Learn to Create a URL Shortener using Next.js, and Supabase

In this tutorial, we will learn how to create a URL shortener using Next.js, a popular React framework, and Supabase, a powerful open-source alternative to Firebase. We will build the backend using Supabase to handle URL creation and redirection, and the frontend using Next.js with Tailwind CSS for a sleek and responsive user interface.

Prerequisites

To follow along with this tutorial, you should have the following prerequisites:

1. Node.js and npm (Node Package Manager) installed on your machine.
2. Basic knowledge of JavaScript and React.
3. A Supabase account to create a database and obtain connection credentials.

Setting Up the Next.js Project

Let's start by setting up a new Next.js project. Open your terminal and execute the following commands:

```
npx create-next-app url-shortener

✔ Would you like to use TypeScript with this project? … No
✔ Would you like to use ESLint with this project? … Yes
✔ Would you like to use Tailwind CSS with this project? … Yes
✔ Would you like to use `src/` directory with this project? … Yes
✔ Use App Router (recommended)? … No
✔ Would you like to customize the default import alias? … No
```

This will create a new Next.js project in a directory named `url-shortener`. Next, open the project in your preferred code editor.

```
cd url-shortener
```

Installing Dependencies

Next, we need to install the required dependencies for our project. In the terminal, navigate to the project directory and run the following command:

```
npm install supabase @supabase/supabase-js dotenv nanoid
```

Setting up Supabase

Now, let's set up Supabase to handle the backend functionality of our URL shortener.

1. Sign up for a free account on **Supabase** and create a new project.
2. Retrieve the URL and public key for your Supabase project.
3. Create a new file called `.env` in the root of your project and add the following environment variables:

```
SUPABASE_URL=<your-supabase-url>
SUPABASE_KEY=<your-supabase-public-key>
```

You'll get these key's in the settings of the project that you created.

4. Select your project and navigate to the "SQL" section in the left sidebar.
5. In the SQL editor, execute the following SQL query to create the `urls` table:

   ```
   create table urls (
     id text primary key,
     original_url text
   );
   ```

   This query creates a table named `urls` with two columns: `id` (to store the short URL identifier) and `original_url` (to store the original URL).
6. After executing the query, you should see the `urls` table listed under the "Tables" section in the dashboard.

Ensure that the table schema and column names match the ones used in your backend code. In this case, the `urls` table should have columns `id` and `original_url`.

Creating the Backend

Next, we will create the backend API routes using Supabase to handle URL creation and redirection.

1. Create a new directory called `src/pages/api` in your project.
2. Inside the `api` directory, create a new file called `create.js` and add the following code:

```
import { createClient } from '@supabase/supabase-js';
import { nanoid } from 'nanoid';

const supabase = createClient(
  process.env.SUPABASE_URL,
  process.env.SUPABASE_KEY
);

export default async function handler(req, res) {
  if (req.method === 'POST') {
    const { originalUrl } = req.body;
    const id = nanoid(8); // Generate a short ID using nanoid library

    const { data, error } = await supabase
      .from('urls')
      .insert([{ id, original_url: originalUrl }])
      .single();

    if (error) {
      res.status(500).json({ error: 'Failed to create short URL' });
    } else {
      const shortUrl = `${req.headers.host}/${data.id}`;
      res.status(200).json({ shortUrl });
    }
  } else {
    res.status(405).json({ error: 'Method not allowed' });
  }
}
```

This code creates a new API route `/api/create` that accepts a POST request with a JSON payload containing the `originalUrl`. It then inserts the `originalUrl` into the `urls` table in Supabase and returns the generated short URL.

