**ErrorActions.js**

export const updateIsError = payload => ({

  type: 'error/isError',

  payload: payload,

});

export const updateErrorMsg = payload => ({

  type: 'error/errorMsg',

  payload: payload,

});

**PlaybackObjectActions.js**

export const updateDidJustFinish = payload => ({

  type: 'playbackObject/statusDidJustFinish',

  payload: payload,

});

export const updateIsPlaying = payload => ({

  type: 'playbackObject/isPlaying',

  payload: payload,

});

export const updateHasLoaded = payload => ({

  type: 'playbackObject/hasLoaded',

  payload: payload,

});

export const updateBtnIcon = payload => ({

  type: 'playbackObject/btnIcon',

  payload: payload,

});

export const updateHasStarted = payload => ({

  type: 'playbackObject/hasStarted',

  payload: payload,

});

export const updateTimeListened = payload => ({

  type: 'playbackObject/timeListened',

  payload: payload,

});

export const updateHasFinished = payload => ({

  type: 'playbackObject/hasFinished',

  payload: payload,

});

export const resetPlaybackObject = () => ({

  type: 'playbackObject/reset',

});

**ProgressActions.js**

export const updateProgressMessage = payload => ({

  type: 'progress/messageText',

  payload: payload,

});

**UserActions.js**

export const updateUserName = payload => ({

  type: 'user/userName',

  payload: payload,

});

export const updateHoursCompleted = payload => ({

  type: 'user/hoursCompleted',

  payload: payload,

});

export const updateSessionsCompleted = payload => ({

  type: 'user/sessionsCompleted',

  payload: payload,

});

export const updateFavoriteSession = payload => ({

  type: 'user/favoriteSession',

  payload: payload,

});

export const updateDayStreak = payload => ({

  type: 'user/dayStreak',

  payload: payload,

});

export const updateSelectedAvatar = payload => ({

  type: 'user/selectedAvatar',

  payload: payload,

});

export const updateShowWarning = payload => ({

  type: 'user/showWarning',

  payload: payload,

});

export const updateShowAvatarModal = payload => ({

  type: 'user/showAvatarModal',

  payload: payload,

});

export const updateResetUser = payload => ({

  type: 'user/resetUser',

  payload: payload,

});

AudioElement.js

import Media from './Media';

import AsyncStorageAPI from '../helpers/AsyncStorageAPI';

import FirebaseFetchAPI from '../helpers/FirebaseFetchAPI';

export default class AudioElement {

  constructor(name, isMainAudio) {

    this.name = name;

    this.isMainAudio = isMainAudio;

    this.Media = null;

  }

  \_loadFromDevice = async item => {

    let storedFile = await AsyncStorageAPI.getItem(item);

    return storedFile;

  };

  \_fetchFromFirebase = async item => {

    let storedFile = await FirebaseFetchAPI.fetchMedia(item.toLowerCase());

    return storedFile;

  };

  \_getAudioFromStoredLocation = async isDownloaded => {

    let storedFile = isDownloaded

      ? await this.\_loadFromDevice(this.name)

      : await this.\_fetchFromFirebase(this.name);

    return storedFile;

  };

  \_checkIfStored = async item => {

    let isDownloaded = await AsyncStorageAPI.isStoredInDevice(item);

    return isDownloaded;

  };

  setupAudioElement = async () => {

    let isDownloaded = await this.\_checkIfStored(this.name);

    let storedFile = await this.\_getAudioFromStoredLocation(isDownloaded);

    this.Media = new Media(storedFile, isDownloaded, this.isMainAudio);

    await this.Media.loadMedia();

  };

}

MainAudio.js

import AudioElement from './AudioElement';

export default class MainAudio extends AudioElement {

  constructor(name, isMainAudio) {

    super(name, isMainAudio);

  }

}

Media.js

import { Audio } from 'expo-av';

import {

  updateDidJustFinish,

  updateInstanceList,

} from '../actions/PlaybackObjectActions';

import ErrorAPI from '../helpers/ErrorAPI';

import store from '../store/Store';

export default class Media {

  constructor(source, isDownloaded, isMainAudio) {

    this.source = source;

    this.isDownloaded = isDownloaded;

    this.isMainAudio = isMainAudio;

    this.playbackInstance = null;

    Audio.setAudioModeAsync({

      staysActiveInBackground: true,

      interruptionModeAndroid: Audio.INTERRUPTION\_MODE\_ANDROID\_DUCK\_OTHERS,

      shouldDuckAndroid: true,

      playThroughEarpieceAndroid: false,

      allowsRecordingIOS: false,

      interruptionModeIOS: Audio.INTERRUPTION\_MODE\_IOS\_DO\_NOT\_MIX,

      playsInSilentModeIOS: true,

    });

