Streamlining Your Next.js Projects with Supabase and Drizzle ORM

This guide showcases how to build an efficient application using Supabase for data handling, complemented by Drizzle ORM for advanced database schema management in TypeScript. Starting with database seeding, we aim to demonstrate effective data retrieval, leveraging Drizzle ORM's intuitive blend of SQL-like and relational data access. Drizzle ORM not only simplifies database interactions but also introduces a suite of tools to enhance development workflows, setting the foundation for a productive and streamlined development experience.

Step 1: Setting Up Your Next.js Environment

To start your Next.js project, execute the command `npx create-next-app@latest syncleaf`. This quickly sets up a fresh Next.js application named `syncleaf`

```
npx create-next-app@latest syncleaf
```

Configuring Drizzle ORM and Environment Variables

Proceed by installing Drizzle ORM, PostgreSQL, and dotenv with the command below in your project directory:

```
npm install drizzle-orm postgres dotenv
```

This step incorporates Drizzle ORM for handling database schemas, `postgres` for interacting with the PostgreSQL database, and `dotenv` for environment variable management, all crucial for a secure and efficient database connection.

Following this, enhance your development workflow by adding Drizzle Kit as a development dependency:

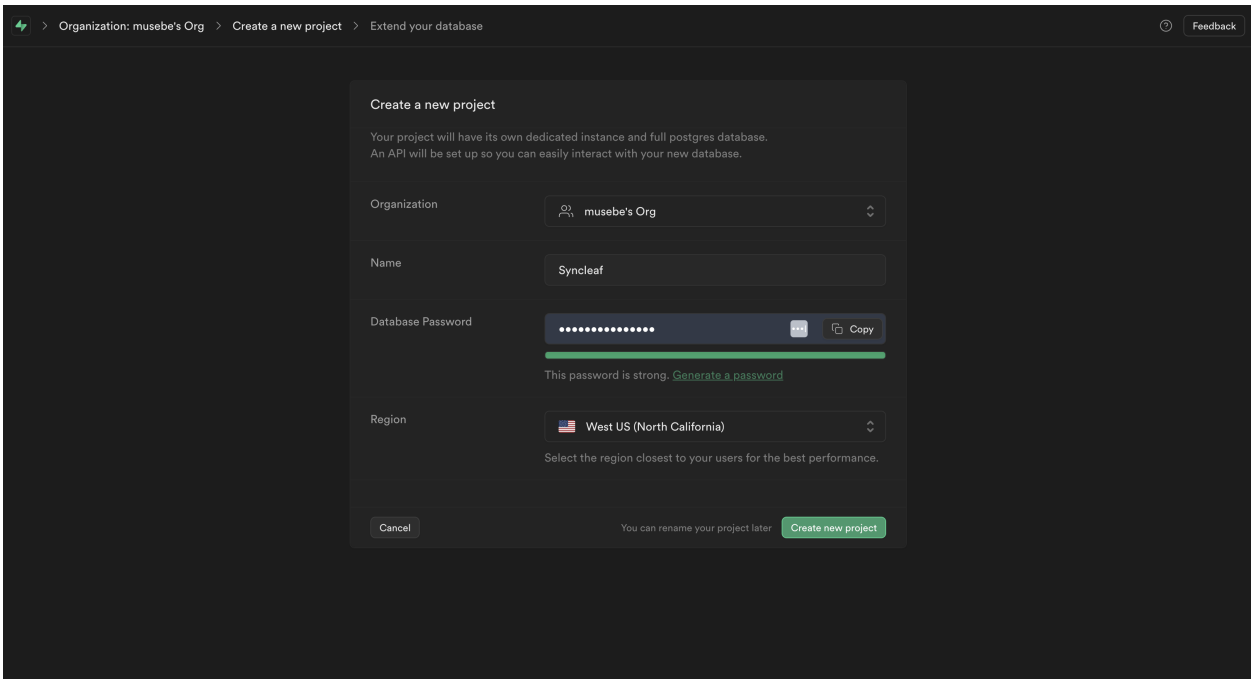
```
npm i -D drizzle-kit -D
```

Think of Drizzle Kit as a magic tool that helps you build and change your database, kind of like building with LEGOs. You tell it how you want your database to look using a special code, and it creates a set of instructions to make or change the database just like that. If you decide to change how your database should look, Drizzle Kit figures out what new instructions are needed and keeps everything organized and safe, so you can always go back and see what changes you made. Plus, you can work on different parts of your database in separate pieces or even work on many databases at once. And if you already have a database, Drizzle Kit can quickly understand how it's built and help you make changes to it super fast!

