```
// Copyright (c) 2017 Tracktunes Inc

import {
    Alert,
    AlertController,
    ActionSheet,
    ActionSheetController,
    Content,
    // ItemSliding,
    ModalController,
    NavController,
    Platform
} from 'ionic-angular';
import {
    ChangeDetectorRef,
    Component,
    ViewChild
} from '@angular/core';
import { AppState } from '../../services/app-state/app-state';
import { ButtonbarButton } from '../../components/button-bar/button-bar';
import { SelectionPage } from '../../pages';
import {
    MoveToPage
    // , TrackPage
} from '../';
// import { Keyboard } from '@ionic-native/keyboard';
import { AppFS } from '../../services';

/**
 * @name OrganizerPage
 * @description
 * Page of file/folder interface to all recorded files. AddFolderPage
 * music organizer.
 */
@Component({
    selector: 'organizer-page',
    templateUrl: 'organizer-page.html'
})
export class OrganizerPage extends SelectionPage {
    @ViewChild(Content) public content: Content;
    // UI uses directoryEntry
    public directoryEntry: DirectoryEntry;
    // we remember this just so we can uncheck it when
    // in dialog of delete button
    public unfiledDirectory: DirectoryEntry;
    // UI uses headerButtons
    public headerButtons: ButtonbarButton[];
    // UI uses footerButtons
    public footerButtons: ButtonbarButton[];
    private navController: NavController;
    // actionSheetController is used by add button
    private actionSheetController: ActionSheetController;
    private alertController: AlertController;
    private modalController: ModalController;
    private changeDetectorRef: ChangeDetectorRef;

    // appState now handled by selectionpage as private
    // private appState: AppState;

    // UI uses selectedPaths

    // selectedPaths now from SelectionPage
    // public selectedPaths: Set<string>;

    /**
     * @constructor
     * @param {NavController}
     * @param {AlertController}
     * @param {ModalController}
     * @param {AppState}
     * @param {Platform}
     */
    constructor(
        // keyboard: Keyboard,
        navController: NavController,
        alertController: AlertController,
        actionSheetController: ActionSheetController,
        modalController: ModalController,
        changeDetectorRef: ChangeDetectorRef,
        appState: AppState,
        appFS: AppFS,
        platform: Platform
    ) {
        super(appState, appFS);

        console.log('constructor():OrganizerPage');
        // this.keyboard = keyboard;
        this.changeDetectorRef = changeDetectorRef;
        this.directoryEntry = null;
        this.actionSheetController = actionSheetController;

        appState.get('lastViewedFolderPath').then(
            (path: string) => {
                this.switchFolder(path, false);
                appState.get('selectedPaths').then(
                    (selectedPaths: Set<string>) => {
                        this.selectedPaths = selectedPaths;
                    });
            } // (path: string) => {..
        ); // appState.get('lastViewedFolderPath').then(..

        this.navController = navController;
        this.alertController = alertController;
        this.modalController = modalController;

        // helper function used in disabledCB below
        const atHome: () => boolean = () => {
            return this.directoryEntry &&
                this.directoryEntry.name === '' &&
                this.directoryEntry.fullPath === '/';
        };

        this.headerButtons = [
            {
                text: 'Select...',
                leftIcon: platform.is('ios') ?
