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

|  |
| --- |
| Un projet NextJS visant à créer une maquette de magasin e-commerce en utilisant le langage NextJS et une API fictive. Il comprend une page de connexion, une page d'affichage des produits, une page pour afficher des produits spécifiques, ainsi que des variantes en tant que pages administratives pour ajouter, modifier et supprimer des produits. |

Table of Contents

[1. Introduction 3](#_Toc194941982)

[1.1 Introduction: 3](#_Toc194941983)

[1.2 Portée: 3](#_Toc194941984)

[1.3 Aperçu: 3](#_Toc194941985)

[2. General description 3](#_Toc194941986)

[2.1 Fonctionnalité du site: 3](#_Toc194941987)

[2.2 User characterization: 3](#_Toc194941988)

[2.3 User objective: 3](#_Toc194941989)

[3. Test objectives 4](#_Toc194941990)

[3.1 Pages to implement: 4](#_Toc194941991)

[3.1.1 Page de connexion: 4](#_Toc194941992)

[3.1.2 Page des produits: 4](#_Toc194941993)

[3.1.3 Page de détails du produit: 4](#_Toc194941994)

[3.1.4 Opérations CRUD: 4](#_Toc194941995)

[3.2 Notes supplémentaires: 4](#_Toc194941996)

[4. Structure et besoins du projet 4](#_Toc194941997)

[4.1 Bibliothèques requises: 4](#_Toc194941998)

[4.1.1 Npm Install: 4](#_Toc194941999)

[4.1.2 React icons: 5](#_Toc194942000)

[4.1.3 Axios: 5](#_Toc194942001)

[4.2 Exécution du projet: 5](#_Toc194942002)

[4.3 Structure des fichiers: 5](#_Toc194942003)

[5. Fonctions du code 7](#_Toc194942004)

[5.1 login.js: 7](#_Toc194942005)

[5.2 MainPage.js: 9](#_Toc194942006)

[5.2 MainPageAdmin.js: 11](#_Toc194942007)

[5.3 products/[id]/page.js: 14](#_Toc194942008)

[5.5 productsAdmin/[id]/page.js: 15](#_Toc194942009)

[6. Technical issues and constriction 16](#_Toc194942010)

[6.1 fakestoreapi: 16](#_Toc194942011)

[6.2 CSS properties: 17](#_Toc194942012)

[Conclusion 18](#_Toc194942013)

# Introduction

## 1.1 Introduction:

L’objectif de ce document est de définir les exigences du projet, ainsi que de décrire les fonctionnalités et les contraintes du site.

## 1.2 Portée:

Le fournisseur de ce test, ainsi que des projets qu’il contient, est l’entreprise Weasydoo. Ce test a été fourni dans le but d’évaluer les compétences du programmeur et de le considérer pour une opportunité d’emploi en tant que développeur Javascript. Ce projet doit être rendu au plus tard le 8 avril 2025.

## 1.3 Aperçu:

Ce projet est un site de magasin e-commerce, conçu en utilisant le framework Javascript appelé NextJS avec l'aide d'une API fictive. Il contient une page principale des produits pour afficher les produits avec une barre de recherche, une page pour afficher des produits spécifiques en détail, des variantes de ces pages pour que l'administrateur puisse apporter des modifications aux produits, et une page de connexion permettant à l'administrateur d'accéder aux pages administratives.

# General description

## 2.1 Fonctionnalité du site:

Le site doit utiliser NextJS pour afficher la liste des produits provenant de fakestoreapi.com, imitant ainsi un véritable site de magasin e-commerce, avec des fonctionnalités de recherche et de connexion pour les utilisateurs.

## 2.2 User characterization:

L'utilisateur sera les employés de Weasydoo dans le but d'évaluer les compétences en Javascript du programmeur, alors qu'ils utilisent les fonctionnalités du site pour naviguer à travers celui-ci.

## 2.3 User objective:

L'utilisateur doit évaluer la structure et la fonctionnalité du site web, s'assurer que chaque composant fonctionne bien, que le site est réactif et bien optimisé, et que tous les objectifs fournis par l'utilisateur sont présents.