  }

  \_onPlaybackStatusUpdate = status => {

    if (status.didJustFinish) {

      store.dispatch(updateDidJustFinish(true));

    }

  };

  loadMedia = async () => {

    // let mediaSource = this.isDownloaded ? this.source : { uri: this.source };

    try {

      let statusUpdate = this.isMainAudio ? this.\_onPlaybackStatusUpdate : null;

      let mediaSource = { uri: this.source };

      const { sound, status } = await Audio.Sound.createAsync(

        mediaSource,

        {},

        statusUpdate,

      );

      this.playbackInstance = sound;

    } catch (error) {

      ErrorAPI.errorHandler(

        error,

        'Sorry, there was an error loading the audio. Please check your internet connect, and try closing and reopening the app.',

      );

    }

  };

  playMedia = async () => {

    try {

      if (this.playbackInstance) {

        await this.playbackInstance.playAsync();

      }

    } catch (error) {

      ErrorAPI.errorHandler(

        error,

        'Sorry, there was an error playing the audio. Please close the app and try again.',

      );

    }

  };

  pauseMedia = async () => {

    if (this.playbackInstance) {

      await this.playbackInstance.pauseAsync();

    }

  };

  stopMedia = async () => {

    if (this.playbackInstance) {

      await this.playbackInstance.stopAsync();

    }

  };

  unloadMedia = async () => {

    if (this.playbackInstance) {

      await this.playbackInstance.unloadAsync();

      this.playbackInstance = null;

    }

  };

}

Session.js

import SoundBiteList from './SoundBiteList';

import MainAudio from './MainAudio';

import store from '../store/Store';

import { updateHasLoaded } from '../actions/PlaybackObjectActions';

import { updateProgressMessage } from '../actions/ProgressActions';

export default class Session {

  constructor(name, soundBites) {

    this.name = name;

    this.soundBites = soundBites;

    this.MainAudio = new MainAudio(this.name, true);

    if (this.soundBites) {

      this.SoundBiteList = new SoundBiteList(this.soundBites);

    }

    this.unsubscribe;

  }

  loadSession = async () => {

    store.dispatch(updateProgressMessage('Loading Main Audio'));

    console.log('Loading Session...');

    await this.MainAudio.setupAudioElement();

    // if statements around SoundBiteList prevent action if session is the intro track

    if (this.soundBites) {

      store.dispatch(updateProgressMessage('Loading Vocal Cues'));

      await this.SoundBiteList.setupSoundBites();

    }

    store.dispatch(updateProgressMessage(''));

  };

  playSession = () => {

    console.log('Playing Session...');

    if (this.MainAudio.Media) {

      this.MainAudio.Media.playMedia();

    }

    if (this.soundBites) {

      this.SoundBiteList.startSoundBites();

    }

  };

  pauseSession = () => {

    if (this.MainAudio.Media) {

      this.MainAudio.Media.pauseMedia();

    }

    if (this.soundBites) {

      this.SoundBiteList.pauseSoundBites();

    }

  };

  endSession = () => {

    if (this.MainAudio.Media) {

      this.MainAudio.Media.stopMedia();

    }

    if (this.soundBites) {

      this.SoundBiteList.stopSoundBites();

    }

  };

  \_unloadLogic = () => {

    /\*

    this is the main logic that will unload all the audio. Packaged in

    a separate function so that it can be used in conjunction with a

    subscription incase the audio hasn't loaded fully when unload is requested

    \*/

    if (this.MainAudio.Media) {

      this.MainAudio.Media.unloadMedia();

    }

    if (this.soundBites) {

      console.log('Unloading SoundBites');

      this.SoundBiteList.unloadSoundBites();

    }

    console.log('Done Unloading Audio');

  };

  \_unloadAudioSubscription = () => {

    /\*

    this subscription only gets activated when unloading of audio is

    requested before the audio has loaded fully loaded. It gets unsubscribed

    immediately after it finishes unloading the audio

    \*/

    let { playbackObject } = store.getState();

    if (playbackObject.hasLoaded) {

      this.\_unloadLogic();

      store.dispatch(updateHasLoaded(false));

      this.unsubscribe();

    }

  };

  unloadSession = () => {

    /\*

    this function gets called first in the unloading. It checks to see if the audio

    has fully loaded. If it has, it unloads normally, if not, it creates a subscription

    that will watch for the audio to load fully and then unload. This prevents errors from

    overloading the phones media player if the user backs out of a session before it loads

    and then loads another session.

