|  |
| --- |
|  |
| NextJS project |
| A Javascript developer technical test |
|  |
| **AFIR Mohamed Belkacem** |
| **4/4/2025** |

|  |
| --- |
| A NextJS project about creating a mockup e-commerce store using the language NextJS and a fake API. It contains a Login page, A products display page, a page to display specific products, and variants as admin pages for adding, editing and removing products. |

Table of Contents

[1. Introduction 3](#_Toc194935509)

[1.1 Introduction: 3](#_Toc194935510)

[1.2 Scope: 3](#_Toc194935511)

[1.3 Overview: 3](#_Toc194935512)

[2. General description 3](#_Toc194935513)

[2.1 Website function: 3](#_Toc194935514)

[2.2 User characterization: 3](#_Toc194935515)

[2.3 User objective: 3](#_Toc194935516)

[3. Test objectives 3](#_Toc194935517)

[3.1 Pages to implement: 4](#_Toc194935518)

[3.1.1 Login Page: 4](#_Toc194935519)

[3.1.2 Products Page: 4](#_Toc194935520)

[3.1.3 Product Details Page: 4](#_Toc194935521)

[3.1.4 CRUD Operations: 4](#_Toc194935522)

[3.2 Additional notes: 4](#_Toc194935523)

[4. Project structure and needs 4](#_Toc194935524)

[4.1 Required libraries: 4](#_Toc194935525)

[4.1.1 Npm Install: 4](#_Toc194935526)

[4.1.2 React icons: 5](#_Toc194935527)

[4.1.3 Axios: 5](#_Toc194935528)

[4.2 Running the project: 5](#_Toc194935529)

[4.3 Files structure: 5](#_Toc194935530)

[5. Code functions 6](#_Toc194935531)

[5.1 login.js: 6](#_Toc194935532)

[5.2 MainPage.js: 9](#_Toc194935533)

[5.2 MainPageAdmin.js: 11](#_Toc194935534)

[5.3 products/[id]/page.js: 13](#_Toc194935535)

[5.5 productsAdmin/[id]/page.js: 15](#_Toc194935536)

[6. Technical issues and constriction 16](#_Toc194935537)

[6.1 fakestoreapi: 16](#_Toc194935538)

[6.2 CSS properties: 17](#_Toc194935539)

[Conclusion 18](#_Toc194935540)

# Introduction

## 1.1 Introduction:

The purpose of this document is to define the requirements of the project, as well as describe the site’s functionalities and constraints.

## 1.2 Scope:

The Provider of this test and the projects it contains is the company of Weasydoo, It was provided for the purpose of testing the programmer’s abilities and consider him for a job opportunity as a Javascript developer. This project is due on the 8th of April 2025.

## 1.3 Overview:

This project is an E-commerce store site, designed using the Javascript framework called NextJS with the help of a mock API. It contains a Products main page for displaying products with a search bar, a page for displaying specific products in full detail, variants of those pages for Admin to make changes to the products, and a login page to allow the admin to access the admin pages.

# General description

## 2.1 Website function:

The site should use NextJS to display the product list from fakestoreapi.com, mimicking an actual e-commerce store site, with also search features and login features for users.

## 2.2 User characterization:

The user will be the employees of Weasydoo for the purpose of evaluating the programmer’s Javascript skills as they use the functionalities of the site to navigate through it.

## 2.3 User objective:

The user is to evaluate the structure and functionality of the website, ensure that each component works well, that the website is responsive and well optimized, and that all of the objectives provided by the user are present.

# Test objectives

In the Next.js project, we'll be using the fakestoreapi.com API for testing and prototyping purposes. This API provides fictitious data for an online store, including products, categories and users.

## 3.1 Pages to implement:

3.1.1 Login Page: A page where users enter their username and password to access the app.

3.1.2 Products Page: A page showing all products from the API. Users can search and filter by category. Each product shows an image, title, price, and category.

3.1.3 Product Details Page: A page with more details about a selected product, including its description, rating, and reviews.

3.1.4 CRUD Operations: Users with permission can perform CRUD operations **(Create, Read, Update, Delete)** on products.

## 3.2 Additional notes:

* Use fakestoreapi.com endpoints to retrieve product, category and user data.
* Implement search and filter functionality on the product page using available endpoints.
* Ensure that CRUD operations are implemented securely and that appropriate authentication and authorization mechanisms are in place.
* Design the user interface with an emphasis on usability and responsiveness, respecting our brand guidelines.
* Use advanced techniques to optimize application performance, including minimizing loading times and reducing API response times.