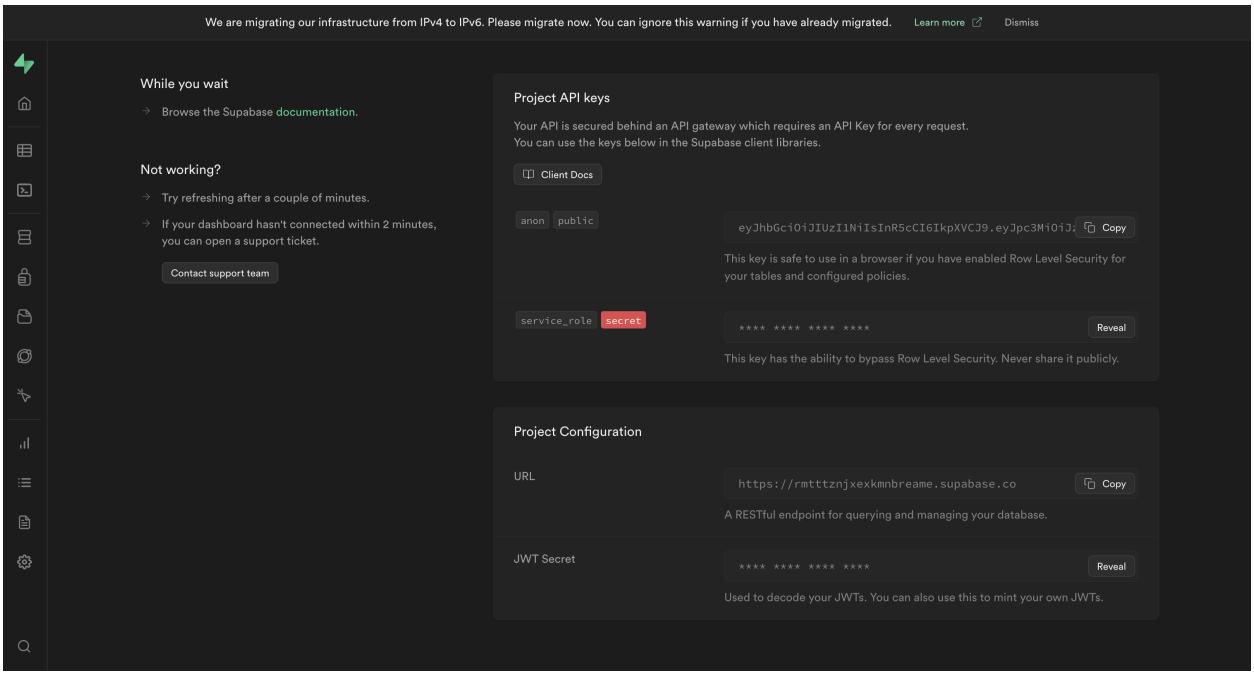
Setting Up Your Supabase Project

Initiate your Supabase project setup by first logging in at Supabase Login. Once logged in, select "New Project" and name it "Syncleaf." It's essential to generate and save a secure database password for later use. Choose the server region that offers the best performance for your target audience. After filling in all the required fields, click "Create new project." Securely store the database password as you will need it for your `.env` file to establish a database connection.

For a visual guide, refer to the image provided below.



After creating your project, you'll be taken to a screen displaying all necessary API keys and configuration details, including your project URL, anon key for client interactions, and service role key for backend operations. These are crucial for connecting your Next.js app securely with Supabase, so make sure to accurately copy them into your `.env` file for future use.



