PUT para cambiar el rol de un usuario. Se proporciona el **ID del usuario, el nuevo rol y un token de autenticación**.

Este código permite gestionar usuarios en el frontend, asegurando que solo usuarios autenticados puedan acceder a la información y modificar roles.



```
ts users.ts x
LitClub-1.0 > frontend > src > api > ts users.ts > [o] changeUserRole
1 import axios from "axios";
2
3 const API_URL = "http://localhost:5001/api/users";
4
5 interface User {
6   id: number;
7   nombre: string;
8   email: string;
9   rol: "administrador" | "usuario";
10  fecha_ingreso: string;
11 }
12
13 export const getAllUsers = async (token: string): Promise<User[]> => {
14   const { data } = await axios.get<User[]>(`${API_URL}/all`, {
15     headers: { Authorization: `Bearer ${token}` },
16   });
17   return data;
18 };
19
20 export const changeUserRole = async (userId: number, newRole: "administrador" | "usuario", token: string) => {
21   return axios.put(
22     `${API_URL}/role/${userId}`,
23     { newRole },
24     {
25       headers: {
26         Authorization: `Bearer ${token}`,
27       },
28     },
29   );
30 };
31
```

Este código gestiona la autenticación de usuarios en el frontend mediante solicitudes a la API del servidor utilizando **Axios**.

- **Definición de la URL de la API:** API_URL apunta a <http://localhost:5001/api/auth>, donde se encuentran las rutas de autenticación.
- **Interfaz LoginResponse:** Representa la estructura de la respuesta al iniciar sesión, incluyendo un **mensaje de éxito, el token JWT y los datos del usuario autenticado**.
- **registerUser:** Envía una solicitud POST a la API para registrar un nuevo usuario con **nombre, email y contraseña**.
- **loginUser:** Envía una solicitud POST para autenticar a un usuario. Si las credenciales son correctas, devuelve un **token JWT** y la información del usuario autenticado.

Este código permite que el frontend gestione el registro e inicio de sesión de usuarios, asegurando la autenticación con **tokens JWT** para futuras solicitudes protegidas.

```
TS auth.ts X
LitClub-1.0 > frontend > src > api > TS auth.ts > ...
1 import axios from "axios";
2
3 const API_URL = "http://localhost:5001/api/auth";
4
5 interface LoginResponse {
6   message: string;
7   token: string;
8   user: {
9     id: number;
10    nombre: string;
11    email: string;
12    rol: "administrador" | "usuario";
13    fecha_ingreso: string;
14  };
15}
16
17 export const registerUser = async (nombre: string, email: string, contraseña: string) => {
18   return axios.post(`${API_URL}/register`, { nombre, email, contraseña });
19 };
20
21 export const loginUser = async (email: string, contraseña: string): Promise<LoginResponse> => {
22   const { data } = await axios.post<LoginResponse>(`${API_URL}/login`, { email, contraseña });
23   return data;
24 };
25
```

Este código maneja las transacciones de compra, renta y devolución de libros en el frontend, comunicándose con la API del servidor mediante **Axios**.

- **Definición de la URL de la API:** API_URL apunta a <http://localhost:5001/api/transactions>, donde se gestionan las transacciones de libros.
- **createTransaction:** Envía una solicitud POST para comprar o rentar un libro. Requiere el **ID del usuario, ID del libro, tipo de transacción (compra o renta), días de renta (opcional) y un token de autenticación**. Antes de realizar la solicitud, verifica que el token esté presente.
- **getUserBooks:** Recupera la lista de libros comprados o rentados por un usuario, identificándolo por su **ID**. Se requiere un **token de autenticación** para la solicitud GET.
- **returnBook:** Envía una solicitud PUT para devolver un libro rentado, identificando la transacción por su **ID**. Verifica que el **token de autenticación** esté presente antes de ejecutar la solicitud.

Este código garantiza que solo los **usuarios autenticados** puedan realizar transacciones, manteniendo la seguridad del sistema y controlando el acceso a los recursos protegidos.

```
ts transactions.ts ×

LitClub-1.0 > frontend > src > api > ts transactions.ts > [o] returnBook
 6  export const createTransaction = async (
15    return axios.post(
16      API_URL,
17      { usuario_id, libro_id, tipo, dias_renta },
18      {
19        headers: { Authorization: `Bearer ${token}`, "Content-Type": "application/json" },
20      }
21    );
22  );
23
24  export const getUserBooks = async (usuario_id: number, token: string): Promise<UserBook[]> => {
25    if (!token) throw new Error("Token de autenticación no encontrado.");
26
27    const { data } = await axios.get(`${API_URL}/${usuario_id}`, {
28      headers: { Authorization: `Bearer ${token}` },
29    );
30
31    return data as UserBook[]; // ✅ Forzamos el tipo para evitar el error `unknown`
32  };
33
34  export const returnBook = async (transaction_id: number, token?: string) => {
35    if (!token) throw new Error("Token de autenticación no encontrado.");
36
37    return axios.put(`${API_URL}/return/${transaction_id}`, {}, [
38      headers: { Authorization: `Bearer ${token}` },
39    );
40  };
41
```

Este código permite gestionar usuarios en el frontend mediante solicitudes a la API del servidor utilizando **Axios**.

- **Definición de la URL de la API:** API_URL apunta a <http://localhost:5001/api/users>, donde se gestionan las operaciones relacionadas con usuarios.
- **Interfaz User:** Define la estructura de los usuarios, incluyendo **ID, nombre, email, rol y fecha de ingreso**.
- **getAllUsers:** Realiza una solicitud GET para obtener la lista de todos los usuarios registrados en el sistema. Se requiere un **token de autenticación** para acceder a esta información.
- **changeUserRole:** Envía una solicitud PUT para cambiar el rol de un usuario. Se debe proporcionar el **ID del usuario, el nuevo rol y un token de autenticación** para validar los permisos.

Este código garantiza que solo los **usuarios autenticados** y con permisos adecuados puedan acceder y modificar los datos de otros usuarios, asegurando la seguridad y el control de roles dentro del sistema.

ts users.ts X

LitClub-1.0 > frontend > src > api > ts users.ts > ...

```

1 import axios from "axios";
2
3 const API_URL = "http://localhost:5001/api/users";
4
5 interface User {
6   id: number;
7   nombre: string;
8   email: string;
9   rol: "administrador" | "usuario";
10  fecha_ingreso: string;
11 }
12
13 export const getAllUsers = async (token: string): Promise<User[]> => {
14   const { data } = await axios.get<User[]>(`${API_URL}/all`, {
15     headers: { Authorization: `Bearer ${token}` },
16   });
17   return data;
18 };
19
20 export const changeUserRole = async (userId: number, newRole: "administrador" | "usuario", token: string) => {
21   return axios.put(
22     `${API_URL}/role/${userId}`,
23     { newRole },
24     {
25       headers: {
26         Authorization: `Bearer ${token}`,
27       },
28     }
29   );
30 };
31

```

EXPLORADOR ...

