**Source Code For Manobal**// =============================================

// BACKEND SOURCE CODE

// =============================================

// =============================================

// server.js

// =============================================

const express = require('express');

const dotenv = require('dotenv');

const cors = require('cors');

const mongoose = require('mongoose');

const authRouter = require('./routes/authRoute');

const mainRouter = require('./routes/mainRoute');

// Important Calls

const app = express();

dotenv.config();

app.use(cors());

app.use(express.json());

// Declarations

const PORT = process.env.PORT || 8000;

const HOST = process.env.HOST;

const MONGO\_URL = process.env.MONGO\_URL;

// API Routes

app.use('/api/auth', authRouter);

app.use('/api/main', mainRouter);

// Server Listen & DB Connection

mongoose

.connect(MONGO\_URL)

.then(() => {

console.log('DB is Connected');

app.listen(PORT, () => {

console.log(`Server is Running on http://${HOST}:${PORT}`);

});

})

.catch((error) => {

console.log(error);

});

// =============================================

// routes/authRoute.js

// =============================================

const express = require('express');

const { register, login } = require('../controllers/authController');

const authRouter = express.Router();

authRouter.post('/register', register);

authRouter.post('/login', login);

module.exports = authRouter;

// =============================================

// routes/mainRoute.js

// =============================================

const express = require('express');

const mainRouter = express.Router();

const {

assessment,

questions,

user,

score,

posts,

myposts,

createpost,

deletepost,

getnexttask,

completeTask

} = require('../controllers/mainController');

const authMiddleware = require('../middlewares/authMiddleware');

mainRouter.post('/assessment', authMiddleware, assessment);

mainRouter.get('/get-questions', questions);

mainRouter.get('/get-user', authMiddleware, user);

mainRouter.get('/get-score', authMiddleware, score);

mainRouter.get('/fetch-posts', authMiddleware, posts);

mainRouter.get('/fetch-my-posts', authMiddleware, myposts);

mainRouter.post('/create-post', authMiddleware, createpost);

mainRouter.delete('/delete-post/:postId', authMiddleware, deletepost);

mainRouter.get('/get-next-task', authMiddleware, getnexttask);

mainRouter.get('/complete-task', authMiddleware, completeTask);

module.exports = mainRouter;

// =============================================

// controllers/authController.js

// =============================================

const userModel = require('../models/userModel');

const bcrypt = require('bcryptjs');

const jwt = require('jsonwebtoken');

// Register API

const register = async (req, res) => {

try {

const { name, email, age, gender, password } = req.body;

if (!name || !email || !age || !gender || !password) {

return res.status(400).send({

sucess: false,

message: 'All fields must me filled',

});

}

const isUserExist = await userModel.findOne({ email });

if (isUserExist) {

return res.status(400).send({

sucess: false,

message: 'This user is already Registered!!',

});

}

const hashedPassword = await bcrypt.hash(password, 10);

const user = await userModel({

name,

email,

age,

gender,

isAssesmentDone:0,

password: hashedPassword,

});

await user.save();

return res.status(200).send({

sucess: true,

message: 'User has been registered sucessfully!!',

});

} catch (error) {

return res.status(200).send({

sucess: false,

message: error

});

}

};

// Login API

const login = async (req, res) => {

try {

const { email, password } = req.body;

if (!email || !password) {

return res.status(400).send({

success: false,

message: 'All fields must be filled',

});

}

const isUserExist = await userModel.findOne({ email });

if (!isUserExist) {

return res.status(400).send({

success: false,

message: 'This user is not registered!',

});

}

const isUserMatched = await bcrypt.compare(password, isUserExist.password);

if (!isUserMatched) {

return res.status(400).send({

success: false,

message: 'Invalid credentials',

});

}

const token = jwt.sign(

{ id: isUserExist.\_id, email: isUserExist.email },

process.env.JWT\_SECRET,

{ expiresIn: '1d' }

);

return res.status(200).send({

success: true,

message: 'Login successful!',

token,

user: {

id: isUserExist.\_id,

name: isUserExist.name,

email: isUserExist.email,

isAssesmentDone:isUserExist.isAssesmentDone

},

});

