App.js --------------------

import React, { useEffect } from 'react';

import Scanner from './screens/Scanner';

import Home from './screens/Home';

import AddProduct from './screens/AddProduct'; // Verificați că numele componentei importate este corect

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

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

import { createTable } from './screens/dbConnection';

const Stack = createStackNavigator();

const App = () => {

  useEffect(() => {

    createTable()

      .then((message) => {

        console.log(message);

      })

      .catch((error) => {

        console.error(error);

      });

  }, []);

  return (

    <NavigationContainer>

      <Stack.Navigator>

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

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

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

      </Stack.Navigator>

    </NavigationContainer>

  );

};

export default App;

Fisierul App.js este fișierul principal al aplicației. Aici se definesc rutele și se creează navigarea între ecranele aplicației folosind react-navigation. De asemenea, în acest fișier se face și crearea tabelei din baza de date, utilizând funcția createTable().

Fisierul Home.js definește ecranul principal al aplicației, unde sunt afișate produsele scaneate și alte acțiuni posibile. Aici sunt definite butoanele pentru scanare, adăugare produs, verificare conexiune cu baza de date și ștergere a tuturor produselor. În acest fișier se folosesc funcții precum getAllProducts(), deleteAllProducts() și checkDatabaseConnection() pentru a realiza aceste acțiuni în baza de date.

Fisierul Scanner.js definește ecranul de scanare a codurilor de bare sau a codurilor QR. Aici se folosește componenta BarCodeScanner din librăria expo-barcode-scanner pentru a permite utilizatorului să scaneze codurile. După scanare, se verifică dacă codul este prezent în baza de date utilizând funcția checkProductByBarcode().

Fisierul dbConnection.js conține funcțiile pentru interacțiunea cu baza de date SQLite. Aici sunt definite funcții pentru verificarea conexiunii cu baza de date, crearea tabelei, inserarea, ștergerea și interogarea produselor. Aceste funcții sunt apelate în celelalte componente pentru a realiza acțiunile în baza de date.

Fisierul AddProduct.js definește ecranul în care utilizatorul poate introduce manual produse în baza de date. Aici sunt afișate câmpurile pentru completarea informațiilor despre produs (nume, cod de bare, preț, monedă și categorie). La apăsarea butonului "Adaugă produs", se verifică dacă toate câmpurile sunt completate și se inserează produsul în baza de date folosind funcția insertProduct().

Aceste fișiere sunt interconectate și alcătuiesc aplicația de citire a codurilor de bare și de gestionare a informațiilor citite.

Codul prezentat este fișierul principal al aplicației, App.js. Voi explica fiecare linie de cod și legătura dintre ele:

Importurile:

import React, { useState } from 'react';

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

import { insertProduct } from './dbConnection';

import { getAllProducts } from './dbConnection';

Aceasta este secțiunea de importuri a fișierului App.js. Aici sunt importate modulele și componentele necesare pentru funcționarea corectă a aplicației. Mai exact:

React și useEffect sunt importate din biblioteca react. React este utilizat pentru a crea componente și useEffect permite efectuarea de acțiuni după ce componenta a fost afișată.

Componentele Scanner, Home și AddProduct sunt importate din fișierele corespunzătoare. Aceste componente vor fi utilizate ulterior în navigație.

NavigationContainer este importat din modulul @react-navigation/native. Acesta este un container pentru navigație și va încadra toate ecranele aplicației.

createStackNavigator este importat din modulul @react-navigation/stack. Aceasta va fi utilizată pentru a crea un navigator de tip stack pentru navigarea între diferite ecrane.

Funcția createTable este importată din fișierul dbConnection și va fi utilizată pentru a crea tabela în baza de date.

javascript

const Stack = createStackNavigator();

const App = () => {

useEffect(() => {

createTable()

.then((message) => {

console.log(message);

})

.catch((error) => {

console.error(error);

});

}, []);

Aceasta este definiția componentei principale App. În interiorul funcției App, se definește un obiect numit Stack, care reprezintă navigatorul de tip stack utilizat pentru navigare.

Apoi, în cadrul funcției useEffect, se utilizează funcția createTable pentru a crea tabela în baza de date. Această funcție este apelată o singură dată, când componenta App este montată în arborele de componente. Dacă tabela este creată cu succes, se afișează un mesaj de confirmare în consolă (console.log(message)). În caz contrar, se afișează o eroare (console.error(error)).

return (

<NavigationContainer>

<Stack.Navigator>

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

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

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

</Stack.Navigator>

</NavigationContainer>

);

};

Aceasta este secțiunea în care este returnată componenta principală App. Este utilizat <NavigationContainer> pentru a încadra toate ecranele și <Stack.Navigator> pentru a defini navigația între ele.

În cadrul navigatorului, sunt definite trei ecrane prin intermediul componentei <Stack.Screen>. Fiecare ecran este asociat cu un nume și o componentă corespunzătoare. Astfel, atunci când navigația se schimbă și se selectează un anumit ecran, componenta asociată va fi afișată în interiorul navigatorului.

Ecranele definite sunt:

Ecranul "Home" cu componenta Home.

Ecranul "Scanner" cu componenta Scanner.