    \*/

    let { playbackObject } = store.getState();

    if (playbackObject.hasLoaded) {

      this.\_unloadLogic();

    } else {

      this.unsubscribe = store.subscribe(this.\_unloadAudioSubscription);

    }

  };

}

SoundBite.js

import Timer from './Timer';

import Media from './Media';

import AudioElement from './AudioElement';

export default class SoundBite extends AudioElement {

  constructor(name, file, isDownloaded, delay) {

    super(name);

    this.file = file;

    this.isDownloaded = isDownloaded;

    this.Timer = null;

    this.delay = delay;

  }

  setupAudioElement = async () => {

    this.Media = new Media(this.file, this.isDownloaded, false);

    await this.Media.loadMedia();

    this.Timer = new Timer(this.Media.playMedia, this.delay);

  };

}

SoundBiteList.js

import SoundBite from './SoundBite';

import FirebaseFetchAPI from '../helpers/FirebaseFetchAPI';

import AsyncStorageAPI from '../helpers/AsyncStorageAPI';

import store from '../store/Store';

import { updateProgressMessage } from '../actions/ProgressActions';

export default class SoundBiteList {

  constructor(listOfSoundBites) {

    this.listOfSoundBites = listOfSoundBites;

    this.soundBiteArray = null;

  }

  \_loadFromDevice = async item => {

    let storedFile = await AsyncStorageAPI.getItem(item);

    return storedFile;

  };

  \_fetchFromFirebase = async item => {

    let storedFile = await FirebaseFetchAPI.fetchMedia(item.toLowerCase());

    return storedFile;

  };

  \_getAudioFromStoredLocation = async (isDownloaded, item) => {

    let storedFile = isDownloaded

      ? await this.\_loadFromDevice(item)

      : await this.\_fetchFromFirebase(item);

    return storedFile;

  };

  \_checkIfStored = async item => {

    let isDownloaded = await AsyncStorageAPI.isStoredInDevice(item);

    return isDownloaded;

  };

  \_timerMath = () => {

    let timers = [];

    for (i = 0; i < 10; i++) {

      let timeInMS = i \* 30000 + 5000;

      timers.push(timeInMS);

    }

    for (i = 0; i < 20; i++) {

      let timeInMS = i \* 15000 + (10 \* 30000 + 5000);

      timers.push(timeInMS);

    }

    return timers;

  };

  \_lengthenArray = array => {

    do {

      array = array.concat(array);

    } while (array.length < 30);

    return array;

  };

  \_shuffleArray = array => {

    // Fisher-Yates shuffle (adapted to not have same elements next to eachother)

    store.dispatch(updateProgressMessage('Shuffling Vocal Cues'));

    for (let i = array.length - 1; i > 0; i--) {

      do {

        const j = Math.floor(Math.random() \* (i + 1));

        [array[i], array[j]] = [array[j], array[i]];

      } while (array[i] === array[i + 1] || array[0] === array[1]);

    }

    return array;

  };

  setupSoundBites = async () => {

    console.log('Setting Up SoundBiteList...');

    let soundBiteNames = this.listOfSoundBites;

    let soundBiteContents = await Promise.all(

      soundBiteNames.map(async element => {

        let isDownloaded = await this.\_checkIfStored(element);

        let storedFile = await this.\_getAudioFromStoredLocation(

          isDownloaded,

          element,

        );

        return [element, storedFile, isDownloaded];

      }),

    );

    soundBiteContents = this.\_lengthenArray(soundBiteContents);

    soundBiteContents = soundBiteContents.slice(0, 30);

    soundBiteContents = this.\_shuffleArray(soundBiteContents);

    // setup a short array of SounBite objects to fetch the audio files

    let executionTimes = this.\_timerMath();

    let soundBiteArray = await Promise.all(

      soundBiteContents.map(async (element, index) => {

        let soundBite = new SoundBite(

          element[0],

          element[1],

          element[2],

          executionTimes[index],

        );

        await soundBite.setupAudioElement();

        return soundBite;

      }),

    );

    let lastSoundBite = await (async () => {

      let isDownloaded = await this.\_checkIfStored('all');

      let storedFile = await this.\_getAudioFromStoredLocation(

        isDownloaded,

        'all',

      );

      let soundBite = new SoundBite('all', storedFile, isDownloaded, 605000);

      await soundBite.setupAudioElement();

      return soundBite;

    })();

    soundBiteArray.push(lastSoundBite);

    console.log('Done setting up SoundBiteList');

    this.soundBiteArray = soundBiteArray;

  };

  startSoundBites = () => {

    if (this.soundBiteArray && this.soundBiteArray.length > 0) {

      this.soundBiteArray.forEach(element => {

        if (element.Timer.gotPaused) {

          element.Media.playMedia();

          element.Timer.gotPaused = false;

        }

        element.Timer.startTimer();

      });

