Front-End with NextJS

Software Engineering, Tutorial

Antonio Bucchiarone - bucchiarone@fbk.eu

Academic year 2023/2024

Building a Next.js shopping cart app

- This tutorial details how to build a shopping cart web app for a store with the ability to
  - **add** or **remove** items from the cart,
  - **view** all products,
  - **view products by category**, and more.

Getting started with create-next-app

- To create a new Next.js app using create-next-app, run the following command on the terminal and wait for the installation process to finish:

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

- open it in VSCode
- run the *dev* script

```
npm run dev
```

- open *http://localhost:3000*

Modify the template

- open the layout.tsx
- change the "title" and "description" parameters
- the "globals.css" must be used to add modify the style of your app as you prefer.

Modify the template (Home) page

- delete the main content and insert this new code
- remove also the import

```
<div>
  <h1>Home Page</h1>
</div>
```

- remove all the predefined styling in globals.css
- do not remove the @tailwind elements (margins, background colors, etc..)
- we should have a white page now.

Installing the npm modules

```
npm i antd axios redux react-redux @reduxjs/toolkit mongoose bcryptjs jsonwebtoken firebase
```

- **antd** for form, table, form validatioms and other components.
- **axios** for the http calls - API management
- **redux, react-redux, @reduxjs/toolkit** for the state management
- **mongoose** for the backend database
- **bcryptjs, jsonwebtoken** for the security aspects
- **firebase** to store the images

```
npm run dev
```

Antd

- module used to build the components like buttons, forms, popup, etc..
- https://ant.design/components/overview/
- https://ant.design/components/button

```
import { Button } from 'antd'
<div>
  <h1>Home Page</h1>
  <Button type='primary'>Button</Button>
</div>
```

module conflict resolution

- conflicts (overlap) between tailwind and Antd
- add the following properties in the tailwind.config.ts

```
plugins: [],
  corePlugins: {
    preflight: false,
  },
```

- add a tailwind button to see if now it works

```
<Button type='primary'>Button</Button>
     <button className="bg-blue-500 text-white p-2">Tailwid Button</button>
```

Overriding AntD Styling

- modify the globals.css file

```
button {
  background-color: black !important;
}
```

- increase the size

  height: 40px !important;

- change the border style

  border-radius: 0 !important;

the need of a Theme Prodiver

- create a default button

```
<div>
  <h1>Home Page</h1>
  <Button type='primary'>Primary Button</Button>
  <Button type='default'>Default Button</Button>
</div>
```

- what it the issue?
- The requiremnt was "To have a black primary button"

Theme Template creation

- create a folder app/providers
- create a file ThemeProvider.tsx with this code

```
import React from "react";
import { ConfigProvider } from "antd";

function ThemeProvider({ children }: { children: React.ReactNode }) {
    return (
        <div><ConfigProvider theme={{
                    token: {
                        colorPrimary: '#000',
                    }
                }}>{children}</ConfigProvider>
        </div>
    )
}
export default ThemeProvider
```

- modify the layout.tsx file

```
<html lang="en">
      <body>
        <ThemeProvider>
          {children}
        </ThemeProvider>
      </body>
    </html>
```

- do not forget to modify the globals.css to this

```
button {
  height: 40px !important;
  border-radius: 0 !important;
}
```

add space between buttons

- modify page.tsx

```
<div>
      <h1>Home Page</h1>

      <div className="flex gap-4">
        <Button type='primary'>Primary Button</Button>
        <Button type='default'>Default Button</Button>
      </div>
    </div>
```

- Note: do not forget to install the extensions in VSCode

  -- npm intellisense

  -- tailwind CSS IntelliSense

Button border color

- change the global.css file

```
button {
  height: 40px !important;
  border-radius: 0 !important;
  border: 1px solid #000000 !important;
}
```

Login Page

-- IMPORTANT: add the VS Code extension redux

- type rfce and see the suggested component
- ex: react Functional Export component template

```
import React from 'react'

function page() {
  return (
    <div>page</div>
  )
}

export default page
```

Login and Register pages

- create the folders /app/auth/login /app/auth/register
- for each folder create the files page.tsx
- in each file create a function using redux (rfce)

```
function Login
    () {
    return (
        <div>Login</div>
    )
}

export default Login
```

- open http://localhost:3000/auth/register and http://localhost:3000/auth/login

division of a page in left and right parts

```
'use client'
import React from 'react'

function Register() {
    return (
        <div className='grid grid-cols-2 min-h-screen'>
            <div className='h-full bg-black'>Left</div>
            <div>Right</div>

        </div>
    )
}

export default Register
```

