**To-DO Lista**

A Mobil Programozás féléves beadandóm témájául egy To-Do Appot választottam, melyet React Native-ban keltettem életre. Az applikáció Android felületen lett tesztelve, de a React Native flexibilis kialakítása miatt gond nélkül futna bármilyen iOS készüléken is (vagy akár iOS emulátoron).

Az alkalmazás Firebase Authentication-t és Storage-t használ, ezzel van lekezelve a felhasználók bejelentkezési és egyéb adatainak tárolása és kezelése.

**Az alkalmazás felülete**

Az alkalmazás használata bejelentkezéshez, és előzetes regisztrációhoz van kötve, melyet egy e-mail jelszó párossal tehetünk meg. Ezek után jön fel a főmenü, melyen később láthatjuk a létrehozott To-Do listánkat.  
  
Új teendőt a jobb alsó sarokban lévő + jellel tudunk felvenni, ahol is megadhatjuk a teendő nevét, leírását, határidejét, választhatunk neki színt, kategóriát és tageket is hozzáadhatunk. Az egyes teendők szerkesztésére vagy törlésére a főmenüben van lehetőségünk.

A főmenüben lehetőségünk nyílik kategóriák szerint szűrni a teendőinket, valamint láthatunk egy Progress bar-t, amely mutatja, hogy a teendők hány százalékával vagyunk eddig készen.

Ha egy teendővel kész lettünk, rányomva a zöld pipára kipipálhatjuk, és átkerül a teljesített teendők listájára, melyet a bal alsó sarokba kattintva érhetünk el. Ha esetleg vissza szeretnénk vonni egy teendő késznek jelölését, azt is megtehetjük a kék nyílra kattintva.

Az alkalmazásban helyet kapott még egy Navbar, ami tartalmaz egy Beállítások menüt, ahol választhatunk sötét és világos téma között, valamint egy kijelentkezés gombot, amivel kiléphetünk az appból.

**Használt technológiák**

**React Native**: Lehetővé teszi natív mobilalkalmazások fejlesztését JavaScript és React segítségével iOS és Android platformokra.

**Firebase**: Ez egy felhő alapú backend szolgáltatás, amely valós idejű adatbázist (Firestore), valamint hitelesítést (Authentication) biztosít mobil- és webalkalmazások számára.

**Javascript**: Egy dinamikus programozási nyelv, amelyet széles körben használnak web- és mobilalkalmazások fejlesztéséhez (beleértve a React Native-t is).

**React Navigation**: Egy navigációs könyvtár, amely segít az oldalak közötti váltásban és a navigációs hierarchia kezelésében React Native alkalmazásokban.

**Ionicons**: Egy ikonkönyvtár, amely népszerű ikonokat biztosít React Native és más webes projektek számára.

**CSS JavaScript API**: Lehetővé teszi a stílusok és vizuális elemek dinamikus módosítását JavaScripten keresztül, például a sötét mód váltását vagy animációk kezelését.

1. **Firebase konfiguráció**

* A **Firebase** konfiguráció teszi lehetővé a Firebase szolgáltatásához való csatlakozást, ebben szerepelnek a különböző kulcsok (pl. apiKey) és az ebből származó exportok is, melyet az applikáció különböző komponenseiben használunk.
* **FirebaseAuth**: A Firebase Authentikációs szolgáltatásáért felelős objektum
* **FirestoreDB**: A Firebase Firestore adatbázis szolgáltatásáért felelős objektum

*// Import the functions you need from the SDKs you need*

import { initializeApp } from "firebase/app";

import { getAuth } from "firebase/auth"

import { getFirestore } from "firebase/firestore";

*// TODO: Add SDKs for Firebase products that you want to use*

*// https://firebase.google.com/docs/web/setup#available-libraries*

*// Your web app's Firebase configuration*

const firebaseConfig = {

  apiKey: "AIzaSyDuZ94VLlLgeF6xi2gJS8ZEJMzW\_L873HQ",

  authDomain: "mytodoapp22.firebaseapp.com",

  projectId: "mytodoapp22",

  storageBucket: "mytodoapp22.firebasestorage.app",

  messagingSenderId: "504648254820",

  appId: "1:504648254820:web:5a579a95b231e43919e671"

