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

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

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

import { Api } from '../providers/api/api';

import AgressionMeterQuestionProvider from '../providers/agressionmeter-question/agressionmeter-question';

import ProfileProvider from '../providers/profile/profile';

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

import RenderHTML from 'react-native-render-html'; // Import RenderHTML

const api = new Api('http://aggressionmanagement.com/api');

const aggressionMeterProvider = new AgressionMeterQuestionProvider(api);

const profileProvider = new ProfileProvider(api);

const QuestionPage: React.FC = () => {

  const route = useRoute();

  const [questions, setQuestions] = useState<any[]>([]);

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

  const [error, setError] = useState<string | null>(null);

  const [firstName, setFirstName] = useState<string | null>(null);

  const [componentName, setComponentName] = useState<string>('');

  const [modalVisible, setModalVisible] = useState<boolean>(false);

  const [selectedExplanation, setSelectedExplanation] = useState<string | null>(null);

  useEffect(() => {

    fetchQuestions();

  }, [fetchQuestions]);

  const fetchQuestions = useCallback(async () => {

    try {

      setLoading(true);

      const userData = await AsyncStorage.getItem('user');

      if (!userData) {

        Alert.alert('Error', 'User not authenticated.');

        return;

      }

      const { user\_id: storedUserId, token: storedToken } = JSON.parse(userData);

      if (!storedUserId || !storedToken) {

        Alert.alert('Error', 'User not authenticated.');

        return;

      }

      const userInfoResponse = await profileProvider.user\_info({

        token: storedToken,

        user\_id: storedUserId,

      });

      setFirstName(userInfoResponse?.data?.firstname || 'Guest');

      const params = route as any;

      const { token, user\_id, type } = params.params.data;

      const pageType = getPageType(type);

      const response = await aggressionMeterProvider.emergencyMeterQuestion({

        token,

        type: pageType,

        user\_id,

      });

      if (response && response.result !== 'failed') {

        const updatedQuestions = response.map((item: any) => ({

          ...item,

          is\_selected: isSelected(item) ? '1' : '0',

        }));

        setQuestions(updatedQuestions);

        setComponentName(params.params.data.name); // Setting the component name

      } else {

        showAlert('Error', response.msg || 'Failed to load questions');

      }

    } catch (err) {

      setError('No internet connection, make sure Wi-Fi or cellular data is turned on, then try again.');

    } finally {

      setLoading(false);

    }

  }, [route.params]);

  const getPageType = (typeId: string) => {

    const pageTypes: any = {

      behavior: 'behavior',

      communication: 'communication',

      interaction: 'interaction',

      demeanor: 'demeanor',

    };

    return pageTypes[typeId] || '';

  };

  const isSelected = (item: any) => {

    return item.is\_selected === '1';

  };

  const showAlert = (title: string, message: string) => {

    Alert.alert(title, message);

  };

  const handleSkip = () => {

    console.log("Skip button pressed");

  };

  const handleQuestionSelect = (qes: any) => {

    console.log('Question selected:', qes);

  };

  const handleInfoPress = (explanation: string) => {

    setSelectedExplanation(explanation);

    setModalVisible(true);

  };

  const closeModal = () => {

    setModalVisible(false);

    setSelectedExplanation(null);

  };

  if (loading) {

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

  }

  return (

    <View style={styles.container}>

      <Text style={styles.header}>Welcome, {firstName}</Text>

      <Text style={styles.title}>Select Level of Aggression</Text>

      <Text style={styles.subtitle}>{componentName} Options</Text>

      {error && <Text style={styles.errorText}>{error}</Text>}

      <TouchableOpacity style={styles.skipButton} onPress={handleSkip}>

        <Text style={styles.skipButtonText}>Skip</Text>

      </TouchableOpacity>

      <FlatList

        data={questions}

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

        renderItem={({ item }) => (

          <View style={styles.questionItem}>

            <Text style={styles.questionText}>{item.text || 'No question available'}</Text>

            <View style={styles.questionActions}>

              <TouchableOpacity

                style={item.is\_selected === '1' ? styles.selectedButton : styles.selectButton}

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

              >

                <Text>{item.is\_selected === '1' ? 'Selected' : 'Select'}</Text>

              </TouchableOpacity>

              <Text style={styles.ratingBadge}>{item.rating}</Text>

              <TouchableOpacity onPress={() => handleInfoPress(item.explanation)}>

                <Text style={styles.infoIcon}>ℹ️</Text>

              </TouchableOpacity>

            </View>

            <Text style={styles.aggressionType}>{item.aggression\_type}</Text>

          </View>

        )}

      />

      {/\* Modal for Explanation \*/}

      <Modal

        animationType="slide"

        transparent={true}

        visible={modalVisible}

        onRequestClose={closeModal}

      >

        <View style={styles.modalContainer}>

          <View style={styles.modalContent}>

            <ScrollView>

              {/\* Render the explanation as HTML \*/}

              <RenderHTML contentWidth={300} source={{ html: selectedExplanation }} />

            </ScrollView>

            <TouchableOpacity style={styles.closeButton} onPress={closeModal}>

              <Text style={styles.closeButtonText}>Close</Text>

            </TouchableOpacity>

          </View>

        </View>

      </Modal>

    </View>

  );

};

const styles = StyleSheet.create({

  container: {

    flex: 1,

    padding: 16,

  },

  header: {

    fontSize: 20,

    fontWeight: 'bold',

    marginBottom: 16,

  },

  title: {

    fontSize: 18,

    textAlign: 'center',

    marginBottom: 8,

  },

  subtitle: {

    fontSize: 16,

    textAlign: 'center',

    marginBottom: 16,

  },

  skipButton: {

    alignSelf: 'flex-end',

    backgroundColor: 'red',

    paddingHorizontal: 16,

    paddingVertical: 8,

    borderRadius: 5,

  },

  skipButtonText: {

    color: 'white',

  },

  questionItem: {

    padding: 10,

    borderBottomWidth: 1,

    borderBottomColor: '#ccc',

  },

  questionText: {

    fontSize: 16,

    color: 'black',

  },

  questionActions: {

    flexDirection: 'row',

    alignItems: 'center',

    marginTop: 8,

  },

  selectButton: {

    backgroundColor: 'blue',

    padding: 8,

    borderRadius: 5,

  },

  selectedButton: {

    backgroundColor: 'red',

    padding: 8,

    borderRadius: 5,

  },

  ratingBadge: {

    marginLeft: 16,

    fontSize: 16,

    color: '#000',

  },

  infoIcon: {

    marginLeft: 16,

    fontSize: 22,

    color: '#9d0808',

  },

  aggressionType: {

    fontSize: 14,

    color: '#555',

    marginTop: 4,

  },

  errorText: {

    color: 'red',

    marginBottom: 16,

  },

  modalContainer: {

    flex: 1,

    justifyContent: 'center',

    alignItems: 'center',

    backgroundColor: 'rgba(0, 0, 0, 0.5)',

  },

  modalContent: {

    width: '80%',

    backgroundColor: 'white',

    borderRadius: 10,

    padding: 20,

    alignItems: 'center',

  },

  closeButton: {

    backgroundColor: 'blue',

    padding: 10,

    borderRadius: 5,

  },

  closeButtonText: {

    color: 'white',

  },

});

export default QuestionPage;