                    'radio-button-off' : 'square-outline',
                rightIcon: 'md-arrow-dropdown',
                clickCB: () => {
                    this.onClickSelectButton();
                },
                disabledCB: () => {
                    return this.entries.length <= 1;
                }
            },
            {
                text: 'Go home',
                leftIcon: 'home',
                clickCB: () => {
                    this.onClickHomeButton();
                },
                disabledCB: atHome
            },
            {
                text: 'Go to parent',
                leftIcon: 'arrow-up',
                rightIcon: 'folder',
                // rightIcon: 'ios-folder-outline',
```

```
                clickCB: () => {
                    this.onClickParentButton();
                },
                disabledCB: atHome
            },
            {
                text: 'Add...',
                leftIcon: 'add',
                clickCB: () => {
                    this.onClickAddButton();
                }
            }
        ];

        this.footerButtons = [
            {
                text: 'Info',
                leftIcon: 'information-circle',
                clickCB: () => {
                    this.onClickInfoButton();
                }
            },
            {
                text: 'Move to...',
                leftIcon: 'share-alt',
                rightIcon: 'folder',
                clickCB: () => {
                    this.onClickMoveButton();
                },
                disabledCB: () => {
                    return this.moveButtonDisabled();
                }
            },
            {
                text: 'Delete',
                leftIcon: 'trash',
                clickCB: () => {
                    this.onClickDeleteButton();
                },
                disabledCB: () => {
                    return this.deleteButtonDisabled();
                }
            },
            {
                text: 'Share',
                leftIcon: 'md-share',
                clickCB: () => {
                    this.onClickShareButton();
                }
            }
        ];

    }

    /**
     * UI calls this when the 'Select...' button is clicked.
     * @returns {void}
     */
    public onClickSelectButton(): void {
        console.log('onClickSelectButton()');

        let selectAlert: Alert = this.alertController.create();
        selectAlert.setTitle('Select which, in ' +
                             this.directoryEntry.fullPath);
        selectAlert.addButton({
            text: 'All',
            handler: () => {
                this.selectAllOrNoneInFolder(true);
            }
        });
        selectAlert.addButton({
            text: 'None',
            handler: () => {
                this.selectAllOrNoneInFolder(false);
            }
        });

        selectAlert.addButton('Cancel');
        selectAlert.present();
    }

    /**
     * UI calls this when the 'Go home' button is clicked.
     * @returns {void}
     */
    public onClickHomeButton(): void {
        console.log('onClickHomeButton()');
        this.switchFolder('/', true);
    }

    /**
     * UI calls this when the 'Go to parent' button is clicked.
     * @returns {void}
     */
    public onClickParentButton(): void {
        console.log('onClickParentButton()');
        const pathParts: string[] = this.directoryEntry.fullPath.split('/')
              .filter((str: string) => { return str !== ''; });
        const parentPath: string = '/' +
              pathParts.splice(0, pathParts.length - 1).join('/') +
              '/';
        this.switchFolder(parentPath, true);
    }

    /**
     * UI calls this when the 'Add...' button is clicked.
     * @returns {void}
     */
    public onClickAddButton(): void {
        console.log('onClickAddButton()');
        let actionSheet: ActionSheet = this.actionSheetController.create({
            title: 'Create new ... in ' + this.directoryEntry.fullPath,
            buttons: [
                {
                    text: 'Folder',
                    icon: 'folder',
                    handler: () => {
                        console.log('Add folder clicked.');
                        this.addFolder();
                    }
                },
                {
                    text: 'URL',
                    icon: 'link',
                    handler: () => {
                        console.log('Add URL clicked.');
                    }
                },
                {
                    text: 'Cancel',
                    role: 'cancel',
                    // icon: 'close',
                    handler: () => {
                        console.log('Cancel clicked.');
                    }
                }
            ]
        });
```

```
        actionSheet.present();
    }

    /**
     * UI calls this when the info button is clicked.
     * Shows cumulative info on all selected items.
     * @returns {void}
     */
    public onClickInfoButton(): void {
        console.log('onClickInfoButton');
    }

    /**
     * UI calls this when move button is clicked.
     * Moves selected items into a folder.
     * @returns {void}
     */
    public onClickMoveButton(): void {
        console.log('onClickMoveButton');
        // this.modalController.create(MoveToPage).present();
        this.navController.push(MoveToPage);
    }

    /**
     * UI calls this to determine whether to disable move button.