SIN TÍT... + ⌂

- LitClub-1.0
 - > backend
 - > frontend
 - > node_modules
 - > public
 - > src
 - > api
 - TS auth.ts
 - TS books.ts
 - TS transactions.ts
 - TS users.ts
 - > assets
 - > components
 - ProtectedRoute.tsx
 - BookCard.tsx
 - HorrorWarning...
 - Navbar.tsx
 - NavbarHome.tsx
 - > context
 - > pages
 - > routes
 - > styles
 - > types
 - # App.css
 - App.tsx
 - # index.css

ProtectedRoute.tsx X

LitClub-1.0 > frontend > src > components > ProtectedRoute.tsx > ...

```

1 import { Navigate } from "react-router-dom";
2 import { JSX, useContext } from "react";
3 import { AuthContext } from "../context/AuthContext";
4
5 interface ProtectedRouteProps {
6   children: JSX.Element;
7   roleRequired?: "administrador" | "usuario";
8 }
9
10 export const ProtectedRoute = ({ children, roleRequired }: ProtectedRouteProps) => {
11   const auth = useContext(AuthContext);
12
13   if (!auth?.token) {
14     return <Navigate to="/login" />;
15   }
16
17   if (roleRequired && auth.user?.rol !== roleRequired) {
18     return <Navigate to="/dashboard" />;
19   }
20
21   return children;
22 };
23

```

Este código implementa un **contexto de autenticación** en **React** utilizando `useContext`, `useState` y `useEffect` para gestionar el estado de autenticación de los usuarios en la aplicación.

- **Definición de la interfaz User:** Representa la estructura de los datos de un usuario autenticado, incluyendo **ID, nombre, email, rol y fecha de ingreso**.
- **Interfaz AuthContextType:** Define las propiedades del contexto de autenticación, que incluyen el **usuario autenticado, el token de sesión, la función de inicio de sesión (login) y la función de cierre de sesión (logout)**.
- **AuthContext:** Se crea el contexto para proporcionar acceso a la autenticación en toda la aplicación.
- **AuthProvider:** Componente proveedor que maneja el estado de **usuario y token**.
 - Usa `useState` para almacenar el usuario autenticado y el token JWT.
 - Usa `useEffect` para **guardar el token en localStorage** cuando cambia, asegurando que persista entre recargas.
 - La función `login` autentica al usuario llamando a `loginUser`, guardando los datos del usuario y el token en el estado.
 - La función `logout` borra los datos de sesión y elimina el token de `localStorage`.
- **Proporciona el contexto** a todos los componentes hijos, permitiendo el acceso a la autenticación en cualquier parte de la aplicación.

Este código asegura que la autenticación sea manejada de manera global en el frontend, facilitando el control de sesión y el acceso a rutas protegidas.

Documentación de Control de Versiones con Git en LitClub

Repositorio: LitClub-1.0

Objetivo: Mantener el control de versiones del proyecto **LitClub**, asegurando que cada versión y actualización se registre correctamente en **GitHub**.

```
LitClub-1.0 > frontend > src > context > AuthContext.tsx > [e]AuthProvider
1 import { createContext, useState, useEffect, ReactNode } from "react";
2 import { loginUser } from "../api/auth";
3
4 interface User {
5   id: number;
6   nombre: string;
7   email: string;
8   rol: "administrador" | "usuario";
9   fecha_ingreso: string;
10 }
11
12 interface AuthContextType {
13   user: User | null;
14   token: string | null;
15   login: (email: string, contraseña: string) => Promise<void>;
16   logout: () => void;
17 }
18
19 // eslint-disable-next-line react-refresh/only-export-components
20 export const AuthContext = createContext<AuthContextType | null>(null);
21
22 export const AuthProvider = ({ children }: { children: ReactNode }) => {
23   const [user, setUser] = useState<User | null>(null);
24   const [token, setToken] = useState<string | null>(localStorage.getItem("token"));
25
26   useEffect(() => {
27     if (token) {
28       localStorage.setItem("token", token);
29     } else {
30       localStorage.removeItem("token");
31     }
32   }, [token]);
33
34   const login = async (email: string, contraseña: string) => {
35     try {
36       const data = await loginUser(email, contraseña);
37       setUser(data.user);
38       setToken(data.token);
39     } catch (error) {
40       console.error("Error en el login", error);
41     }
42   };
43
44   const logout = () => {
45     setUser(null);
46     setToken(null);
47     localStorage.removeItem("token");
48   };
49
50   return <AuthContext.Provider value={{ user, token, login, logout }}>{children}</AuthContext.Provider>;
51 }
52
```

Buenas Prácticas para Manejar Secretos en Docker

Introducción

El manejo seguro de secretos en Docker es fundamental para evitar la exposición de credenciales y configuraciones sensibles. Este documento describe buenas prácticas para proteger información sensible tanto en entornos de desarrollo como

de producción, asegurando la integridad y seguridad de las aplicaciones.

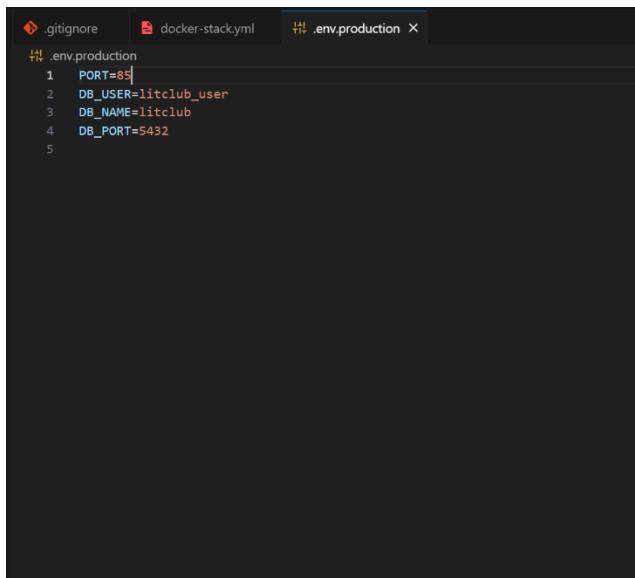
Evitar Almacenar Credenciales en Imágenes de Docker

- No incluir claves de API, credenciales de bases de datos o tokens en el Dockerfile.
- No copiar archivos .env dentro de la imagen.
- Usar variables de entorno o volúmenes para injectar configuraciones sin exponerlas.

Uso de Archivos .env.production en Desarrollo

En entornos de producción, se recomienda usar archivos .env montados como volumen para evitar exponer credenciales en la imagen de Docker.

