Documentación del Proyecto PeriConecta

Este documento proporciona una guía completa para la instalación, configuración y ejecución de los proyectos backend y frontend de PeriConecta.

1. Descripción General

PeriConecta es una aplicación de red social que consta de dos componentes principales:

- **`periconecta-api` (Backend):** Una API RESTful construida con NestJS que gestiona usuarios, autenticación, publicaciones y likes. Utiliza PostgreSQL como base de datos.
- **`perisocial-front` (Frontend):** Una aplicación de una sola página (SPA) desarrollada con React y Vite, que consume la API del backend para proporcionar la interfaz de usuario.

2. Prerrequisitos

Antes de comenzar, asegúrate de tener instalado el siguiente software en tu sistema:

- **Node.js:** Versión 18 o superior.
- **Docker y Docker Compose:** Para ejecutar la base de datos PostgreSQL de forma aislada.
- **npm:** Generalmente se instala junto con Node.js.

---

3. Backend (`periconecta-api`)

3.1. Tecnologías Principales

- **Framework:** NestJS
- **Base de Datos:** PostgreSQL
- **ORM:** TypeORM
- **Autenticación:** JWT (JSON Web Tokens)
- **Lenguaje:** TypeScript

3.2. Instalación y Configuración

1. **Navega al directorio del proyecto:**

```
cd periconecta-api
```

2. **Instala las dependencias:**

```
npm install
```

3. **Configura las variables de entorno:** Crea un archivo `.env` en la raíz del directorio `periconecta-api` y añade las siguientes variables. Puedes usar los valores por defecto para un entorno de desarrollo local.

```
# Configuración de la Base de Datos
DB_HOST=localhost
DB_PORT=5432
DB_USERNAME=admin
DB_PASSWORD=admin123
DB_NAME=periconecta_db

# Configuración de JWT
JWT_SECRET=your-super-secret-jwt-key
JWT_EXPIRES_IN=24h

# Configuración de la Aplicación
PORT=3000
NODE_ENV=development
```

4. **Inicia la base de datos con Docker:** El proyecto incluye un archivo `docker-compose.yml` para levantar fácilmente un contenedor de PostgreSQL y PgAdmin.

```
docker-compose up -d
```

   - La base de datos PostgreSQL estará disponible en `localhost:5432`.
   - PgAdmin (herramienta de gestión de DB) estará en `http://localhost:8080`.

3.3. Ejecución de la Aplicación

Una vez completada la instalación y configuración, puedes iniciar la API en modo de desarrollo:

```
npm run start:dev
```

La API se ejecutará en `http://localhost:3000`. Al iniciarse, ejecutará un "seeder" para poblar la base de datos con usuarios y publicaciones de prueba.

La documentación de la API (Swagger) estará disponible en `http://localhost:3000/api`.

---

4. Frontend (`perisocial-front`)

4.1. Tecnologías Principales

- **Framework/Librería:** React 18 + Vite
- **Lenguaje:** TypeScript
- **Gestión de Estado:** Zustand
- **Estilos:** Tailwind CSS
- **Componentes UI:** ShadCN UI
- **Cliente HTTP:** Axios

4.2. Instalación

1. **Navega al directorio del proyecto:**

```
cd perisocial-front
```

2. **Instala las dependencias:**

```
npm install
```

4.3. Ejecución de la Aplicación

Para iniciar el servidor de desarrollo de Vite, ejecuta:

```
npm run dev
```

La aplicación frontend estará disponible en `http://localhost:5173`. Se conectará automáticamente a la API del backend que se ejecuta en `http://localhost:3000`.

---

5. Ejecución del Proyecto Completo

Para tener la aplicación funcionando en su totalidad, sigue estos pasos:

1. **Inicia el Backend:**
    - Abre una terminal en `periconecta-api`.
    - Ejecuta `docker-compose up -d` para la base de datos.

- Ejecuta `npm run start:dev` para iniciar la API.

2. **Inicia el Frontend:**

   - Abre una segunda terminal en `perisocial-front`.
   - Ejecuta `npm run dev`.

Ahora puedes acceder a la aplicación en `http://localhost:5173` y utilizar todas sus funcionalidades.

---

6. Explicación de los Servicios

6.1. Servicios del Backend (`periconecta-api`)

Los servicios en NestJS contienen la lógica de negocio principal. Se encargan de procesar los datos, interactuar con la base de datos y responder a las solicitudes de los controladores.

- **`auth/auth.service.ts`**
  - **Responsabilidad:** Gestionar toda la lógica de autenticación y registro de usuarios.
  - **Métodos clave:** `register`, `login`, `validateUser`.
- **`users/users.service.ts`**
  - **Responsabilidad:** Manejar la lógica para obtener información de los usuarios.
  - **Métodos clave:** `findById`, `findByAlias`, `getProfile`.
- **`posts/posts.service.ts`**
  - **Responsabilidad:** Gestionar todo lo relacionado con las publicaciones (posts) y los "me gusta" (likes).
  - **Métodos clave:** `create`, `findAll`, `findMyPosts`, `findOne`, `likePost`, `unlikePost`.
- **`posts/posts.gateway.ts`**
  - **Responsabilidad:** Manejar conexiones en tiempo real con WebSockets para notificar a los clientes sobre eventos de likes de forma instantánea.
  - **Métodos clave:** `emitLikeAdded`, `emitLikeRemoved`, `emitLikeCountUpdate`.

6.2. Servicios del Frontend (`perisocial-front`)

En el frontend, los servicios son módulos que encapsulan la comunicación con la API del backend, abstrayendo las llamadas HTTP.

- **`services/api.ts`**
  - **Responsabilidad:** Es la base de la capa de servicios. Crea y configura una instancia de **Axios**.

    - **Funcionalidad:** Establece la URL base de la API y utiliza **interceptores** para adjuntar automáticamente el token JWT a las peticiones y para manejar errores de autenticación (401) de forma global.

- **`services/auth.ts`**
    - **Responsabilidad:** Centraliza todas las llamadas a la API relacionadas con la autenticación.
    - **Métodos clave:** `login`, `register`, `getProfile`, y utilidades para manejar el `localStorage` (`storeAuth`, `logout`, etc.).

- **`services/posts.ts`**
    - **Responsabilidad:** Encapsula todas las llamadas a la API para la gestión de publicaciones.
    - **Métodos clave:** `getAllPosts`, `getMyPosts`, `createPost`, `likePost`, `unlikePost`.

- **`services/users.ts`**
    - **Responsabilidad:** Maneja las llamadas a la API para obtener datos de otros usuarios.
    - **Métodos clave:** `getUserById`, `getUserByAlias`.