     * @returns {boolean}
     */
    public moveButtonDisabled(): boolean {
        // if the only thing selected is the unfiled folder
        // disable delete and move
        if (this.selectedPaths.size === 1 &&
            this.selectedPaths.has('/Unfiled/')) {
            return true;
        }
        return false;
    }

    /**
     * @returns {void}
     */
    private confirmAndDeleteSelected(): void {
        let nSelectedEntries: number = this.selectedPaths.size,
            itemsStr: string = nSelectedEntries.toString() + ' item' +
            ((nSelectedEntries > 1) ? 's' : ''),
            entries: string[] = Array.from(this.selectedPaths),
            // sortFun: (a: string, b: string) => number =
            // (a: string, b: string) => {
            //     const lenA: number = a.split('/').length,
            //           lenB: number = b.split('/').length;
            //     if (lenA < lenB) {
            //         return -1;
            //     }
            //     else if (lenA === lenB) {
            //         return 0;
            //     }
            //     else {
            //         return 1;
            //     }
            // },
            deleteAlert: Alert = this.alertController.create();

        // entries.sort(sortFun);
        entries.sort();
        console.log(entries);
        deleteAlert.setTitle('Are you sure you want to delete ' +
                             itemsStr + '?');
        deleteAlert.addButton('Cancel');
        deleteAlert.addButton({
            text: 'Yes',
            handler: () => {
                this.appFS.removeEntries(entries).subscribe(() => {
                    this.selectedPaths.clear();
                    this.appState.set(
                        'selectedPaths',
                        this.selectedPaths
                    ).then(
                        () => {
                            this.switchFolder(
                                this.getFullPath(this.directoryEntry),
                                false
                            );
                        });
                });
            }
        });

        deleteAlert.present();
    }

    /**
     * UI calls this when delete button is clicked.
     * @returns {void}
     */
    public onClickDeleteButton(): void {
        console.log('onClickDeleteButton()');
        if (this.selectedPaths.has('/Unfiled/')) {
            let deleteAlert: Alert = this.alertController.create();

            deleteAlert.setTitle('/Unfiled folder cannot be deleted. But it' +
                                 '\'s selected. Automatically unselect it?');
            deleteAlert.addButton('Cancel');
            deleteAlert.addButton({
                text: 'Yes',
                handler: () => {
                    this.selectedPaths.delete('/Unfiled/');
                    this.selectedPaths.delete(
                        this.getFullPath(this.unfiledDirectory)
                    );
                    this.confirmAndDeleteSelected();
                }
            });
            deleteAlert.present();
        }
        else {
            this.confirmAndDeleteSelected();
        }
    }

    /**
     * UI calls this to determine whether disable the delete button
     * @returns {boolean}
     */
    public deleteButtonDisabled(): boolean {
        // if the only thing selected is the unfiled folder
        // disable delete and move
        if (this.selectedPaths.size === 1 &&
            this.selectedPaths.has('/Unfiled/')) {
            return true;
        }
        return false;
    }

    /**
     * UI calls this when social sharing button is clicked
     * @returns {void}
     */
    public onClickShareButton(): void {
        console.log('onClickShareButton()');
```

```
    }

    /**
     * UI calls this when selected badge on top right is clicked
     * @returns {void}
     */
    public onClickSelectedBadge(): void {
        console.log('onClickSelectedBadge()');
        if (this.selectedPaths.size) {
            // only go to edit selections if at least one is selected
            this.navController.push(SelectionPage);
        }
    }

    /**
     * Switch to a new folder
     * @param {number} key of treenode corresponding to folder to switch to
     * @param {boolean} whether to update app state 'lastFolderViewed' property
     * @returns {void}
     */
    private switchFolder(
        path: string,
        bUpdateAppState: boolean = true
    ): void {
        console.log('OrganizerPage.switchFolder(' + path + ', ' +
                bUpdateAppState + ')');
        this.appFS.getPathEntry(path, false).subscribe(
            (directoryEntry: DirectoryEntry) => {
                this.directoryEntry = directoryEntry;
                if (!directoryEntry) {
                    alert('!directoryEntry!');
                }
                this.appFS.readDirectory(directoryEntry).subscribe(
                    (entries: Entry[]) => {
                        console.log('OrganizerPage.switchFolder() entries: ' +
                                entries);
                        console.log(this.selectedPaths);
                        console.dir(entries);
                        this.entries = entries;
                        this.detectChanges();
                        if (bUpdateAppState) {
                            this.appState.set(
                                'lastViewedFolderPath',
                                path
                            ).then();
                        }
                    },
                    (err1: any) => {
                        alert('err1: ' + err1);
                    }
                ); // this.appFS.readDirectory().susbscribe(..