Ecranul "AddProduct" cu componenta AddProduct.

Aceasta este structura de bază a aplicației. Utilizând navigația, utilizatorul va putea accesa diferite ecrane și componente, precum ecranul principal "Home", ecranul de scanare "Scanner" și ecranul de adăugare a produsului "AddProduct". De asemenea, se va crea tabela în baza de date la încărcarea aplicației, utilizând funcția createTable din fișierul dbConnection.

Home.js:

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

import { View, StyleSheet, TouchableHighlight, Text, ScrollView, Dimensions } from 'react-native';

import { BarCodeScanner } from 'expo-barcode-scanner';

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

import { checkDatabaseConnection, getAllProducts, deleteAllProducts } from './dbConnection';

export default function Home() {

  const navigation = useNavigation();

  const [products, setProducts] = useState([]);

  const loadProducts = useCallback(() => {

    getAllProducts()

      .then((result) => {

        setProducts(result.rows.\_array);

      })

      .catch((error) => {

        console.error(error);

      });

  }, []);

  useEffect(() => {

    const unsubscribe = navigation.addListener('focus', loadProducts);

    return unsubscribe;

  }, [navigation, loadProducts]);

  const handleScanPress = async () => {

    const { status } = await BarCodeScanner.requestPermissionsAsync();

    if (status === 'granted') {

      navigation.navigate('Scanner');

    } else {

      alert('Camera permission denied');

    }

  };

  const handleDBConnectionPress = async () => {

    try {

      await checkDatabaseConnection();

      alert('Conectat la baza de date SQLite!');

    } catch (error) {

      alert('Eroare la conectarea la baza de date SQLite!');

      console.error(error);

    }

  };

  const handleAddProductPress = () => {

    navigation.navigate('AddProduct');

  };

  const handleDeleteAllProducts = () => {

    deleteAllProducts()

      .then(() => {

        alert('Toate produsele au fost șterse din tabelă!');

        loadProducts(); // Reîncarcă produsele după ștergerea lor

      })

      .catch((error) => {

        console.error(error);

        alert('Eroare la ștergerea produselor din tabelă!');

      });

  };

  return (

    <View style={styles.container}>

      <TouchableHighlight

        style={styles.scanButton}

        onPress={handleScanPress}

        underlayColor="#FFD700"

      >

        <Text style={styles.scanButtonText}>Scan</Text>

      </TouchableHighlight>

      <TouchableHighlight

        style={styles.addButton}

        onPress={handleAddProductPress}

        underlayColor="#FFD700"

      >

        <Text style={styles.addButtonText}>Adaugă produs</Text>

      </TouchableHighlight>

      <TouchableHighlight

        style={styles.deleteButton}

        onPress={handleDeleteAllProducts}

        underlayColor="#FFD700"

      >

        <Text style={styles.deleteButtonText}>Șterge toate produsele(!)</Text>

      </TouchableHighlight>

      <TouchableHighlight

        style={styles.scanButton}

        onPress={handleDBConnectionPress}

        underlayColor="#FFD700"

      >

        <Text style={styles.scanButtonText}>Verifica conexiunea cu DB!</Text>

      </TouchableHighlight>

      <ScrollView style={styles.productListContainer}>

        <View style={styles.productList}>

          {products.map((product) => (

            <View key={product.id} style={styles.productItem}>

              <Text style={[styles.productText, styles.column]}>{product.id}</Text>

              <Text style={[styles.productText, styles.column]}>{product.product}</Text>

              <Text style={[styles.productText, styles.column]}>{product.barcode}</Text>

              <Text style={[styles.productText, styles.column]}>{product.price}</Text>

              <Text style={[styles.productText, styles.column]}>{product.currency}</Text>

              <Text style={[styles.productText, styles.column]}>{product.category}</Text>

            </View>

          ))}

        </View>

      </ScrollView>

    </View>

  );

}

const windowWidth = Dimensions.get('window').width;

const styles = StyleSheet.create({

  container: {

    flex: 1,

    backgroundColor: '#983dd9',

    alignItems: 'center',

    justifyContent: 'center',

  },

  scanButton: {

    backgroundColor: '#FFD700',

    paddingHorizontal: 16,

    paddingVertical: 8,

    borderRadius: 8,

    marginBottom: 10,

  },

  scanButtonText: {

    color: '#FFFFFF',

    fontWeight: 'bold',

  },

  addButton: {

    backgroundColor: '#FFD700',

    paddingHorizontal: 16,

    paddingVertical: 8,

    borderRadius: 8,

    marginBottom: 10,

  },

  addButtonText: {

    color: '#FFFFFF',

    fontWeight: 'bold',

  },

  deleteButton: {

    backgroundColor: '#FF0000',

    paddingHorizontal: 16,

    paddingVertical: 8,

    borderRadius: 8,

    marginBottom: 10,

  },

  deleteButtonText: {

    color: '#FFFFFF',

    fontWeight: 'bold',

  },

  productListContainer: {

    flex: 1,

    width: '100%',

    paddingHorizontal: 16,

  },

  productList: {

    backgroundColor: '#FFFFFF',

    borderRadius: 8,

  },

  productItem: {

    flexDirection: 'row',

    alignItems: 'center',

    justifyContent: 'space-between',

    marginBottom: 8,

    paddingHorizontal: 16,

  },

  productText: {

    flex: 1,

    fontSize: 16,

  },

  column: {

    width: windowWidth / 6,

    textAlign: 'center',

  },

});