# Project structure and needs

## 4.1 Required libraries:

4.1.1 Npm Install: NextJS is a ReactJs variant, and as such requires node\_modules folder to work.

Open cmd or powershell and go to the directory in which the project folder is. If the folder doesn’t contain (node\_modules) folder, then enter the folder and then type **npm install.**

$ npm install

4.1.2 React icons: The icons required for the login input icons can be installed using npm install react-icons.

$ npm install react-icons –save

4.1.3 Axios: Axios is an important library, it allows for the usage of API calls more easily. Type npm install axios.

$ npm install axios

## 4.2 Running the project:

To run the project, Go to the project folder and then type:

$ npm run dev

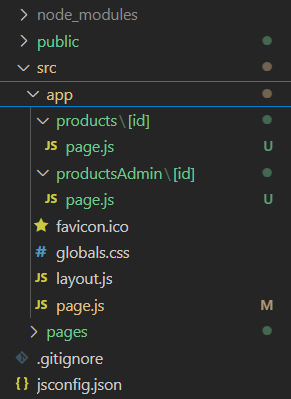
# or

$ yarn dev

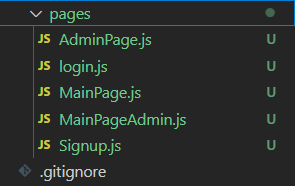
Open <http://localhost:3000> with your browser to see the result.

## 4.3 Files structure:

The files are structured as a normal NextJS project would be, there is the apps folder and the pages folder.



The apps folder is where the main display page is, the very first thing that appears in the project and the main file the /src/app/page.js file.

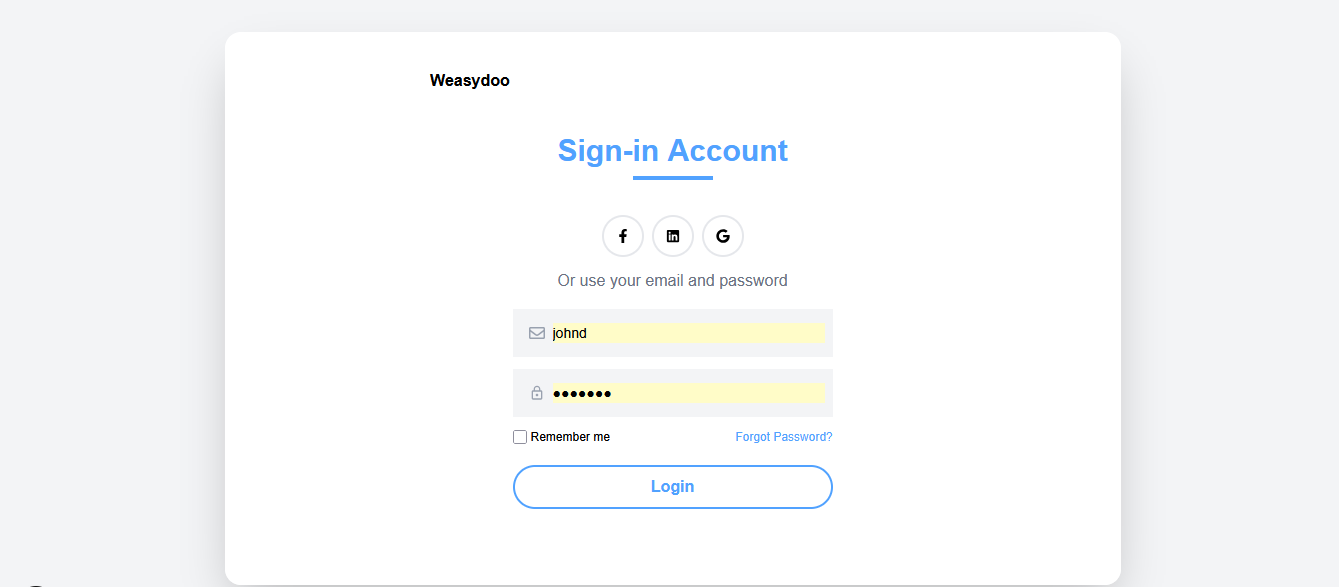


* **MainPage.js:** for the main page that displays the products without logging in.
* **Login.js:** for the login page.
* **MainPageAdmin.js:** for the admin version of the main page where editing can occur.
* **Signup.js & AdminPage.js:** obsolete files.