# Test objectives

Dans le projet Next.js, nous utiliserons l'API fakestoreapi.com à des fins de test et de prototypage. Cette API fournit des données fictives pour un magasin en ligne, y compris des produits, des catégories et des utilisateurs.

## 3.1 Pages to implement:

3.1.1 Page de connexion: Une page où les utilisateurs saisissent leur nom d'utilisateur et leur mot de passe pour accéder à l'application.

3.1.2 Page des produits: Une page affichant tous les produits de l'API. Les utilisateurs peuvent rechercher et filtrer par catégorie. Chaque produit affiche une image, un titre, un prix et une catégorie.

3.1.3 Page de détails du produit: Une page avec plus de détails sur un produit sélectionné, y compris sa description, sa note et ses avis.

3.1.4 Opérations CRUD: Les utilisateurs autorisés peuvent effectuer des opérations **CRUD (Créer, Lire, Mettre à jour, Supprimer)** sur les produits.

## 3.2 Notes supplémentaires:

* Utiliser les points de terminaison de fakestoreapi.com pour récupérer les données des produits, des catégories et des utilisateurs.
* Implémenter la fonctionnalité de recherche et de filtrage sur la page des produits en utilisant les points de terminaison disponibles.
* S'assurer que les opérations CRUD sont implémentées de manière sécurisée et que des mécanismes appropriés d'authentification et d'autorisation sont en place.
* Concevoir l'interface utilisateur en mettant l'accent sur l'ergonomie et la réactivité, en respectant nos directives de marque.
* Utiliser des techniques avancées pour optimiser les performances de l'application, notamment en minimisant les temps de chargement et en réduisant les temps de réponse de l'API.

# Structure et besoins du projet

## 4.1 Bibliothèques requises:

4.1.1 Npm Install: React Native est une variante de ReactJS destinée à la création d’applications mobiles, et nécessite donc le dossier **node\_modules** pour fonctionner.

Ouvrez l’invite de commande (cmd) ou PowerShell et accédez au répertoire contenant le dossier du projet. Si ce dossier ne contient pas de dossier (node\_modules) , entrez dans le dossier du projet, puis tapez la commande suivante: **npm install.**

$ npm install

4.1.2 React icons: Les icônes nécessaires pour les champs de saisie de la page de connexion peuvent être installées à l’aide de la commande suivante: npm install react-icons.

$ npm install react-icons –save

4.1.3 Axios: Axios est une bibliothèque importante, car elle permet de faciliter l’utilisation des appels à des API. Pour l’installer, tapez la commande suivante: npm install axios.

$ npm install axios

## 4.2 Exécution du projet:

Pour exécuter le projet, accédez au dossier du projet, puis tapez la commande suivante:

$ npm run dev

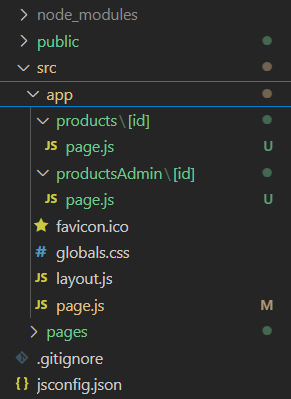
# or

$ yarn dev

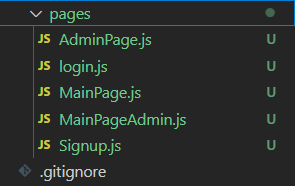
Ouvrez <http://localhost:8081> dans votre navigateur pour voir le résultat.

## 4.3 Structure des fichiers:

Les fichiers sont structurés comme dans un projet NextJS classique. Il y a un dossier **apps** et un dossier **pages**.



Le dossier **apps** est l'endroit où se trouve la page d'affichage principale, c'est la première chose qui apparaît dans le projet, ainsi que le fichier principal **/src/app/page.js**.

