```
// 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 { alertAndDo } from '../../models/utils/alerts';
import { AppState } from '../../services/app-state/app-state';
import { ButtonbarButton } from '../../components/button-bar/button-bar';
import {
    DB_KEY_PATH,
    KeyDict,
    ParentChild,
    ROOT_FOLDER_KEY,
    TreeNode
} from '../../models/idb/idb-fs';
import { EditSelectionPage } from '../edit-selection-page/edit-selection-page';
import { FS } from '../../models/filesystem/filesystem';
import { isPositiveWholeNumber, isUndefined } from '../../models/utils/utils';
import { MoveToPage, TrackPage } from '../';

const REQUEST_SIZE: number = 1024 * 1024 * 1024;

/**
 * @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 {
    @ViewChild(Content) public content: Content;
    private fileSystem: FileSystem;
    public entries: Entry[];
    // UI uses directoryEntry
    public directoryEntry: 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;
    private appState: AppState;
    // UI uses selectedEntries
    private selectedEntries: Set<string>;

    /**
     * @constructor
     * @param {NavController}
     * @param {AlertController}
     * @param {ModalController}
     * @param {AppState}
     * @param {Platform}
     */
    constructor(
        navController: NavController,
        alertController: AlertController,
        actionSheetController: ActionSheetController,
        modalController: ModalController,
        changeDetectorRef: ChangeDetectorRef,
        appState: AppState,
        platform: Platform
    ) {
        console.log('constructor():OrganizerPage');
        this.appState = appState;
        this.changeDetectorRef = changeDetectorRef;
        this.fileSystem = null;
        this.entries = [];
        this.directoryEntry = null;
        this.selectedEntries = new Set<string>();
        this.actionSheetController = actionSheetController;

        appState.getProperty('selectedEntries').then(
            (selectedEntries: Set<string>) => {
                this.selectedEntries = selectedEntries;
                this.detectChanges();
            }
        );

        // get the filesystem
        FS.getFileSystem(true, REQUEST_SIZE).subscribe(
            (fileSystem: FileSystem) => {
                // remember the filesystem you got
                this.fileSystem = fileSystem;
                // create the /Unfiled/ folder if not already there
                FS.getPathEntry(fileSystem, '/Unfiled/', true).subscribe(
                    (directoryEntry: DirectoryEntry) => {
                        console.log('Created /Unfiled/');
                        // get last viewed folder to switch to it
                        appState.getProperty('lastViewedFolderPath').then(
                            (path: string) => {
                                this.switchFolder(path, false);
                            }
                        ); // State.getProperty('lastViewedFolderPath').then(..
                    },
                    (err3: any) => {
                        alert('err3: ' + err3);
                    }
                ); // FS.getPathEntry(..).subscribe(..
            }
        ); // FS.getFileSystem(true).subscribe(..

        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.selectAllInFolder();
        }
    });
    selectAlert.addButton({
        text: 'None',
        handler: () => {
            this.selectNoneInFolder();
        }
    });

    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('/');
}

/**
 * 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 = '/' +
        pathParts.splice(0, pathParts.length - 1).join('/') +
        '/';
    this.switchFolder(parentPath);
}

/**
 * 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.selectedEntries.size === 1 &&
        this.selectedEntries.has('/Unfiled')) {
        return true;
    }
    return false;
}

/**
 * UI calls this when delete button is clicked.
 * @returns {void}
 */
public onClickDeleteButton(): void {
    console.log('onClickDeleteButton()');
    console.dir(this.selectedEntries);
    FS.removeEntries(this.fileSystem, Array.from(this.selectedEntries))
        .subscribe(() => { this.detectChanges(); });
}

/**
 * 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.selectedEntries.size === 1 &&
        this.selectedEntries.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()');
    this.navController.push(EditSelectionPage);
}

/**
 * UI calls this to determine the icon for an entry.
