Tutorial: Todo App con Next.js + Drizzle + PostgreSQL (Docker)

1) Introducción

En este tutorial construiremos una Todo app simple usando:

- **Next.js** (App Router) para el frontend y server actions.
- **Drizzle ORM** para modelar y consultar la base de datos.
- **PostgreSQL en Docker** para la base de datos local.

Al finalizar tendrás una página que permite crear, listar, completar y eliminar todos persistidas en PostgreSQL.

---

2) Crear el proyecto base

Genera la estructura inicial del proyecto con Next.js:

```
pnpm create next-app@latest my-app --yes
cd my-app
pnpm dev
```

---

3) Configurar el ORM (Drizzle)

3.a) Instalar dependencias

Instala Drizzle, el driver de PostgreSQL y los tipos. También añadiremos la CLI de Drizzle y `dotenv` para leer variables de entorno:

```
pnpm add drizzle-orm pg dotenv
pnpm add -D drizzle-kit @types/pg
```

3.b) Configurar `drizzle.config.ts`

Crea `drizzle.config.ts` en la raíz del proyecto con esta configuración:

```
import { defineConfig } from "drizzle-kit";

export default defineConfig({
  out: "./src/db/migrations",
  schema: "./src/db/schema.ts",
  dialect: "postgresql",
  dbCredentials: {
    url: process.env.DATABASE_URL!,
  },
});
```

Qué significa cada opción:

- **out**: carpeta donde la CLI de Drizzle guarda las migraciones generadas.
- **schema**: ruta al archivo TypeScript que define tus tablas (Drizzle lo lee para saber qué crear/alterar en la BD).
- **dbCredentials.url**: cadena de conexión a la base de datos. La leemos de una variable de entorno. El CLI de drizzle la usa para conectarse cuando haces `drizzle-kit push/generate`.

3.c) Crear la tabla `todo` en `src/db/schema.ts`

Crea el archivo `src/db/schema.ts` y define una tabla simple `todo`:

```
import { pgTable, integer, text, boolean } from "drizzle-orm/pg-core";

export const todo = pgTable("todo", {
  id: integer("id").primaryKey().generatedAlwaysAsIdentity(),
  title: text("title").notNull(),
  completed: boolean("completed").notNull().default(false),
});
```

3.d) Crear la conexión en `src/db/index.ts` leyendo la variable de entorno

```
import "dotenv/config";
import * as schema from "./schema";
import { drizzle } from "drizzle-orm/node-postgres";

export const db = drizzle({
  schema,
  connection: process.env.DATABASE_URL!,
});
```

---

4) Configurar Docker Compose para la base de datos local

Crea `docker-compose.yml` en la raíz del proyecto con el servicio de PostgreSQL:

```
version: "3.9"

services:
  db:
    image: postgres:16-alpine
    container_name: lieou-postgres
    restart: unless-stopped
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres
      POSTGRES_DB: lieou
    ports:
      - "5432:5432"
    volumes:
      - lieou_pg_data:/var/lib/postgresql/data
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U postgres -d lieou"]
      interval: 5s
      timeout: 5s
      retries: 5

volumes:
  lieou_pg_data:
```

---

5) Levantar el contenedor de la base de datos

```
docker compose up -d
```

---

6) Crear la variable de entorno

En la raíz del proyecto crea un archivo `.env` con la variable DATABASE_URL que usaremos en el ambiente de desarollo:

```
DATABASE_URL="postgres://postgres:postgres@localhost:5432/lieou"
```

---

7) Implementar la Todo App en `src/app/page.tsx` usando Server Actions

Breve contexto del código:

- **Server Actions**: funciones marcadas con la directiva `"use server"` que se ejecutan en el servidor (no en el navegador). Se pueden asignar directamente a `form action={miAccion}`; Nextjs serializa el `FormData` y llama la función en el server, donde tenemos acceso seguro a la BD y variables de entorno, sin crear rutas API manuales.
- **revalidatePath("/")**: invalida el caché/ruta indicada en el App Router para que la página se vuelva a renderizar con datos frescos después de una mutación (insert/update/delete).

Borra el contenido de `src/app/page.tsx` generado por Next.js y sustitúyelo por lo siguiente:

```
 import { revalidatePath } from "next/cache";
import { db } from "@/db";
import { todo as todoTable } from "@/db/schema";
import { eq } from "drizzle-orm";

async function addTodo(formData: FormData) {
  "use server";
  const rawTitle = formData.get("title");
  const title = typeof rawTitle === "string" ? rawTitle.trim() : "";
  if (!title) return;
  await db.insert(todoTable).values({ title });
  revalidatePath("/");
}

async function toggleTodo(formData: FormData) {
  "use server";
  const id = Number(formData.get("id"));
  const nextCompleted = String(formData.get("completed")) === "true";
  if (!Number.isFinite(id)) return;
  await db
    .update(todoTable)
    .set({ completed: nextCompleted })
    .where(eq(todoTable.id, id));
  revalidatePath("/");
}

async function deleteTodo(formData: FormData) {
  "use server";
  const id = Number(formData.get("id"));
  if (!Number.isFinite(id)) return;
  await db.delete(todoTable).where(eq(todoTable.id, id));
  revalidatePath("/");
}

export default async function Home() {
  const todos = await db.select().from(todoTable).orderBy(todoTable.id);

  return (
    <div className="min-h-screen p-8 font-sans">
      <div className="mx-auto max-w-md space-y-6">
        <h1 className="text-2xl font-semibold">Todos</h1>

        <form action={addTodo} className="flex gap-2">
          <input
            type="text"
            name="title"
            placeholder="Add a todo..."
            className="w-full rounded border px-3 py-2"
          />
          <button type="submit" className="rounded bg-black px-4 py-2 text-white">
            Add
          </button>
        </form>

        <ul className="space-y-2">
          {todos.length === 0 ? (
            <li className="text-sm text-gray-500">No todos yet.</li>
          ) : (
            todos.map((t) => (
              <li key={t.id} className="flex items-center justify-between rounded border p-2">
                <div className={t.completed ? "line-through text-gray-500" : ""}>{t.title}</div>
                <div className="flex items-center gap-2">
                  <form action={toggleTodo}>
                    <input type="hidden" name="id" value={String(t.id)} />
```

```
                    <input type="hidden" name="id" value={String(t.id)} />
                    <input type="hidden" name="completed" value={String(!t.completed)} />
                    <button type="submit" className="rounded border px-2 py-1 text-sm">
                      {t.completed ? "Undo" : "Done"}
                    </button>
                  </form>
                  <form action={deleteTodo}>
                    <input type="hidden" name="id" value={String(t.id)} />
                    <button type="submit" className="rounded border px-2 py-1 text-sm text-red-600">
                      Delete
                    </button>
                  </form>
                </div>
              </li>
            ))
          )}
        </ul>
      </div>
    </div>
  );
}
```

---

8) "Cargar las tablas en la base de datos

Usaremos la CLI de Drizzle para inicializar la BD:

```
pnpm run db push
```

---

9) Correr el servidor de desarrollo

```
pnpm run dev
```

---

10) Ver la Todo App

Abre `http://localhost:3000` en el navegador. Deberías poder:

- Crear nuevos todos.
- Marcarlos como completados o deshacer.
- Eliminarlos.