Next, create another file called `middleware.js` in the `src` directory and add the following code:

```
import { createClient } from "@supabase/supabase-js";
import { NextResponse } from "next/server";

const supabase = createClient(
  process.env.SUPABASE_URL,
  process.env.SUPABASE_KEY
);

function getValidURL(url) {
    if(url.includes('http://') || url.includes('https://')) {
        return url
    }
    return 'https://' + url
}

export default async function handler(req) {
    if (req.method === "GET") {

        let pathname = req.nextUrl.pathname
        let parts = pathname.split('/')
        let id = parts[parts.length - 1]
        try {
            const { data, error } = await supabase
                .from("urls")
                .select("original_url")
                .eq("id", id)
                .single();
            // console.log(data, error);
            if (!error) {
                const shortUrl = data.original_url;
                console.log("SHORT URL", shortUrl)
                return NextResponse.redirect(getValidURL(shortUrl));
            }
        } catch (error) {
            console.log(error.message)
        }

    }
}
```

This code serves as middleware that intercepts incoming GET requests and handles the redirection logic based on the ID parameter.

- The middleware retrieves the `SUPABASE_URL` and `SUPABASE_KEY` from environment variables and creates a Supabase client using `createClient()`.
- The `getValidURL()` function ensures that the original URL has either `http://` or `https://` prefix. If it doesn't, it appends `https://` to the URL.

- The `handler()` function checks if the incoming request method is GET.
- It extracts the ID parameter from the URL path.
- Using the Supabase client, it queries the `urls` table to fetch the original URL corresponding to the ID.
- If there are no errors and the original URL is found, it performs a redirection using `NextResponse.redirect()` by passing the valid URL obtained from `getValidURL()`.
- If there is an error or the original URL is not found, it logs the error message to the console.

This middleware can be used in your Next.js application to handle redirection based on the ID parameter and the corresponding original URL stored in the Supabase database.

Creating the Frontend

Now that the backend is set up, let's create the frontend interface using Next.js and Tailwind CSS.

1. Open the `pages/index.js` file in your project and replace the existing code with the following:

```
import { useState } from 'react';

export default function Home() {
  const [originalUrl, setOriginalUrl] = useState('');
  const [shortUrl, setShortUrl] = useState('');

  const handleSubmit = async (e) => {
    e.preventDefault();

    const response = await fetch('/api/create', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({ originalUrl }),
    });

    const data = await response.json();
    setShortUrl(data.shortUrl);
  };

  function getValidURL(url) {
    if(url.includes('http://') || url.includes('https://')) {
```

```
        return url
    }
    return 'https://' + url
}

  return (
    <div className="container mx-auto p-4 flex justify-center items-center flex-col min-h-
screen">
      <h1 className="text-2xl font-bold mb-4">URL Shortener</h1>
      <form onSubmit={handleSubmit}>
        <input
          type="text"
          placeholder="Enter URL"
          className="p-2 mr-2 border border-gray-300 text-black"
          value={originalUrl}
          onChange={(e) => setOriginalUrl(e.target.value)}
        />
        <button
          type="submit"
          className="bg-blue-500 text-white px-4 py-2 rounded"
        >
          Shorten
        </button>
      </form>
      {shortUrl && (
        <div className="mt-4">
          <label className="font-bold">Short URL:</label>
          <a href={shortUrl} target="_blank" rel="noopener noreferrer">
            {getValidURL(shortUrl)}
          </a>
        </div>
      )}
    </div>
  );
}
```

This code sets up a basic form where users can enter a URL to be shortened. On form submission, it sends a POST request to the `/api/create` route, receives the generated short URL, and displays it on the page.

Conclusion

In summary, we have learned how to create a URL shortener using Next.js and Supabase. By combining Next.js for the frontend, Supabase for the backend, and Tailwind CSS for styling, we built a functional application.

We covered the creation of API routes to handle URL creation and redirection. Supabase served as the data storage solution, allowing us to store and retrieve URLs

efficiently.

By following this tutorial, you now have the knowledge to develop your own URL shortener, which can be extended with additional features based on your needs.

In conclusion, with Next.js and Supabase, you can quickly build a URL shortener that is scalable, easy to maintain, and customizable. Furthermore, deploying your application on Vercel provides a seamless and efficient way to make it accessible to users worldwide. Enjoy exploring the possibilities of your new URL shortening application!