|  |
| --- |
|  |
| React Native project |
| A Javascript developer technical test |
|  |
| **AFIR Mohamed Belkacem** |
| **5/4/2025** |

|  |
| --- |
| Un projet React Native créé en utilisant le code du projet NextJS pour créer une application mobile. Le projet doit être optimisé et réactif pour fonctionner rapidement et charger tous les produits depuis une API, de manière similaire au projet NextJS. |

Contents

[1. Introduction 3](#_Toc194940117)

[1.1 Introduction: 3](#_Toc194940118)

[1.2 Portée: 3](#_Toc194940119)

[1.3 Aperçu: 3](#_Toc194940120)

[2. General description 3](#_Toc194940121)

[2.1 Fonctionnalité du site: 3](#_Toc194940122)

[2.2 User characterization: 3](#_Toc194940123)

[2.3 User objective: 3](#_Toc194940124)

[3. Objectifs du test 3](#_Toc194940125)

[3.1 Pages à implémenter: 4](#_Toc194940126)

[3.1.1 Application mobile: 4](#_Toc194940127)

[3.1.2 Interface utilisateur adaptative: 4](#_Toc194940128)

[3.1.3 Optimisation des performances: 4](#_Toc194940129)

[4. Structure et besoins du projet 4](#_Toc194940130)

[4.1 Bibliothèques requises: 4](#_Toc194940131)

[4.1.1 Npm Install: 4](#_Toc194940132)

[4.1.2 React icons: 4](#_Toc194940133)

[4.1.3 Axios: 4](#_Toc194940134)

[4.1.4 asyncStorage: 4](#_Toc194940135)

[4.1.5 useNavigation: 5](#_Toc194940136)

[4.2 Exécution du projet: 5](#_Toc194940137)

[4.3 Structure des fichiers: 5](#_Toc194940138)

[5. Fonctions du code 7](#_Toc194940139)

[5.1 login.js: 7](#_Toc194940140)

[5.2 MainPage.js: 10](#_Toc194940141)

[5.2 MainPageAdmin.js: 13](#_Toc194940142)

[5.3 products/[id]/page.js: 16](#_Toc194940143)

[5.5 productsAdmin/[id]/page.js: 18](#_Toc194940144)

[5.6 app.js: 19](#_Toc194940145)

[6. Problèmes techniques et contraintes 20](#_Toc194940146)

[6.1 fakestoreapi: 20](#_Toc194940147)

[6.2 CSS properties: 21](#_Toc194940148)

[Conclusion 22](#_Toc194940149)

# 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 une application mobile de boutique e-commerce, conçue à l’aide du framework Javascript appelé React Native, avec l’appui d’une API factice (mock API). Il comprend une page principale de produits avec une barre de recherche, une page pour afficher les détails complets d’un produit spécifique, des variantes de ces pages permettant à l’administrateur de modifier les produits, ainsi qu’une page de connexion permettant à l’administrateur d’accéder aux pages d’administration.

# General description

## 2.1 Fonctionnalité du site:

L’application doit utiliser React Native pour afficher la liste des produits provenant de fakestoreapi.com, en simulant un véritable site de boutique en ligne. Elle doit également inclure des fonctionnalités de recherche et de connexion pour les utilisateurs.

## 2.2 User characterization:

Les utilisateurs seront les employés de Weasydoo, dans le but d’évaluer les compétences du programmeur en Javascript, en utilisant les fonctionnalités du site pour y naviguer.

## 2.3 User objective:

L’utilisateur doit évaluer la structure et le fonctionnement de l’application mobile, s’assurer que chaque composant fonctionne correctement, que l’application est réactive et bien optimisée, et que tous les objectifs définis par l’utilisateur sont atteints.

# Objectifs du test

Dans le projet React Native, nous utiliserons l’API de fakestoreapi.com à des fins de test et de prototypage. Cette API fournit des données fictives pour une boutique en ligne, incluant des produits, des catégories et des utilisateurs.

## 3.1 Pages à implémenter:

3.1.1 Application mobile: Convertir l’application existante développée avec Next.js en une application mobile utilisant React Native. Reproduire fidèlement toutes les fonctionnalités du site web.

