

Backend CRUD - Gestión Empresarial

Backend completo con Node.js, TypeScript, Express, Prisma y PostgreSQL para gestionar contactos y empresas. Incluye autenticación JWT opcional, dockerización completa y datos de prueba precargados.

Autor

Isaías Fernández Lozano  [GitHub](#) [isaiasfl](#)

Módulo: Desarrollo Web Entorno Cliente (DWECE) **Curso:** 2025-2026

Tabla de Contenidos

- [Características](#)
 - [Stack Tecnológico](#)
 - [Requisitos Previos](#)
 - [Instalación y Configuración](#)
 - [Poblar la Base de Datos \(Seed\)](#)
 - [Uso](#)
 - [Estructura del Proyecto](#)
 - [Endpoints de la API](#)
 - [Autenticación JWT](#)
 - [Ejemplos de Uso](#)
 - [pgAdmin](#)
 - [Troubleshooting](#)
-

☆☆ Características

- ✓ **CRUD completo** para Contactos y Empresas
 - ✓ **Autenticación JWT** opcional (variable AUTH_REQUIRED)
 - ✓ **PostgreSQL** con Prisma ORM
 - ✓ **Dockerizado** (PostgreSQL + pgAdmin + Backend)
 - ✓ **TypeScript** moderno (enero 2026)
 - ✓ **Validaciones** con Zod
 - ✓ **Seed data** con empresas reales (Google, Microsoft, Amazon, Meta)
 - ✓ **CORS** configurado para desarrollo
 - ✓ **Hot reload** en desarrollo
 - ✓ **Documentación** completa de endpoints
-

Stack Tecnológico

Tecnología	Versión	Propósito
Node.js	20 LTS	Runtime de JavaScript
TypeScript	5.7+	Lenguaje con tipado estático
Express	5.x	Framework web
Prisma	6.x	ORM para PostgreSQL
PostgreSQL	16	Base de datos relacional
Zod	3.x	Validaciones de esquemas
bcrypt	5.x	Hash de contraseñas
JWT	9.x	Tokens de autenticación
Docker	-	Containerización
pgAdmin	4	Administrador visual de PostgreSQL

Requisitos Previos

Antes de comenzar, asegúrate de tener instalado:

- [Docker](#) (versión 20+)
- [Docker Compose](#) (versión 3.9+)
- [Node.js](#) 20 LTS (opcional, solo si ejecutas sin Docker)
- [npm](#) o [pnpm](#)

Instalación y Configuración

1. Clonar el repositorio

```
cd crud-gestion-empresarial-bd/backend
```

2. Configurar variables de entorno

El proyecto ya viene con un archivo `.env` preconfigurado. Si necesitas cambiarlo, edita `.env`:

```
# Database
POSTGRES_USER=admin
POSTGRES_PASSWORD=admin123
POSTGRES_DB=contacts_db
DATABASE_URL=postgresql://admin:admin123@postgres:5432/contacts_db
```

```
# Server
PORT=3000 # Puerto interno del contenedor (se mapea a 3001 externamente)
NODE_ENV=development

# Authentication (false = sin auth, true = con JWT)
AUTH_REQUIRED=false
JWT_SECRET=your-super-secret-jwt-key-change-in-production
JWT_EXPIRES_IN=24h

# pgAdmin
PGADMIN_EMAIL=admin@admin.com
PGADMIN_PASSWORD=Admin.123456
```

3. Levantar el proyecto con Docker

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

Este comando:

- ✓ Descarga las imágenes necesarias (PostgreSQL, pgAdmin, Node)
- ✓ Crea los contenedores
- ✓ Ejecuta las migraciones de Prisma
- ✓ Inicia el backend en <http://localhost:3001>

4. Verificar que todo funciona

```
# Ver logs del backend
docker-compose logs -f backend

# Ver estado de los contenedores
docker ps
```

Deberías ver 3 contenedores corriendo:

- `crud_postgres` (PostgreSQL)
- `crud_pgadmin` (pgAdmin)
- `crud_backend` (API)

🔗 Poblar la Base de Datos (Seed)

IMPORTANTE: El seed NO se ejecuta automáticamente al levantar los contenedores debido a problemas de compatibilidad de bcrypt con Alpine Linux. Debes ejecutarlo manualmente.

¿Qué datos incluye el seed?