Paso 1: Crear el Archivo .env.production



```
.gitignore docker-stack.yml .env.production
.env.production
1 PORT=88
2 DB_USER=litclub_user
3 DB_NAME=litclub
4 DB_PORT=5432
5
```

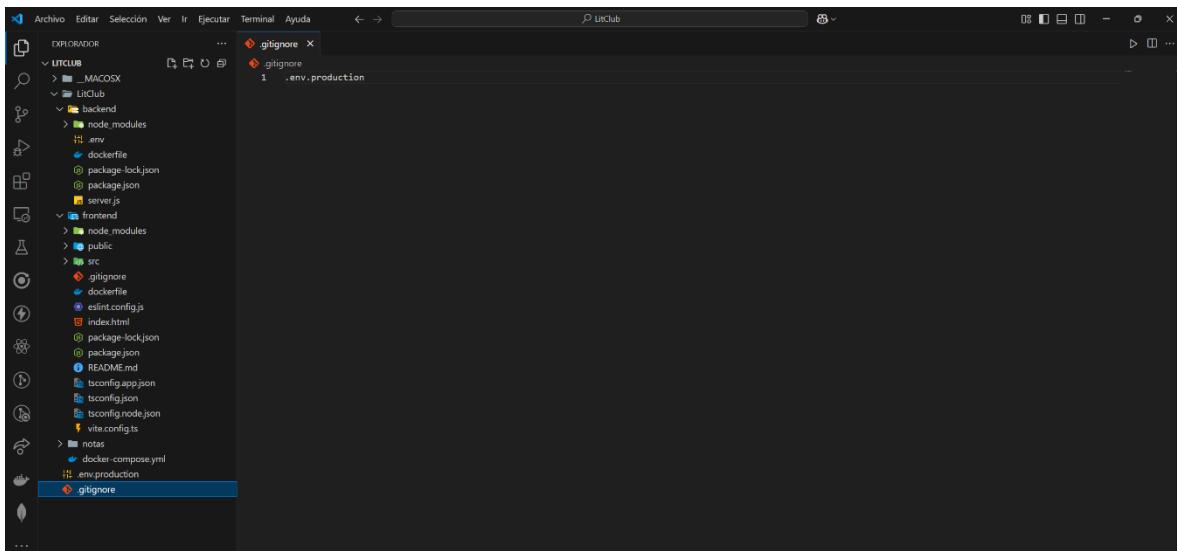
Ubicar el archivo en la raíz del proyecto:

Contiene las variables de entorno

Paso 2: Agregar .env a .gitignore

Para evitar que se suba al repositorio:

.env.production



Esto garantiza que el archivo .env no se suba al repositorio. Todo está listo para que las variables de entorno permanezcan locales y seguras.

Paso 3: Modificar docker-compose.yml para Cargar .env

```
services:  
  backend:  
    env_file:  
      - .env  
  db:  
    env_file:  
      - .env
```

Paso 4: Usar Volúmenes para Montar .env

```
services:  
  backend:  
    volumes:  
      - ./envs/backend.env:/app/.env:ro # Solo lectura  
  db:  
    volumes:  
      - ./envs/db.env:/app/.env:ro # Solo lectura
```

Uso de Docker Secrets en Producción con Docker Swarm

En producción, Docker Secrets proporciona una forma segura de almacenar y gestionar credenciales sin exponerlas en variables de entorno.

Paso 1: Inicializar Docker Swarm

docker swarm init



```
Terminal https://aka.ms/PowerShell-Release?tag=v7.5.0  
PS C:\Users\UsuarioPrueba> docker swarm init  
Swarm initialized: current node (4dj84979t5ik0nxxp9gtlfsn) is now a manager.  
To add a worker to this swarm, run the following command:  
  docker swarm join --token SWMTK-1-5odxzjk2716torwo41sqp22wdhkqqcysrw1cyi45n8mqcp3gb-yau4v5834fm4ammabndpcal20 192.168.65.3:2377  
To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.  
PS C:\Users\UsuarioPrueba>  
RAM 1.50 GB CPU 0.25% Disk: 9.05 GB used (limit 1006.85 GB)
```

Acabas de inicializar Docker Swarm exitosamente. Ahora tienes un nodo mánager en tu máquina, y si más adelante deseas añadir más nodos (trabajadores o incluso otro manager), puedes hacerlo utilizando los comandos que te mostró Docker al final del mensaje.

Paso 2: Crear Secretos en Docker Swarm

```
echo "litclub_jwt_secret_value" | docker secret create litclub_jwt_secret -
echo "litclub_db_password_value" | docker secret create litclub_db_password -
```

```
PS C:\Users\UsuarioPrueba> echo "litclub_jwt_secret_value" | docker secret create litclub_jwt_secret -
4ts3ylzhrvyj97bos9wg6flvw
PS C:\Users\UsuarioPrueba> echo "litclub_db_password_value" | docker secret create litclub_db_password -
vhurp8qj5zafoedif3gj6kx6
PS C:\Users\UsuarioPrueba>
```

RAM 1.52 GB CPU 1.37% Disk: 9.05 GB used (limit 1006.85 GB) > Terminal New version available

Resumen: Estos dos comandos sirven para crear secretos en Docker Swarm para almacenar de manera segura las credenciales necesarias para tu aplicación (jwt_secret y db_password). Los secretos se gestionan de manera segura, sin exponerlos en las variables de entorno o archivos de configuración. Posteriormente, estos secretos pueden ser utilizados dentro de los contenedores a través de la opción secrets en tu archivo docker-compose.yml para garantizar que tu aplicación en producción tenga acceso a los valores sensibles de manera segura.

```
PS C:\Users\UsuarioPrueba> docker secret ls
ID           NAME      DRIVER    CREATED          UPDATED
vhurp8qj5zafoedif3gj6kx6  litclub_db_password
4ts3ylzhrvyj97bos9wg6flvw  litclub_jwt_secret
PS C:\Users\UsuarioPrueba>
```

RAM 1.53 GB CPU 0.38% Disk: 9.05 GB used (limit 1006.85 GB) > Terminal New version available

Esto mostrará los secretos que están disponibles en tu Docker Swarm.

Paso 3: Modificar docker-stack.yml para Usar Secretos

```
version: '3.8'

services:
  litclub-backend-1:
    image: your-backend-image
    secrets:
      - jwt_secret
      - db_password
    environment:
      JWT_SECRET_FILE: /run/secrets/jwt_secret
      DB_PASSWORD_FILE: /run/secrets/db_password
    ports:
      - "5001:5001" # Puerto para el backend
    command: npm run start

  litclub-bd-1:
    image: postgres:latest
    secrets:
      - db_password
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD_FILE: /run/secrets/db_password
      POSTGRES_DB: litclub
    ports:
      - "5432:5432" # Puerto para la base de datos

  litclub-frontend-1:
    image: your-frontend-image
    ports:
      - "5173:5173" # Puerto para el frontend
    depends_on:
      - litclub-backend-1

secrets:
  jwt_secret:
    external: true
  db_password:
    external: true
```

El archivo docker-stack.yml es un archivo de configuración utilizado en Docker Swarm para definir y desplegar una pila de servicios en un clúster de Docker. Su función principal es especificar cómo se deben ejecutar los servicios, redes, volúmenes y secretos dentro del entorno Swarm.