3.1.2 Interface utilisateur adaptative: Concevoir une interface utilisateur adaptative pour les appareils mobiles, en tenant compte des spécificités de la navigation et de l’interaction sur smartphones et tablettes.

3.1.3 Optimisation des performances: Optimiser les performances de l’application mobile afin d’assurer des temps de chargement rapides et une réactivité fluide, même avec des connexions réseau faibles.

# 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

### 4.1.4 asyncStorage:

Cette bibliothèque est utile pour stocker des chaînes de caractères, telles que le jeton d’authentification, dans le stockage local, ainsi que pour les extraire ou les supprimer. Pour l’installer, tapez : npm install @react-native-async-storage/async-storage

$ npm install @react-native-async-storage/async-storage

4.1.5 useNavigation: Cette bibliothèque est utilisée pour naviguer entre les pages dans une application React Native. Pour l’installer, tapez: npm install @react-navigation/native @react-navigation/native-stack

npm install @react-navigation/native @react-navigation/native-stack

## 4.2 Exécution du projet:

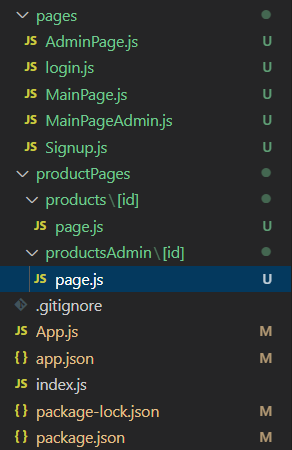
Pour exécuter le projet, accédez au dossier du projet, puis tapez la commande suivante:

$ npm run web

Ouvrez [http://localhost:8081](http://localhost:8081" \t "_new) dans votre navigateur pour voir le résultat.

## 4.3 Structure des fichiers:

Les fichiers sont structurés de manière similaire à un projet NextJS, avec quelques différences mineures : il y a un dossier **apps** et un dossier **pages**.



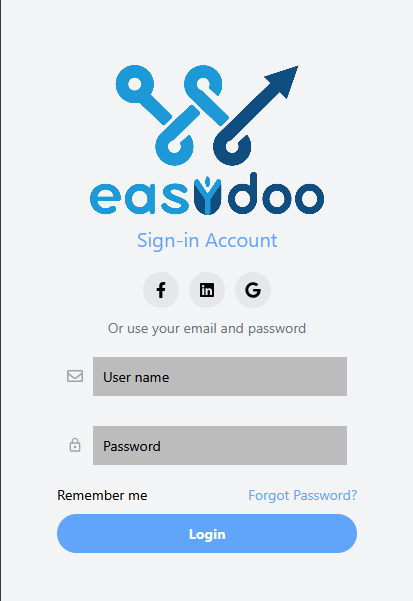
Il y a deux dossiers qui contiennent la majeure partie du code : le dossier **pages** et le dossier **productPages**. Le dossier **pages** contient les pages principales de l'application, tandis que le dossier **productPages** contient les pages qui affichent les produits en fonction de leur ID.

* **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 aussi les fichiers **/productPage/products/[id]/page.js** et **/productPage/productsAdmin/[id]/page.js**. Ces deux fichiers sont utilisés lorsque l'utilisateur clique sur l'un des produits pour être redirigé vers la page contenant plus de détails sur le produit. Le premier est pour la page classique, et le deuxième est destiné à l’administrateur afin de lui permettre d’éditer les détails du produit.

# Fonctions du code

## login.js:



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

import { View, Text, TextInput, TouchableOpacity, StyleSheet, Image } from 'react-native';

import { useNavigation } from '@react-navigation/native';

import axios from 'axios';

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

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

import AsyncStorage from '@react-native-async-storage/async-storage';

export default function Login() {

  const navigation = useNavigation(); //for navigating between pages

  const [credentials, setCredentials] = useState({

    username: '',

    password: ''

  }); //variable that stores username and password

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

Les imports sont utilisés pour des hooks importants: **useState** pour les variables. **useEffect** pour les actions à réaliser lorsque la page se charge. **Axios** pour les appels API. **useNavigation** pour gérer les liens de navigation. **Les icônes** pour l'affichage des icônes. **asyncStorage** pour stocker des chaînes de caractères dans le stockage local.