El script de seed ([prisma/seed.ts](#)) inserta datos de prueba realistas:

Tabla	Cantidad	Descripción
Users	2 usuarios	Usuarios de prueba para autenticación
Companies	4 empresas	Google, Microsoft, Amazon, Meta
Contacts	10 contactos	8 contactos con empresa + 2 freelance

Credenciales de los usuarios de prueba:

Email	Password	Uso
demo@example.com	Demo123!	Usuario de demostración
test@example.com	Test123!	Usuario de testing

Método 1: Ejecutar seed desde el host (Recomendado) ☆

Requisitos: Tener Node.js instalado en tu máquina local.

```
# Asegúrate de estar en el directorio backend/  
cd crud-gestion-empresarial-bd/backend  
  
# Ejecutar el seed  
npm run prisma:seed
```

Verás una salida similar a:

```
🌀 Iniciando seed de la base de datos...  
✓ Datos anteriores eliminados  
✓ Usuarios creados: demo@example.com test@example.com  
✓ Empresas creadas: Google Microsoft Amazon Meta  
✓ Contactos creados (10 en total)  
🔮 Seed completado exitosamente!  
  
📊 Resumen:  
  - 2 usuarios de prueba  
  - 4 empresas  
  - 10 contactos (8 con empresa, 2 freelance)  
  
🔑 Credenciales de prueba:  
  Email: demo@example.com | Password: Demo123!  
  Email: test@example.com | Password: Test123!
```

Método 2: Ejecutar seed desde Docker

```
# Desde el directorio backend/  
docker-compose exec backend npx prisma db seed
```

Nota: Este método puede fallar debido a problemas de bcrypt con Alpine Linux. Si falla, usa el Método 1.

Método 3: Ejecutar el script TypeScript directamente

```
# Desde el directorio backend/  
npm run prisma:seed
```

Verificar que los datos fueron insertados

```
# Verificar empresas (debería devolver 4)  
curl http://localhost:3001/api/companies | jq '.total'  
  
# Verificar contactos (debería devolver 10)  
curl http://localhost:3001/api/contacts | jq '.total'  
  
# Ver los nombres de las empresas  
curl http://localhost:3001/api/companies | jq '.companies[].name'
```

Salida esperada:

```
"Meta"  
"Amazon"  
"Microsoft"  
"Google"
```

Re-poblar la base de datos (resetear datos)

Si necesitas volver a ejecutar el seed (por ejemplo, después de hacer pruebas):

```
# El seed automáticamente elimina los datos existentes antes de insertar  
nuevos  
npm run prisma:seed
```

El script hace limpieza automática:

1. Elimina todos los contactos
2. Elimina todas las empresas
3. Elimina todos los usuarios
4. Inserta los datos nuevos

Troubleshooting del Seed

Error: "Port is already allocated" o problemas con puerto 3000

- El backend se ejecuta en el puerto **3001** (no 3000)
- Verifica con: `docker ps | grep crud_backend`

Error: "bcrypt segmentation fault" en Docker

- Usa el Método 1 (desde el host) en lugar de dentro del contenedor
- Alpine Linux + bcrypt nativo tienen problemas de compatibilidad

Error: "Cannot find module tsx"

- Asegúrate de haber ejecutado `npm install` primero
- Verifica que estás en el directorio `backend/`

Los datos no aparecen después del seed

- Verifica que el backend está corriendo: `docker ps`
- Revisa los logs: `docker-compose logs backend`
- Prueba los endpoints directamente con curl

Uso

Servicios disponibles

Servicio	URL	Credenciales
Backend API	http://localhost:3001	-
pgAdmin	http://localhost:5050	Email: admin@admin.com Password: Admin.123456
PostgreSQL	localhost:5432	User: admin Password: admin123 DB: contacts_db

Probar la API

```
# Health check
curl http://localhost:3001/health

# Listar empresas
curl http://localhost:3001/api/companies

# Listar contactos
curl http://localhost:3001/api/contacts
```

Scripts útiles

```
# Levantar todos los servicios
npm run docker:up

# Detener todos los servicios
npm run docker:down

# Reconstruir contenedores
npm run docker:rebuild

# Ver logs del backend
npm run docker:logs

# Ejecutar migraciones de Prisma
npm run prisma:migrate

# Ejecutar seed (datos de prueba)
npm run prisma:seed

# Abrir Prisma Studio (GUI para la BD)
npm run prisma:studio
```

📁 Estructura del Proyecto

```
backend/
├── src/
│   ├── controllers/          # Lógica de negocio
│   │   ├── auth.controller.ts
│   │   ├── companies.controller.ts
│   │   └── contacts.controller.ts
│   ├── routes/              # Definición de rutas
│   │   ├── auth.routes.ts
│   │   └── companies.routes.ts
```

```
├── contacts.routes.ts
├── middleware/           # Middlewares personalizados
│   ├── auth.middleware.ts
│   └── errorHandler.middleware.ts
├── validators/          # Validaciones con Zod
│   ├── auth.validator.ts
│   ├── companies.validator.ts
│   └── contacts.validator.ts
├── utils/               # Utilidades
│   └── jwt.utils.ts
├── lib/                 # Clientes y configuraciones
│   └── prisma.ts
├── types/               # Tipos TypeScript
│   └── express.d.ts
├── app.ts               # Configuración de Express
└── server.ts            # Entry point

├── prisma/
│   ├── schema.prisma    # Modelos de datos
│   └── seed.ts          # Datos de prueba
├── docker-compose.yml    # Orquestación de servicios
├── Dockerfile            # Imagen del backend
├── package.json
├── tsconfig.json
├── .env.example
└── README.md
```

🔗 Endpoints de la API

Base URL

```
http://localhost:3001/api
```

Nota: El backend usa el puerto **3001** para evitar conflictos con otros servicios (como open-webui u otros backends que usan 3000).