} catch (error) {

return res.status(200).send({

success: false,

message: error

});

}

};

module.exports = { register, login };

// =============================================

// controllers/mainController.js

// =============================================

const jwt = require('jsonwebtoken');

const questionModel = require('../models/questionModel');

const scoreModel = require('../models/scoreModel');

const userModel = require('../models/userModel');

const postModel = require('../models/postModel');

const prevTaskModel = require('../models/prevtasksModel');

const userTasksModel = require('../models/userTasksModel')

const TaskModel = require('../models/tasksModel');

const assessment = async (req, res) => {

try {

const { answers } = req.body;

if (!answers || !Array.isArray(answers)) {

return res.status(400).send({

success: false,

message: 'Invalid input: answers must be an array.',

});

}

const traits = {

Openness: [],

Conscientiousness: [],

Extraversion: [],

Agreeableness: [],

Neuroticism: [],

};

for (const answer of answers) {

const { id, \_id, answer: score } = answer;

if (score < 1 || score > 5) {

return res.status(400).send({

success: false,

message: `Invalid score for question ID ${id}: must be between 1 and 5.`,

});

}

const ques = await questionModel.findOne({ id });

const trait = ques.trait;

if (trait) {

traits[trait].push(score);

}

}

const result = {};

for (const trait in traits) {

const scores = traits[trait];

let sum = 0;

for (const score of scores) {

sum += score;

}

const average = sum / scores.length;

result[trait] = average.toFixed(2);

}

const userEmail = req.user.email;

const scoreExist = await scoreModel.findOne({ userEmail });

if (scoreExist) {

await scoreModel.deleteOne({ userEmail });

}

const userScore = new scoreModel({ userEmail, scores: result });

await userScore.save();

await userModel.updateOne({ email: userEmail }, { isAssesmentDone: true });

return res.status(200).send({

success: true,

message: 'Personality assessment completed successfully!',

data: result,

});

} catch (error) {

return res.status(500).send({

success: false,

message: error.message,

});

}

};

const questions = async (req, res) => {

try {

const ques = await questionModel.find();

const quesLen = ques.length;

return res.status(200).send({

success: true,

message: 'Successful!',

noOfQuestions: quesLen,

questions: ques,

});

} catch (error) {

return res.status(500).send({

success: false,

message: error.message,

});

}

};

const user = async (req, res) => {

try {

const userEmail = req.user.email;

const user = await userModel.findOne({ email: userEmail });

if (!user) {

return res.status(400).send({

success: false,

message: 'This user is not registered!',

});

}

return res.status(200).send({

success: true,

message: 'User fetched sucessfuly!',

user: user,

});

} catch (error) {

return res.status(500).send({

success: false,

message: error.message,

});

}

};

const score = async (req, res) => {

try {

const userEmail = req.user.email;

const userScore = await scoreModel.findOne({ userEmail });

if (!userScore) {

return res.status(400).send({

success: false,

message: 'Score is not calculated yet!',

});

}

return res.status(200).send({

success: true,

message: 'Score fetched sucessfuly!',

score: userScore,

});

} catch (error) {

return res.status(500).send({

success: false,

message: error.message,

});

}

};

const posts = async (req, res) => {

try {

const posts = await postModel.find().sort({ createdAt: -1 });

return res.status(200).send({

success: true,

message: 'posts fetched sucessfuly!',

posts: posts,

});

} catch (err) {

return res.status(500).send({

success: false,

message: error.message,

});

}

};

const myposts = async (req, res) => {

try {

const posts = await postModel

.find({ author: req.user.email })

.sort({ createdAt: -1 });

return res.status(200).send({

success: true,

message: 'posts fetched sucessfuly!',

posts: posts,

});

} catch (err) {

return res.status(400).send({

success: false,

message: error.message,

});

}

};

const createpost = async (req, res) => {

try {

if (!req.body.title || !req.body.content) {

return res.status(400).json({ error: 'Title and content are required' });

}

const post = await postModel({

title: req.body.title,

content: req.body.content,

author: req.user.email,

});

post.save();

return res.status(200).send({

success: true,

message: 'Post created sucessfuly!',

});