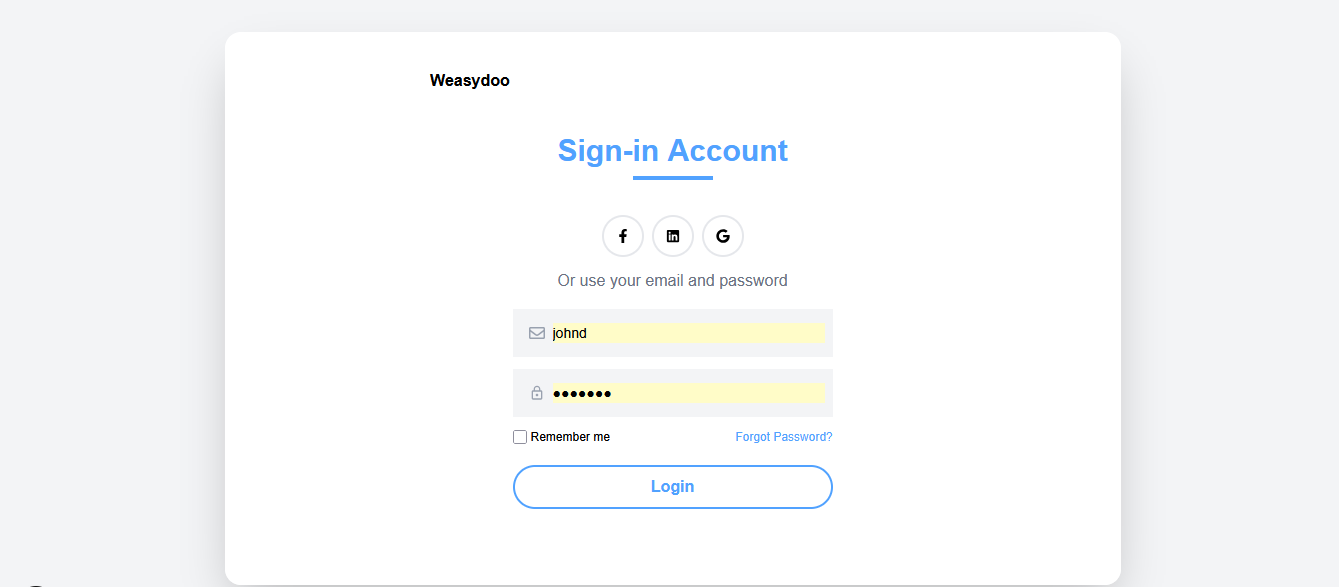


* **MainPage.js** : pour la page principale qui affiche les produits sans connexion.
* **Login.js** : pour la page de connexion.
* **MainPageAdmin.js** : pour la version administrateur de la page principale où l'édition peut avoir lieu.
* **Signup.js** et **AdminPage.js** : fichiers obsolètes.

Dans la première image, il y a également **/src/apps/products/[id]/page.js** et **/src/app/productsAdmin/[id]/page.js**. Ces deux fichiers sont utilisés lorsque l'utilisateur clique sur un des produits pour être redirigé vers la page contenant plus de détails sur le produit. Le premier est pour la page normale, et le second est destiné à l'admin pour lui permettre de modifier les détails du produit.

# Fonctions du code

## 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

Les imports sont pour des hooks importants : **useState** pour les variables et **useEffect** pour les actions lors du chargement de la page. **Axios** pour les appels API et **useRouter** pour la gestion des liens. Les **icônes** pour les icônes, et l'import **CSS** pour le CSS de Tailwind.

  //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** est la fonction utilisée pour modifier la valeur des champs de saisie pour le nom d'utilisateur et le mot de passe dans la variable **credentials**.

Le code HTML pour les champs de saisie du nom d'utilisateur et du mot de passe :

<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** est la fonction qui, lorsqu'on clique sur le bouton de connexion, prend la variable des informations de connexion et l'envoie à l'API. En réponse, un token de connexion est reçu et stocké dans le localStorage, puis l'utilisateur est redirigé vers la page **MainPageAdmin**. Si la connexion échoue, la variable **error** affichera un message via le HTML pour informer l'utilisateur que le nom d'utilisateur ou le mot de passe est incorrect.