- refresh the page http://localhost:3000/auth/register

delete the white space

- modify the global.css file

```
*,
html,
body {
  box-sizing: border-box;
  margin: 0;
  padding: 0;
}

button {
  height: 40px !important;
  border-radius: 0 !important;
  border: 1px solid #000000 !important;
}
```

add text

```
'use client'
import React from 'react'

function Register() {
    return (
        <div className='grid grid-cols-2 min-h-screen'>
            <div className='h-full bg-black flex items-center justify-center'>
                <h1 className='text-7xl font-bold text-red-500'>MY</h1>
                <h1 className='text-7xl font-bold text-gray-500'>-</h1>
                <h1 className='text-7xl font-bold text-yellow-700'>SHOP</h1>

            </div>
            <div>Right</div>

        </div>
    )
}

export default Register
```

Form with a style

```
<Form className='w-[500px]' layout='vertical' >
                    <Form.Item name="name" label="Name">
                        <input type='text' />
                    </Form.Item>
                    <Form.Item name="email" label="Email">
                        <input type='email' />
                    </Form.Item>
                    <Form.Item name="password" label="password">
                        <input type='password' />
                    </Form.Item>

                    <Button type='primary' htmlType='submit' block>
                        Register
                    </Button>
                </Form>
```

input style in global.css

```css
input {
  width: 100% !important;
  height: 40px !important;
  padding: 0 10px !important;
}
```

add the "Register" header

```
<Form className='w-[500px] gap-5' layout='vertical'  >
    <h1 className='text-2x1 font-bold'>Register</h1>
      <hr />
      <br />
      <Form.Item name="name" label="Name">
        <input type='text' />
      </Form.Item>
      <Form.Item name="email" label="Email">
        <input type='email' />
      </Form.Item>
      <Form.Item name="password" label="password">
        <input type='password' />
      </Form.Item>
      <Button type='primary' htmlType='submit' block>
        Register
      </Button>
</Form>
```

Navigation link

```
<Button type='primary' htmlType='submit' block>
    Register
</Button>

<Link href="/auth/login" className='text-black'>
Already have an account? Login
</Link>
```

Form Inputs and Inspect for input values

```
interface userType {
    name: string;
    email: string;
    password: string;
}
function Register() {

    const onRegister = (values: userType) => {
        console.log(values);
    }
...
...
 <Form className='w-[500px] gap-5' layout='vertical'
 onFinish={onRegister} >
```

- Insert user data and inspect the page

Now do the same for your Login Page

Mandatory fields validation

```
 <Form.Item name="email" label="Email"
  rules={[{
    required: true,
    message: "Please input your email"
  }]}>

...

  <Form.Item name="password" label="password"
  rules={[{
    required: true,
    message: "Please input your password"
    }]}>
```

use reusable functions

- create a folder and a tsx file /app/helpers/validation.ts
- create a function like this

```
export const getAntdFieldRequiredRule = (message: string) => {
    return [
        {
            required: true,
            message,
        }
    ]
}
```

Modify the rules in the Form field

```
<Form.Item name="email" label="Email"
    rules={getAntdFieldRequiredRule('Please input your email')}>

  <input type='email' />
</Form.Item>
<Form.Item name="password" label="password"
    rules={getAntdFieldRequiredRule('Please input your password')}>
  <input type='password' />
</Form.Item>
```

Add the rules to your Register page

Basic API

- create a folder for the APIs
- app/api/users/route.ts
- create a GET API /api/users

```
import { NextResponse } from "next/server";

export async function GET() {
    return NextResponse.json({
        success: true,
        data: [
            {
                id: 1,
                name: "Antonio Bucchiarone"
            },
        ],
    });
}
```

GET API - dynamic user id

- /api/users/[userid]
- createa a file /api/users/[userid]/route.ts

```
import { NextRequest, NextResponse } from "next/server";

interface Params {
    userid: string;
}

export async function GET(request: NextRequest, { params }: { params: Params }) {
    const userid = params.userid;
    return NextResponse.json({
        success: true,
        data: {
            id: userid,
            name: 'Antonio Bucchiarone'
        },
    });
}
```

From static data to MongoDB data

1. create a .env file
2. define the mongodb url constant

```
mongo_url = mongodb+srv://bucchiarone:<password>@cluster0.qfrrpwf.mongodb.net/myshop
```

3. code the connectDB function in the dbConfig.ts file

connectDB function

```
import mongoose from "mongoose";

export const connectDB = async () => {
    try {
        await mongoose.connect(process.env.mongo_url!);
        console.log("Mongo DB connected");

    } catch (error) {
        console.log(error);
    }

};
```

- call the api **/api/users** and see the console message in VS Code

User Authentication API

1. Create the User Model in /models/userModel.ts
2. Create the Schema for the User data

```
import mongoose from "mongoose";

export const userSchema = new mongoose.Schema(
    {
        name: {
            type: String,
            required: true,
        },
        email: {
            type: String,
            required: true,
        },
        password: {
            type: String,
            required: true
        },
        deliveryAddresses: {
            type: Array,
            default: [],
            required: false,
        },
        isActive: {
            type: Boolean,
            default: true,
            required: true,
        },
        isAdmin: {
            type: Boolean,
            default: false,
            required: false,
        },

    },
    {
        timestamps: true,
    }
);
```