} catch (error) {

return res.status(500).send({

success: false,

message: error.message,

});

}

};

const deletepost = async (req, res) => {

const { postId } = req.params;

try {

const deletedPost = await postModel.findByIdAndDelete(postId);

if (!deletedPost) {

return res.status(400).send({

success: false,

message: 'This post is not registered!',

});

}

return res.status(200).send({

success: true,

message: 'Post deleted sucessfuly!',

});

} catch (error) {

return res.status(500).send({

success: false,

message: error.message,

});

}

};

const getnexttask = async (req, res) => {

try {

const orderList = ['openness', 'conscientiousness', 'extraversion', 'agreeableness', 'neuroticism'];

const userEmail = req.user.email;

const existingPendingTask = await userTasksModel.findOne({ userEmail, status: 'pending' }).populate('taskId');

if (existingPendingTask) {

return res.status(200).send({

success: true,

message: 'You have an ongoing task!',

task: existingPendingTask.taskId,

});

}

const userScore = await scoreModel.findOne({ userEmail });

if (!userScore) {

return res.status(400).send({

success: false,

message: 'Score is not registered!',

});

}

} catch (error) {

return res.status(500).send({

success: false,

message: error.message,

});

}

};

const completeTask = async (req, res) => {

try {

const userEmail = req.user.email;

const taskId = req.query.taskId;

if (!taskId) {

return res.status(400).send({

success: false,

message: 'Task ID is required!',

});

}

const userTask = await userTasksModel.findOne({ userEmail, taskId });

if (!userTask) {

return res.status(400).send({

success: false,

message: 'Task not found!',

});

}

userTask.status = 'completed';

await userTask.save();

return res.status(200).send({

success: true,

message: 'Task completed successfully!',

});

} catch (error) {

return res.status(500).send({

success: false,

message: error.message,

});

}

};

module.exports = {

assessment,

questions,

user,

score,

posts,

myposts,

createpost,

deletepost,

getnexttask,

completeTask

};

// =============================================

// middlewares/authMiddleware.js

// =============================================

const jwt = require('jsonwebtoken');

const authMiddleware = (req, res, next) => {

const token = req.header('manobal');

if (!token) {

return res.status(401).json({ message: 'No token provided, access denied' });

}

try {

const decoded = jwt.verify(token, process.env.JWT\_SECRET);

req.user = decoded;

next();

} catch (err) {

return res.status(401).json({ message: 'Invalid token' });

}

};

module.exports = authMiddleware;

// =============================================

// models/userModel.js

// =============================================

const mongoose = require('mongoose');

const userSchema = new mongoose.Schema(

{

name: String,

email: {

type:String,

required: true,

unique:true

},

gender: String,

isAssesmentDone:Boolean,

age: Number,

password: {

type:String,

required: true,

},

},

{

timestamps: true,

}

);

const userModel = new mongoose.model('user', userSchema);

module.exports = userModel;

// =============================================

// models/questionModel.js

// =============================================

const mongoose = require('mongoose');

const questionSchema = new mongoose.Schema({

id: {

type: Number,

required: true,

unique: true

},

question: {

type: String,

required: true,

trim: true

},

trait: {

type: String,

required: true,

enum: ['Openness', 'Conscientiousness', 'Extraversion', 'Agreeableness', 'Neuroticism']

},

});

module.exports = mongoose.model('question', questionSchema);

// =============================================

// models/scoreModel.js

// =============================================

const mongoose = require("mongoose");

const scoreSchema = new mongoose.Schema(

{

userEmail: {

type: String,

required: true,

},

scores: {

Openness: {

type: Number,

default: 0,

},

Conscientiousness: {

type: Number,

default: 0,

},

Extraversion: {

type: Number,

default: 0,

},

Agreeableness: {

type: Number,

default: 0,

},

Neuroticism: {

type: Number,

default: 0,

},

},

},

{ timestamps: true }

);

const scoreModel = mongoose.model("score", scoreSchema);

module.exports = scoreModel;