In the first picture there is also the **/src/apps/products/[id]/page.js** and **/src/app/productsAdmin/[id]/page.js**, these two are for when the user clicks on one of the products to be directed to the page that contains more details of the product. The first one is for the normal page, and the second one for the admin to allow them to edit the product’s details.

# Code functions

## login.js:



"use client"

import React, {useState, useEffect} from 'react';

import {FaFacebookF, FaLinkedin,FaGoogle, FaRegEnvelope} from 'react-icons/fa';

import {MdLockOutline} from 'react-icons/md';

import axios from 'axios';

import { useRouter } from 'next/navigation';

import '../app/globals.css';

function Login() {

  const router = useRouter(); //handles routing

  const [credentials, setCredentials] = useState({

    username: '',

    password: ''

  }); //stores username and password as object

  const [error, setError] = useState(""); //error variable

The imports are for important hooks, **useState** for variables and **useEffect** for when the page loads. **Axios** for API call and **useRouter** for managing links. The **icons** for icons, and the **CSS** import for tailwind CSS.

  //handles changes in input value

  const handleChange = (e) => {

    setCredentials({ ...credentials, [e.target.name]: e.target.value });

  };

  //function that handles logging in

  const handleLogin = async() =>{

    //if you want the full list of users, use this code:

    // axios.get('https://fakestoreapi.com/users')

    // .then(response => console.log(response.data))

    // example: { username: 'johnd', password: 'm38rmF$' };

    try{

      setError('');

      const response = await axios.post('https://fakestoreapi.com/auth/login', credentials, {

        headers: { 'Content-Type': 'application/json' }

      });

      const token = response.data.token //stores the login token in a variable.

      if (token){

        localStorage.setItem("authToken", token); //stores the login token in local storage.

        router.push("/MainPageAdmin"); //go to the admin page

      }

    }

    catch(error) {

      setError('Username or Password incorrect')

    }

  }

**handleChange** is the function used to change the value of the input of the username and password input HTML in the credentials variable.

The HTML for username and password inputs:

<input type="text" name="username" placeholder="User name" onChange={handleChange} className="bg-gray-100 outline-none text-sm flex-1" />

<input type="password" name="password" placeholder="Password" onChange={handleChange} className="bg-gray-100 outline-none text-sm flex-1" />

**handleLogin** is the function where upon clicking on the login button, will take the credentials variable and send it to the API, receiving a login token in response that is stored in the **localStorage**, then going to the **MainPageAdmin** page. If the login fails then the error variable will display a message through the HTML to inform the user that the username or password is incorrect.

The HTML for **handleLogin**:

            {/\* Error Message \*/}

            {error && <p className="text-red-500 text-sm">{error}</p>}

            {/\* Login Button \*/}

            <a onClick={handleLogin} className="cursor-pointer w-full max-w-xs border-2 border-blue-400 text-blue-400 px-12 py-2 rounded-full font-semibold hover:bg-blue-400 hover:text-white">

              Login

            </a>

**useEffect** checks to see if the page already has the login token, and if it does then to take the user out of the login page into the main admin page:

  //code that runs when the page loads

  useEffect(() => {

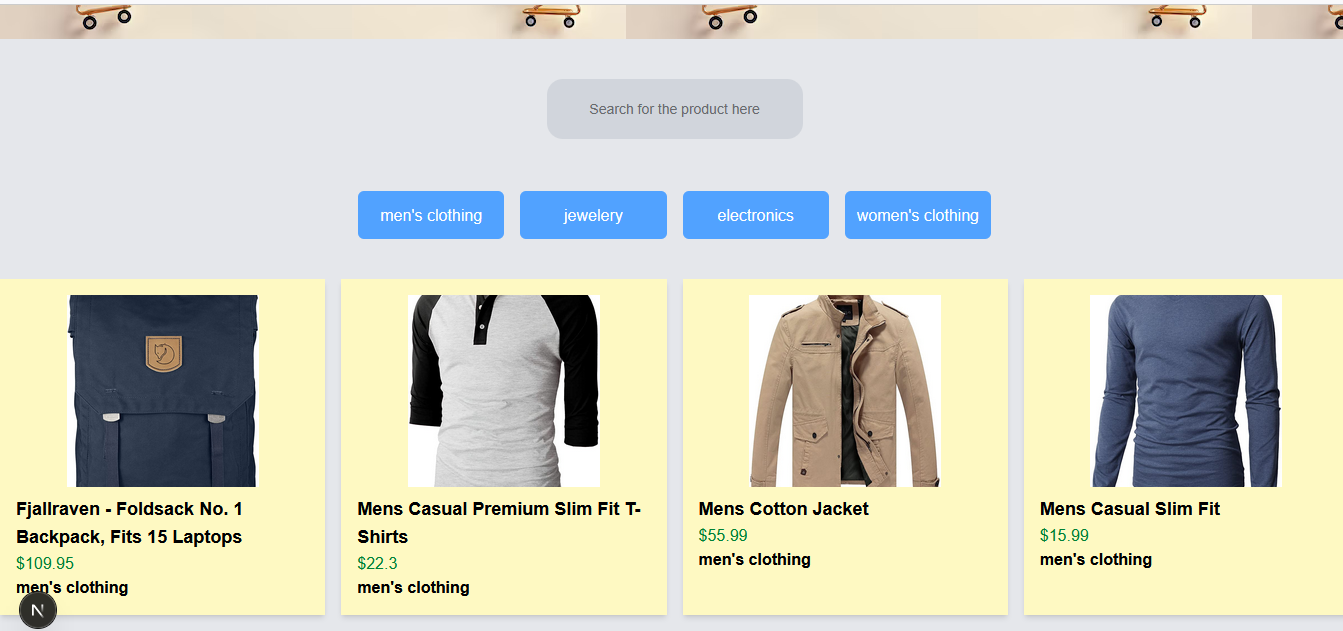
    if (localStorage.getItem("authToken")) {

      router.push("/MainPageAdmin"); //prevents user from getting into this page if they're already logged in.

    }

  }, []);

## 5.2 MainPage.js:



'use client'

import React, { useEffect, useMemo, useState } from 'react';

import '../app/globals.css';

import axios from 'axios';

import { useRouter } from 'next/navigation';

function MainPage() {

    const router = useRouter(); //handles routing

    const [items, setItems] = useState([]); //variable to store the products

    const [Query, setQuery] = useState(""); //variable for handling search filter

    const [selectedCategory, setSelectedCategory] = useState(""); //variale for categories

The important variables here are **items**, which is used to display and manage items retrieved from the API. **Query**, used for dealing with the search bar and filtering. **SelectedCategory**, used to filter items based on the category value.

    //code that runs when the page loads

    useEffect(() =>{

        apiCall();

    },[])

    //function that calls all product from the API

    const apiCall= async ()=>{

        try{

            const response = await axios.get("https://fakestoreapi.com/products");

            setItems(response.data); //puts the products in the items variable.

        }

        catch(error){

            console.error("Error fetching products:", error);

        }

    }

**useEffect** calls the **apiCall** function, which is responsible for calling all the products from the API and **setItems** into the items variable. The use of **useEffect** is to have the API call when the page is loaded.

    //function that routes to the specific product's page

    function desc(productId){

        router.push(`/products/${productId}`);

    }

//the HTML component

<h2 onClick={() => desc(item.id)} className="text-lg font-bold mt-2 cursor-pointer">{item.title}</h2>

This function is an **onClick** function, where if the user clicks on the product’s name, it will take him to the product’s page.

    //handles items filtering for search bar and categories

    const filterItems = useMemo(() => { return items.filter((item =>{

        const matchesQuery = item.title.toLowerCase().includes(Query.toLowerCase()); //for the search bar