    }

  };

  pauseSoundBites = () => {

    if (this.soundBiteArray && this.soundBiteArray.length > 0) {

      this.soundBiteArray.forEach((element, index, array) => {

        if (

          (index !== array.length - 1 &&

            element.Timer.hasStarted &&

            !array[index + 1].hasStarted) ||

          (index === array.length - 1 && element.Timer.hasStarted)

        ) {

          element.Media.pauseMedia();

          element.Timer.gotPaused = true;

        }

        element.Timer.pauseTimer();

      });

    }

  };

  stopSoundBites = () => {

    if (this.soundBiteArray && this.soundBiteArray.length > 0) {

      this.pauseSoundBites();

      this.soundBiteArray.forEach(element => {

        element.Timer.stopTimer();

      });

    }

  };

  unloadSoundBites = () => {

    if (this.soundBiteArray && this.soundBiteArray.length > 0) {

      this.soundBiteArray.forEach(element => {

        element.Media.unloadMedia();

        element.Timer.destroyTimer();

      });

    }

  };

}

Timer.js

import BackgroundTimer from 'react-native-background-timer';

export default class Timer {

  constructor(callback, delay) {

    this.delay = delay;

    this.callback = callback;

    this.instance = null;

    this.remaining = delay;

    this.startTime = null;

    this.totalTimePlayed = 0;

    this.hasStarted = false;

    this.gotPaused = false;

  }

  \_callbackFunction = () => {

    this.callback();

    this.hasStarted = true;

  };

  startTimer = () => {

    this.startTime = Date.now();

    this.remaining -= Date.now() - this.startTime;

    BackgroundTimer.clearTimeout(this.instance);

    if (this.remaining > 0) {

      clearTimeout(this.instance);

      // calling BackgroundTimer for Android

      this.instance = BackgroundTimer.setTimeout(

        this.\_callbackFunction,

        this.remaining,

      );

    }

  };

  pauseTimer = () => {

    this.remaining -= Date.now() - this.startTime;

    if (this.delay != this.remaining) {

      this.totalTimePlayed = this.delay - this.remaining;

    }

    BackgroundTimer.clearTimeout(this.instance);

  };

  stopTimer = () => {

    this.totalTimePlayed = this.delay - this.remaining;

    this.remaining = this.delay;

    BackgroundTimer.clearTimeout(this.instance);

  };

  destroyTimer = () => {

    this.stopTimer();

    this.instance = null;

  };

}

AboutComponent.js

import React from 'react';

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

import ArrowButton from '../components/ArrowButton';

const AboutComponent = props => {

  return (

    <View style={styles.Hero}>

      <Image resizeMode="contain" source={props.image} style={styles.Image} />

      <View style={styles.Info}>

        <Text style={styles.Header}>{props.header}</Text>

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

      </View>

      <View style={styles.Controls}>

        <ArrowButton

          screen={props.prevScreen}

          direction={'arrowleft'}

          shouldShow={props.shouldShowLeft}

          color={'white'}

          backgroundColor={'rgb(255, 101, 132)'}

        />

        <ArrowButton

          screen={props.nextScreen}

          direction={'arrowright'}

          shouldShow={props.shouldShowRight}

          color={'white'}

          backgroundColor={'rgb(255, 101, 132)'}

        />

      </View>

    </View>

  );

};

const styles = StyleSheet.create({

  Hero: {

    flex: 1,

    alignItems: 'center',

    justifyContent: 'center',

  },

  Header: {

    fontSize: 28,

    textAlign: 'left',

    color: 'rgb(30, 27, 57)',

    marginBottom: 7.5,

    fontFamily: 'JosefinSans-Bold',

  },

  Text: {

    fontSize: 21,

    lineHeight: 25,

    textAlign: 'left',

    color: 'rgb(30, 27, 57)',

    fontFamily: 'JosefinSans-Regular',

    marginTop: 7.5,

  },

  Button: {

    height: 50,

    width: 50,

    borderRadius: 50,

    alignItems: 'center',

    justifyContent: 'center',

    backgroundColor: 'rgb(30, 27, 57)',

  },

  Image: {

    height: '50%',

  },

  Controls: {

    width: Dimensions.get('window').width,

    paddingHorizontal: 20,

    flexDirection: 'row',

    justifyContent: 'space-around',

    top: -25,

  },

  Info: {

    display: 'flex',

    alignItems: 'center',

    justifyContent: 'center',

    width: '100%',

    height: 200,

    top: -25,

    paddingHorizontal: 30,

    marginLeft: 25,

    elevation: 15,

    borderRadius: 10,

    shadowColor: 'black',

    shadowOffset: {

      width: 10,

      height: 10,

    },

    shadowOpacity: 0.4,

    shadowRadius: 10,

    borderTopLeftRadius: 25,

    borderBottomLeftRadius: 25,

    marginBottom: 30,

    backgroundColor: 'white',

  },

});

export default AboutComponent;