};

*// Initialize Firebase*

export const FirebaseApp = initializeApp(firebaseConfig);

export const FirebaseAuth = getAuth(FirebaseApp);

export const FirestoreDB = getFirestore(FirebaseApp);

1. **Login felület létrehozása**

* A Login felületen tud bejelentkezni, vagy regisztrálni a felhasználó

A képen szöveg, képernyőkép, Betűtípus, szoftver látható

Automatikusan generált leírás

import {

    View,

    Text,

    StyleSheet,

    TextInput,

    ActivityIndicator,

    Pressable,

    KeyboardAvoidingView,

    Image,

  } from "react-native";

  import React, { useState } from "react";

  import { FirebaseAuth } from "../../FirebaseConfig";

  import { signInWithEmailAndPassword, createUserWithEmailAndPassword } from "firebase/auth";

  const Login = () => {

    const [email, setEmail] = useState("");

    const [password, setPassword] = useState("");

    const [loading, setLoading] = useState(false);

    const auth = FirebaseAuth;

    const signIn = async () => {

      setLoading(true);

      try {

        const res = await signInWithEmailAndPassword(auth, email, password);

        console.log(res);

      } catch (error) {

        console.log(error);

        alert("Failed to login. Please check your credentials and try again." + error.message);

      }

      setLoading(false);

    };

    const signUp = async () => {

      setLoading(true);

      try {

        const res = await createUserWithEmailAndPassword(auth, email, password);

        console.log(res);

        alert("Sign up successful. Please check your emails!");

      } catch (error) {

        console.error("Error Code:", error.code);

        console.error("Error Message:", error.message);

        alert("Failed to sign up: " + error.message);

      }

      setLoading(false);

    };

    return (

      <View style={styles.container}>

        <KeyboardAvoidingView behavior="padding">

          {*/\* <Image source={require("../../assets/logo.png")} style={styles.logo} /> \*/*}

          <Text style={styles.welcomeText}>Welcome to ToDoIst</Text>

          <TextInput

            value={email}

            style={styles.input}

            placeholder="Email"

            autoCapitalize="none"

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

          />

          <TextInput

            value={password}

            style={styles.input}

            placeholder="Password"

            autoCapitalize="none"

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

            secureTextEntry={true}

          />

          {loading ? (

            <ActivityIndicator size="large" color="#0000ff" />

          ) : (

            <>

              <Pressable style={styles.button} onPress={() => signIn()}>

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

              </Pressable>

              <Pressable style={styles.button} onPress={() => signUp()}>

                <Text style={styles.text}>Create account</Text>

              </Pressable>

            </>

          )}

        </KeyboardAvoidingView>

      </View>

    );

  };

  export default Login;

1. **Navigáció létrehozása**

* Az App.js-ben lett létrehozva a navigációs stack, amely definiálja, hogy a felhasználó mely oldalakról mely oldalakra tud eljutni.

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

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

import Login from "./app/screens/Login";

import List from "./app/screens/List";

import Settings from "./app/screens/Settings";

import AddToDo from "./app/screens/AddToDo";

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

import { onAuthStateChanged } from "firebase/auth";

import { FirebaseAuth } from "./FirebaseConfig";

import { View } from "react-native";

import CompletedTasks from "./app/screens/CompletedTasks";

const Stack = createNativeStackNavigator();

const InsideStack = createNativeStackNavigator();

function InsideLayout() {

  return (

    <View style={{ flex: 1, marginTop: 40 }}>

    <InsideStack.Navigator screenOptions={{ headerShown: false }}>

      <InsideStack.Screen name="List" component={List} />

      <InsideStack.Screen name="Settings" component={Settings} />

      <InsideStack.Screen name="AddToDo" component={AddToDo} />

      <InsideStack.Screen name="CompletedTasks" component={CompletedTasks} />

    </InsideStack.Navigator>

    </View>

  );