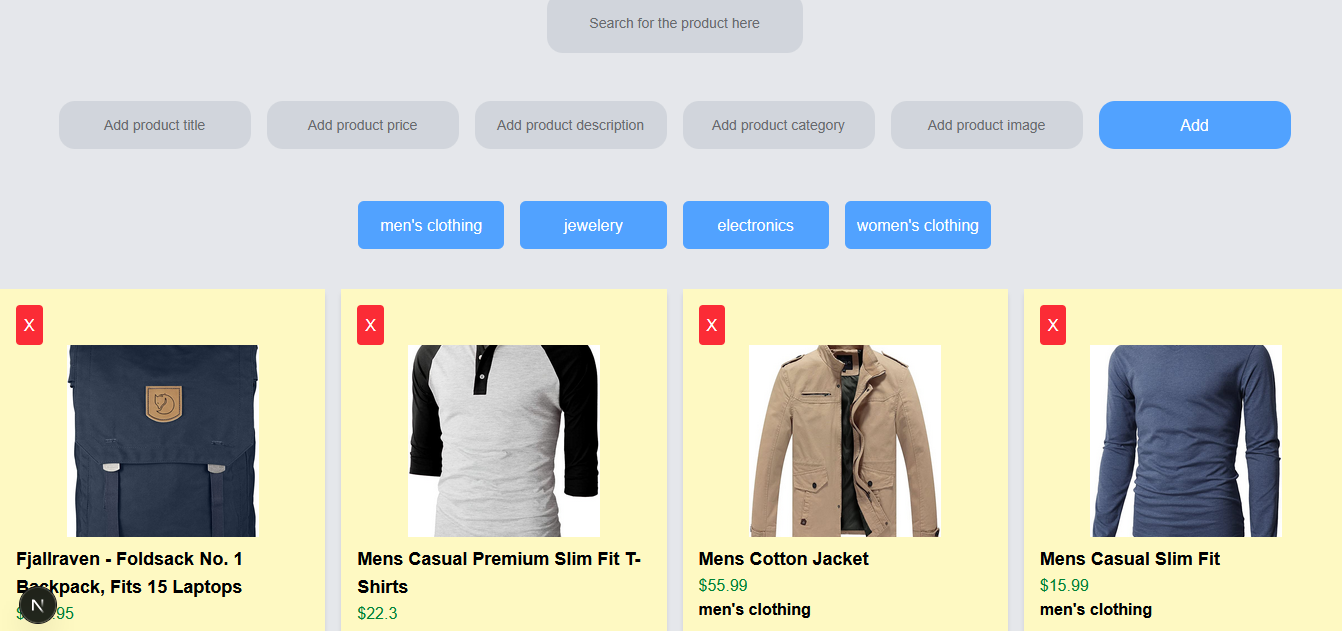
        const matchesCategory = selectedCategory ? item.category.toLowerCase() === selectedCategory.toLowerCase() : true;

        return matchesQuery && matchesCategory; //for the categories

    }))},[items,Query, selectedCategory])

This function is for filtering products depending on what was typed in the search bar. **useMemo** is to cache the values so that they’re not recalculated. **toLowerCase** allows for the search feature to not care about the casing of the letters.

## MainPageAdmin.js:



    const router = useRouter() //handles routing

    const [items, setItems] = useState([]) //variable to store the products

    const [add,setAdd] = useState({

        title: '',

        price: 0.0,

        description: '',

        category: '',

        image: '#'

    }) //variable for adding a new product

    const [Query, setQuery] = useState(""); //variable for handling search filter

    const [selectedCategory, setSelectedCategory] = useState(""); //variale for categories

    const [isAuthenticated, setIsAuthenticated] = useState(false); //for authentification

This page is similar to the **MainPage** while adding the functionalities of adding products through the add variable, a **delete** button to remove products, and **isAuthenticated** variable to ensure the user has a login token to access this page.

    //function that handles logging out

    const handleLogout = () => {

        localStorage.removeItem("authToken"); //remove the login token from local storage

        console.log("Logged out successfully");

        router.push('/'); //route to the main page

    };

**HandleLogout** removes the token from local storage and takes the user back to the original **MainPage**.

    //function that handles adding new products

    const handleAdd = async () => {

        try {

            const response = await axios.post('https://fakestoreapi.com/products', add);

            setItems(prev => [...prev, response.data]); //add the new product

            setAdd({ title: '', price: 0.0, description: '', category: '', image: '' }); //reset the add variable

        } catch (error) {

            console.error("Error adding product:", error);

        }

    };

            <div className='grid grid-cols-1 md:grid-cols-6 gap-4 my-2 mb-10'>

                <input onChange={handleChange}  type="text" name='title' placeholder='Add product title' className='bg-gray-300 text-center outline-none rounded-2xl text-sm flex-1 p-3'/>

                <input onChange={handleChange}  type="price" name='price' placeholder='Add product price' className='bg-gray-300 text-center outline-none rounded-2xl text-sm flex-1 p-3'/>