// =============================================

// models/postModel.js

// =============================================

const mongoose = require('mongoose');

const postSchema = new mongoose.Schema({

title: { type: String, required: true },

content: { type: String, required: true },

author: { type: String, default:null },

likes: { type: Number, default: 0 },

createdAt: { type: Date, default: Date.now },

updatedAt: { type: Date, default: Date.now },

});

const postModel = mongoose.model('Post', postSchema);

module.exports = postModel;

// =============================================

// models/tasksModel.js

// =============================================

const mongoose = require('mongoose');

const Schema = mongoose.Schema;

const TaskSchema = new Schema(

{

title: {

type: String,

required: true,

},

description: {

type: String,

required: true,

},

trait: {

type: String,

required: true,

enum: [

'openness',

'conscientiousness',

'extraversion',

'agreeableness',

'neuroticism',

],

},

level: {

type: String,

required: true,

enum: ['beginner', 'intermediate', 'advanced'],

},

category: {

type: String,

required: true,

},

},

{ timestamps: true }

);

const Tasks = mongoose.model('tasks', TaskSchema);

module.exports = Tasks;

// =============================================

// models/userTasksModel.js

// =============================================

const mongoose = require('mongoose');

const Schema = mongoose.Schema;

const UserTaskSchema = new Schema(

{

userEmail: {

type: String,

required: true,

index: true,

},

taskId: {

type: mongoose.Schema.Types.ObjectId,

ref: 'tasks',

required: true,

},

status: {

type: String,

enum: ['pending', 'completed'],

default: 'pending',

},

},

{ timestamps: true }

);

const UserTasks = mongoose.model('user\_tasks', UserTaskSchema);

module.exports = UserTasks;

// =============================================

// models/prevtasksModel.js

// =============================================

const mongoose = require('mongoose');

const Schema = mongoose.Schema;

const TaskFeedbackSchema = new Schema({

userEmail: {

type: String,

required: true,

},

taskId: {

type: String,

required: true,

},

feedback: {

type: String,

default: '',

},

completedAt: {

type: Date,

default: Date.now,

},

}, { timestamps: true });

const prevTasks = mongoose.model('prevTasks', TaskFeedbackSchema);

module.exports = prevTasks;  
  
  
  
// =============================================

// FRONTEND SOURCE CODE

// =============================================

// =============================================

// app/\_layout.jsx

// =============================================

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

import { useFonts } from "expo-font";

import React from 'react';

import { Stack } from 'expo-router';

import "./global.css";

const RootLayout = () => {

const [loaded] = useFonts({

"Jakarta-Bold": require("../assets/fonts/PlusJakartaSans-Bold.ttf"),

"Jakarta-ExtraBold": require("../assets/fonts/PlusJakartaSans-ExtraBold.ttf"),

"Jakarta-ExtraLight": require("../assets/fonts/PlusJakartaSans-ExtraLight.ttf"),

"Jakarta-Light": require("../assets/fonts/PlusJakartaSans-Light.ttf"),

"Jakarta-Medium": require("../assets/fonts/PlusJakartaSans-Medium.ttf"),

Jakarta: require("../assets/fonts/PlusJakartaSans-Regular.ttf"),

"Jakarta-SemiBold": require("../assets/fonts/PlusJakartaSans-SemiBold.ttf"),

});

return (

<Stack>

<Stack.Screen name="index" options={{ headerShown: false }} />

<Stack.Screen name="(auth)" options={{ headerShown: false }} />

<Stack.Screen name="(tabs)" options={{ headerShown: false }} />

<Stack.Screen name="createpost" options={{ headerShown: false }} />

<Stack.Screen name="allmineposts" options={{ headerShown: false }} />

<Stack.Screen name="viewprofile" options={{ headerShown: false }} />

</Stack>

);

};

export default RootLayout;

const styles = StyleSheet.create({

container: {

flex: 1,

backgroundColor: '#fff',

alignItems: 'center',

justifyContent: 'center',

},

});

// =============================================

// app/global.css

// =============================================

@tailwind base;

@tailwind components;

@tailwind utilities;