**handleLogin** HTML:

            {/\* 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** vérifie si la page possède déjà le token de connexion. Si c'est le cas, elle redirige l'utilisateur hors de la page de connexion vers la page principale de l'administrateur:

  //code that runs when the page loads

  useEffect(() => {

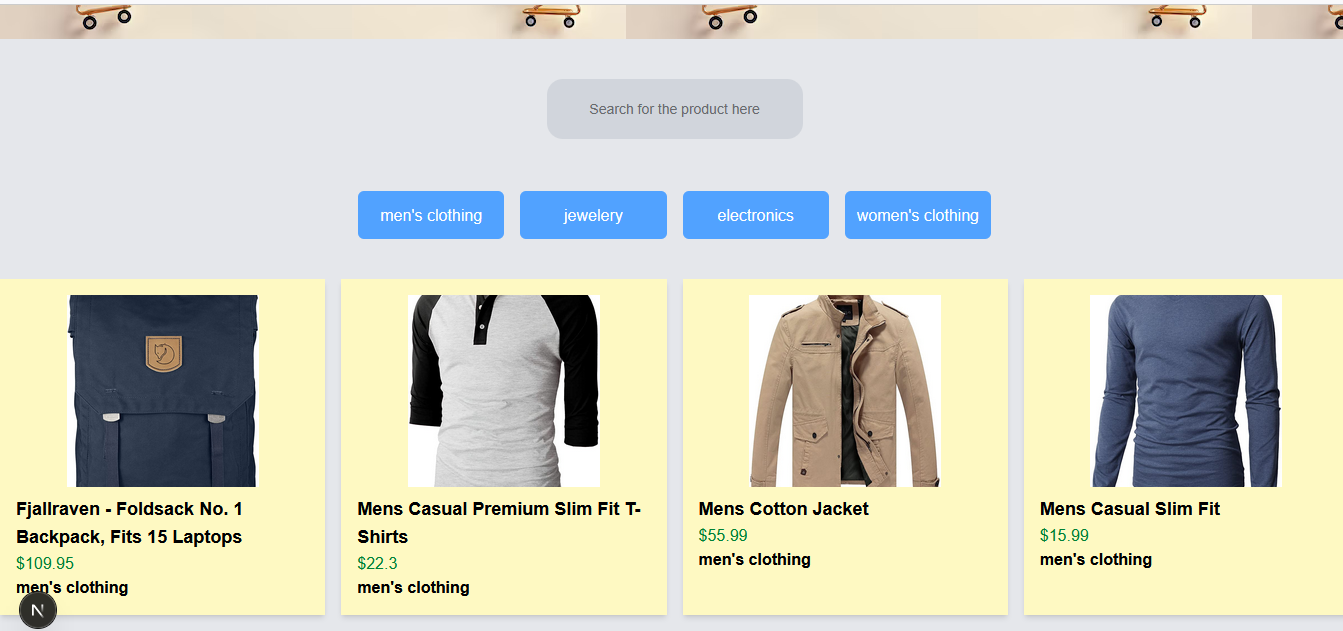
    if (localStorage.getItem("authToken")) {

      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

Les variables importantes ici sont **items**, qui est utilisée pour afficher et gérer les articles récupérés depuis l'API. **Query**, utilisée pour gérer la barre de recherche et le filtrage. **SelectedCategory**, utilisée pour filtrer les articles en fonction de la catégorie sélectionnée.

    //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** appelle la fonction **apiCall**, qui est responsable de récupérer tous les produits depuis l'API et de les stocker dans la variable **items**. L'utilisation de **useEffect** permet d'exécuter l'appel à l'API lorsque la page est chargée.

    //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>

Cette fonction est une fonction **onClick**, où si l'utilisateur clique sur le nom du produit, il sera redirigé vers la page du produit.