}

export default function App() {

  const [user, setUser] = useState(null);

  useEffect(() => {

    onAuthStateChanged(FirebaseAuth, (user) => {

      setUser(user);

    });

  }, []);

  return (

    <NavigationContainer>

      <Stack.Navigator initialRouteName="Login">

        {user ? (

          <Stack.Screen

            name="Home"

            component={InsideLayout}

            options={{ headerShown: false }}

          />

        ) : (

          <Stack.Screen

            name="Login"

            component={Login}

            options={{ headerShown: false }}

          />

        )}

      </Stack.Navigator>

    </NavigationContainer>

  );

}

1. **Home Page komponens**

* **Kijelentkezés**: Egy gomb a navbar felületén, amelyet megnyomva a felhasználó kijelentkezik
* **Kategória választás**: Az egyes kategória gombok megnyomásával leszűrhetjük a teendőket a kiválasztott kategóriájú teendőkre
* **Beállítások fül**: Egy gomb a navbar felületén, amelyet megnyomva megnyílik a Beállítások menü
* **Haladás mutató**: Egy százalékos mutató, amely jelzi, hogy a megírt teendőnknek eddig hány százalékát teljesítettük
* **Teendő kártyák**: Az egyes kártyákon láthatjuk a teendő nevét, leírását, határidejét valamint kategóriáját. A kártyákon lévő gombokkal lehetőségünk nyílik az adott teendőt módosítani, készként megjelölni vagy törölni.
* **Új teendő**: Egy gomb a jobb alsó sarokban, amelyet megnyomva új teendőt adhatunk hozzá.
* **Elvégzett teendők listája**: Egy gomb a bal alsó sarokban, amelyet megnyomva megnézhetjük a már elvégzett teendők listáját.

A képen szöveg, képernyőkép, szoftver, Operációs rendszer látható

Automatikusan generált leírás

import {

  View,

  Text,

  StyleSheet,

  Pressable,

  FlatList,

  TouchableOpacity,

  Alert,

} from "react-native";

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

import { FirebaseAuth, FirestoreDB } from "../../FirebaseConfig";

import {

  collection,

  getDocs,

  deleteDoc,

  doc,

  updateDoc,

} from "firebase/firestore";

import { Ionicons } from "@expo/vector-icons";

import { useFocusEffect } from "@react-navigation/native";

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

import { ProgressBar } from "react-native-paper";

const List = ({ navigation }) => {

  const [todos, setTodos] = useState([]);

  const [isDarkMode, setIsDarkMode] = useState(false);

  const [cardColor, setCardColor] = useState("#ffffff");

  const [progress, setProgress] = useState(0);

  const [selectedCategory, setSelectedCategory] = useState("All");

  const fetchTodos = async () => {

    try {

      const querySnapshot = await getDocs(collection(FirestoreDB, "todos"));

      const todoList = querySnapshot.docs.map((doc) => ({

        id: doc.id,

        ...doc.data(),

      }));

      setTodos(todoList);

    } catch (error) {

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

    }

  };

  const loadSettings = async () => {

    try {

      const theme = await AsyncStorage.getItem("theme");

      const color = await AsyncStorage.getItem("cardColor");

      if (theme) setIsDarkMode(theme === "dark");

      if (color) setCardColor(color);

    } catch (error) {

      console.error("Error loading settings:", error);

    }

  };

  useFocusEffect(

    useCallback(() => {

      fetchTodos();

      loadSettings();

    }, [])

  );

  useEffect(() => {

    if (todos.length > 0) {

      const completedTasks = todos.filter((todo) => todo.completed).length;

      setProgress(completedTasks / todos.length);

    }

  }, [todos]);

  const handleComplete = async (id) => {

    try {

      const todoRef = doc(FirestoreDB, "todos", id);

      await updateDoc(todoRef, { completed: true });

      fetchTodos();