                <input onChange={handleChange}  type="description" name='description' placeholder='Add product description' className='bg-gray-300 text-center rounded-2xl outline-none text-sm flex-1 p-3'/>

                <input onChange={handleChange}  type="category" name='category' placeholder='Add product category' className='bg-gray-300 text-center outline-none rounded-2xl text-sm flex-1 p-3'/>

                <input onChange={handleChange}  type="url" name='image' placeholder='Add product image' className='bg-gray-300 text-center outline-none rounded-2xl text-sm flex-1 p-3'/>

                <button onClick={handleAdd} className='p-3 text-center bg-blue-400 text-white rounded-2xl cursor-pointer'>Add</button>

            </div>

**HandleAdd** allows for adding a new product with the data inputted in the inputs HTML.

            {[...new Set(items.map(item => item.category))].map((category, index) => (

                <span

                    key={index}

                    onClick={() => setSelectedCategory(category)}

                    className={`p-3 text-center cursor-pointer rounded-md text-white ${

                        selectedCategory === category ? 'bg-blue-600' : 'bg-blue-400'

                    }`}

                >

                    {category}

                </span>

            ))}

            {selectedCategory && (

                <button onClick={() => setSelectedCategory("")} className="bg-red-400 text-white p-2 rounded-md">

                    Clear Filter

                </button>

            )}

This HTML allows for filtering the products by categories. The second button clears the category and displays the full list with clicked.

    //function that handles deleting the product

    const handleDelete = async (id) => {

        try {

            await axios.delete(`https://fakestoreapi.com/products/${id}`)

            setItems(prevItems => prevItems.filter(item => item.id !== id)); //filter the product from the products list

        } catch (error) {

            console.error("Error adding product:", error);

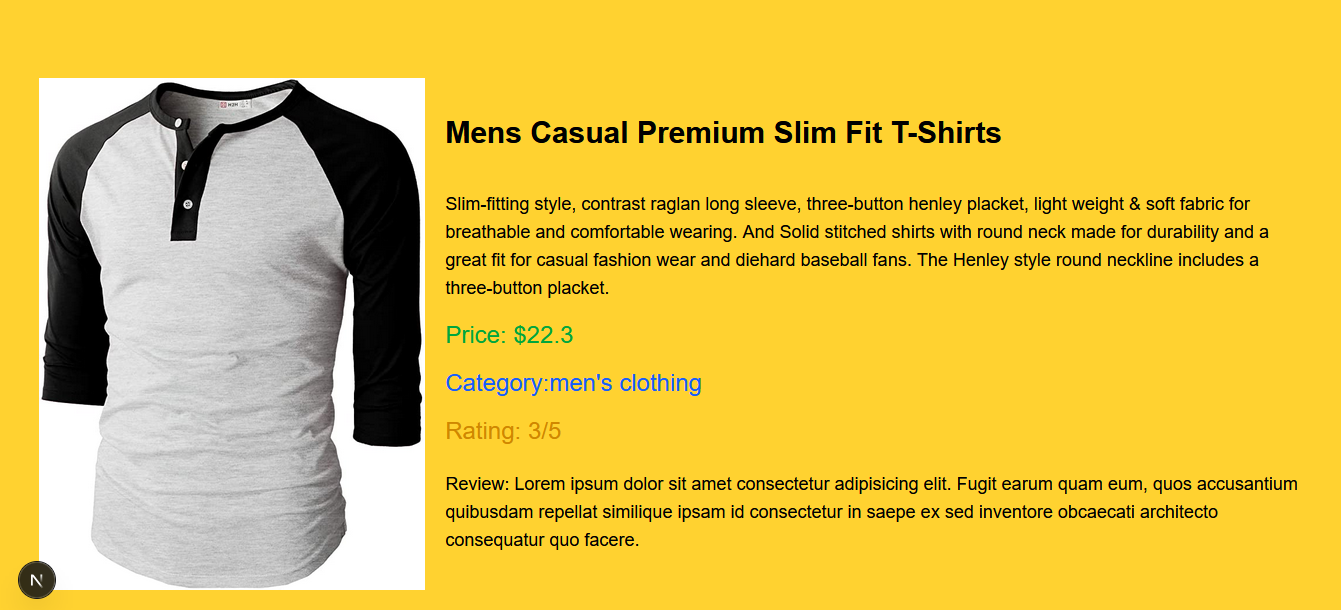
        }

    }

This function handles deleting the product from the product’s list through the HTML button.