    //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

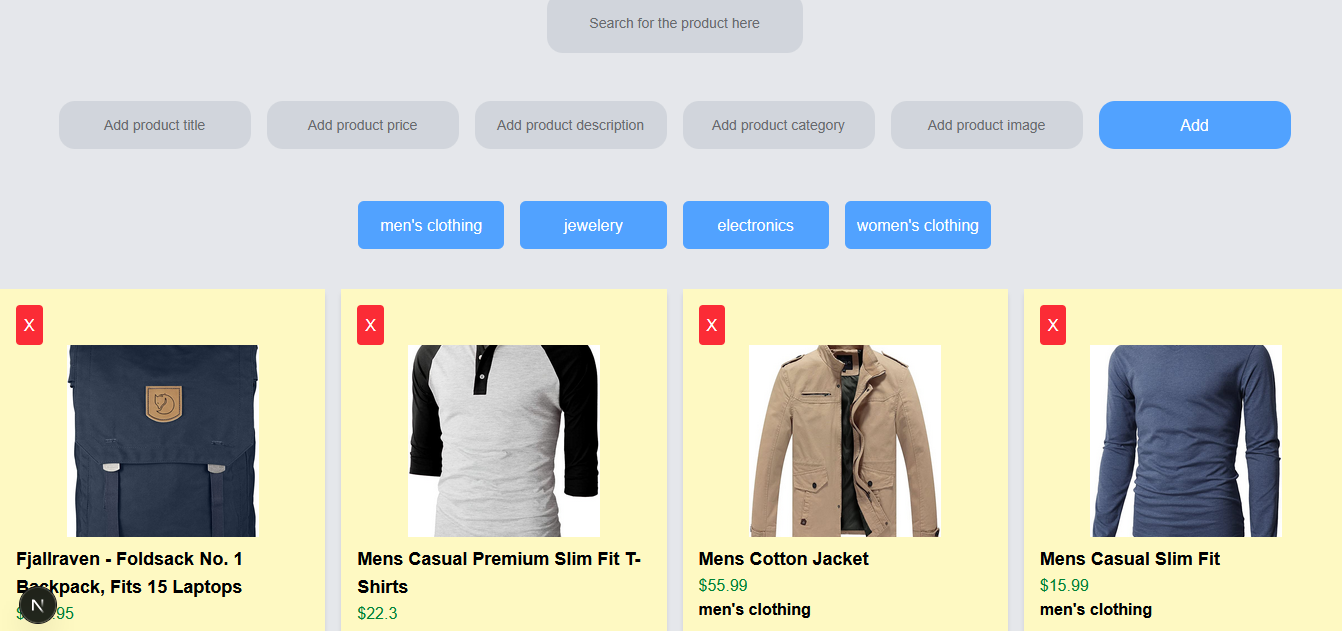
        const matchesCategory = selectedCategory ? item.category.toLowerCase() === selectedCategory.toLowerCase() : true;

        return matchesQuery && matchesCategory; //for the categories

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

Cette fonction sert à filtrer les produits en fonction de ce qui a été tapé dans la barre de recherche. **useMemo** permet de mettre en cache les valeurs afin qu'elles ne soient pas recalculées. **toLowerCase** permet à la fonctionnalité de recherche de ne pas tenir compte de la casse des lettres.

## 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

Cette page est similaire à la **MainPage** tout en ajoutant les fonctionnalités d'ajout de produits via la variable **add**, un bouton de suppression pour retirer des produits, et la variable **isAuthenticated** pour s'assurer que l'utilisateur possède un jeton de connexion afin d'accéder à cette 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** supprime le jeton du stockage local et redirige l'utilisateur vers la **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** permet d'ajouter un nouveau produit avec les données saisies dans les champs de saisie 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>

            )}

Ce code HTML permet de filtrer les produits par catégories. Le second bouton efface la catégorie et affiche la liste complète lorsque cliqué.