3. export the User Model

```
// if the model is already defined, use that model else create a new one
export default mongoose.models["Users"] || mongoose.model("users", userSchema);
```

User REGISTER API

1. Install bcrypt

```
npm i --save-dev @types/bcryptjs
```

2. modify the api file api/auth/register/route.ts

```
import { connectDB } from "@/configs/dbConfig";
import User from "@/app/models/userModel";
import { NextRequest, NextResponse } from "next/server";
import bcrypt from "bcryptjs";


connectDB();

export async function POST(request: NextRequest) {
    try {
        const reqBody = await request.json();

        //check if the user already exists
        const userExists = await User.findOne({ email: reqBody.email });
        if (userExists) {
            throw new Error("User already exists");
        }
        // create new user
        // random string
        const salt = await bcrypt.genSalt(10);
        // hashing the pwd
        const hashedPassword = await bcrypt.hash(reqBody.password, salt);
        reqBody.password = hashedPassword;
        const newUser = new User(reqBody);

        await newUser.save();

        return NextResponse.json({
            message: "User created successfully",
            data: newUser,
        })

    } catch (error) {
        return NextResponse.error();

    }
}
```

API use in the UI

1. Modify the auth/register/page.tsx - the onRegister function using the post API to register the new user.

```
function Register() {
    const [loading, setLoading] = React.useState(false);
    const onRegister = async (values: userType) => {
        try {
            setLoading(true);
            await axios.post("/api/auth/register", values);
            message.success("Registration successful, please login to continue");
        } catch (error: any) {
            message.error(error.response.data.message);

        } finally {
            setLoading(false);
        }

    };
```

add the loading to the button object

```
<Button type='primary' htmlType='submit' block loading={loading}>
   Register
</Button>
```

2. open the Register page and add a new user
3. check the DB in Atlas Collection if the user has been added correctly

- add the Error message if the User already exists

1. open the file api/auth/register/route.ts
2. extend the catch part with this code

```
catch (error: any) {
        return NextResponse.json({
            message: error.message,
        },
            {
                status: 400
            }
        );

    }
```

3. try to create an existing user, with the same pwd and see the message.

Router to navigate after the registration

- open the page auth/register/page.tsx and import the userRouter

```
import Router from 'next/router';
```

- modify the onRegister function

```
function Register() {
    const router = useRouter();
    const [loading, setLoading] = React.useState(false);
    const onRegister = async (values: userType) => {
        try {
            setLoading(true);
            await axios.post("/api/auth/register", values);
            message.success("Registration successful, please login to continue");
            router.push("/auth/login");
        } catch (error: any) {
            message.error(error.response.data.message);

        } finally {
            setLoading(false);
        }
    };
```

Login API

1. create the page /api/auth/login/route.ts
2. import the jwt module installing it

```
npm i --save-dev @types/jsonwebtoken
```

```
import jwt from "jsonwebtoken";
```

3. add the jwt secret in the .env file

```
jwt_secret = myshop
```

```
export async function POST(request: NextRequest) {
    try {
        const reqBody = await request.json();

        // check if user exists in the DB or not
        const user = await User.findOne({ email: reqBody.email });
        if (!user) {
            throw new Error("User does not exist");
        }
        // password match
        const passwordMatch = await bcrypt.compare(reqBody.password, user.password);

        if (!passwordMatch) {
            throw new Error("Invalid credentials");
        }

        // create token
        const token = jwt.sign({ id: user._id }, process.env.jwt_secret!, { expiresIn: "7d" });

        const response = NextResponse.json({ message: "Login successfull", })
        response.cookies.set("token", token, {
            httpOnly: true,
            path: "/",
        });

        return response;

    } catch (error: any) {
        return NextResponse.json({
            message: error.message,
        },
            {
                status: 400
            }
        );

    }
}
```

Use the Login API

- restart the app
- modify the /auth/login/page.tsx

```
function Login() {
    const [loading, setLoading] = React.useState(false);
    const router = useRouter();
    const onLogin = async (values: userType) => {
        try {
            setLoading(true);
            await axios.post("/api/auth/login", values);
            message.success("Login successful");
            router.push("/");
        } catch (error: any) {
            message.error(error.response.data.message);
        } finally {
            setLoading(false);
        }
    };
```

add the loading to the Button element

```
  <Button type='primary' htmlType='submit' block loading={loading}>
     Login
 </Button>
```

- re-buil the app
- try all the scenarios

1. User that does not exist
2. Invalid password
3. Valid Login - Token in the inspect of the browser (Application/Cookies Tab)
4. Copy the Token and past in jwt.io web site and check in MongoDB if the ID exists

Questions?

bucchiarone@fbk.eu