Paso 4: Acceder a los Secretos en la Aplicación

En lugar de leer variables de entorno directamente, la aplicación debe leerlas desde los archivos de Docker Secrets:

Se creo el archivo server.docker.js en el backend para modo de produccion

```

LitClub > backend > server.docker.js > ...
1 const fs = require("fs");
2 const express = require("express");
3 const cors = require("cors");
4 const bcrypt = require("bcryptjs");
5 const jwt = require("jsonwebtoken");
6 const { Pool } = require("pg");
7
8 const app = express();
9 app.use(express.json());
10 app.use(cors({
11   origin: "http://localhost:5173",
12   methods: "GET,POST,PUT,DELETE",
13   allowedHeaders: "Content-Type,Authorization"
14 }));
15
16 // Función para leer secretos de Docker o variables de entorno en desarrollo
17 const readSecret = (path, envVar) => {
18   try {
19     return fs.readFileSync(path, "utf8").trim();
20   } catch (err) {
21     return process.env[envVar] || null;
22   }
23 };
24
25 // Leer secretos de Docker Secrets
26 const DB_PASSWORD = readSecret("/run/secrets/litclub_db_password_value", "DB_PASSWORD");
27 const JWT_SECRET = readSecret("/run/secrets/litclub_jwt_secret_value", "JWT_SECRET");
28
29 if (!DB_PASSWORD || !JWT_SECRET) {
30   console.warn("⚠️ ADVERTENCIA: No se encontraron credenciales en Docker Secrets, usando variables de entorno.");
31 }
32
33 // Configuración de PostgreSQL en Swarm
34 const pool = new Pool({
35   user: "postgres",
36   host: "db", // En Swarm, usa el nombre del servicio en lugar de "localhost"
37   database: "litclub",
38   password: DB_PASSWORD,
39   port: 5432,
40 });
41
42 // Rutas de autenticación
43 app.post("/register", async (req, res) => {
44   const { nombre, email, contraseña, rol } = req.body;
45   try {
46     const hashedPassword = await bcrypt.hash(contraseña, 10);

```

Lín. 80, col. 1 Espacios: 2 UTF-8 CRLF ⓘ JavaScript ⓘ Go Live ⓘ Quokka ⚡ Ninja ⓘ tabnine basic ⚡ ⓘ Prettier ⓘ

Paso 5: Use el comando docker info

docker info

```

PS C:\Users\UsuarioPrueba> docker info
Client:
  Version: 27.4.0
  Context: desktop-linux
  Debug Mode: false
  Plugins:
    at: Ask Gordon - Docker Agent (Docker Inc.)
    Version: v0.5.1
    Path: C:\Program Files\Docker\cli-plugins\docker-ai.exe
  buildx: Docker Buildx (Docker Inc.)
    Version: v0.19.2-desktop.1
    Path: C:\Program Files\Docker\cli-plugins\docker-buildx.exe

```

El comando docker info muestra una gran cantidad de información detallada sobre la instalación de Docker y su estado actual. Esta información es útil para diagnosticar problemas o entender cómo está configurado tu entorno Docker.

Paso 6: Use el comando

`docker stack deploy -c docker-stack.yml litclub`

```

PS C:\Users\UsuarioPrueba\LitClub> docker stack deploy -c docker-stack.yml litclub
Ignoring unsupported options: restart
Since --detach=false was not specified, tasks will be created in the background.
In a future release, --detach=false will become the default.
Creating service litclub_litclub-bd-1
Creating service litclub_litclub-frontend-1
Creating service litclub_litclub-backend-1
PS C:\Users\UsuarioPrueba\LitClub>

```

Los servicios backend, frontend y base de datos fueron creados correctamente en Docker Swarm. Ahora, los contenedores correspondientes deberían estar en ejecución en tu clúster. Puedes comprobar el estado con el comando docker service ls para ver los servicios activos y asegurarte de que todo esté funcionando.

```

PS C:\Users\UsuarioPrueba\LitClub> docker service ls
ID          NAME      MODE      REPLICAS  IMAGE
ppwte3vmqbqq  litclub_litclub-backend-1  replicated  0/1      your-backend-image:latest *:5001->5001/tcp
q4ktjnyuaawe  litclub_litclub-bd-1     replicated  1/1      postgres:latest      *:5432->5432/tcp
5682hv1zs4dy  litclub_litclub-frontend-1  replicated  0/1      your-frontend-image:latest *:5173->5173/tcp
PS C:\Users\UsuarioPrueba\LitClub>

```

Paso 7: Comandos para construir las imágenes de docker

```
docker build -f LitClub/backend/dockerfile.swarm -t litclub-backend .
```

El comando construye una imagen de Docker para tu backend usando un Dockerfile ubicado en LitClub/backend/dockerfile.swarm, y asigna el nombre litclub-backend a la imagen. El contexto de la construcción es el directorio actual, lo que significa que Docker podrá acceder a los archivos en esa carpeta para incluirlos en la imagen.

```
docker build -f LitClub/frontend/dockerfile.swarm -t litclub-frontend .
```

Paso 8: Si solo quieres detener litclub-litclub-frontend sin eliminar el stack, usa

```
docker service scale litclub_litclub-frontend-1=0
```

Para volver a activarlo, solo escala de nuevo:

```
docker service scale litclub_litclub-frontend-1=1
```

Lo mismo para backend sin eliminar el stack,

```
docker service scale litclub_litclub-backend-1=0
```

Para Volver a activarlo, solo escala de nuevo:



```
PS C:\Users\UsuarioPrueba> docker service scale litclub_litclub-backend-1=1
litclub_litclub-backend-1 scaled to 1
overall progress: 1 out of 1 tasks
1/1: running  [=====>]
verify: Service litclub_litclub-backend-1 converged
RAM 1.77 GB CPU 4.88% Disk: 11.15 GB used (limit 1006.85 GB)
```

```
docker service scale litclub_litclub-backend-1=1
```

Lo mismo para base de datos sin eliminar el stack

```
docker service scale litclub_litclub-bd-1=0
```

Para Volver a activarlo, solo escala de nuevo:

```
docker service scale litclub_litclub-bd-1=1
```

Paso 9: Use el comando docker service update –image litclub-backend litclub_litclub-backend-1

El comando ha actualizado exitosamente el servicio en Docker Swarm con la nueva imagen litclub-backend:latest, pero con la advertencia de que, si no tienes un registro centralizado para las imágenes, los nodos del clúster podrían estar utilizando versiones distintas de la imagen. Sin embargo, el servicio ha quedado funcionando correctamente después de la actualización.



```
PS C:\Users\UsuarioPrueba\LitClub> docker service logs litclub_litclub-backend-1
litclub_litclub-backend-1.1.55w7pd01f9yz@docker-desktop | 
litclub_litclub-backend-1.1.55w7pd01f9yz@docker-desktop | > backend@1.0.0 start
litclub_litclub-backend-1.1.55w7pd01f9yz@docker-desktop | > node server.js
litclub_litclub-backend-1.1.55w7pd01f9yz@docker-desktop | 
litclub_litclub-backend-1.1.55w7pd01f9yz@docker-desktop | Servidor corriendo en el puerto 5001
PS C:\Users\UsuarioPrueba\LitClub>
```

The terminal window shows the command `docker service logs litclub_litclub-backend-1` being run. The output displays log entries from three Docker containers. The first two lines show the command being run and the container ID. The third line shows the container starting the `backend@1.0.0` service, which runs the `node server.js` script. The final line indicates that the backend service is running on port 5001.

El comando `docker service logs litclub_litclub-backend-1` se utiliza para ver los logs de un servicio en ejecución en Docker Swarm. En este caso, el servicio `litclub_litclub-backend-1` es el backend de tu aplicación LitClub.

Descripción del resultado

> `backend@1.0.0 start`: Este es el script de inicio definido en el archivo `package.json` de tu backend. Está ejecutando el comando `node server.js`, que inicia tu servidor Node.js.

> `node server.js`: El comando ejecutado dentro del contenedor, que inicia el servidor backend utilizando el archivo `server.js`.