Configure Environment Variables

Create a `.env` file at the root of your project to securely store your database credentials. These values are provided by Supabase as highlighted above :

```
DATABASE_URL=
NEXT_PUBLIC_SUPABASE_URL=
NEXT_PUBLIC_SUPABASE_ANON_KEY=
SERVICE_ROLE_KEY=
PW=
```

Drizzle Configuration Setup

Create a `drizzle.config.ts` file at the root of your project to configure Drizzle ORM's interaction with your database.

Here is the content for the configuration file:

```
import type { Config } from 'drizzle-kit';
import * as dotenv from 'dotenv';
dotenv.config({ path: '.env' });

if (!process.env.DATABASE_URL) {
  console.log('🔴 Cannot find database url');
}

export default {
  schema: './src/lib/supabase/schema.ts',
  out: './migrations',
  driver: 'pg',
  dbCredentials: {
    connectionString: process.env.DATABASE_URL || '',
  },
} satisfies Config;
```

This configuration file serves as a map for Drizzle ORM, pointing it to the location of your database schema files, where to store migration files, and which database driver to use. It also securely pulls in the database connection string from your `.env` file. This setup is essential for enabling Drizzle ORM to manage your database schema and migrations effectively.

Following the structure outlined in our `drizzle.config.ts` configuration, let's proceed to create the files and directories:

Defining the Database Schema

For schema definition, place a `schema.ts` file within the `src/lib/supabase/` directory. To set up this file and its required directory structure, execute the command:

```
mkdir -p src/lib/supabase && touch src/lib/supabase/schema.ts
```

The `schema.ts` file is used to define and export data models that closely represent the structure of your database tables. These models facilitate type-safe database operations, ensuring that the data types used in your application match those in your database. This approach significantly enhances development efficiency by enabling autocompletion, reducing runtime errors, and making the codebase easier to understand and maintain.

Add this to it:

```
import { pgTable, uuid, text, decimal, integer, timestamp } fro

export const product = pgTable("product", {
    id: uuid('id').defaultRandom().primaryKey().notNull(),
    name: text("name"),
    description: text("description"),
    price: decimal("price", { precision: 10, scale: 2 }),
    quantity: integer("quantity"),
    image: text("image"),
    created_at: timestamp("created_at").defaultNow(),
    updated_at: timestamp("updated_at").defaultNow(),
});
```

This code snippet employs `drizzle-orm/pg-core` to create a `product` table model for PostgreSQL integration with Supabase, ensuring operations adhere to specified data types and schema constraints. This method enhances the reliability and scalability of your application's data layer without detailing individual fields.

Setting Up Database Connection and Utility Functions

Continuing with our setup, we'll now add a `db.ts` file to the `src/lib/supabase` directory, crucial for our database connection and utility functions. This step simplifies database interactions, improving maintainability and scalability.

To create the `db.ts` file:

```
touch src/lib/supabase/db.ts
```

This prepares us to define our connection and utilities.

Add the following content to the
`db.ts` file to set up your database connection and utilities:

```
import { drizzle } from 'drizzle-orm/postgres-js';
import postgres from 'postgres';
import dotenv from 'dotenv';
import * as schema from '../../../migrations/schema';

dotenv.config();

if (!process.env.DATABASE_URL) {
  console.error('❌ Error: Database URL is not specified in the
  process.exit(1);
}

const client = postgres(process.env.DATABASE_URL, { max: 1 });

const db = drizzle(client, { schema });

console.log('Database connection successfully established.');

export default db;
```

This script sets up the database connection using `drizzle-ORM` and `postgres`, with configurations managed via environment variables. It ensures the `DATABASE_URL` is available, initializes the connection, and indicates a successful setup. The `drizzle` client is then made available for application-wide usage.

Enhancing Database Management with Drizzle Scripts

To efficiently manage and interact with your database using Drizzle, add the following scripts to your `package.json`. These scripts provide convenient commands for database operations such as schema synchronization, introspection, generation, migration, and seeding:

```
"scripts": {
  "push": "drizzle-kit push:pg",
  "pull": "drizzle-kit introspect:pg",
  "generate": "drizzle-kit generate:pg",
  "drop": "drizzle-kit drop",
  "check": "drizzle-kit check:pg",
  "up": "drizzle-kit up:pg",
  "migrate": "npx tsx scripts/migrations/migration.ts",
  "studio": "drizzle-kit studio",
  "seed": "npx tsx scripts/seed.ts"
}
```

These scripts simplify the process of keeping your database schema in sync with your codebase, managing migrations, and seeding data for development and testing.