            },
            (err2: any) => {
                alert('err2: ' + err2);
            }
        ); // this.appFS.getPathEntry(..).subscribe(..
    }

    private detectChanges(): void {
        console.log('OrganizerPage.detectChanges()');
        setTimeout(
            () => {
                this.changeDetectorRef.detectChanges();
                this.content.resize();
            },
            0
        );
    }

    /**
     * UI calls this when the new folder button is clicked
     * @returns {void}
     */
    public onClickEntry(entry: Entry): void {
        console.log('onClickEntry()');
        const dirPath: string = [
            this.directoryEntry.fullPath,
            '/',
            entry.name,
            '/'
        ].join('');
        this.switchFolder(dirPath, true);
    }

    /**
     * UI calls this when the new folder button is clicked
     * @returns {void}
     */
    public addFolder(): void {
        let parentPath: string = this.getFullPath(this.directoryEntry),
            newFolderAlert: Alert = this.alertController.create({
                title: 'Create a new folder in ' + parentPath,
                // message: 'Enter the folder name',
                inputs: [{
                    name: 'folderName',
                    placeholder: 'Enter folder name...'
                }],
                buttons: [
                    {
                        text: 'Cancel',
                        role: 'cancel',
                        handler: () => {
                            console.log('Cancel clicked in new-folder alert');
                            // this.keyboard.close();
                        }
                    },
                    {
                        text: 'Done',
                        handler: (data: any) => {
                            let folderName: string = data.folderName;
                            if (!folderName.length) {
                                // this code should never be reached
                                alert('how did we reach this code?');
                                return;
                            }
                            if (folderName[folderName.length - 1] !== '/') {
                                // last char isn't a slash, add a
                                // slash at the end
                                folderName += '/';
                            }
                            // create the folder via getPathEntry()
                            this.appFS.getPathEntry(
                                parentPath + folderName,
                                true
                            ).subscribe(
                                (directoryEntry: DirectoryEntry) => {
                                    // re-read parent
                                    // to load in new info
                                    this.switchFolder(parentPath, false);
                                    // this.keyboard.close();
                                }
                            );
                        }
                    }
                ]
            });
        newFolderAlert.present();
        // this.keyboard.show();
```

```
    }

    /**
     * Select all or no items in current folder, depending on 'all; argument
     * @param {boolean} if true, select all, if false, select none
     * @returns {void}
     */
    private selectAllOrNoneInFolder(bSelectAll: boolean): void {
        console.log('selectAllOrNoneInFolder(' + bSelectAll + ')');
        let bChanged: boolean = false;
        this.entries.forEach((entry: Entry) => {
            const fullPath: string = this.getFullPath(entry),
                  isSelected: boolean = this.isSelected(entry);
            if (bSelectAll && !isSelected) {
                this.selectedPaths.add(fullPath);
                bChanged = true;
            }
            else if (!bSelectAll && isSelected) {
                this.selectedPaths.delete(fullPath);
                bChanged = true;
            }
        });
        if (bChanged) {
            this.appState.set(
                'selectedPaths',
                this.selectedPaths
            ).then();
        }
    }
}
```