`Servidor corriendo en el puerto 5001`: El servidor backend se ha iniciado correctamente y está escuchando en el puerto 5001, lo que significa que el backend está disponible en ese puerto para manejar solicitudes.

Paso 10: En este proceso actualizará el servicio de frontend a la última versión de la imagen `litclub-frontend`, al igual que hiciste con el backend,

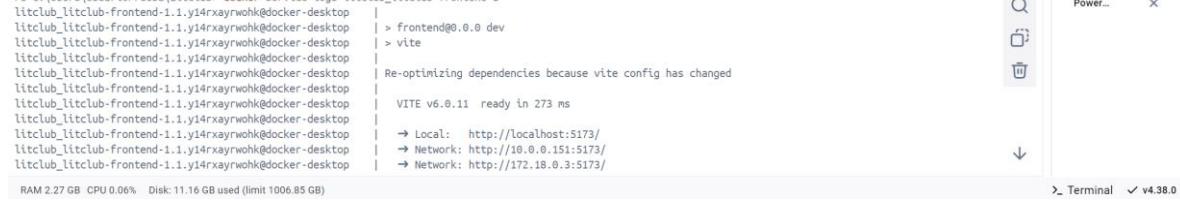
asegurando que los contenedores usen la versión más reciente de tu aplicación frontend.



```
PS C:\Users\UsuarioPrueba\LitClub> docker service create --name litclub_litclub-frontend-1 --replicas 1 -p 5173:5173 litclub-frontend:latest
image litclub-frontend:latest could not be accessed on a registry to record
its digest. Each node will access litclub-frontend:latest independently,
possibly leading to different nodes running different
versions of the image.

5b34zny10odkys7c7musbz3n7
overall progress: 1 out of 1 tasks

RAM 2.25 GB CPU 0.75% Disk: 11.16 GB used (limit 1006.85 GB)
```



```
PS C:\Users\UsuarioPrueba\LitClub> docker service logs litclub_litclub-frontend-1
litclub_litclub-frontend-1.1.y14rxayrwohk@docker-desktop | > frontend@0.0.0 dev
litclub_litclub-frontend-1.1.y14rxayrwohk@docker-desktop | > vite
litclub_litclub-frontend-1.1.y14rxayrwohk@docker-desktop | Re-optimizing dependencies because vite config has changed
litclub_litclub-frontend-1.1.y14rxayrwohk@docker-desktop |
litclub_litclub-frontend-1.1.y14rxayrwohk@docker-desktop | VITE v6.0.11 ready in 273 ms
litclub_litclub-frontend-1.1.y14rxayrwohk@docker-desktop | → Local: http://localhost:5173/
litclub_litclub-frontend-1.1.y14rxayrwohk@docker-desktop | → Network: http://10.0.0.151:5173/
litclub_litclub-frontend-1.1.y14rxayrwohk@docker-desktop | → Network: http://172.18.0.3:5173/

RAM 2.27 GB CPU 0.06% Disk: 11.16 GB used (limit 1006.85 GB)
```

El servicio `litclub_litclub-frontend-1` se ha creado correctamente y está en ejecución. La advertencia sobre la imagen `litclub-frontend:latest` sugiere que no está en un registro centralizado, por lo que cada nodo accederá a la imagen de forma independiente.

Ahora puedes verificar si tu servicio está funcionando correctamente en el puerto 5173. Intenta abrir un navegador y accede a `http://localhost:5173` o `http://<tu-ip-local>:5173` para verificar que la aplicación frontend se está ejecutando.

Paso 11: Verificación de la Base de Datos en Docker

Descripción del Estado Actual de la Base de Datos:

Tabla usuarios: La tabla usuarios en tu base de datos PostgreSQL tiene las siguientes columnas:

id: Identificador único para cada usuario (tipo SERIAL).

nombre: Nombre del usuario (hasta 100 caracteres).

email: Correo electrónico del usuario (hasta 150 caracteres), debe ser único.

rol: Rol del usuario (hasta 50 caracteres).

fecha_ingreso: Marca de tiempo que registra la fecha y hora del ingreso (por defecto, se asigna la fecha y hora actuales).

contraseña: Contraseña del usuario (tipo TEXT).

Índices:

usuarios_pkey: Clave primaria para la columna id.

usuarios_email_key: Restricción de unicidad en el campo email para evitar registros duplicados.

Explicación del Comando docker service logs:

El comando docker service logs se utiliza para obtener los registros (logs) de un servicio ejecutándose en un clúster de Docker Swarm. En tu caso, el servicio especificado es litclub_litclub-bd-1, que corresponde a tu base de datos PostgreSQL.

Qué hace el comando:

Mostrar los registros del servicio: Proporciona información sobre el estado y eventos importantes relacionados con el servicio. Por ejemplo, cuándo la base de datos está lista para aceptar conexiones o si hay problemas durante su ejecución.

Logs de los contenedores: Si el servicio tiene múltiples contenedores (como es común en Docker Swarm), el comando mostrará los registros de todos esos contenedores que forman parte del servicio.

```

PS C:\Users\UsuarioPrueba\LitClub> docker service logs litclub_litclub-bd-1
litclub_litclub-bd-1.1.zbu0c5h9grp@docker-desktop | PostgreSQL Database directory appears to contain a database; Skipping initialization
litclub_litclub-bd-1.1.zbu0c5h9grp@docker-desktop | 2025-02-09 23:26:30.991 UTC [1] LOG:  starting PostgreSQL 17.2 (Debian 17.2-1.pgdg120+1) on x86_64-pc-linux-gnu, compiled by gcc (Debian 12.2.0-14) 12.2.0, 64-bit
litclub_litclub-bd-1.1.zbu0c5h9grp@docker-desktop | 2025-02-09 23:26:30.991 UTC [1] LOG:  listening on IPv4 address "0.0.0.0", port 5432
litclub_litclub-bd-1.1.zbu0c5h9grp@docker-desktop | 2025-02-09 23:26:30.991 UTC [1] LOG:  listening on IPv6 address "::", port 5432
litclub_litclub-bd-1.1.zbu0c5h9grp@docker-desktop | 2025-02-09 23:26:31.000 UTC [1] LOG:  listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
litclub_litclub-bd-1.1.zbu0c5h9grp@docker-desktop | 2025-02-09 23:26:31.031 UTC [1] LOG:  database system was shut down at 2025-02-09 07:13:34 UTC
litclub_litclub-bd-1.1.zbu0c5h9grp@docker-desktop | 2025-02-09 23:31:31.082 UTC [27] LOG:  database system is ready to accept connections
litclub_litclub-bd-1.1.zbu0c5h9grp@docker-desktop | 2025-02-09 23:31:31.082 UTC [27] LOG:  checkpoint starting: time

```

RAM 2.37 GB CPU 0.56% Disk: 11.16 GB used (limit 1006.85 GB)

Power... X

Terminal v4.38.0

Los registros muestran que tu base de datos PostgreSQL en el contenedor `litclub_litclub-bd-1` ha iniciado correctamente y está funcionando como se esperaba.

Inicialización de la base de datos: Se ha detectado que el directorio de la base de datos ya contiene una base de datos, por lo que se ha omitido la inicialización.