  //function for handling the input from username and password inputs

  const handleChange = (name, value) => {

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

  };

  //function that logs the user in

  const handleLogin = async () => {

    try {

      setError('');

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

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

      });

      const token = response.data.token; //takes the login token from the response

      if (token) {

        await AsyncStorage.setItem("authToken", token); //add the token to the local storage

        navigation.navigate("MainAdmin"); //navigate to the admin page

      }

    } catch (error) {

      setError('Username or Password incorrect');

    }

  };

**handleChange** est la fonction utilisée pour changer la valeur des champs de saisie du nom d'utilisateur et du mot de passe dans la variable credentials.

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

        <View style={styles.inputBox}>

          <FaRegEnvelope style={styles.iconInline} />

          <TextInput placeholder="User name" style={styles.input} onChangeText={text => handleChange('username', text)}/>

        </View>

        <View style={styles.inputBox}>

          <MdLockOutline style={styles.iconInline} />

          <TextInput placeholder="Password" secureTextEntry style={styles.input} onChangeText={text => handleChange('password', text)}/>

        </View>

**handleLogin** est la fonction qui, lorsqu'on clique sur le bouton de connexion, prend la variable **credentials**, l'envoie à **l'API**, et reçoit un jeton de connexion en réponse, qui est ensuite stocké dans le **localStorage**. Ensuite, 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:

        {/\* if username or password is wrong \*/}

        {error ? <Text style={styles.error}>{error}</Text> : null}

        {/\* login button \*/}

        <TouchableOpacity style={styles.loginButton} onPress={handleLogin}>

          <Text style={styles.loginButtonText}>Login</Text>

        </TouchableOpacity>

**useEffect** vérifie si la page dispose déjà du jeton de connexion. Si c'est le cas, il redirige l'utilisateur hors de la page de connexion vers la page principale administrateur:

  //code that runs when the page loads

  useEffect(() => {

    //gets the list of users ex. username: johnd, password:m38rmF$

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

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

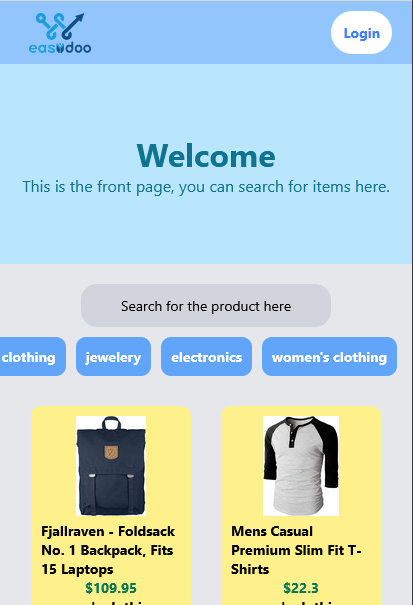
    AsyncStorage.getItem("authToken").then(token => {

      if (token) navigation.navigate("MainAdmin"); //prevents the user from accessing this page if they're already logged in.

    });

  }, []);

## 5.2 MainPage.js:



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

import { View, Text, TextInput, Image, TouchableOpacity, ScrollView, StyleSheet } from 'react-native';

import axios from 'axios';

import { useNavigation } from '@react-navigation/native';

import AsyncStorage from '@react-native-async-storage/async-storage';

function MainPage() {

  const navigation = useNavigation(); //for navigating between pages

  const [items, setItems] = useState([]); // the products list

  const [Query, setQuery] = useState(""); //for the search bar

  const [selectedCategory, setSelectedCategory] = useState(""); //for the category filter

**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 valeur de la catégorie.

  //code that runs when loading the page

  useEffect(() => {

    const checkToken = async () => {

      try {

        const token = await AsyncStorage.getItem('authToken'); //gets the login tokem form local storage

        if (token) {

          navigation.navigate('MainAdmin'); //navigate to the admin page if it exists

        } else {

          apiCall(); //calls this function

        }

      } catch (error) {

        console.error('error message:', error)

      }

    };

    checkToken();

  }, []);

**useEffect** appelle la fonction **apiCall**, qui est responsable de récupérer tous les produits depuis l'API et de les assigner à la variable **items** via **setItems**. L'utilisation de **useEffect** permet d'effectuer l'appel API lors du chargement de la page. Il vérifie également si un jeton de connexion existe, et si c'est le cas, il redirige l'utilisateur vers la page administrateur.

  const desc = (productId) => {

    navigation.navigate('ProductDetails', { id: productId }); // You need to set up this screen

  };

//the HTML component

<TouchableOpacity onPress={() => desc(item.id)}>

    <Text style={styles.itemTitle}>{item.title}</Text>

</TouchableOpacity>

Cette fonction est une fonction **onClick** qui, lorsqu'un utilisateur clique sur le nom d'un produit, le redirige vers la page du produit.