    //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);

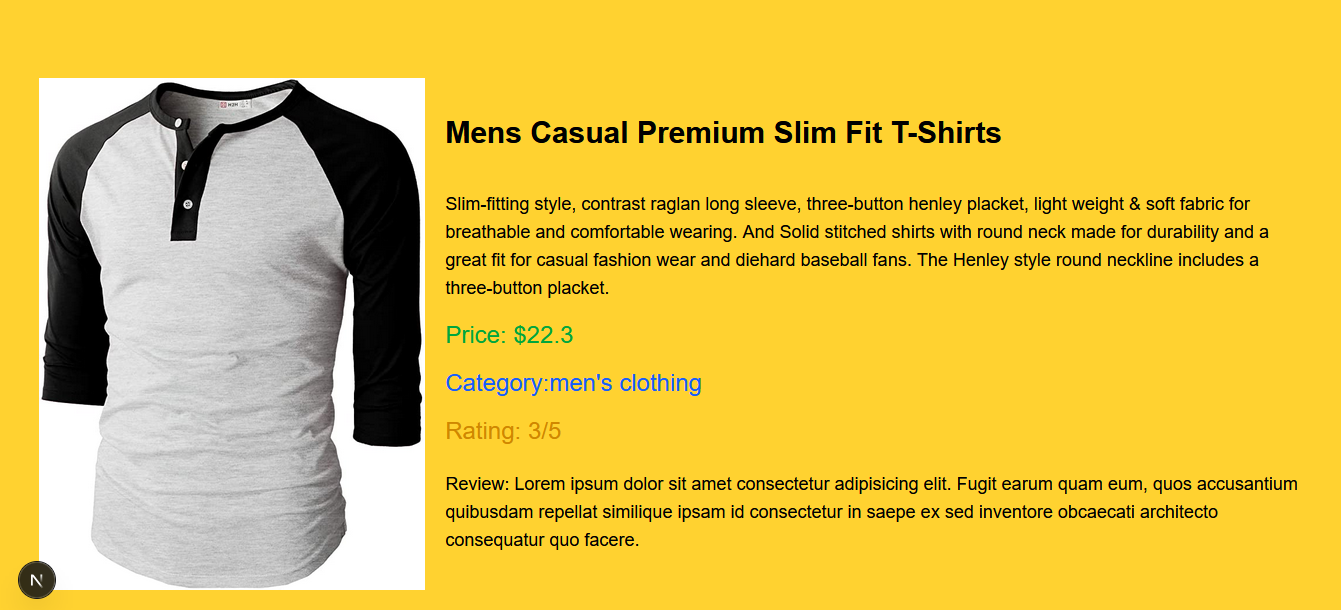
        }

    }

Cette fonction gère la suppression du produit de la liste des produits via le bouton HTML.

<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

Cette page contient tous les détails du produit, y compris le titre, la description, le prix, la catégorie et l'image.

**Params** sert à obtenir l'ID du produit sélectionné dans la **MainPage** ou la **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) {