 * @param {Entry} entry
 */
public entryIcon(entry: Entry): string {
    return entry.isDirectory ? 'folder' : 'play';
}

/**
 * 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) {
    console.log('OrganizerPage.switchFolder(' + path + ', ' + bUpdateAppState +
')');
    FS.getPathEntry(
        this.fileSystem,
        path,
        false
    ).subscribe(
        (directoryEntry: DirectoryEntry) => {
            this.directoryEntry = directoryEntry;
            if (!directoryEntry) {
                alert('!directoryEntry!');
            }
            FS.readDirectory(
                directoryEntry
            ).subscribe(
                (entries: Entry[]) => {
                    console.log('entries: ' + entries);
                    console.dir(entries);
                    this.entries = entries;
                    this.detectChanges();
                    if (bUpdateAppState) {
                        this.appState.updateProperty(
                            'lastViewedFolderPath',
                            path
                        ).then();
                    }
                },
```

```
                    (err1: any) => {
                        alert('err1: ' + err1);
                    }
                ); // FS.readDirectory().susbscribe(..
            },
            (err2: any) => {
                alert('err2: ' + err2);
            }
        ); // FS.getPathEntry(..).subscribe(..
    }

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

    public ionViewWillEnter(): void {
        console.log('OrganizerPage.ionViewWillEnter()');
        this.detectChanges();
    }

    public ionViewDidEnter(): void {
        console.log('OrganizerPage.ionViewDidEnter()');
        this.detectChanges();
    }

    /**
     * UI calls this when the new folder button is clicked
     * @returns {void}
     */
    public onClickRename(node: TreeNode, item: ItemSliding): void {
        console.log('onClickRename()');
    }

    /**
     * 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);
    }

    public toggleSelect(entry: Entry): void {
        console.log('toggleSelect()');
        const fullPath: string = entry.fullPath +
            (entry.isDirectory ? '/' : '');
        if (fullPath === '/') {
            alert('fullPath === \'/\'');
            debugger;
        }
        if (this.selectedEntries.has(fullPath)) {
            this.selectedEntries.delete(fullPath);
        }
        else {
            this.selectedEntries.add(fullPath);
        }
        this.appState.updateProperty('selectedEntries', this.selectedEntries)
            .then();
        this.detectChanges();
    }

    public isSelected(entry: Entry): boolean {
        // return entry.fullPath in this.selectedEntries;
        return this.selectedEntries.has(entry.fullPath);
    }

    public onRenameEntry(entry: Entry): void {
        console.log('onRenameEntry(' + entry + '');
    }

    /**
     * UI calls this when the new folder button is clicked
     * @returns {void}
     */
    public addFolder(): void {
        let parentPath: string =this.directoryEntry.fullPath+'/',
            newFolderAlert: Alert = this.alertController.create({
            title: 'Create a new folder in ' + this.directoryEntry.fullPath,
            // 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');
                    }
                },
                {
                    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()
                        FS.getPathEntry(
                            this.fileSystem,
                            parentPath + folderName,
                            true
                        ).subscribe(
                            (directoryEntry: DirectoryEntry) => {
                                // re-read parent
                                // to load in new info
                                this.switchFolder(parentPath, false);
                            }
                        );
                    }
                }
            ]
        });
        newFolderAlert.present();
    }

    /**
     * Select all items in current folder
     * @returns {void}
```

```
     */
    private selectAllInFolder(): void {
        this.selectAllOrNoneInFolder(true);
    }

    /**
     * Get rid of selection on all nodes in current folder
     * @returns {void}
     */
    private selectNoneInFolder(): void {
        this.selectAllOrNoneInFolder(false);
    }

    /**
     * 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(bSelecting: boolean): void {
        for (let i: number = 0; i < this.entries.length; i++) {
            const entry: Entry = this.entries[i],
                isSelected: boolean = this.isSelected(entry);
            if ((bSelecting && !isSelected) || (!bSelecting && isSelected)) {
                // reverse (toggle) node selection status
                this.toggleSelect(entry);
                // remember that something, at least one, has changed
            }
        }
        this.detectChanges();
    }

    public reorderEntries(indexes: any): void {
        console.log('reorderEntries(' + indexes + ')');
        console.log(typeof(indexes));
        console.dir(indexes);
        let entry: Entry = this.entries[indexes.from];
        this.entries.splice(indexes.from, 1);
        this.entries.splice(indexes.to, 0, entry);
    }
}
```