<button className='bg-red-500 border-white p-2 text-white rounded-sm cursor-pointer' onClick={() =>handleDelete(item.id)}>X</button>

## products/[id]/page.js:



"use client";

import React, { useEffect, useState } from "react";

import { useParams, useRouter } from "next/navigation";

import axios from "axios";

function Page() {

  const params = useParams();

  const router = useRouter(); //handles routing

  const id = params.id;  //gets the id from the url

  const [product, setProduct] = useState(null); //the product's variables stored here

This page contains all the details of the product including title, description, price, category and image.

**Params** is to get the id from the product selected in **MainPage** or **MainPageAdmin**.

  //fetches the product's data

  const fetchProduct = async () => {

    try {

      const response = await axios.get(`https://fakestoreapi.com/products/${id}`);

      setProduct(response.data); //puts the product's object in the product variable

    } catch (error) {

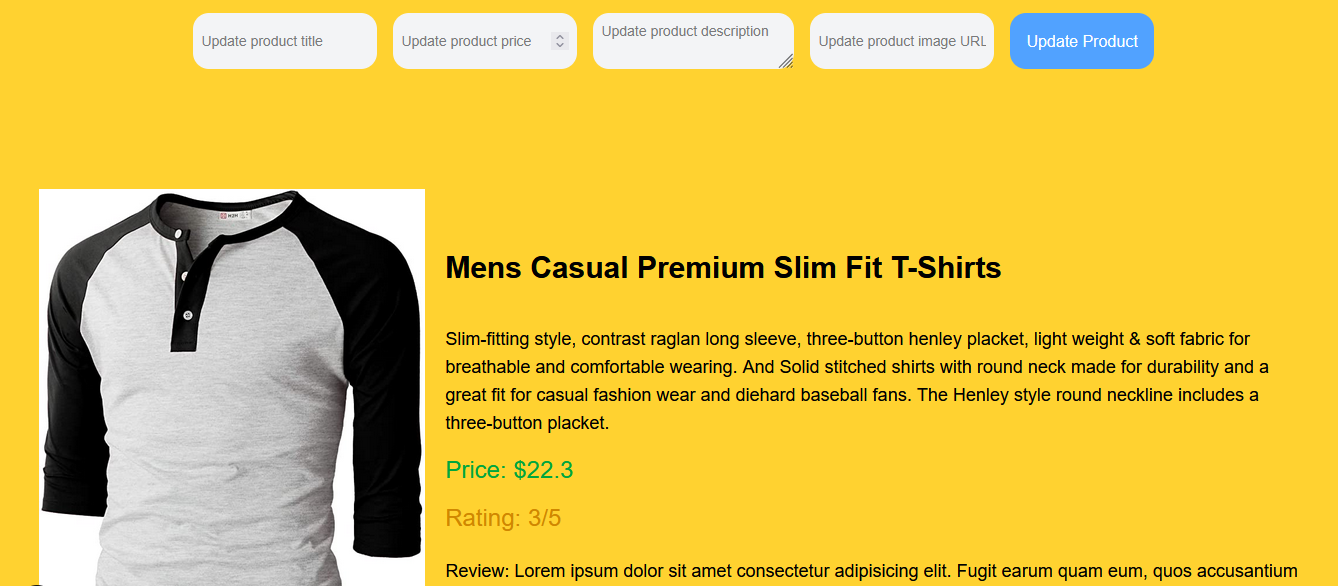
      console.error("Error fetching product:", error);

    }

  };

This fetches the specific product depending on the id from the url and places the object in the **product** variable.

## 5.5 productsAdmin/[id]/page.js:



Similar to the above page; with added input HTML for updating the product properties.