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

import { View, StyleSheet, TouchableHighlight, Text, ScrollView, Dimensions } from 'react-native';

import { BarCodeScanner } from 'expo-barcode-scanner';

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

import { checkDatabaseConnection, getAllProducts, deleteAllProducts } from './dbConnection';

export default function Home() {

const navigation = useNavigation();

const [products, setProducts] = useState([]);

const loadProducts = useCallback(() => {

getAllProducts()

.then((result) => {

setProducts(result.rows.\_array);

})

.catch((error) => {

console.error(error);

});

}, []);

useEffect(() => {

const unsubscribe = navigation.addListener('focus', loadProducts);

return unsubscribe;

}, [navigation, loadProducts]);

Componenta Home este exportată ca fiind componenta implicită a acestui fișier.

useNavigation este utilizat pentru a obține obiectul de navigare, astfel încât să putem naviga către alte ecrane.

Folosim useState pentru a gestiona starea produselor. Inițial, lista de produse este goală ([]).

Funcția loadProducts este definită utilizând useCallback. Această funcție este responsabilă pentru încărcarea produselor din baza de date și actualizarea stării produselor utilizând setProducts. getAllProducts este apelat pentru a obține toate produsele din baza de date, iar rezultatul este actualizat în starea produselor.

Utilizăm useEffect pentru a se asigura că funcția loadProducts este apelată atunci când ecranul "Home" primește focalizarea (navigation.addListener('focus', loadProducts)). Astfel, lista de produse este încărcată și actualizată atunci când utilizatorul se întoarce pe ecranul "Home". Funcția useEffect returnează un funcționalitate de dezabonare (return unsubscribe), care se va asigura că eventualele abonări sunt anulate atunci când componenta este dezmontată.

Continuând cu restul codului:

const handleScanPress = async () => {

const { status } = await BarCodeScanner.requestPermissionsAsync();

if (status === 'granted') {

navigation.navigate('Scanner');

} else {

alert('Camera permission denied');

}

};

const handleDBConnectionPress = async () => {

try {

await checkDatabaseConnection();

alert('Conectat la baza de date SQLite!');

} catch (error) {

alert('Eroare la conectarea la baza de date SQLite!');

console.error(error);

}

};

const handleAddProductPress = () => {

navigation.navigate('AddProduct');

};

const handleDeleteAllProducts = () => {

deleteAllProducts()

.then(() => {

alert('Toate produsele au fost șterse din tabelă!');

loadProducts(); // Reîncarcă produsele după ștergerea lor

})

.catch((error) => {

console.error(error);

alert('Eroare la ștergerea produselor din tabelă!');

});

};

handleScanPress este apelat atunci când utilizatorul apasă butonul "Scan". În acest cod, întâi solicităm permisiunea de a utiliza camera utilizând BarCodeScanner.requestPermissionsAsync(). Dacă permisiunea este acordată, utilizăm navigation.navigate('Scanner') pentru a naviga către ecranul "Scanner". În caz contrar, afișăm un mesaj de eroare.

handleDBConnectionPress este apelat atunci când utilizatorul apasă butonul "Verifică conexiunea cu DB!". În acest cod, încercăm să verificăm conexiunea la baza de date utilizând checkDatabaseConnection(). Dacă conexiunea este realizată cu succes, afișăm un mesaj de succes. În caz contrar, afișăm un mesaj de eroare și afișăm detaliile erorii în consolă.

handleAddProductPress este apelat atunci când utilizatorul apasă butonul "Adaugă produs". În acest cod, utilizăm navigation.navigate('AddProduct') pentru a naviga către ecranul "AddProduct".

handleDeleteAllProducts este apelat atunci când utilizatorul apasă butonul "Șterge toate produsele(!)". În acest cod, apelăm funcția deleteAllProducts() care șterge toate produsele din tabela bazei de date. După ștergere, afișăm un mesaj de succes și reîncărcăm lista de produse utilizând loadProducts(). În caz de eroare, afișăm un mesaj de eroare și afișăm detalii despre eroare în consolă.

Continuăm cu restul codului:

return (

<View style={styles.container}>

<TouchableHighlight

style={styles.scanButton}

onPress={handleScanPress}

underlayColor="#FFD700"

>

<Text style={styles.scanButtonText}>Scan</Text>

</TouchableHighlight>

<TouchableHighlight

style={styles.addButton}

onPress={handleAddProductPress}

underlayColor="#FFD700"

>

<Text style={styles.addButtonText}>Adaugă produs</Text>

</TouchableHighlight>

<TouchableHighlight

style={styles.deleteButton}

onPress={handleDeleteAllProducts}

underlayColor="#FFD700"

>

<Text style={styles.deleteButtonText}>Șterge toate produsele(!)</Text>

</TouchableHighlight>

<TouchableHighlight

style={styles.scanButton}

onPress={handleDBConnectionPress}

underlayColor="#FFD700"