Generating Migration Files for PostgreSQL with Drizzle

Execute the `npm run generate` command to initiate migration file creation:

```
npm run generate
```

Running `npm run generate` triggers `drizzle-kit generate:pg`, analyzing your PostgreSQL schema and auto-generating a migration file for streamlined schema management. Following this command, a `migrations` folder will be created at the root of your project, as directed by the `out: './migrations'` setting in `drizzle.config.ts`, ensuring an organized approach to tracking database schema changes.

Setting Up the Migration Script

To organize your project's migration scripts, first create a `scripts` folder at the root of your project directory, then add a `migrations` folder within it, and finally create a `migration.ts` file inside this folder. Use the following command to set up this structure:

```
mkdir -p scripts/migrations && touch scripts/migrations/migra
tion.ts
```

This command ensures the necessary directories and file are created in your project, ready for you to add your migration logic.

Add the following content to your `migration.ts` file to handle database migrations:

```
import db from '../../src/lib/supabase/db';
import { migrate } from 'drizzle-orm/postgres-js/migrator';
import dotenv from 'dotenv';

dotenv.config();
```

```
const migrateDatabase = async (): Promise<void> => {
    console.log('🚀 Starting database migration...');

    try {
        await migrate(db, { migrationsFolder: 'migrations'
});
        console.log('✅ Successfully completed the database m
igration.');
        process.exit(0);
    } catch (error) {
        console.error('❌ Error during the database migratio
n:', error);
        process.exit(1);
    }
};

migrateDatabase();
```

This script initializes the environment variables, then defines and executes a function to migrate the database using `drizzle-ORM`. It logs the start and successful completion of the migration process or catches and logs any errors encountered, ensuring a clear status update during the migration process.

Executing the Migration Script

To execute the migration script and apply your database changes, run the following command:

```
npm run migrate
```

Upon successful execution, you'll notice new files within the `migrations` folder, indicating that the migration scripts have been generated and run. Additionally, by checking your Supabase database, you should find the `products` table created, complete with all the fields you've previously defined.

For a more interactive view of your database schema and to manage your data directly, use the command:

```
npm run studio
```

This will launch Drizzle-Kit Studio, utilizing your project's Drizzle configuration file to connect to your database. Drizzle Studio provides a user-friendly interface for browsing your database, as well as adding, deleting, and updating entries according to your defined Drizzle SQL schema.

Populating the Products Table with Seed Data

With the products table in place, it's time to populate it with some sample data. To achieve this, we'll utilize the `faker` library to generate realistic product information seamlessly. This approach not only simplifies the process of creating diverse data sets but also enhances the testing and development experience by providing a rich dataset to work with.

Ensure `faker` is installed in your project by running:

```
npm install @faker-js/faker
```

Next, create the
`seed.ts` file by executing the following command:

```
touch scripts/seed.ts
```

Now, add the following contents to your `seed.ts` file:

```
import { drizzle } from 'drizzle-orm/node-postgres';
import { Pool } from 'pg';
import { product } from '../src/lib/supabase/schema';
import { faker } from '@faker-js/faker';
import * as dotenv from 'dotenv';

dotenv.config({ path: './.env' });

if (!process.env.DATABASE_URL) {
    console.error('DATABASE_URL not found in .env');
    process.exit(1);
}

const main = async () => {
    const pool = new Pool({
        connectionString: process.env.DATABASE_URL,
    });
    const db = drizzle(pool);

    const productsData = [];

    for (let i = 0; i < 20; i++) {
        productsData.push({
            name: faker.commerce.productName(),
            description: faker.commerce.productDescription(),
            price: faker.commerce.price({ min: 100, max: 1000, d
            quantity: faker.number.int({ min: 1, max: 100 }),
            image: faker.image.url(),
```