  //code for handling filtration, both search bar and category

  const filterItems = useMemo(() => { //useMemo so that the API is not called upon multiple times

    return items.filter((item) => {

      const matchesQuery = item.title.toLowerCase().includes(Query.toLowerCase()); //converts the input to lowercase

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

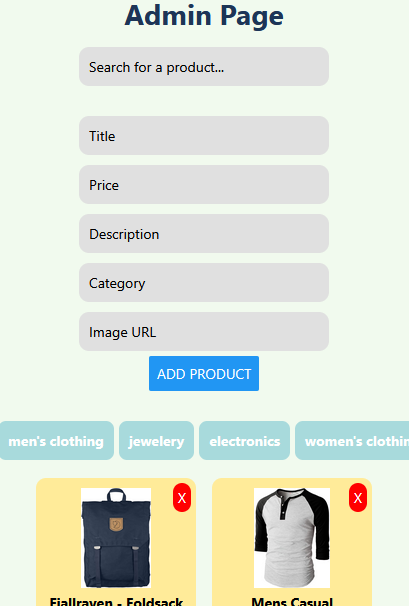
      return matchesQuery && matchesCategory;

    });

  }, [items, Query, selectedCategory]);

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

## MainPageAdmin.js:



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

import { View, Text, TextInput, Button, ScrollView, Image, TouchableOpacity, StyleSheet,

} from 'react-native';

import axios from 'axios';

import { useNavigation } from '@react-navigation/native';

import AsyncStorage from '@react-native-async-storage/async-storage';

const MainPageAdmin = () => {

  const navigation = useNavigation(); //for navigating between pages

  const [items, setItems] = useState([]); // the products list

  const [add, setAdd] = useState({

    title: '',

    price: '',

    description: '',

    category: '',

    image: '',

  }); //the variable for adding a new product

  const [query, setQuery] = useState(''); //for the search bar

  const [selectedCategory, setSelectedCategory] = useState(''); //for the category filter

  const [loading, setLoading] = useState(true); //for loading before the products list is loaded

Cette page est similaire à la **MainPage**, tout en ajoutant les fonctionnalités suivantes : ajout de produits via la variable **add**, un bouton de suppression pour retirer les produits, et la variable **isAuthenticated** pour s'assurer que l'utilisateur dispose d'un jeton de connexion afin d'accéder à cette page.

  //handles the code for logging out from the admin page

  const handleLogout = async () => {

    try {

      await AsyncStorage.removeItem("authToken"); //removes the login token from local storage

      //use this to check if the token was removed

      // console.log("Logged out successfully");

      navigation.navigate('Main'); //go to the main page

    } catch (error) {

      console.error("Logout failed:", error)

    }

  };

**HandleLogout** supprime le jeton du stockage local et redirige l'utilisateur vers la page principale d'origine **MainPage**.

  //the function that adds a new product

  const handleAdd = async () => {

    try {

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

        setItems(prev => [...prev, response.data]); //modifies items to add this new item

        setAdd({ title: '', price: 0.0, description: '', category: '', image: '' }); //return the add variable to default values

    } catch (error) {

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

    }

};

      {/\* Add Product Form \*/}

      <View style={styles.form}>

        <TextInput placeholder="Title" value={add.title} onChangeText={text => handleChange('title', text)} style={styles.input}/>

        <TextInput placeholder="Price" value={add.price} onChangeText={text => handleChange('price', text)} keyboardType="numeric" style={styles.input}/>

        <TextInput placeholder="Description" value={add.description} onChangeText={text => handleChange('description', text)} style={styles.input}/>

        <TextInput placeholder="Category" value={add.category} onChangeText={text => handleChange('category', text)} style={styles.input}/>

        <TextInput placeholder="Image URL" value={add.image} onChangeText={text => handleChange('image', text)} style={styles.input}/>