// =============================================

// app/index.jsx

// =============================================

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

import React from 'react';

import { useRouter } from 'expo-router';

import { StatusBar } from 'expo-status-bar';

const Index = () => {

const router = useRouter();

return (

<View className="flex-1 bg-white">

<StatusBar style="dark" />

<View className="flex-1 items-center justify-center px-4">

<Image

source={require('../assets/logo.png')}

className="w-40 h-40"

resizeMode="contain"

/>

<Text className="text-3xl font-bold text-center mt-4">

Welcome to Manobal

</Text>

<Text className="text-gray-500 text-center mt-2">

Your personal mental health companion

</Text>

<View className="w-full mt-8 space-y-4">

<TouchableOpacity

onPress={() => router.push('/(auth)/sign-in')}

className="bg-blue-500 p-4 rounded-full"

>

<Text className="text-white text-center font-semibold text-lg">

Sign In

</Text>

</TouchableOpacity>

<TouchableOpacity

onPress={() => router.push('/(auth)/sign-up')}

className="bg-white p-4 rounded-full border border-blue-500"

>

<Text className="text-blue-500 text-center font-semibold text-lg">

Sign Up

</Text>

</TouchableOpacity>

</View>

</View>

</View>

);

};

export default Index;

// =============================================

// app/(auth)/\_layout.jsx

// =============================================

import { Stack } from 'expo-router';

import React from 'react';

const AuthLayout = () => {

return (

<Stack>

<Stack.Screen name="sign-in" options={{ headerShown: false }} />

<Stack.Screen name="sign-up" options={{ headerShown: false }} />

<Stack.Screen name="assessment" options={{ headerShown: false }} />

</Stack>

);

};

export default AuthLayout;

// =============================================

// app/(auth)/sign-in.jsx

// =============================================

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

import React, { useState } from 'react';

import { useRouter } from 'expo-router';

import { mainService } from '../../services/mainService';

const SignIn = () => {

const router = useRouter();

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

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

const handleSignIn = async () => {

try {

const response = await mainService.login({ email, password });

if (response.success) {

router.replace('/(tabs)');

} else {

Alert.alert('Error', response.message);

}

} catch (error) {

Alert.alert('Error', 'Something went wrong');

}

};

return (

<View className="flex-1 bg-white p-4">

<View className="flex-1 justify-center">

<Text className="text-3xl font-bold text-center mb-8">Sign In</Text>

<View className="space-y-4">

<TextInput

className="border border-gray-300 p-4 rounded-lg"

placeholder="Email"

value={email}

onChangeText={setEmail}

keyboardType="email-address"

autoCapitalize="none"

/>

<TextInput

className="border border-gray-300 p-4 rounded-lg"

placeholder="Password"

value={password}

onChangeText={setPassword}

secureTextEntry

/>

<TouchableOpacity

onPress={handleSignIn}

className="bg-blue-500 p-4 rounded-lg"

>

<Text className="text-white text-center font-semibold">Sign In</Text>

</TouchableOpacity>

<TouchableOpacity

onPress={() => router.push('/(auth)/sign-up')}

className="mt-4"

>

<Text className="text-blue-500 text-center">

Don't have an account? Sign Up

</Text>

</TouchableOpacity>

</View>

</View>

</View>

);

};

export default SignIn;

// =============================================

// app/(auth)/sign-up.jsx

// =============================================

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

import React, { useState } from 'react';

import { useRouter } from 'expo-router';

import { mainService } from '../../services/mainService';

const SignUp = () => {

const router = useRouter();

const [name, setName] = useState('');

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

const [age, setAge] = useState('');

const [gender, setGender] = useState('');

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

const handleSignUp = async () => {

try {

const response = await mainService.register({

name,

email,

age: parseInt(age),

gender,

password,

});

if (response.success) {

router.replace('/(auth)/assessment');

} else {

Alert.alert('Error', response.message);

}

} catch (error) {

Alert.alert('Error', 'Something went wrong');

}

};

return (

<View className="flex-1 bg-white p-4">

<View className="flex-1 justify-center">

<Text className="text-3xl font-bold text-center mb-8">Sign Up</Text>

<View className="space-y-4">

<TextInput

className="border border-gray-300 p-4 rounded-lg"

placeholder="Name"

value={name}

onChangeText={setName}

/>

<TextInput

className="border border-gray-300 p-4 rounded-lg"

placeholder="Email"

value={email}

onChangeText={setEmail}

keyboardType="email-address"

autoCapitalize="none"

