Periferia Social

Aplicación de red social interna donde los colaboradores pueden iniciar sesión, revisar su perfil, publicar mensajes y reaccionar con "likes". La solución está compuesta por un backend unificado en Nest.js + Prisma y un frontend en React + Vite.

Arquitectura

- `backend` : API REST en Nest.js (TypeScript) con Prisma ORM y JWT para autenticación.
- `frontend` : SPA en React + TypeScript con Zustand para manejo de estado global.
- `postgres` : Base de datos PostgreSQL gestionada mediante migraciones Prisma.

Prerrequisitos

- Node.js 20 o superior
- npm 10 o superior
- Docker y Docker Compose
- Acceso a PowerShell (Windows) o Bash (Linux/Mac)

Instalación local

1. Clona el repositorio y entra en la carpeta del proyecto:

   ```
   git clone https://github.com/markus993/pruebaFullstackPeriferia.git
   cd pruebaFullstackPeriferia
   ```

2. Duplica el archivo `env.example` y renombra la copia a `.env` .

3. Revisa y ajusta las variables de entorno críticas (ver sección siguiente).

Variables de entorno

Las variables compartidas residen en `.env` ; `env.example` es la plantilla base. Asegúrate de validar estos valores clave antes de ejecutar cualquier servicio:

Variable	Descripción
`POSTGRES_USER`, `POSTGRES_PASSWORD`, `POSTGRES_HOST`, `POSTGRES_PORT`	Configuración de PostgreSQL
`POSTGRES_DB`	Nombre de la base de datos principal (`periferia_social`)
`DATABASE_URL`	Cadena de conexión usada por Prisma
`API_PORT`	Puerto expuesto del backend Nest (3000 por defecto)
`FRONTEND_PORT`	Puerto expuesto del frontend Vite (5173 por defecto)
`JWT_SECRET`	Clave simétrica para firmar tokens JWT

Para desarrollo del frontend fuera de Docker puedes exportar `VITE_API_URL` (por ejemplo `http://localhost:3000`).

Despliegue con Docker Compose

1. Personaliza las variables de `.env` si lo necesitas.
2. Construye y levanta la solución completa:

```
docker compose up --build
```

3. Espera a que el contenedor `api` muestre en los logs los mensajes “Executing seed” e “Iniciando servidor Nest”.
4. Accede a los servicios:
   - Frontend: `http://localhost:${FRONTEND_PORT}` (por defecto `http://localhost:5173`)
   - API: `http://localhost:${API_PORT}/api` (por defecto `http://localhost:3000/api`)

Usuarios de prueba

Alias	Usuario / Email	Contraseña
`@anar`	aromero / ana.romero@periferia.it	`Periferia123!`
`@carlitos`	cmendez / carlos.mendez@periferia.it	`Periferia123!`
`@lauca`	lcastillo / laura.castillo@periferia.it	`Periferia123!`

Pruebas unitarias

Backend (Jest)

Ejecución local

1. Instala dependencias (solo la primera vez):

```
cd backend
npm install
```

2. Ejecuta la suite:

```
npm run test
```

3. Genera cobertura:

```
npm run test:cov
```

Ejecución con Docker Compose

1. Levanta los servicios si aún no lo hiciste:

```
docker compose up --build
```

2. Corre las pruebas dentro del contenedor:

```
docker compose exec api npm run test
```

3. Genera cobertura:

```
docker compose exec api npm run test:cov
```

Frontend (Vitest)

Ejecución local

1. Instala dependencias (solo la primera vez):

```
cd frontend
npm install
```

2. Ejecuta las pruebas (modo una sola corrida):

```
npm run test -- --run
```

3. Obtén cobertura:

```
npm run test:coverage
```

Ejecución con Docker Compose

1. Asegúrate de que el servicio frontend esté levantado:

```
docker compose up --build frontend -d
```

2. Corre las pruebas dentro del contenedor `frontend`:

```
docker compose exec frontend npm run test -- --run
```

3. Genera cobertura:

```
docker compose exec frontend npm run test:coverage
```

Endpoints relevantes

- `GET /api/health` – Health check del backend.
- `POST /api/auth/login` – Devuelve `{ token, user }` tras validar credenciales.
- `GET /api/users/me` – Retorna el perfil del usuario autenticado.

- `GET /api/posts` – Feed de publicaciones de otros usuarios (likes agregados).
- `POST /api/posts` – Crea una publicación propia.
- `POST /api/posts/:id/like` – Envía un like idempotente.

Las rutas protegidas requieren el encabezado `Authorization: Bearer <token>`.

Documentación interactiva de la API (Swagger)

- Swagger UI se expone en `http://localhost:${API_PORT}/api/docs` (por defecto `http://localhost:3000/api/docs`).
- Incluye esquema OpenAPI con autenticación Bearer; puedes probar endpoints e incluir tu token JWT desde la interfaz.
- Si ejecutas con Docker, levanta los servicios con `docker compose up --build` y luego abre la URL anterior en tu navegador.

Scripts útiles

Backend (`backend`)

- `npm run prisma:generate` – Generar cliente Prisma a partir del esquema.
- `npm run prisma:migrate:dev` / `npm run prisma:migrate:deploy` – Aplicar migraciones.
- `npm run prisma:seed` – Poblar datos iniciales.
- `npm run start:dev` – Modo desarrollo con recarga.
- `npm run build` / `npm run start:prod` – Compilar y ejecutar en producción.

Frontend (`frontend`)

- `npm run dev` – Levantar el frontend en modo desarrollo.
- `npm run build` – Generar la versión productiva.
- `npm run lint` – Ejecutar reglas de ESLint.
- `npm run test` – Ejecutar pruebas unitarias con Vitest.

Estructura del proyecto

```
├── backend/                # Backend unificado Nest.js + Prisma
│   ├── Dockerfile
│   ├── docker-entrypoint.sh
│   ├── prisma/         # schema.prisma, migraciones y seed.ts
│   └── src/            # Módulos Auth, Users, Posts y configuración
├── frontend/               # SPA React + Zustand + Vite
├── config/                 # Scripts de init DB (archivos .env viven en la raíz)
├── docs/                   # Manual en Markdown/PDF
├── docker-compose.yml
└── README.md
```

Documentación adicional

- `docs/manual_instalacion.md` : Guía editable con pasos de instalación.
- `docs/manual_instalacion.pdf` : Versión en PDF generada desde la guía.