        <Button title="Add Product" onPress={handleAdd} style={styles.button}/>

      </View>

**HandleAdd** permet d'ajouter un nouveau produit avec les données saisies dans les champs de saisie HTML.

      {/\* Category Filters \*/}

      <ScrollView horizontal style={styles.categoryRow}>

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

          <TouchableOpacity key={index} onPress={() => setSelectedCategory(category)} style={[ styles.categoryBtn, selectedCategory === category && styles.activeCategory,]}>

            <Text style={styles.categoryText}>{category}</Text>

          </TouchableOpacity>

        ))}

        {selectedCategory !== '' && (

          <TouchableOpacity onPress={() => setSelectedCategory('')} style={styles.clearBtn}>

            <Text style={styles.categoryText}>Clear</Text>

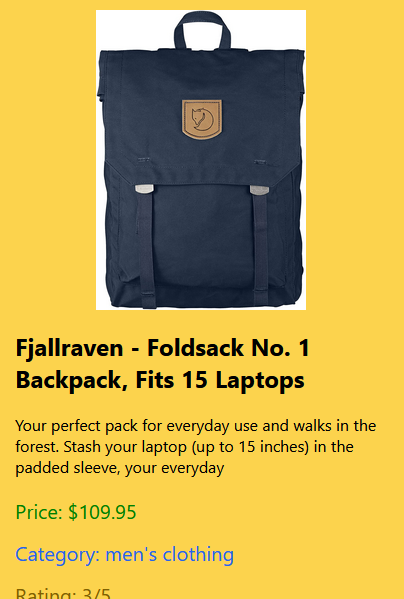
          </TouchableOpacity>

        )}

      </ScrollView>

Ce code HTML permet de filtrer les produits par catégories. Le deuxième bouton efface la catégorie sélectionnée et affiche la liste complète lorsqu'il est cliqué.

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



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

import { View, Text, Image, StyleSheet, TouchableOpacity, ScrollView, ActivityIndicator } from 'react-native';

import axios from 'axios';

function ProductScreen({ route, navigation }) {

  const { id } = route.params || {}; //gets the id from the url

  const [product, setProduct] = useState(null); //holds the product's information

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

**Params** récupère l'ID du produit soit de la **MainPage** soit de la **MainPageAdmin**.

  //code that runs when the page is called

  useEffect(() => {

    if (id) {

      fetchProduct(); //get this function if there is an id

    }

  }, [id]);

  //function that fetches product information

  const fetchProduct = async () => {

    try {

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

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

    } catch (error) {

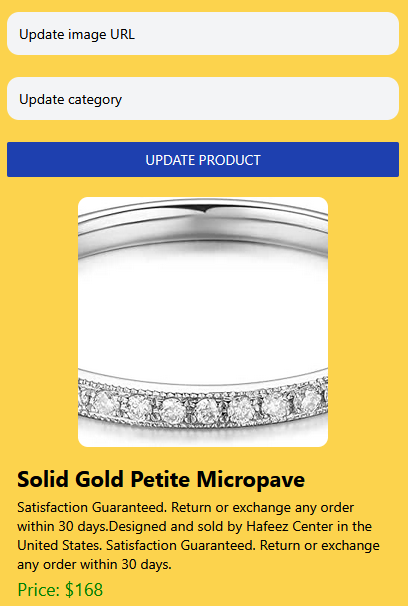
      console.error('Error fetching product:', error);

    }

  };

Cette fonction récupère le produit spécifique en utilisant l'ID de l'URL et le stocke dans la variable **product.** Elle s'exécute lorsque la page se charge.

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



Semblable à la page ci-dessus, avec des champs de saisie HTML supplémentaires pour mettre à jour les propriétés du produit.

      <View style={styles.inputContainer}>

        <TextInput placeholder="Update title" onChangeText={text => handleChange('title', text)} style={styles.input} />

        <TextInput placeholder="Update price" onChangeText={text => handleChange('price', text)} keyboardType="numeric" style={styles.input} />

        <TextInput placeholder="Update description" onChangeText={text => handleChange('description', text)} multiline style={styles.input} />

        <TextInput placeholder="Update image URL" onChangeText={text => handleChange('image', text)} style={styles.input} />

        <TextInput placeholder="Update category" onChangeText={text => handleChange('category', text)} style={styles.input} />

        <Button title="Update Product" onPress={handleUpdate} color="#1e40af" />

      </View>

Le code ci-dessous gère la mise à jour des paramètres du produit en utilisant la fonction PUT de l'API. Le deuxième **setUpdate** sert à effacer les champs de saisie de mise à jour après la mise à jour.