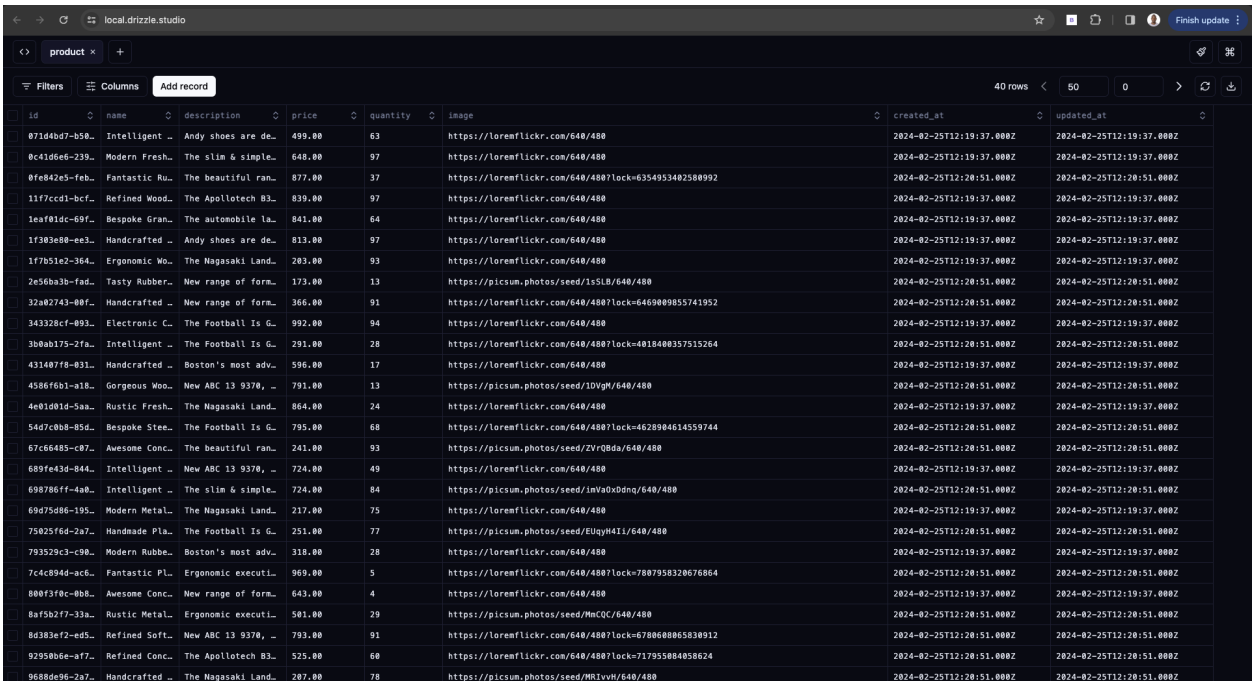
```
        });
    }

    console.log('Seed start');
    await db.insert(product).values(productsData).execute();
    console.log('Seed done');
    await pool.end();
};

main().catch((error) => {
    console.error('Failed to seed products:', error);
    process.exit(1);
});
```

To populate your database with 20 unique product entries, execute the command `npm run seed`. This command triggers a script that connects to your database, generates product entries using faker, and inserts them into the products table, creating a foundational dataset for development and testing.

After running `npm run seed`, review your Supabase database or drizzle-kit studio to confirm the successful population of product entries, as shown in the provided screenshot. This confirms the success of your migration and seeding efforts, setting the stage for application development.



Wrapping Up

In conclusion, leveraging Drizzle ORM has empowered us to streamline database population, schema evolution, and data manipulation seamlessly. This efficiency has greatly expedited our development journey, furnishing us with a sturdy groundwork for constructing and expanding our application.

Reference

- GitHub Repository - https://github.com/musebe/Drizzle-supabase
- Deployed Demo - https://drizzle-supabase.vercel.app/
- Supabase - https://supabase.io
- Drizzle ORM Documentation - https://orm.drizzle.team/
- Drizzle ORM Kit Documentation - https://orm.drizzle.team/kit-docs/overview