Next.js with Prisma and SQLite

MARCH 21, 2024 BY ROBIN WIERUCH - EDIT THIS POST

A short tutorial about setting up a Next.js application with Prisma and SQLite. I decided to write this tutorial, because many of my other Next.js tutorials depend on a database and I want to reference it. Why this ORM and database? Prisma is a great tool to interact with databases and SQLite is a simple database to get started with.

This tutorial is part 1 of 2 in this series.

Part 1: Next.js with Prisma and SQLite
Part 2: Server Actions with Next.js

This tutorial is part 1 of 2 in this series.

Part 1: Next.js with Prisma and SQLite
Part 2: Authentication in Next.js

NEXT.JS: SETUP

If you don't have a Next.js application yet, you can create one on the command line:

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

Personally I like to use all the defaults when creating a new Next.js application. After your project is created, you can navigate into the project folder:

```
cd my-next-app
```

Next change the default page to something more minimalistic:

```
// src/app/page.tsx
```

```
export default Home;
```

Finally you can start the Next.js application:

```
npm run dev
```

From here we will setup Prisma and SQLite before reading from the database.

SQLITE

SQLite is a database that is often used for its simplicity. Go ahead and install it:

```
npm install sqlite3
```

When using SQLite, you don't have to set up an extra database, because SQLite is just a file in your project. This makes SQLite a great choice for starting out with a database.

PRISMA

Prisma is the most popular JavaScript/TypeScript ORM to interact with databases. Go ahead and install it for your project:

```
npm install prisma --save-dev
```

We can initialize Prisma with SQLite as the data source provider:

```
npx prisma init --datasource-provider sqlite
```

The *prisma/schema.prisma* file in the project should look similar to this one:

```
// prisma/schema.prisma
```

```
datasource db {
  provider = "sqlite"
  url      = env("DATABASE_URL")
}
```

There should also be a *.env* file which holds the path to the SQLite database file:

```
DATABASE_URL="file:./dev.db"
```

Do not forget to create this database file, because Prisma will not do it for you:

```
cd prisma
touch dev.db
```

Essentially this file is the SQLite database.

PRISMA MIGRATIONS

Prisma Migrations are needed to introduce new data in the database. For example, when you want to add a new table to the database, you can create it in the Prisma schema file. We will have a `Post` table in this example, but you can choose a different name or properties if you like:

```
// prisma/schema.prisma

generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "sqlite"
  url      = env("DATABASE_URL")
}

model Post {
  id      String   @id @default(cuid())
  name    String
}
```

Last, you can always check the database with Prisma Studio:

```
npx prisma studio
```

There you should see the empty Post table. Optionally you can seed this database with some initial data to see it later in your Next.js application.

PRISMA CLIENT

We will need a Prisma Client instance in the application eventually to read and write from/to the database. Therefore, initialize a Prisma Client instance in the project:

```
// src/lib/prisma.ts
import { PrismaClient } from '@prisma/client';

export const prisma = new PrismaClient();
```

When working with a framework like Next.js, it is important to use a singleton pattern for the Prisma Client instance. Otherwise, you may run into issues with hot reloading and multiple instances of the Prisma Client in development mode. Use the following code snippet to create a singleton for the Prisma Client instance:

```
// src/lib/prisma.ts
import { PrismaClient } from "@prisma/client";

const prismaClientSingleton = () => {
  return new PrismaClient();
};

declare const globalThis: {
  prismaGlobal: ReturnType<typeof prismaClientSingleton>;
} & typeof global;

export const prisma = globalThis.prismaGlobal ?? prismaClientSingletc

if (process.env.NODE_ENV !== "production") globalThis.prismaGlobal =
```

NEXT.JS: READING FROM THE DATABASE

With Next's App Router, every component is by default a React Server Component. This means that you can use the Prisma Client instance in these components to read from the database. Here is how you would read from the database on the root page:

```
// src/app/page.tsx

import { prisma } from '@/lib/prisma';

const Home = async () => {
  const posts = await prisma.post.findMany();

  return (
    <div className="p-4 flex flex-col gap-y-4">
      <h2>Home</h2>

      <ul className="flex flex-col gap-y-2">
        {posts.map((post) => (
          <li key={post.id}>{post.name}</li>
        ))}
      </ul>
    </div>
  );
};

export default Home;
```

That's it. If you have done the database seeding in between, you should see the seeded data in your Next.js application.

. . .

You can find the repository for this tutorial over here. If you want to go beyond this, check out **"The Road to Next"** and get on the waitlist!

This tutorial is part 1 of 2 in this series.

Part 1: Next.js with Prisma and SQLite
Part 2: Server Actions with Next.js

Part 2: Authentication in Next.js

Discuss on Twitter Share on Twitter

KEEP READING ABOUT NODE

HOW TO SETUP EXPRESS.JS IN NODE.JS

Express.js is the most popular choice when it comes to building web applications with Node.js. However, when saying web applications with Node.js, it's often not for anything visible in the browser...

SEEDING A DATABASE WITH PRISMA (TYPESCRIPT)

A short tutorial about seeding a database with Prisma in a TypeScript application. I decided to write this tutorial, because many of my other tutorials depend on a database with an initial seeding and...

GET THE BOOK

Get it on Amazon.

TAKE PART

NEVER MISS AN ARTICLE ABOUT WEB DEVELOPMENT AND JAVASCRIPT.

- ✔ Join 50.000+ Developers
- ✔ Learn Web Development
- ✔ Learn JavaScript
- ✔ Access Tutorials, eBooks and Courses
- ✔ Personal Development as a Software Engineer

SUBSCRIBE

View our Privacy Policy.