🔒 Autenticación

Método	Endpoint	Descripción	Requiere Auth
POST	/auth/register	Registrar nuevo usuario	✗
POST	/auth/login	Iniciar sesión (obtener JWT)	✗
GET	/auth/me	Obtener usuario autenticado	✓

🏢 Empresas

Método	Endpoint	Descripción	Requiere Auth
GET	/companies	Listar todas las empresas	⚙️
GET	/companies/:id	Obtener empresa por ID	⚙️
POST	/companies	Crear nueva empresa	⚙️
PUT	/companies/:id	Actualizar empresa	⚙️
DELETE	/companies/:id	Eliminar empresa	⚙️
GET	/companies/:id/contacts	Contactos de una empresa	⚙️

👤 Contactos

Método	Endpoint	Descripción	Requiere Auth
GET	/contacts	Listar todos los contactos	⚙️
GET	/contacts/:id	Obtener contacto por ID	⚙️
POST	/contacts	Crear nuevo contacto	⚙️
PUT	/contacts/:id	Actualizar contacto	⚙️
DELETE	/contacts/:id	Eliminar contacto	⚙️

Leyenda:

- ✓ = Siempre requiere autenticación
- ✗ = No requiere autenticación
- ⚙️ = Depende de `AUTH_REQUIRED` (false = no requiere, true = requiere)

🔑 Autenticación JWT

Modo Híbrido (AUTH_REQUIRED)

Este backend tiene un sistema de autenticación **opcional** controlado por la variable `AUTH_REQUIRED`:

Modo 1: Sin autenticación (por defecto)

```
AUTH_REQUIRED=false
```

- Los endpoints de CRUD funcionan **sin token**
- Ideal para desarrollo y alumnos principiantes
- Pueden practicar CRUD sin implementar login

Modo 2: Con autenticación

```
AUTH_REQUIRED=true
```

- Los endpoints de CRUD **requieren token JWT**
- Los alumnos deben implementar login en el frontend
- Token se envía en header: `Authorization: Bearer <token>`

Usuarios de prueba

El seed incluye 2 usuarios para testing:

Email	Password
demo@example.com	Demo123!
test@example.com	Test123!

Ejemplos de Uso

1. Listar empresas (sin auth)

```
curl http://localhost:3001/api/companies
```

Respuesta:

```
{
  "companies": [
    {
      "id": 1,
      "name": "Google",
      "industry": "Technology",
      "website": "https://google.com",
      "createdAt": "2026-02-05T...",
      "updatedAt": "2026-02-05T..."
    }
  ]
}
```

```
    "_count": { "contacts": 2 }
  }
],
"total": 4
}
```

2. Crear un contacto (sin auth)

```
curl -X POST http://localhost:3001/api/contacts \
-H "Content-Type: application/json" \
-d '{
  "firstName": "Juan",
  "lastName": "Pérez",
  "email": "juan.perez@example.com",
  "phone": "+34-600-123-456",
  "position": "Developer",
  "companyId": 1,
  "linkedin": "https://linkedin.com/in/juan-perez"
}'
```

Respuesta:

```
{
  "message": "Contacto creado exitosamente",
  "contact": {
    "id": 11,
    "firstName": "Juan",
    "lastName": "Pérez",
    "email": "juan.perez@example.com",
    "phone": "+34-600-123-456",
    "position": "Developer",
    "companyId": 1,
    "linkedin": "https://linkedin.com/in/juan-perez",
    "notes": null,
    "createdAt": "2026-02-05T...",
    "updatedAt": "2026-02-05T...",
    "company": {
      "id": 1,
      "name": "Google",
      "industry": "Technology"
    }
  }
}
```

3. Registrar un usuario

```
curl -X POST http://localhost:3001/api/auth/register \  
-H "Content-Type: application/json" \  
-d '{  
  "email": "nuevo@example.com",  
  "password": "Password123!",  
  "name": "Usuario Nuevo"  
}'
```

Respuesta:

```
{  
  "message": "Usuario registrado exitosamente",  
  "user": {  
    "id": 3,  
    "email": "nuevo@example.com",  
    "name": "Usuario Nuevo",  
    "createdAt": "2026-02-05T..."  
  },  
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."  
}
```