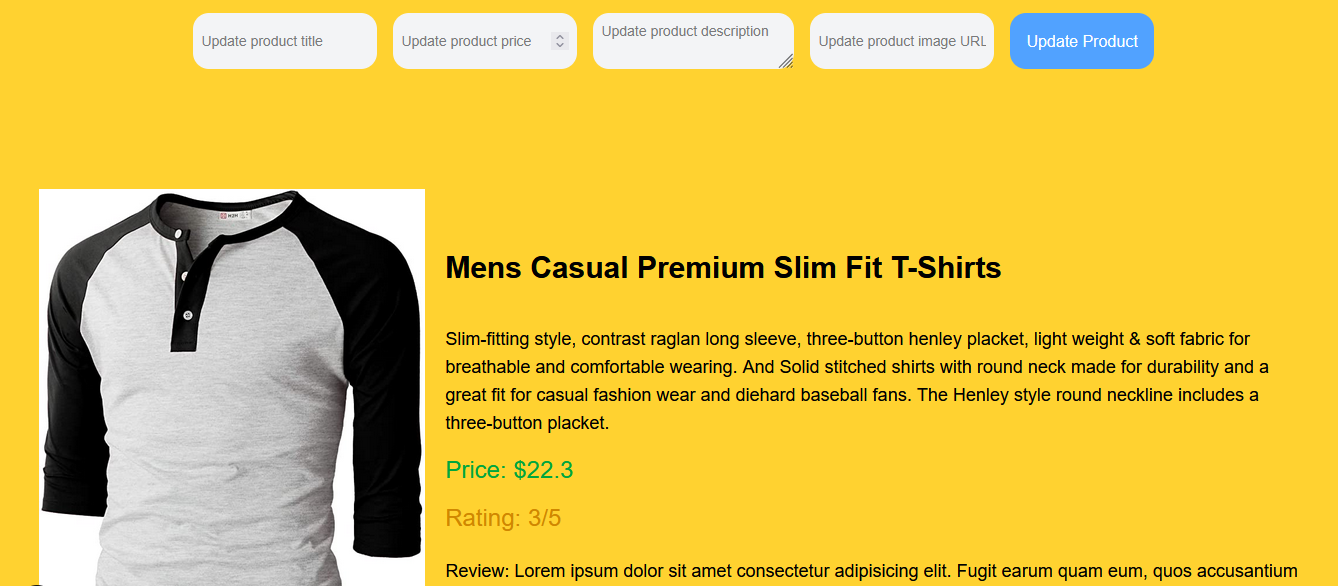
      console.error("Error fetching product:", error);

    }

  };

Cela récupère le produit spécifique en fonction de l'ID dans l'URL et place l'objet dans la variable **products**.

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



Similaire à la page ci-dessus, avec l'ajout de balises HTML d'entrée pour mettre à jour les propriétés du produit.

<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>

Le code ci-dessous gère la mise à jour des paramètres du produit en utilisant la fonction API PUT. Le deuxième **setUpdate** est utilisé pour vider les champs de mise à jour après la mise à jour.

  //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

La plupart des contraintes proviennent soit du fait que l'API utilisée est une **fakestoreapi**, qui ne permet pas réellement de valider les opérations **CRUD** lorsqu'elles sont appelées, soit du temps limité qui ne permettait pas d'explorer d'autres approches pour le codage ou le **CSS**.

## fakestoreapi:

Étant donné que l'API utilisée dans le projet n'est pas une API réelle, elle ne se met pas réellement à jour lorsqu'on effectue des opérations CRUD.

  //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);

    }

  }

Ce code dans la page du produit ne mettra à jour les propriétés du produit que sur la page spécifique, mais il ne le mettra pas à jour sur la page principale. La fonction d’update du **CRUD** se trouve dans la page du produit, car elle permet de modifier toutes les propriétés.

    //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);

        }

    }

Ces deux fonctions dans la page **MainPageAdmin** rencontrent également le même problème. Il a été déterminé que cela fait partie d'un test, et l'objectif principal est de montrer que ces fonctions **CRUD** fonctionnent correctement.

## 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>

Le code ci-dessus concerne les éléments HTML et CSS de **MainPageAdmin**. L'inclusion de l'élément **handleDelete** a permis de simplifier la fonction de suppression du produit.

Le plan initial pour la fonction **update** consistait à ce que le panneau du produit soit cliquable sur un bouton spécifique, ce qui permettrait de changer les éléments en un état modifiable. Cela s'est avéré non seulement un défi, mais aussi limitatif, car tous les éléments à mettre à jour ne sont pas présents sur la page principale.

# Conclusion

Le projet est une vitrine de commerce électronique simple. Le fait de le créer en NextJS a offert certains avantages par rapport à la création d'un projet dans un modèle ReactJS classique. Le test a mis l'accent sur les éléments front-end et leur communication avec l'API back-end.

**Améliorations futures:**

* Améliorer l'utilisation de CSS.
* Utilisation accrue du stockage local.
* Optimisation et simplification supplémentaires du code.

Remerciements:

* (https://fakestoreapi.com/) pour les données fictives des produits.
* React Native et Axios pour la base de ce projet.
* Weasydoo pour avoir fourni ce test.