    } catch (error) {

      console.error("Error marking todo as completed:", error);

    }

  };

  const handleDelete = async (id) => {

    Alert.alert("Delete To-Do", "Are you sure you want to delete this?", [

      { text: "Cancel", style: "cancel" },

      {

        text: "Delete",

        onPress: async () => {

          try {

            await deleteDoc(doc(FirestoreDB, "todos", id));

            fetchTodos();

          } catch (error) {

            console.error("Error deleting todo:", error);

          }

        },

        style: "destructive",

      },

    ]);

  };

  const handleEdit = (todo) => {

    navigation.navigate("AddToDo", { todo });

  };

  const sortedTodos = [...todos]

    .filter((todo) => !todo.completed)

    .sort((a, b) => new Date(a.date) - new Date(b.date));

  const filteredTodos =

    selectedCategory === "All"

      ? sortedTodos

      : sortedTodos.filter((todo) => todo.category === selectedCategory);

  return (

    <View style={[styles.container, isDarkMode && styles.containerDark]}>

      <View style={[styles.header, isDarkMode && styles.headerDark]}>

        <Pressable

          onPress={() => FirebaseAuth.signOut()}

          style={styles.headerButton}

        >

          <Ionicons

            name="log-out-outline"

            size={24}

            color={isDarkMode ? "white" : "black"}

          />

        </Pressable>

        <Text style={[styles.headerTitle, isDarkMode && styles.textDark]}>

          To-Do List

        </Text>

        <Pressable

          onPress={() => navigation.navigate("Settings")}

          style={styles.headerButton}

        >

          <Ionicons

            name="settings-outline"

            size={24}

            color={isDarkMode ? "white" : "black"}

          />

        </Pressable>

      </View>

      <View style={styles.categoryContainer}>

        {["All", "Work", "Personal", "University"].map((category) => (

          <TouchableOpacity

            key={category}

            style={[

              styles.categoryButton,

              selectedCategory === category && styles.selectedCategory,

            ]}

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

          >

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

          </TouchableOpacity>

        ))}

      </View>

      <View style={styles.progressContainer}>

        <Text style={styles.progressText}>Task Progress</Text>

        <ProgressBar

          progress={progress}

          color="blue"

          style={{ height: 10, borderRadius: 5 }}

        />

      </View>

      <FlatList

        data={filteredTodos}

        keyExtractor={(item) => item.id}

        renderItem={({ item }) => (

          <View

            style={[styles.card, { backgroundColor: item.color || "white" }]}

          >

            <View style={styles.categoryBadge}>

              <Text style={styles.categoryBadgeText}>

                {item.category || "Uncategorized"}

              </Text>

            </View>

            <Text style={[styles.todoTitle, isDarkMode && styles.textDark]}>

              {item.name}

            </Text>

            <Text style={[styles.todoDate, isDarkMode && styles.textDark]}>

              {item.date}

            </Text>

            <Text

              style={[styles.todoDescription, isDarkMode && styles.textDark]}

            >

              {item.description}

            </Text>

            <View style={styles.actionButtons}>

              <TouchableOpacity

                onPress={() => handleEdit(item)}

                style={styles.editButton}

              >

                <Ionicons name="create-outline" size={24} color="blue" />

              </TouchableOpacity>

              <TouchableOpacity

                onPress={() => handleComplete(item.id)}

                style={styles.completeButton}

              >

                <Ionicons

                  name="checkmark-circle-outline"

                  size={24}

                  color="green"

                />

              </TouchableOpacity>

              <TouchableOpacity

                onPress={() => handleDelete(item.id)}

                style={styles.deleteButton}

              >

                <Ionicons name="trash-outline" size={24} color="red" />

              </TouchableOpacity>

            </View>

          </View>

        )}

        contentContainerStyle={{ paddingBottom: 80 }}

      />

      <TouchableOpacity

        style={styles.fab}

        onPress={() => navigation.navigate("AddToDo")}

      >

        <Ionicons name="add" size={32} color="white" />

      </TouchableOpacity>

      <TouchableOpacity

        style={styles.completedTasksButton}

        onPress={() => navigation.navigate("CompletedTasks")}

      >

        <Ionicons name="checkmark-done-outline" size={28} color="white" />

      </TouchableOpacity>

    </View>

  );