>

<Text style={styles.scanButtonText}>Verifica conexiunea cu DB!</Text>

</TouchableHighlight>

<ScrollView style={styles.productListContainer}>

<View style={styles.productList}>

{products.map((product) => (

<View key={product.id} style={styles.productItem}>

<Text style={[styles.productText, styles.column]}>{product.id}</Text>

<Text style={[styles.productText, styles.column]}>{product.product}</Text>

<Text style={[styles.productText, styles.column]}>{product.barcode}</Text>

<Text style={[styles.productText, styles.column]}>{product.price}</Text>

<Text style={[styles.productText, styles.column]}>{product.currency}</Text>

<Text style={[styles.productText, styles.column]}>{product.category}</Text>

</View>

))}

</View>

</ScrollView>

</View>

);

}

const windowWidth = Dimensions.get('window').width;

const styles = StyleSheet.create({

container: {

flex: 1,

backgroundColor: '#983dd9',

alignItems: 'center',

justifyContent: 'center',

},

scanButton: {

backgroundColor: '#FFD700',

paddingHorizontal: 16,

paddingVertical: 8,

borderRadius: 8,

marginBottom: 10,

},

scanButtonText: {

color: '#FFFFFF',

fontWeight: 'bold',

},

addButton: {

backgroundColor: '#FFD700',

paddingHorizontal: 16,

paddingVertical: 8,

borderRadius: 8,

marginBottom: 10,

},

addButtonText: {

color: '#FFFFFF',

fontWeight: 'bold',

},

deleteButton: {

backgroundColor: '#FF0000',

paddingHorizontal: 16,

paddingVertical: 8,

borderRadius: 8,

marginBottom: 10,

},

deleteButtonText: {

color: '#FFFFFF',

fontWeight: 'bold',

},

productListContainer: {

flex: 1,

width: '100%',

paddingHorizontal: 16,

},

productList: {

backgroundColor: '#FFFFFF',

borderRadius: 8,

},

productItem: {

flexDirection: 'row',

alignItems: 'center',

justifyContent: 'space-between',

marginBottom: 8,

paddingHorizontal: 16,

},

productText: {

flex: 1,

fontSize: 16,

},

column: {

width: windowWidth / 6,

textAlign: 'center',

},

});

În acest bloc de cod se întoarce structura de vizualizare a componentei Home.

Utilizăm View pentru a încadra conținutul componentei.

Utilizăm TouchableHighlight pentru a crea butoane cu funcționalități diferite. Proprietatea onPress specifică funcția de apel atunci când butonul este apăsat.

Folosim ScrollView pentru a crea o listă de produse care poate fi derulată.

Utilizăm styles pentru a gestiona stilurile vizuale ale componentei.

Aceasta este o prezentare detaliată a codului din fișierul Home.js. Acesta definește comportamentul și aspectul ecranului "Home" în cadrul unei aplicații React Native.

dbConnection.js:

import \* as SQLite from 'expo-sqlite';

const db = SQLite.openDatabase('barcodeapp.db');

export const checkDatabaseConnection = async () => {

  return new Promise((resolve, reject) => {

    db.transaction((tx) => {

      tx.executeSql(

        'SELECT 1 FROM sqlite\_master LIMIT 1',

        [],

        () => {

          resolve('Conectat la baza de date SQLite!');

        },

        (\_, error) => {

          reject(new Error('Eroare la conectarea la baza de date SQLite: ' + error));

        }

      );

    });

  });

};

export const createTable = async () => {

  return new Promise((resolve, reject) => {

    db.transaction((tx) => {

      tx.executeSql(

        `CREATE TABLE IF NOT EXISTS products (

          id INTEGER PRIMARY KEY AUTOINCREMENT,

          product TEXT,

          barcode TEXT,

          price TEXT,

          currency TEXT,

          category TEXT

        );`,

        [],

        () => {

          resolve('Tabela a fost creată sau există deja');

        },

        (\_, error) => {

          reject(new Error('Eroare la crearea tabelei: ' + error));

        }

      );

    });

  });

};

export const getAllProducts = async () => {

  return new Promise((resolve, reject) => {

    db.transaction((tx) => {

      tx.executeSql(

        'SELECT \* FROM products',

        [],

        (\_, result) => {

          resolve(result);

        },

        (\_, error) => {

          reject(new Error('Eroare la preluarea produselor: ' + error));

        }

      );

    });

  });

};

export const insertProduct = async (product) => {

  return new Promise((resolve, reject) => {

    db.transaction((tx) => {

      tx.executeSql(

        `INSERT INTO products (product, barcode, price, currency, category) VALUES (?, ?, ?, ?, ?)`,

        [product.product, product.barcode, product.price, product.currency, product.category],

        (\_, result) => {

          resolve(result);

        },

        (\_, error) => {

          reject(new Error('Eroare la inserarea produsului: ' + error));

        }

      );

    });

  });

};

export const deleteAllProducts = async () => {

  return new Promise((resolve, reject) => {

    db.transaction((tx) => {

      tx.executeSql(

        'DELETE FROM products',

        [],

        (\_, result) => {

          resolve(result);

        },

        (\_, error) => {

          reject(new Error('Eroare la ștergerea produselor: ' + error));