Inicio de PostgreSQL: La base de datos ha comenzado correctamente, con PostgreSQL 17.2, y está escuchando en el puerto 5432 tanto en IPv4 como en IPv6.

Conexiones: El sistema de base de datos está listo para aceptar conexiones.

Puntos de control (checkpoints): Los registros incluyen detalles sobre los puntos de control, donde se realizan operaciones de escritura para garantizar la consistencia y rendimiento de la base de datos.

El comando `\dt` en PostgreSQL muestra una lista de todas las tablas en la base de datos actual. Proporciona información sobre el esquema, el nombre de la tabla, el

```

litclub=# \dt
List of relations
 Schema | Name   | Type  | Owner
-----+-----+-----+-----
 public | usuarios | table | postgres
(1 row)

```

Power... X

tipo de objeto y el propietario de cada tabla. Es útil para ver qué tablas están presentes en la base de datos.

El comando `\d` usuarios en PostgreSQL muestra la estructura de la tabla usuarios, incluyendo las columnas y sus tipos de datos, así como los índices y restricciones asociadas. Es útil para ver detalles como claves primarias y únicas, además de los tipos de datos de las columnas.



```
litclub=# \d usuarios
              Table "public.usuarios"
   Column   |      Type       | Collation | Nullable | Default
-----+-----+-----+-----+-----+
  id    | integer        |           | not null | nextval('usuarios_id_seq'::regclass)
 nombre | character varying(100) |           |           |
 email  | character varying(100) |           |           |
 rol    | character varying(50) |           |           |
 fechaingreso | timestamp without time zone |           |           | CURRENT_TIMESTAMP
 password | character varying(255) |           |           |
```

Se ha insertado y verificado un usuario de prueba en la base de datos litclub con el rol "cliente". Sus datos incluyen nombre, email y fecha de ingreso generada automáticamente. Este usuario servirá para probar autenticación y permisos en LitClub.



```
litclub=# INSERT INTO usuarios (nombre, email, rol, password)
VALUES ('Usuario Prueba', 'prueba@ejemplo.com', 'cliente', 'password123');
INSERT 0 1
litclub=# SELECT * FROM usuarios WHERE email = 'prueba@ejemplo.com';
 id | nombre  |   email    | rol  | fechaingreso | password
----+-----+-----+-----+-----+-----+
 1 | Usuario Prueba | prueba@ejemplo.com | cliente | 2025-02-10 01:15:55.929624 | password123
(1 row)

litclub#
```

Paso 12: Comando Docker secret ls

El comando `docker secret ls` muestra una lista de los secretos almacenados en el entorno de Docker Swarm. Los secretos son datos sensibles que se utilizan en contenedores o servicios, como contraseñas o claves de acceso, y se gestionan

de manera segura dentro de Docker para evitar su exposición.

En la salida que compartes, vemos dos secretos:

`litclub_db_password`: Este es el secreto relacionado con la contraseña de la base de datos de la aplicación. Se utiliza para acceder a la base de datos de manera segura, sin exponer la contraseña en el código o archivos de configuración.

`litclub_jwt_secret`: Este secreto es una clave utilizada para firmar y verificar los tokens JWT (JSON Web Tokens) en la aplicación. Los tokens JWT son comunes en la autenticación de aplicaciones, y mantener segura la clave utilizada para firmarlos es esencial para garantizar la integridad de las sesiones de usuario.

Ambos secretos están gestionados por Docker Swarm y se utilizan en los servicios de la aplicación para asegurar el acceso a recursos críticos sin comprometer la seguridad de las credenciales.

Es importante destacar que estos secretos están protegidos por Docker y solo pueden ser accesibles por los servicios que los necesitan, evitando así que sean expuestos accidentalmente.

```
PS C:\Users\UsuarioPrueba> docker secret ls
ID           NAME      DRIVER    CREATED        UPDATED
vwhurp8qj5zafoedif3gj6kx6  litclub_db_password
4ts3ylzhrvyj97bbs9wg6flvw  litclub_jwt_secret
```

Paso 13: Comando Docker secret inspect litclub_jwt_secret

El comando `docker secret inspect litclub_jwt_secret` muestra detalles sobre el secreto `litclub_jwt_secret` que tienes en tu sistema Docker. La salida proporciona la siguiente información clave:

ID: Identificador único del secreto (4ts3ylzhrvyj97bbs9wg6flvw).

Version: Número de versión del secreto, en este caso, la versión 11.

CreatedAt y UpdatedAt: Fechas en que el secreto fue creado y actualizado (ambas a las 23:11:42 UTC del 8 de febrero de 2025).

Spec: Detalles sobre la especificación del secreto, como su nombre (litclub_jwt_secret), y las etiquetas asociadas (en este caso, no hay etiquetas).

Este comando no revela el valor del secreto, sino que muestra solo metadatos sobre el mismo. El secreto litclub_jwt_secret probablemente esté siendo utilizado para gestionar información sensible (como una clave JWT) en tu entorno de



```
PS C:\Users\UsuarioPrueba> docker secret inspect litclub_jwt_secret
[{"ID": "4ts3ylzhrvyj97bbs9wg6flvw", "Version": {"Index": 11}, "CreatedAt": "2025-02-08T23:11:42.410926941Z", "UpdatedAt": "2025-02-08T23:11:42.410926941Z", "Spec": {"Name": "litclub_jwt_secret"}, "Labels": []}]
```

RAM 1.78 GB CPU 3.95% Disk: 11.15 GB used (limit 1006.85 GB) Terminal v4.38.0

Docker Swarm.

Paso 14: Comando docker secret inspect litclub_db_password

El comando docker secret inspect litclub_db_password muestra la información sobre el secreto llamado litclub_db_password en Docker, que contiene

credenciales sensibles, como la contraseña de la base de datos. Esta información es esencial para garantizar que las credenciales se mantengan seguras y no sean expuestas en el código o la configuración del contenedor.

En resumen, lo que se muestra con este comando es:

ID: Identificador único del secreto.

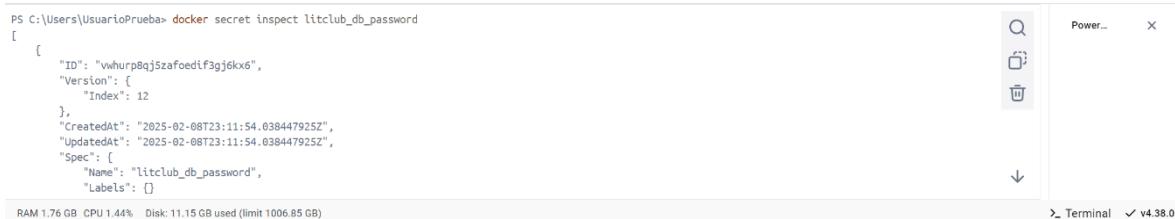
Version: La versión actual del secreto.

CreatedAt: Fecha y hora de creación del secreto.

UpdatedAt: Fecha y hora de la última actualización del secreto.

Spec: Contiene el nombre del secreto (litclub_db_password) y cualquier etiqueta asociada.