};

export default List;

1. A képen szöveg, képernyőkép, szoftver, Operációs rendszer látható

   Automatikusan generált leírás**AddToDo komponens létrehozása**

* **Teendő elnevezése**: Minden teendőnek adnunk kell egy nevet, érdemes olyat, ami jól leírja, hogy mi is a teendőnk.
* **Leírás**: A leírással kiegészíthetjük a teendő nevét, és jobban kifejthetjük azt.
* **Határidő**: A bevitt teendő határideje, ameddig el kell végeznünk.
* **Színválasztás**: Egyedi színek közül választhatunk a teendőink számára, az alkalmazás ehhez 8 féle színt biztosít, hogy könnyen el tudjuk különíteni az egyes kategóriákat.
* **Kategória**: Különböző kategóriájú teendőket hozhatunk létre, legyen szó munkáról, személyes dolgokról vagy egyetemi teendőkről.
* **Tagek**: Hozzáadhatunk különféle egyéb tageket a teendőinkhez.
* **Új teendő felvétele**: A legalsó gomb megnyomásával felvehetjük a megszerkesztett teendőt a teendőink listájára.

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

import {

  View, Text, TextInput, StyleSheet, Pressable, Alert, TouchableOpacity

} from "react-native";

import { collection, addDoc, updateDoc, doc } from "firebase/firestore";

import { FirestoreDB } from "../../FirebaseConfig";

import DateTimePicker from "@react-native-community/datetimepicker";

const categories = ["Work", "Personal", "University"];

const AddToDo = ({ navigation, route }) => {

  const existingTodo = route.params?.todo || null;

  const [name, setName] = useState(existingTodo ? existingTodo.name : "");

  const [description, setDescription] = useState(existingTodo ? existingTodo.description : "");

  const [date, setDate] = useState(existingTodo ? new Date(existingTodo.date) : new Date());

  const [showDatePicker, setShowDatePicker] = useState(false);

  const [color, setColor] = useState(existingTodo ? existingTodo.color : "white");

  const [category, setCategory] = useState(existingTodo ? existingTodo.category : "Work");

  const [tags, setTags] = useState(existingTodo ? existingTodo.tags || [] : []);

  const [newTag, setNewTag] = useState("");

  useEffect(() => {

    if (existingTodo) {

      setName(existingTodo.name);

      setDescription(existingTodo.description);

      setDate(new Date(existingTodo.date));

      setColor(existingTodo.color);

      setCategory(existingTodo.category);

      setTags(existingTodo.tags || []);

    }

  }, [existingTodo]);

  const handleSaveToDo = async () => {

    if (!name.trim() || !description.trim()) {

      Alert.alert("Error", "Please enter a name and description for the to-do.");

      return;

    }

    try {

      if (existingTodo) {

        const todoRef = doc(FirestoreDB, "todos", existingTodo.id);

        await updateDoc(todoRef, {

          name,

          description,

          date: date.toISOString().split("T")[0],

          color,

          category,

          tags,

        });

      } else {

        await addDoc(collection(FirestoreDB, "todos"), {

          name,

          description,

          date: date.toISOString().split("T")[0],

          color,

          category,

          tags,

        });

      }

      Alert.alert("Success", existingTodo ? "To-Do updated successfully!" : "To-Do added successfully!");

      navigation.goBack();

    } catch (error) {

      console.error("Error saving to-do:", error);

      Alert.alert("Error", "Could not save to-do. Try again.");

    }

  };

  const handleAddTag = () => {

    if (newTag.trim() && !tags.includes(newTag.trim())) {

      setTags([...tags, newTag.trim()]);

      setNewTag("");