        }

      );

    });

  });

};

export const checkProductByBarcode = async (barcode) => {

  return new Promise((resolve, reject) => {

    db.transaction((tx) => {

      tx.executeSql(

        'SELECT \* FROM products WHERE barcode = ?',

        [barcode],

        (\_, result) => {

          if (result.rows.length > 0) {

            resolve(result.rows.item(0));

          } else {

            resolve(null);

          }

        },

        (\_, error) => {

          reject(new Error('Eroare la verificarea produsului după codul de bare: ' + error));

        }

      );

    });

  });

};

import \* as SQLite from 'expo-sqlite';

// Deschide conexiunea către baza de date SQLite

const db = SQLite.openDatabase('barcodeapp.db');

Importăm modulul expo-sqlite pentru a utiliza funcționalitățile bazei de date SQLite.

Folosim funcția openDatabase pentru a deschide o conexiune la baza de date SQLite cu numele 'barcodeapp.db'. Aceasta va deschide baza de date existentă sau va crea una nouă dacă nu există deja.

export const checkDatabaseConnection = async () => {

return new Promise((resolve, reject) => {

db.transaction((tx) => {

tx.executeSql(

'SELECT 1 FROM sqlite\_master LIMIT 1',

[],

() => {

resolve('Conectat la baza de date SQLite!');

},

(\_, error) => {

reject(new Error('Eroare la conectarea la baza de date SQLite: ' + error));

}

);

});

});

};

checkDatabaseConnection este o funcție asincronă exportată.

Această funcție verifică conexiunea la baza de date SQLite. Folosim o promisiune pentru a returna rezultatul.

În interiorul promisiunii, folosim db.transaction pentru a executa o tranzacție.

Folosim tx.executeSql pentru a executa o interogare SQL care verifică existența tabelei 'sqlite\_master'.

Dacă interogarea este executată cu succes, rezolvăm promisiunea cu un mesaj de succes. În caz contrar, respingem promisiunea și afișăm o eroare.

export const createTable = async () => {

return new Promise((resolve, reject) => {

db.transaction((tx) => {

tx.executeSql(

`CREATE TABLE IF NOT EXISTS products (

id INTEGER PRIMARY KEY AUTOINCREMENT,

product TEXT,

barcode TEXT,

price TEXT,

currency TEXT,

category TEXT

);`,

[],

() => {

resolve('Tabela a fost creată sau există deja');

},

(\_, error) => {

reject(new Error('Eroare la crearea tabelei: ' + error));

}

);

});

});

};

createTable este o funcție asincronă exportată.

Această funcție creează tabela 'products' dacă nu există deja.

Folosim o promisiune pentru a returna rezultatul.

În interiorul promisiunii, folosim db.transaction pentru a executa o tranzacție.

Folosim tx.executeSql pentru a executa o interogare SQL care creează tabela 'products' cu coloanele specificate.

Dacă interogarea este executată cu succes, rezolvăm promisiunea cu un mesaj de succes. În caz contrar, respingem promisiunea și afișăm o eroare.

export const getAllProducts = async () => {

return new Promise((resolve, reject) => {

db.transaction((tx) => {

tx.executeSql(

'SELECT \* FROM products',

[],

(\_, result) => {

resolve(result);

},

(\_, error) => {

reject(new Error('Eroare la preluarea produselor: ' + error));

}

);

});

});

};

getAllProducts este o funcție asincronă exportată.

Această funcție returnează toate produsele din tabela 'products'.

Folosim o promisiune pentru a returna rezultatul.

În interiorul promisiunii, folosim db.transaction pentru a executa o tranzacție.

Folosim tx.executeSql pentru a executa o interogare SQL care selectează toate înregistrările din tabela 'products'.

Dacă interogarea este executată cu succes, rezolvăm promisiunea cu rezultatul. În caz contrar, respingem promisiunea și afișăm o eroare.

export const insertProduct = async (product) => {

return new Promise((resolve, reject) => {

db.transaction((tx) => {

tx.executeSql(

`INSERT INTO products (product, barcode, price, currency, category) VALUES (?, ?, ?, ?, ?)`,

[product.product, product.barcode, product.price, product.currency, product.category],

(\_, result) => {

resolve(result);

},

(\_, error) => {

reject(new Error('Eroare la inserarea produsului: ' + error));

}

);

});

});

};

insertProduct este o funcție asincronă exportată.

Această funcție inserează un produs în tabela 'products' cu valorile specifice.

Folosim o promisiune pentru a returna rezultatul.

În interiorul promisiunii, folosim db.transaction pentru a executa o tranzacție.

Folosim tx.executeSql pentru a executa o interogare SQL care inserează valorile produsului în tabela 'products'.

Dacă interogarea este executată cu succes, rezolvăm promisiunea cu rezultatul. În caz contrar, respingem promisiunea și afișăm o eroare.

export const deleteAllProducts = async () => {

return new Promise((resolve, reject) => {

db.transaction((tx) => {

tx.executeSql(

'DELETE FROM products',

[],

(\_, result) => {

resolve(result);

},

(\_, error) => {

reject(new Error('Eroare la ștergerea produselor: ' + error));

}

);

});

});

};

deleteAllProducts este o funcție asincronă exportată.