Este secreto se usa típicamente en entornos de producción dentro de servicios Docker, garantizando que las contraseñas y otras credenciales se gestionen de forma segura, sin exponerlas directamente en el código.



```
PS C:\Users\UsuarioPrueba> docker secret inspect litclub_db_password
[{"ID": "vwhurp8qj5zaf0edif3gj6kx6", "Version": {"Index": 12}, "CreatedAt": "2025-02-08T23:11:54.038447925Z", "UpdatedAt": "2025-02-08T23:11:54.038447925Z", "Spec": {"Name": "litclub_db_password", "Labels": {}}}
```

The screenshot shows a terminal window with the command `docker secret inspect litclub_db_password` and its output. The output is a JSON array containing one object. The object has fields: ID, Version, CreatedAt, UpdatedAt, and Spec. The ID is `vwhurp8qj5zaf0edif3gj6kx6`. The Version field contains an index of 12. The CreatedAt and UpdatedAt fields show the same timestamp: `2025-02-08T23:11:54.038447925Z`. The Spec field contains a Name of `litclub_db_password` and an empty Labels array. The terminal also shows system status at the bottom: RAM 1.76 GB, CPU 1.44%, Disk: 11.15 GB used (limit 1006.85 GB).

Conclusión

Al aplicar estas mejores prácticas, se asegura que las credenciales y configuraciones sensibles estén adecuadamente protegidas tanto en entornos de desarrollo como de producción. Utilizar archivos .env en el entorno de desarrollo y Docker Secrets en producción facilita una gestión segura y eficiente de los secretos, evitando la exposición accidental de información confidencial y garantizando la integridad de las aplicaciones desplegadas en contenedores.

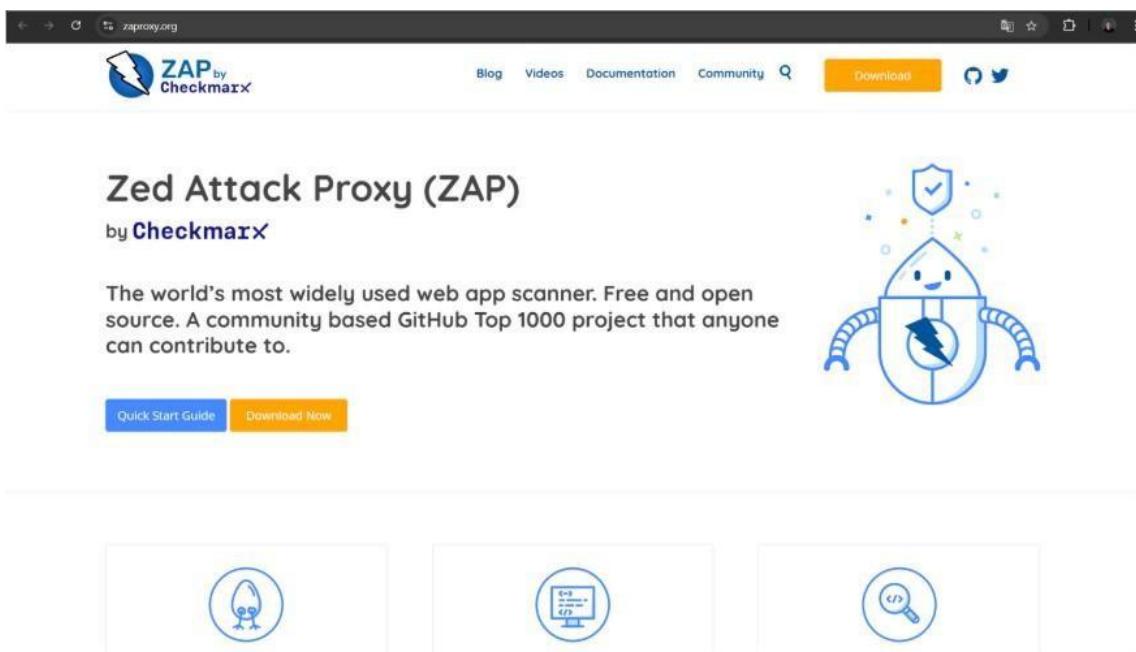
Docker.

Pruebas y Seguridad

OWASP ZAP (versión 2.13.0)

Descarga desde <https://www.zaproxy.org/>

Sigue el asistente para instalar y abre la herramienta.



Postman (versión 10.x)

Descarga desde Postman.

Instala y usa colecciones para organizar las pruebas de tus APIs.

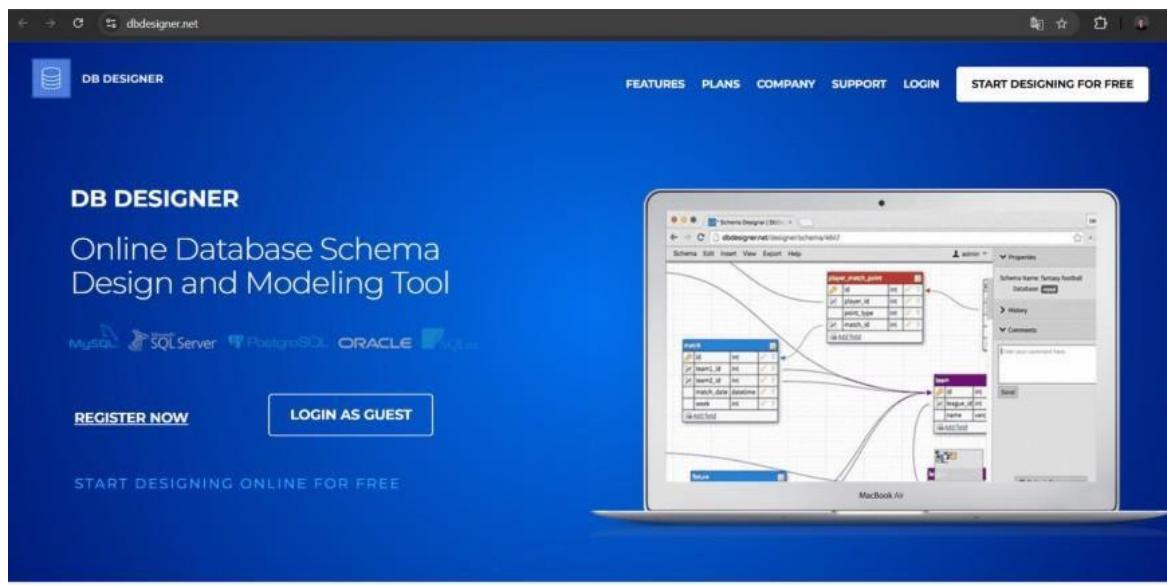
The screenshot shows the Postman website homepage. At the top, there's a banner with the text "The future of AI is Agentic. Discover how APIs are powering the next wave of AI innovation in our latest blog. [Read Now →](#)". Below the banner, the navigation bar includes links for Product, Pricing, Enterprise, Resources, Public API Network, Contact Sales, Sign In, and Sign Up for Free. The main headline reads "AI is powered by APIs. APIs are powered by Postman." Below the headline, a subtext states: "Postman is your single platform for collaborative API development. Join 35+ million devs building great APIs together, across the entire API lifecycle." There are two buttons: "name@company.com" and "Sign Up for Free". To the right, there's a large graphic of three 3D cubes stacked, with the letters "API" on them. Below the cubes, there's a section titled "Speed up API development through team collaboration" with a "Manage Cookies" button, "Reject All Cookies", and "Accept All Cookies" button. A cookie consent banner is also visible.