    }

  };

  const handleRemoveTag = (tag) => {

    setTags(tags.filter((t) => t !== tag));

  };

  return (

    <View style={styles.container}>

      <Text style={styles.label}>To-Do Name</Text>

      <TextInput

        style={styles.input}

        placeholder="Enter to-do name"

        value={name}

        onChangeText={setName}

      />

      <Text style={styles.label}>Description</Text>

      <TextInput

        style={[styles.input, styles.textArea]}

        placeholder="Enter description"

        value={description}

        onChangeText={setDescription}

        multiline

      />

      <Text style={styles.label}>Due Date</Text>

      <Pressable style={styles.datePicker} onPress={() => setShowDatePicker(true)}>

        <Text style={styles.dateText}>{date.toDateString()}</Text>

      </Pressable>

      {showDatePicker && (

        <DateTimePicker

          value={date}

          mode="date"

          display="default"

          onChange={(event, selectedDate) => {

            setShowDatePicker(false);

            if (selectedDate) setDate(selectedDate);

          }}

        />

      )}

    <Text style={styles.label}>Select To-Do Color</Text>

      <View style={styles.colorOptions}>

        {["white", "lightblue", "lightgreen", "lightcoral", "lightyellow", "lightgray", "lightpink", "lightcyan"].map((item) => (

          <Pressable

            key={item}

            style={[styles.colorBox, { backgroundColor: item }, color === item && styles.selectedColor]}

            onPress={() => setColor(item)}

          />

        ))}

      </View>

      <Text style={styles.label}>Category</Text>

      <View style={styles.categoryContainer}>

        {categories.map((cat) => (

          <Pressable

            key={cat}

            style={[styles.categoryButton, category === cat && styles.selectedCategory]}

            onPress={() => setCategory(cat)}

          >

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

          </Pressable>

        ))}

      </View>

      <Text style={styles.label}>Tags</Text>

      <View style={styles.tagsContainer}>

        {tags.map((tag) => (

          <View key={tag} style={styles.tag}>

            <Text style={styles.tagText}>{tag}</Text>

            <TouchableOpacity onPress={() => handleRemoveTag(tag)}>

              <Text style={styles.removeTag}> ✕ </Text>

            </TouchableOpacity>

          </View>

        ))}

      </View>

      <View style={styles.tagInputContainer}>

        <TextInput

          style={styles.tagInput}

          placeholder="Add a tag"

          value={newTag}

          onChangeText={setNewTag}

        />

        <Pressable style={styles.addTagButton} onPress={handleAddTag}>

          <Text style={styles.addTagText}>Add</Text>

        </Pressable>

      </View>

      <Pressable style={styles.button} onPress={handleSaveToDo}>

        <Text style={styles.buttonText}>{existingTodo ? "Update To-Do" : "Add To-Do"}</Text>

      </Pressable>

    </View>

  );

};

export default AddToDo;

1. A képen szöveg, képernyőkép, szoftver, Operációs rendszer látható

   Automatikusan generált leírás**CompletedTasks komponens létrehozása**

* **Teendő kártyák**: Az itt lévő kártyák a már teljesített teendők, az egyes kártyákon lévő nyíllal visszavonhatjuk a teljesítésüket, és ismét teljesítésre váró teendők lesznek, vagy pedig a kuka ikonra kattintva törölhetjük őket a teljesítettek közül, ebben az esetben véglegesen törlésre kerülnek.

import {

    View, Text, StyleSheet, FlatList, TouchableOpacity, Alert

} from "react-native";

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

import { FirestoreDB } from "../../FirebaseConfig";

import { collection, getDocs, doc, updateDoc, deleteDoc, query, where } from "firebase/firestore";

import { Ionicons } from "@expo/vector-icons";

import { useFocusEffect } from "@react-navigation/native";

const CompletedTasks = ({ navigation }) => {

    const [completedTodos, setCompletedTodos] = useState([]);

    const fetchCompletedTodos = async () => {

        try {

            const q = query(collection(FirestoreDB, "todos"), where("completed", "==", true));

            const querySnapshot = await getDocs(q);

            const todoList = querySnapshot.docs.map(doc => ({

                id: doc.id,

                ...doc.data(),

            }));

            setCompletedTodos(todoList);