Această funcție șterge toate produsele din tabela 'products'.

Folosim o promisiune pentru a returna rezultatul.

În interiorul promisiunii, folosim db.transaction pentru a executa o tranzacție.

Folosim tx.executeSql pentru a executa o interogare SQL care șterge toate înregistrările din tabela 'products'.

Dacă interogarea este executată cu succes, rezolvăm promisiunea cu rezultatul. În caz contrar, respingem promisiunea și afișăm o eroare.

export const checkProductByBarcode = async (barcode) => {

return new Promise((resolve, reject) => {

db.transaction((tx) => {

tx.executeSql(

'SELECT \* FROM products WHERE barcode = ?',

[barcode],

(\_, result) => {

if (result.rows.length > 0) {

resolve(result.rows.item(0));

} else {

resolve(null);

}

},

(\_, error) => {

reject(new Error('Eroare la verificarea produsului după codul de bare: ' + error));

}

);

});

});

};

checkProductByBarcode este o funcție asincronă exportată.

Această funcție verifică existența unui produs în tabela 'products' pe baza codului de bare specificat.

Folosim o promisiune pentru a returna rezultatul.

În interiorul promisiunii, folosim db.transaction pentru a executa o tranzacție.

Folosim tx.executeSql pentru a executa o interogare SQL care selectează înregistrările din tabela 'products' unde codul de bare coincide cu valoarea specificată.

Dacă există înregistrări în rezultat, rezolvăm promisiunea cu prima înregistrare. În caz contrar, rezolvăm promisiunea cu null.

Dacă interogarea nu poate fi executată cu succes, respingem promisiunea și afișăm o eroare.

Sacanner.js:

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

import { Text, View, StyleSheet, Button, Linking } from 'react-native';

import { BarCodeScanner } from 'expo-barcode-scanner';

import { checkProductByBarcode } from './dbConnection';

export default function Scanner() {

  const [hasPermission, setHasPermission] = useState(null);

  const [scanned, setScanned] = useState(false);

  useEffect(() => {

    (async () => {

      const { status } = await BarCodeScanner.requestPermissionsAsync();

      setHasPermission(status === 'granted');

    })();

  }, []);

  const handleBarCodeScanned = async ({ type, data }) => {

    setScanned(true);

    //console.log(data);

    try {

      const product = await checkProductByBarcode(data);

      if (product) {

        alert(`Codul ${data} a fost găsit în baza de date. Produs: ${product.product}, Preț: ${product.price} Lei`);

      } else {

        alert(`Codul ${data} nu a fost găsit în baza de date.`);

      }

    } catch (error) {

      console.error('Eroare la verificarea produsului:', error);

    }

  };

  if (hasPermission === null) {

    return <Text>Requesting for camera permission</Text>;

  }

  if (hasPermission === false) {

    return <Text>No access to camera</Text>;

  }

  return (

    <View style={styles.container}>

      <BarCodeScanner onBarCodeScanned={scanned ? undefined : handleBarCodeScanned} style={StyleSheet.absoluteFillObject} />

      {scanned && <Button title="Tap to scan again" onPress={() => setScanned(false)} />}

    </View>

  );

}

const styles = StyleSheet.create({

  container: {

    flex: 1,

    flexDirection: 'column',

    justifyContent: 'center',

    backgroundColor: '#983dd9',

  },

});

Codul de mai jos este o componenta React Native pentru scanarea codurilor de bare utilizând camera dispozitivului și verificarea acestora în baza de date. Iată o explicație linie cu linie a codului:

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

import { Text, View, StyleSheet, Button, Linking } from 'react-native';

import { BarCodeScanner } from 'expo-barcode-scanner';

import { checkProductByBarcode } from './dbConnection';

Importăm modulele și componentele necesare din biblioteca react-native și expo.

Importăm funcția checkProductByBarcode din fișierul dbConnection.js pe care l-am definit anterior pentru a verifica produsul în baza de date.

export default function Scanner() {

const [hasPermission, setHasPermission] = useState(null);

const [scanned, setScanned] = useState(false);

Definim componenta Scanner ca o funcție React.

Folosim hook-urile useState pentru a defini stările componente.

hasPermission este o stare pentru a stoca permisiunea de acces la cameră (true sau false).

scanned este o stare pentru a indica dacă un cod de bare a fost scanat sau nu (true sau false).

useEffect(() => {

(async () => {

const { status } = await BarCodeScanner.requestPermissionsAsync();

setHasPermission(status === 'granted');

})();

}, []);

Folosim hook-ul useEffect pentru a solicita permisiunea de acces la cameră la încărcarea componentei.

În interiorul useEffect, definim o funcție asincronă care apelează requestPermissionsAsync() pentru a solicita permisiunea.

Starea hasPermission este actualizată pe baza rezultatului permisiunii.

const handleBarCodeScanned = async ({ type, data }) => {

setScanned(true);

try {

const product = await checkProductByBarcode(data);

if (product) {

alert(`Codul ${data} a fost găsit în baza de date. Produs: ${product.product}, Preț: ${product.price} Lei`);

} else {

alert(`Codul ${data} nu a fost găsit în baza de date.`);

}

} catch (error) {

console.error('Eroare la verificarea produsului:', error);

}

};

handleBarCodeScanned este o funcție asincronă care este apelată atunci când un cod de bare este scanat cu succes.