4. Login

```
curl -X POST http://localhost:3001/api/auth/login \  
-H "Content-Type: application/json" \  
-d '{  
  "email": "demo@example.com",  
  "password": "Demo123!"  
}'
```

Respuesta:

```
{  
  "message": "Login exitoso",  
  "user": {  
    "id": 1,  
    "email": "demo@example.com",  
    "name": "Usuario Demo",  
    "createdAt": "2026-02-05T..."  
  },  
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."  
}
```

5. Usar un endpoint con token (AUTH_REQUIRED=true)

```
curl http://localhost:3001/api/contacts \
-H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
```

6. Actualizar una empresa

```
curl -X PUT http://localhost:3001/api/companies/1 \
-H "Content-Type: application/json" \
-d '{
  "industry": "Technology & AI",
  "website": "https://google.com"
}'
```

7. Eliminar un contacto

```
curl -X DELETE http://localhost:3001/api/contacts/11
```

8. Obtener contactos de una empresa

```
curl http://localhost:3001/api/companies/1/contacts
```

pgAdmin

Acceder a pgAdmin

1. Abrir <http://localhost:5050>
2. Login con:
 - Email: `admin@admin.com`
 - Password: `admin123`

Conectar a PostgreSQL

1. Click derecho en "Servers" → "Register" → "Server"
2. En la pestaña "General":
 - Name: `CRUD Database`
3. En la pestaña "Connection":
 - Host: `postgres`

- Port: 5432
- Maintenance database: contacts_db
- Username: admin
- Password: admin123

4. Click "Save"

Explorar las tablas

```
Servers → CRUD Database → Databases → contacts_db → Schemas → public → Tables
```

Verás 3 tablas:

- users (usuarios para autenticación)
- companies (empresas)
- contacts (contactos)

🔧 Troubleshooting

El backend no inicia

```
# Ver logs
docker-compose logs backend

# Reiniciar contenedor
docker-compose restart backend
```

Error de conexión a PostgreSQL

```
# Verificar que PostgreSQL está corriendo
docker-compose ps

# Reiniciar PostgreSQL
docker-compose restart postgres

# Esperar unos segundos y reiniciar backend
docker-compose restart backend
```

Las migraciones fallan

```
# Entrar al contenedor del backend
docker-compose exec backend sh

# Ejecutar migraciones manualmente
npx prisma migrate deploy

# Ejecutar seed
npx prisma db seed
```

Quiero resetear la base de datos

```
# Detener servicios
docker-compose down

# Eliminar volúmenes (¡Esto borrará todos los datos!)
docker-compose down -v

# Volver a levantar
docker-compose up -d
```

Puerto 3001/5432/5050 ya está en uso

Nota: El backend ya usa el puerto **3001** (no 3000) para evitar conflictos con otros servicios.

Si necesitas cambiar los puertos, editar `docker-compose.yml`:

```
services:
  backend:
    ports:
      - "3002:3000" # Cambiar 3001 a 3002

  postgres:
    ports:
      - "5433:5432" # Cambiar 5432 a 5433

  pgadmin:
    ports:
      - "5051:80" # Cambiar 5050 a 5051
```

Importante: No cambies el segundo número (puerto interno del contenedor), solo el primero (puerto externo de tu máquina).

Ejecutar sin Docker (desarrollo local)

```
# Instalar dependencias
npm install

# Asegúrate de tener PostgreSQL corriendo localmente
# Actualizar DATABASE_URL en .env:
DATABASE_URL=postgresql://admin:admin123@localhost:5432/contacts_db

# Generar cliente Prisma
npm run prisma:generate

# Ejecutar migraciones
npm run prisma:migrate

# Ejecutar seed
npm run prisma:seed

# Iniciar en modo desarrollo
npm run dev
```

Recursos Adicionales

- [Documentación de Prisma](#)
- [Documentación de Express](#)
- [Documentación de Zod](#)
- [Documentación de JWT](#)

Para Alumnos

Ejercicio 1: CRUD sin autenticación (principiantes)

1. Mantener `AUTH_REQUIRED=false`
2. Crear un frontend que consuma los endpoints de `/api/companies` y `/api/contacts`
3. Practicar:
 - Listar datos (GET)
 - Crear registros (POST)
 - Actualizar registros (PUT)
 - Eliminar registros (DELETE)

Ejercicio 2: Añadir autenticación (avanzados)

1. Cambiar `AUTH_REQUIRED=true`
2. Implementar login/register en el frontend
3. Guardar el token en localStorage o Context
4. Enviar token en todas las peticiones: `Authorization: Bearer <token>`

5. Manejar errores 401 (token inválido/expirado)
6. Implementar logout

Licencia

MIT

Autor

Backend desarrollado para práctica educativa de CRUD con React 19 + TypeScript + Prisma + PostgreSQL.

¡Listo para usar! 