        } catch (error) {

            console.error("Error fetching completed todos:", error);

        }

    };

    useFocusEffect(

        useCallback(() => {

            fetchCompletedTodos();

        }, [])

    );

    const handleRestore = async (id) => {

        try {

            await updateDoc(doc(FirestoreDB, "todos", id), { completed: false });

            setCompletedTodos(completedTodos.filter(todo => todo.id !== id));

        } catch (error) {

            console.error("Error restoring todo:", error);

        }

    };

    const handleDelete = async (id) => {

        Alert.alert("Delete Completed Task", "Are you sure?", [

            { text: "Cancel", style: "cancel" },

            {

                text: "Delete",

                onPress: async () => {

                    try {

                        await deleteDoc(doc(FirestoreDB, "todos", id));

                        setCompletedTodos(completedTodos.filter(todo => todo.id !== id));

                    } catch (error) {

                        console.error("Error deleting todo:", error);

                    }

                },

                style: "destructive"

            },

        ]);

    };

    return (

        <View style={styles.container}>

            <Text style={styles.header}>Completed Tasks</Text>

            <FlatList

                data={completedTodos}

                keyExtractor={(item) => item.id}

                renderItem={({ item }) => (

                    <View style={styles.card}>

                        <Text style={styles.todoTitle}>{item.name}</Text>

                        <Text style={styles.todoDate}>{item.date}</Text>

                        <Text style={styles.todoDescription}>{item.description}</Text>

                        <View style={styles.actionButtons}>

                            <TouchableOpacity onPress={() => handleRestore(item.id)} style={styles.restoreButton}>

                                <Ionicons name="refresh-outline" size={24} color="blue" />

                            </TouchableOpacity>

                            <TouchableOpacity onPress={() => handleDelete(item.id)} style={styles.deleteButton}>

                                <Ionicons name="trash-outline" size={24} color="red" />

                            </TouchableOpacity>

                        </View>

                    </View>

                )}

            />

        </View>

    );

};

export default CompletedTasks;

1. A képen szöveg, képernyőkép, szoftver, Multimédiás szoftver látható

   Automatikusan generált leírás**Beállítások komponens létrehozása**

* **Sötét mód**: A sötét mód bekapcsolásával az alkalmazás átvált a világos helyett a sötét módú háttérre.

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

import { View, Text, Switch, StyleSheet, Pressable } from "react-native";

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

import { Ionicons } from "@expo/vector-icons";

const Settings = ({ navigation }) => {

  const [darkMode, setDarkMode] = useState(false);

  useEffect(() => {

    const loadSettings = async () => {

      const savedDarkMode = await AsyncStorage.getItem("darkMode");

      if (savedDarkMode !== null) setDarkMode(JSON.parse(savedDarkMode));

    };

    loadSettings();

  }, []);

  const toggleDarkMode = async () => {

    setDarkMode((prev) => {

      AsyncStorage.setItem("darkMode", JSON.stringify(!prev));

      return !prev;

    });

  };

  return (

    <View style={[styles.container, darkMode && styles.darkBackground]}>

      <View style={styles.header}>

        <Pressable onPress={() => navigation.goBack()} style={styles.headerButton}>

          <Ionicons name="arrow-back" size={24} color={darkMode ? "white" : "black"} />

        </Pressable>

        <Text style={[styles.title, darkMode && styles.darkText]}>Settings</Text>

      </View>

      <View style={styles.settingRow}>

        <Text style={[styles.label, darkMode && styles.darkText]}>Dark Mode</Text>

        <Switch value={darkMode} onValueChange={toggleDarkMode} />

      </View>

    </View>

  );

};

export default Settings;