The screenshot shows the Draw.io diagram editor interface. The title bar says "Diagrama sin título.drawio". The menu bar includes Archivo, Editar, Vista, Organizar, Extras, Ayuda. The toolbar has various drawing tools like rectangles, circles, arrows, etc. On the left, there are category lists for General, Miscelánea, Avanzado, Básico, Flechas, Diagrama de flujo, and a "Más formas" button. The main workspace is a grid. On the right, there are several panels: "Diagrama" (with "Vista" and "Estilo" tabs), "Opciones" (with checkboxes for Gridlines, View page style, Rellenos, Colores de fondo, Sombras, Flechas de conexión, Puntos de conexión, Guias, and Guardar automático), "Tamaño del papel" (set to A4 (210 mm x 297 mm) with Orientation vertical selected), and "Editar datos" and "Quitar estilo predeterminado" buttons.

DBDesigner

Accede a DB Designer.

Diseña la estructura de tu base de datos de manera visual.



GESTION

Reporte de Avance - Desarrollo del Proyecto de la Biblioteca

Estado de la Base de Datos:

La base de datos está diseñada y configurada en PostgreSQL. El esquema principal está compuesto por las siguientes tablas:

usuario: Almacena información de los usuarios del sistema, incluyendo nombre de usuario, contraseña, correo, rol y estado.

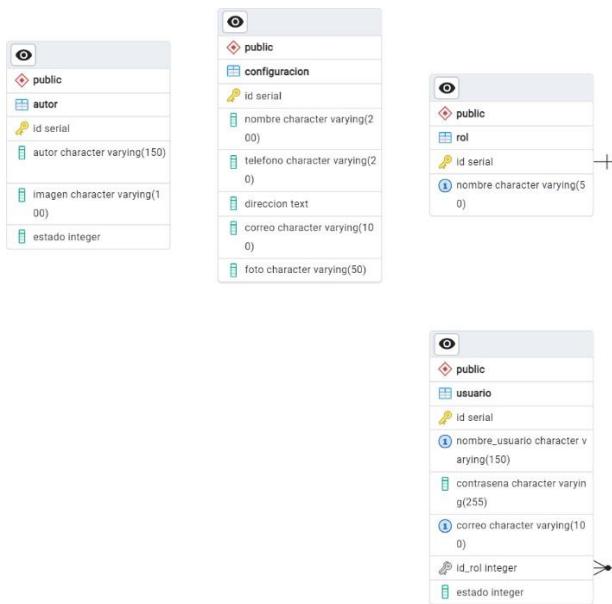
rol: Define los diferentes roles que pueden tener los usuarios dentro del sistema.

autor: Contiene información sobre los autores, incluyendo nombre, imagen y estado.

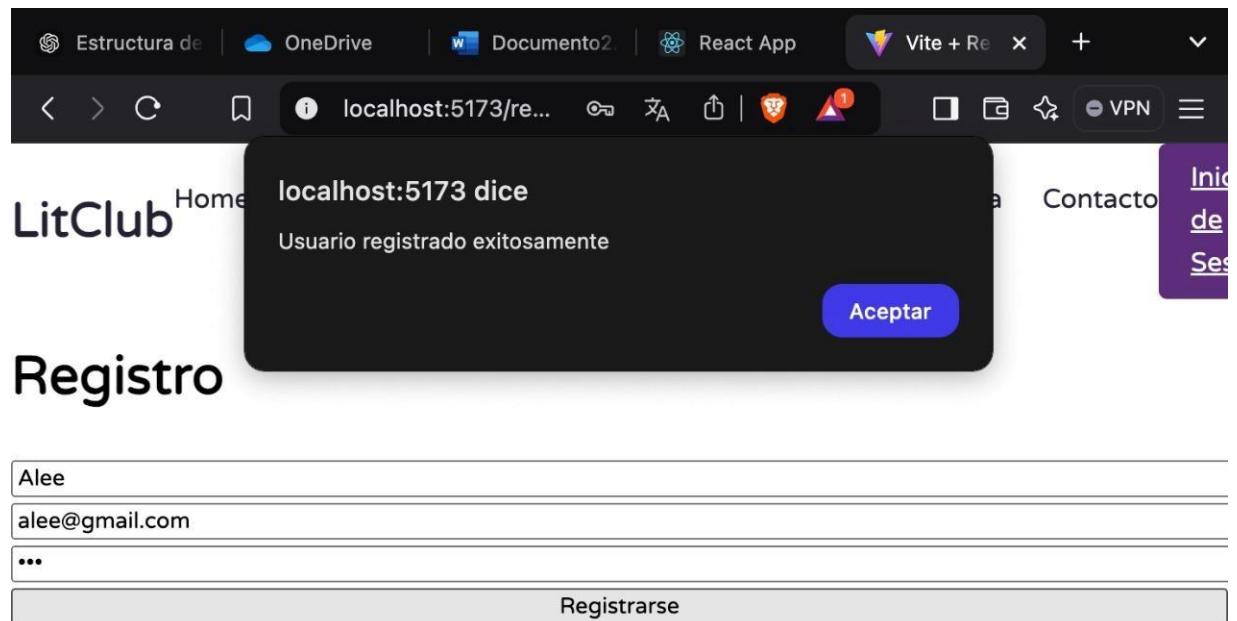
configuracion: Gestiona configuraciones generales del sistema, como nombre, teléfono, dirección, correo y foto.

Cada usuario tiene asignado un rol, estableciendo así la relación entre las tablas usuario y rol.

ESQUEMA DE LA BASE DE DATOS



DISEÑODELOGIN



The screenshot shows a web browser window with multiple tabs open at the top, including 'Estructura de sitio web', 'OneDrive', 'Documento2.docx', 'React App', 'Comenzando | Vite', and 'Vite + React + TS'. The main content area displays the LitClub homepage. The header features the 'LitClub' logo and a navigation menu with links to 'Catálogo', 'Comunidad', 'Ofertas', 'Sobre Nosotros', 'Ayuda', 'Contacto', 'Inicio de Sesión' (which is highlighted in purple), and 'Registro'. The main section contains the text: '¡Descubre el próximo capítulo de tu vida con nosotros!' followed by a small book icon. Below this, a subtext reads: 'Explora nuevas historias, aprende y crece con nuestra increíble colección de libros.' A purple button labeled 'Explorar ahora' is present. To the right, there is a cartoon illustration of an astronaut in a purple suit reading a purple book titled 'Math'. A stack of books is visible next to the astronaut's feet.

Pantalla del Home

The screenshot shows a dark-themed web application interface. At the top, there's a header bar with several tabs: "Estructura de sitio web", "OneDrive", "Documento2.docx", "React App", "Comenzando | Vite", and "Vite + React + TS". Below the header, the main content area has a title "Catálogo de Libros". It displays two book entries side-by-side.

El Principito
Un clásico lleno de enseñanzas sobre la vida, la amistad y el amor.

Cien Años de Soledad
Una obra maestra de Gabriel García Márquez sobre el realismo mágico.

Comentarios de la Comunidad



Aquí podemos observar el catálogo de libros de la Aplicación LIT CLUB