/>

<TextInput

className="border border-gray-300 p-4 rounded-lg"

placeholder="Age"

value={age}

onChangeText={setAge}

keyboardType="numeric"

/>

<TextInput

className="border border-gray-300 p-4 rounded-lg"

placeholder="Gender"

value={gender}

onChangeText={setGender}

/>

<TextInput

className="border border-gray-300 p-4 rounded-lg"

placeholder="Password"

value={password}

onChangeText={setPassword}

secureTextEntry

/>

<TouchableOpacity

onPress={handleSignUp}

className="bg-blue-500 p-4 rounded-lg"

>

<Text className="text-white text-center font-semibold">Sign Up</Text>

</TouchableOpacity>

<TouchableOpacity

onPress={() => router.push('/(auth)/sign-in')}

className="mt-4"

>

<Text className="text-blue-500 text-center">

Already have an account? Sign In

</Text>

</TouchableOpacity>

</View>

</View>

</View>

);

};

export default SignUp;

// =============================================

// app/(auth)/assessment.jsx

// =============================================

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

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

import { useRouter } from 'expo-router';

import { mainService } from '../../services/mainService';

const Assessment = () => {

const router = useRouter();

const [questions, setQuestions] = useState([]);

const [answers, setAnswers] = useState({});

const [currentQuestion, setCurrentQuestion] = useState(0);

useEffect(() => {

fetchQuestions();

}, []);

const fetchQuestions = async () => {

try {

const response = await mainService.getQuestions();

if (response.success) {

setQuestions(response.questions);

}

} catch (error) {

Alert.alert('Error', 'Failed to fetch questions');

}

};

const handleAnswer = (score) => {

setAnswers({

...answers,

[questions[currentQuestion].id]: score,

});

};

const handleNext = () => {

if (currentQuestion < questions.length - 1) {

setCurrentQuestion(currentQuestion + 1);

} else {

submitAssessment();

}

};

const submitAssessment = async () => {

try {

const formattedAnswers = Object.entries(answers).map(([id, answer]) => ({

id,

answer,

}));

const response = await mainService.submitAssessment(formattedAnswers);

if (response.success) {

router.replace('/(tabs)');

}

} catch (error) {

Alert.alert('Error', 'Failed to submit assessment');

}

};

if (questions.length === 0) {

return (

<View className="flex-1 justify-center items-center">

<Text>Loading questions...</Text>

</View>

);

}

return (

<View className="flex-1 bg-white p-4">

<ScrollView className="flex-1">

<Text className="text-2xl font-bold mb-4">

Question {currentQuestion + 1} of {questions.length}

</Text>

<Text className="text-lg mb-8">{questions[currentQuestion].question}</Text>

<View className="space-y-4">

{[1, 2, 3, 4, 5].map((score) => (

<TouchableOpacity

key={score}

onPress={() => handleAnswer(score)}

className={`p-4 rounded-lg border ${

answers[questions[currentQuestion].id] === score

? 'bg-blue-500 border-blue-500'

: 'border-gray-300'

}`}

>

<Text

className={`text-center ${

answers[questions[currentQuestion].id] === score

? 'text-white'

: 'text-gray-700'

}`}

>

{score}

</Text>

</TouchableOpacity>

))}

</View>

<TouchableOpacity

onPress={handleNext}

className="bg-blue-500 p-4 rounded-lg mt-8"

>

<Text className="text-white text-center font-semibold">

{currentQuestion < questions.length - 1 ? 'Next' : 'Submit'}

</Text>

</TouchableOpacity>

</ScrollView>

</View>

);

};

export default Assessment;