<div className="flex flex-row gap-4 mt-10 max-md:flex-col">

        <input

          type="text" name="title" placeholder="Update product title" onChange={handleChange} className="bg-gray-100 outline-none text-sm p-2 rounded-2xl"

        />

        <input

          type="number" name="price" placeholder="Update product price" onChange={handleChange} className="bg-gray-100 outline-none text-sm p-2 rounded-2xl"

        />

        <textarea

          name="description" placeholder="Update product description" onChange={handleChange} className="bg-gray-100 outline-none text-sm p-2 rounded-2xl"

        />

        <input

          type="url" name="image" placeholder="Update product image URL" onChange={handleChange} className="bg-gray-100 outline-none text-sm p-2 rounded-2xl"

        />

        <button onClick={handleUpdate} className="p-4 bg-blue-400 text-white cursor-pointer rounded-2xl">

          Update Product

        </button>

</div>

The code below handles updating the product’s parameters using the API put function. The second **setUpdate** is to clear the update inputs after updating.

  //function that handles the updating of product's details

  const handleUpdate = async () =>{

    if (!product || Object.keys(update).length === 0) return; // Don't update if no changes

    try{

      const response = await axios.put(`https://fakestoreapi.com/products/${id}`, {...product,...update});

      setProduct(response.data); //update the product variable

      setUpdate({});

    }catch (error) {

      console.error("Error fetching product:", error);

    }

  }

# Technical issues and constriction

Most constraints came from either the API being a **fakestoreapi**, and as such would not actually commit the **CRUD** operations when called, or time being limited as to not allow for different approaches to the coding or **CSS**.

## fakestoreapi:

Since the API used in the project isn’t a real API, it does not truly update itself when using CRUD operations.

  //function that handles the updating of product's details

  const handleUpdate = async () =>{

    if (!product || Object.keys(update).length === 0) return; // Don't update if no changes

    try{

      const response = await axios.put(`https://fakestoreapi.com/products/${id}`, {...product,...update});

      setProduct(response.data); //update the product variable

      setUpdate({});

    }catch (error) {

      console.error("Error fetching product:", error);

    }

  }

This code in the product page will only update the product’s properties in the specific page, but it will not update it in the main page. The **update** function of **CRUD** is in the product page as it allows for the all properties to be changed.

    //function that handles adding new products

    const handleAdd = async () => {

        try {

            const response = await axios.post('https://fakestoreapi.com/products', add);

            setItems(prev => [...prev, response.data]); //add the new product

            setAdd({ title: '', price: 0.0, description: '', category: '', image: '' }); //reset the add variable

        } catch (error) {

            console.error("Error adding product:", error);

        }

    };

    //function that handles deleting the product

    const handleDelete = async (id) => {

        try {

            await axios.delete(`https://fakestoreapi.com/products/${id}`)

            setItems(prevItems => prevItems.filter(item => item.id !== id)); //filter the product from the products list

        } catch (error) {

            console.error("Error adding product:", error);

        }

    }

These two functions in the **MainPageAdmin** also have the same problem, it was determined that this is a test and it’s more important to showcase that these **CRUD** functions work properly.

## 6.2 CSS properties:

<div className="grid grid-cols-2 md:grid-cols-4 gap-4">

                {items.length >0?(

                    filterItems.map((item)=>(

                        <div key={item.id} className="bg-yellow-100 p-4 shadow-md">

                             <button className='bg-red-500 border-white p-2 text-white rounded-sm cursor-pointer' onClick={() =>handleDelete(item.id)}>X</button>

                            <img src={item.image} alt={item.title} loading="lazy" className="w-48 h-48 object-cover mx-auto" />

                            <h2 onClick={() => desc(item.id)} className="text-lg font-bold mt-2 cursor-pointer">{item.title}</h2>

                            <p className="text-green-700">${item.price}</p>

                            <p className='font-bold'>{item.category}</p>

                        </div>

                    ))

                ):(

                    <p>Loading products....</p>

                )}

</div>

The code above is for the HTML and CSS elements of **MainPageAdmin**. The inclusion of the **handleDelete** element allowed for easier **Delete** function of the product.

The initial plan for the **Update** function was for the product panel to be clickable on a certain button that would change the elements into an editable state. This proved not just a challenge but also limiting as not all **Update** elements are present in the main page.

# Conclusion

The project is a simple e-commerce storefront, creating it in NextJS provided some advantages over creating it in normal ReactJS template. The test included heavy focus on front-end elements and how they communicate with the back-end API.

Further improvements:

* Improving CSS usage.
* Further usage of local storage.
* Further optimization and simplification of code.

Acknowledgement:

* (https://fakestoreapi.com/) for dummy product data.
* React Native and Axios for the core foundation of this project.
* Weasydoo for providing this test.