  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);

      setUpdate({});

    }catch (error) {

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

    }

  }

## 5.6 app.js:

import MainPage from './pages/MainPage';

import { NavigationContainer } from '@react-navigation/native';

import { createNativeStackNavigator } from '@react-navigation/native-stack';

import Login from './pages/login';

import ProductScreen from './productPages/products/[id]/page';

import MainPageAdmin from './pages/MainPageAdmin';

import ProductPage from './productPages/productsAdmin/[id]/page';

export default function App() {

  const Stack = createNativeStackNavigator();

  return (

    <NavigationContainer>

      <Stack.Navigator initialRouteName='Main' >

        <Stack.Screen name="Main" component={MainPage}/>

        <Stack.Screen name="login" component={Login}/>

        <Stack.Screen name="MainAdmin" component={MainPageAdmin}/>

        <Stack.Screen name="ProductDetails" component={ProductScreen}/>

        <Stack.Screen name="ProductAdmin" component={ProductPage}/>

      </Stack.Navigator>

    </NavigationContainer>

// add options={{ headerShown: false }} to the stacks if you want to remove the header.

  );}

Le fichier principal qui affiche le projet dans l'application. Il contient le code de navigation encapsulé dans le composant **NavigationContainer** . D'autres pages peuvent y être ajoutées.

# Problèmes techniques et contraintes

La plupart des contraintes provenaient soit de l'API étant une **fakestoreapi**, et donc ne réalisant pas réellement les opérations **CRUD** lorsqu'elles sont appelées, soit du temps limité, ce qui ne permettait pas d'essayer différentes approches pour le codage ou le **CSS**.

## fakestoreapi:

Étant donné que l'API utilisée dans le projet n'est pas une véritable API, elle ne se met pas réellement à jour lors de l'utilisation des opérations CRUD.

  //function that handles updating the product's properties

  const handleUpdate = async () => {

    if (!product || Object.keys(update).length === 0) return; //if there is no product or the inputs are empty, do nothing

    try {

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

        ...product,

        ...update,

      });

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

      setUpdate({}); //set the update variable to empty object

    } catch (error) {

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

    }

  };

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

  //the function that adds a new product

  const handleAdd = async () => {

    try {

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

        setItems(prev => [...prev, response.data]); //modifies items to add this new item

        setAdd({ title: '', price: 0.0, description: '', category: '', image: '' }); //return the add variable to default values

    } catch (error) {

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

    }

};

**Ces deux fonctions dans la MainPageAdmin présentent également le même problème. Il a été déterminé que, puisque ceci est un test, il est plus important de démontrer que ces fonctions CRUD fonctionnent correctement.**

## CSS properties:

Un changement soudain de conception entre NextJS et React Native est que Tailwind est légèrement différent en Native, et par conséquent, l'utilisation de la syntaxe CSS normale était préférable.

EX:

const styles = StyleSheet.create({

  container: {

    flex: 1,

    backgroundColor: '#f3f4f6',

    alignItems: 'center',

    justifyContent: 'center',

    padding: 20,

  },

  logo: {

    fontSize: 24,

    fontWeight: 'bold',

    marginBottom: 10,

    cursor: "pointer",

  },

  header: {

    fontSize: 20,

    color: '#60a5fa',

    marginBottom: 10,

  },

  socialIcons: {

    flexDirection: 'row',

    marginVertical: 10,

  },

  icon: {

    padding: 10,

    marginHorizontal: 5,

    backgroundColor: '#e5e7eb',

    borderRadius: 50,

  },

# Conclusion

**Ce produit met en évidence les nombreuses différences entre NextJS et React Native, ce qui a entraîné des approches différentes pour certaines pratiques courantes de JavaScript lors du passage d'un projet à l'autre.**

**Améliorations futures :**

* Améliorer l'utilisation de CSS.
* Utiliser davantage le stockage local.
* La performance peut encore être améliorée.

**Remerciements :**

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