Aceasta setează starea scanned pe true.

Apoi, verificăm produsul în baza de date utilizând funcția checkProductByBarcode.

Dacă produsul există în baza de date, afișăm un mesaj cu detalii despre produs.

Dacă produsul nu există în baza de date, afișăm un mesaj de avertizare.

În caz de eroare, afișăm un mesaj de eroare în consolă.

if (hasPermission === null) {

return <Text>Requesting for camera permission</Text>;

}

if (hasPermission === false) {

return <Text>No access to camera</Text>;

}

return (

<View style={styles.container}>

<BarCodeScanner onBarCodeScanned={scanned ? undefined : handleBarCodeScanned} style={StyleSheet.absoluteFillObject} />

{scanned && <Button title="Tap to scan again" onPress={() => setScanned(false)} />}

</View>

);

}

if (hasPermission === null) {

return <Text>Requesting for camera permission</Text>;

}

if (hasPermission === false) {

return <Text>No access to camera</Text>;

}

return (

<View style={styles.container}>

<BarCodeScanner onBarCodeScanned={scanned ? undefined : handleBarCodeScanned} style={StyleSheet.absoluteFillObject} />

{scanned && <Button title="Tap to scan again" onPress={() => setScanned(false)} />}

</View>

);

}

Verificăm starea hasPermission pentru a determina dacă permisiunea de acces la cameră a fost acordată sau nu.

Dacă permisiunea este null, returnăm un mesaj care solicită permisiunea.

Dacă permisiunea este false, returnăm un mesaj care indică că nu există acces la cameră.

În caz contrar, returnăm componenta <BarCodeScanner> care este responsabilă de scanarea codului de bare.

Proprietatea onBarCodeScanned este setată la handleBarCodeScanned pentru a procesa evenimentul de scanare a codului de bare.

Dacă un cod de bare a fost deja scanat (scanned este true), afișăm un buton care permite scanarea din nou.

const styles = StyleSheet.create({

container: {

flex: 1,

flexDirection: 'column',

justifyContent: 'center',

backgroundColor: '#983dd9',

},

});

Definim stilurile componente utilizând StyleSheet.create.

În acest exemplu, setăm backgroundColor la #983dd9, dar stilurile pot fi personalizate în funcție de nevoile tale.

AddProduct.js:

import React, { useState } from 'react';

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

import { insertProduct } from './dbConnection';

import { getAllProducts } from './dbConnection';

export default function AddProduct() {

  const [product, setProduct] = useState('');

  const [barcode, setBarcode] = useState('');

  const [price, setPrice] = useState('');

  const [currency, setCurrency] = useState('');

  const [category, setCategory] = useState('');

  const handleInsertProduct = () => {

    if (product && barcode && price && currency && category ) {

      const newProduct = {

        product: product,

        barcode: barcode,

        price: price,

        currency: currency,

        category: category,

      };

      insertProduct(newProduct)

        .then(() => {

          alert('Produsul a fost inserat în baza de date!');

          setProduct('');

          setBarcode('');

          setPrice('');

          setCurrency('');

          setCategory('');

        })

        .catch((error) => {

          console.error(error);

          alert('Eroare la inserarea produsului în baza de date!');

        });

    } else {

      alert('Te rugăm să completezi toate câmpurile!');

    }

  };

  return (

    <View style={styles.container}>

      <TextInput

        style={styles.input}

        placeholder="Nume produs"

        value={product}

        onChangeText={(text) => setProduct(text)}

        placeholderTextColor="#FFFFFF"

      />

      <TextInput

        style={styles.input}

        placeholder="Cod de bare"

        value={barcode}

        onChangeText={(text) => setBarcode(text)}

        keyboardType="numeric"

        placeholderTextColor="#FFFFFF"

      />

      <TextInput

        style={styles.input}

        placeholder="Preț"

        value={price}

        onChangeText={(text) => setPrice(text)}

        keyboardType="numeric"

        placeholderTextColor="#FFFFFF"

      />

      <TextInput

        style={styles.input}

        placeholder="Monedă"

        value={currency}

        onChangeText={(text) => setCurrency(text)}

        placeholderTextColor="#FFFFFF"

      />

      <TextInput

        style={styles.input}

        placeholder="Categorie"

        value={category}

        onChangeText={(text) => setCategory(text)}

        placeholderTextColor="#FFFFFF"

      />

      <TouchableOpacity style={styles.button} onPress={handleInsertProduct}>

        <Text style={styles.buttonText}>Inserare produs</Text>

      </TouchableOpacity>

    </View>

  );

}

const styles = StyleSheet.create({

  container: {

    flex: 1,

    padding: 16,

    backgroundColor: '#983dd9',

    paddingHorizontal: 16,

    paddingVertical: 8,

    borderRadius: 8,

    marginBottom: 10,

  },

  input: {

    height: 40,

    borderColor: '#FFFFFF',

    borderWidth: 1,

    marginBottom: 12,

    color: '#FFFFFF',

    fontWeight: 'bold',

    paddingHorizontal: 16,

    paddingVertical: 8,

    borderRadius: 8,

  },

  button: {

    backgroundColor: '#FFD700',

    borderRadius: 10,

    paddingHorizontal: 16,

    paddingVertical: 8,

  },

  buttonText: {

    color: '#FFFFFF',

    fontWeight: 'bold',

    textAlign: 'center',

  },

});

Codul prezentat este un component React Native pentru adăugarea unui produs în baza de date. Iată o explicație linie cu linie a codului:

import React, { useState } from 'react';

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

import { insertProduct } from './dbConnection';

import { getAllProducts } from './dbConnection';

Importăm modulele și componentele necesare din biblioteca react-native.