// =============================================

// app/(tabs)/\_layout.jsx

// =============================================

import { Tabs } from 'expo-router';

import { FontAwesome } from '@expo/vector-icons';

import React from 'react';

const TabLayout = () => {

return (

<Tabs

screenOptions={{

tabBarActiveTintColor: '#3b82f6',

tabBarInactiveTintColor: '#6b7280',

tabBarStyle: {

backgroundColor: '#ffffff',

borderTopWidth: 1,

borderTopColor: '#e5e7eb',

},

}}

>

<Tabs.Screen

name="home"

options={{

title: 'Home',

tabBarIcon: ({ color }) => (

<FontAwesome name="home" size={24} color={color} />

),

}}

/>

<Tabs.Screen

name="tasks"

options={{

title: 'Tasks',

tabBarIcon: ({ color }) => (

<FontAwesome name="tasks" size={24} color={color} />

),

}}

/>

<Tabs.Screen

name="community"

options={{

title: 'Community',

tabBarIcon: ({ color }) => (

<FontAwesome name="users" size={24} color={color} />

),

}}

/>

<Tabs.Screen

name="chats"

options={{

title: 'Chats',

tabBarIcon: ({ color }) => (

<FontAwesome name="comments" size={24} color={color} />

),

}}

/>

<Tabs.Screen

name="contact\_us"

options={{

title: 'Contact',

tabBarIcon: ({ color }) => (

<FontAwesome name="envelope" size={24} color={color} />

),

}}

/>

</Tabs>

);

};

export default TabLayout;

// =============================================

// services/mainService.js

// =============================================

import axios from 'axios';

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

const BASE\_URL = 'http://localhost:8000/api';

export const mainService = {

async login(credentials) {

try {

const response = await axios.post(`${BASE\_URL}/auth/login`, credentials);

if (response.data.success) {

await AsyncStorage.setItem('token', response.data.token);

}

return response.data;

} catch (error) {

throw error;

}

},

async register(userData) {

try {

const response = await axios.post(`${BASE\_URL}/auth/register`, userData);

return response.data;

} catch (error) {

throw error;

}

},

async getQuestions() {

try {

const response = await axios.get(`${BASE\_URL}/main/get-questions`);

return response.data;

} catch (error) {

throw error;

}

},

async submitAssessment(answers) {

try {

const token = await AsyncStorage.getItem('token');

const response = await axios.post(

`${BASE\_URL}/main/assessment`,

{ answers },

{

headers: {

manobal: token,

},

}

);

return response.data;

} catch (error) {

throw error;

}

},

async getUser() {

try {

const token = await AsyncStorage.getItem('token');

const response = await axios.get(`${BASE\_URL}/main/get-user`, {

headers: {

manobal: token,

},

});

return response.data;

} catch (error) {

throw error;

}

},

async getScore() {

try {

const token = await AsyncStorage.getItem('token');

const response = await axios.get(`${BASE\_URL}/main/get-score`, {

headers: {

manobal: token,

},

});

return response.data;

} catch (error) {

throw error;

}

},

async getPosts() {

try {

const token = await AsyncStorage.getItem('token');

const response = await axios.get(`${BASE\_URL}/main/fetch-posts`, {

headers: {

manobal: token,

},

});

return response.data;

} catch (error) {

throw error;

}

},

async createPost(postData) {

try {

const token = await AsyncStorage.getItem('token');

const response = await axios.post(

`${BASE\_URL}/main/create-post`,

postData,

{

headers: {

manobal: token,

},

}

);

return response.data;

} catch (error) {

throw error;

}

},

async deletePost(postId) {

try {

const token = await AsyncStorage.getItem('token');

const response = await axios.delete(

`${BASE\_URL}/main/delete-post/${postId}`,

{

headers: {

manobal: token,

},

}

);

return response.data;

} catch (error) {

throw error;

}

},

async getNextTask() {

try {

const token = await AsyncStorage.getItem('token');

const response = await axios.get(`${BASE\_URL}/main/get-next-task`, {

headers: {

manobal: token,

},

});

return response.data;

} catch (error) {

throw error;

}

},

async completeTask(taskId) {

try {

const token = await AsyncStorage.getItem('token');

const response = await axios.get(

`${BASE\_URL}/main/complete-task?taskId=${taskId}`,

{

headers: {

manobal: token,

},

}

);

return response.data;

} catch (error) {

throw error;

}

},

};