Importăm funcțiile insertProduct și getAllProducts din fișierul dbConnection.js pe care l-am definit anterior pentru a insera un produs în baza de date și a obține toate produsele din baza de date.

export default function AddProduct() {

const [product, setProduct] = useState('');

const [barcode, setBarcode] = useState('');

const [price, setPrice] = useState('');

const [currency, setCurrency] = useState('');

const [category, setCategory] = useState('');

Definim componenta AddProduct ca o funcție React.

Folosim hook-urile useState pentru a defini stările componente.

Stările product, barcode, price, currency și category sunt utilizate pentru a stoca valorile introduse în câmpurile de introducere.

const handleInsertProduct = () => {

if (product && barcode && price && currency && category ) {

const newProduct = {

product: product,

barcode: barcode,

price: price,

currency: currency,

category: category,

};

insertProduct(newProduct)

.then(() => {

alert('Produsul a fost inserat în baza de date!');

setProduct('');

setBarcode('');

setPrice('');

setCurrency('');

setCategory('');

})

.catch((error) => {

console.error(error);

alert('Eroare la inserarea produsului în baza de date!');

});

} else {

alert('Te rugăm să completezi toate câmpurile!');

}

};

handleInsertProduct este o funcție care este apelată atunci când utilizatorul apasă butonul "Inserare produs".

Verificăm dacă toate câmpurile (product, barcode, price, currency și category) sunt completate.

Dacă da, definim un obiect newProduct care conține valorile introduse în câmpuri.

Apelăm funcția insertProduct pentru a insera noul produs în baza de date.

Dacă inserarea este reușită, afișăm un mesaj de succes și resetăm valorile câmpurilor la șirul gol.

În caz de eroare, afișăm un mesaj de eroare în consolă și alertăm utilizatorul.

return (

<View style={styles.container}>

<TextInput

style={styles.input}

placeholder="Nume produs"

value={product}

onChangeText={(text) => setProduct(text)}

placeholderTextColor="#FFFFFF"

/>

<TextInput

style={styles.input}

placeholder="Cod de bare"

value={barcode}

onChangeText={(text) => setBarcode(text)}

keyboardType="numeric"

placeholderTextColor="#FFFFFF"

/>

<TextInput

style={styles.input}

placeholder="Preț"

value={price}

onChangeText={(text) => setPrice(text)}

keyboardType="numeric"

placeholderTextColor="#FFFFFF"

/>

<TextInput

style={styles.input}

placeholder="Monedă"

value={currency}

onChangeText={(text) => setCurrency(text)}

placeholderTextColor="#FFFFFF"

/>

<TextInput

style={styles.input}

placeholder="Categorie"

value={category}

onChangeText={(text) => setCategory(text)}

placeholderTextColor="#FFFFFF"

/>

<TouchableOpacity style={styles.button} onPress={handleInsertProduct}>

<Text style={styles.buttonText}>Inserare produs</Text>

</TouchableOpacity>

</View>

);

Componenta returnează o vizualizare (View) care conține câmpuri de introducere (TextInput) și un buton (TouchableOpacity).

Fiecare câmp de introducere (TextInput) are un stil (style) asociat, un text de substituție (placeholder), o valoare (value), un handler pentru modificarea textului (onChangeText) și un stil pentru textul de substituție (placeholderTextColor).

Butonul are un stil, un text și un handler pentru evenimentul de apăsare (onPress).

const styles = StyleSheet.create({

container: {

flex: 1,

padding: 16,

backgroundColor: '#983dd9',

paddingHorizontal: 16,

paddingVertical: 8,

borderRadius: 8,

marginBottom: 10,

},

input: {

height: 40,

borderColor: '#FFFFFF',

borderWidth: 1,

marginBottom: 12,

color: '#FFFFFF',

fontWeight: 'bold',

paddingHorizontal: 16,

paddingVertical: 8,

borderRadius: 8,

},

button: {

backgroundColor: '#FFD700',

borderRadius: 10,

paddingHorizontal: 16,

paddingVertical: 8,

},

buttonText: {

color: '#FFFFFF',

fontWeight: 'bold',

textAlign: 'center',

},

});

Definim stilurile pentru componenta utilizând StyleSheet.create.

container reprezintă stilul pentru containerul principal al componentei.

input reprezintă stilul pentru câmpurile de introducere.

button reprezintă stilul pentru buton.

buttonText reprezintă stilul pentru textul butonului.