Devdactic

How to Build an Ionic 4 File Explorer

OCTOBER 1, 2019 BY SIMON (HTTPS://DEVDACTIC.COM/AUTHOR/SIMON-REIMLER/)

Working with files in Ionic and Cordova applications can be painful and sometimes complicated, so today we want to go all in on the topic!

In this tutorial we will build a full file explorer with Ionic 4.

We'll implement the basic functionalities to create and delete files and folders, and also implement an intelligent navigation to create a tree of folders to navigate around to navigate around.

Get the Ionic 4 File Explorer App Code

Enter your Email below to receive the needed files directly to your inbox!

Powered by ConvertKit (http://mbsy.co/convertkit/23139740)

Setup Our Ionic File Explorer

To get started we just need a blank new app and 2 additional packages:

- The File plugin (https://ionicframework.com/docs/native/file) to perform all of our operations on the file system
- The File opener plugin

- The File opener plugin (https://ionicframework.com/docs/native/file-opener) to open some of our files

Make sure you install both the npm packages and also the cordova plugin:

```
ionic start devdacticExplorer blank
cd ./devdacticExplorer
npm install @ionic-native/file @ionic-native/file-ope
ionic cordova plugin add cordova-plugin-file
ionic cordova plugin add cordova-plugin-file-opener2
```

To use all of this also makre sure to add both packages to your **app/app.module.ts** like this:

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-bro
import { RouteReuseStrategy } from '@angular/router'

import { IonicModule, IonicRouteStrategy } from '@ic
import { SplashScreen } from '@ionic-native/splash-s
import { StatusBar } from '@ionic-native/status-bar/

import { AppComponent } from './app.component';
import { AppRoutingModule } from './app-routing.modu

import { File } from '@ionic-native/file/ngx';
import { FileOpener } from '@ionic-native/file-opene

@NgModule({
  declarations: [AppComponent],
  entryComponents: [],
  imports: [BrowserModule, IonicModule.forRoot(), Ap
  providers: [
    StatusBar,
    SplashScreen,
    { provide: RouteReuseStrategy, useClass: IonicRo
    File,
    FileOpener
  ],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

Now to one of the cool things of our Ionic file explorer: We will actually use only one single page, but reuse it so it works for all levels of our directory structure!

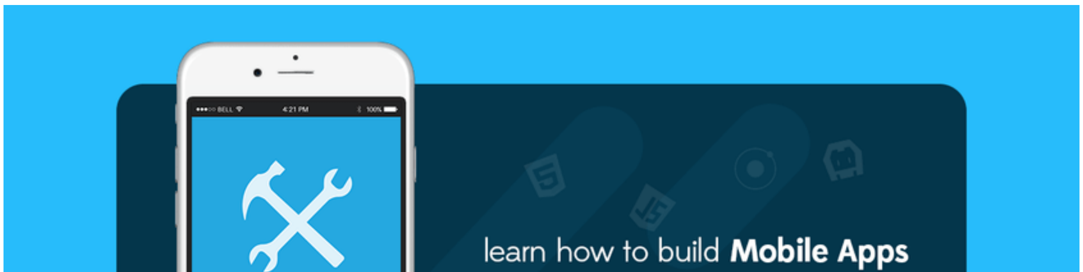
To do so, we can simply create another routing entry which also uses the default page, but with a different path that will contain a **folder** value that indicates in which folder we currently are.

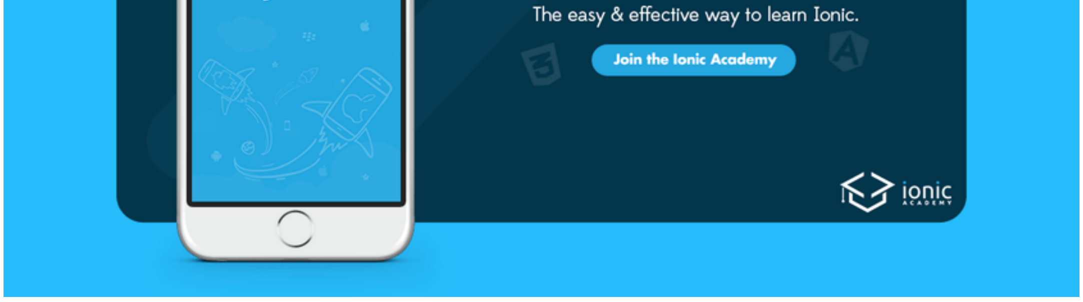
Therefore change the **app/app-routing.module.ts** and notice the usage of the Angular 8 syntax for loading our module:

```
import { NgModule } from '@angular/core';
import { PreloadAllModules, RouterModule, Routes } f

const routes: Routes = [
  { path: '', redirectTo: 'home', pathMatch: 'full'
  { path: 'home', loadChildren: () => import('./home
  { path: 'home/:folder', loadChildren: () => import
];

@NgModule({
  imports: [
    RouterModule.forRoot(routes, { preloadingStrateg
  ],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Now we got all the basic things in place, and because we basically only use Cordova plugins **I highly recommend you start the app on a connected device** using livereload by running the command like this:

```
ionic cordova run ios --consolelogs --livereload --ad
```





All of our plugins won't work inside the browser, and having the app on a device with livereload is your best bet to develop an app that makes heavy use of native functionality!

The Full Explorer View – In One Page

We could go about this one by one, but the changes we would have to apply along the way would be actually more confusing so let's do the view in one take.

The main part of the view consists of an iteration over all the entries we find in a directory – be it files or folders. All entries have a **click event**, and can **swipe** in either the **delete button or a copy/move button** that starts a copy operation.

Talking of copy & move, we will simply **select a first file** which then sets a `copyFile` variable as a reference which file should be moved. We are then in sort of a **transition phase**, and in that phase also change the color of our toolbar.

Also, we will display either our generic title or the current path of the folder we navigated to. And if the current folder is not the root folder anymore, we also show the default back button so we can **navigate one level up** our directories again!

Now go ahead and change the **app/home/home.page.html** to:

```
<ion-header>
  <ion-toolbar [color]="copyFile ? 'secondary' : 'pr
    <ion-buttons slot="start" *ngIf="folder != ''">
      <ion-back-button></ion-back-button>
    </ion-buttons>
    <ion-title>
      {{ folder || 'Devdactic Explorer' }}
    </ion-title>
  </ion-toolbar>
</ion-header>

<ion-content>
  <ion-text color="medium" *ngIf="directories.length
    <p>No documents found</p>
  </ion-text>

  <ion-list>
    <ion-item-sliding *ngFor="let f of directories">
      <ion-item (click)="itemClicked(f)">
        <ion-icon name="folder" slot="start" *ngIf='

        <ion-icon name="document" slot="start" *ngIf
        <ion-label text-wrap>
          {{ f.name }}
          <p>{{ f.fullPath }}</p>
        </ion-label>
      </ion-item>
      </ion-item>

      <ion-item-options side="start" *ngIf="!f.isDir
        <ion-item-option (click)="deleteFile(f)" col
          <ion-icon name="trash" slot="icon-only"></
        </ion-item-option>
      </ion-item-options>

      <ion-item-options side="end">
        <ion-item-option (click)="startCopy(f)" colo
          Copy
        </ion-item-option>
        <ion-item-option (click)="startCopy(f, true)
          Move
        </ion-item-option>
      </ion-item-options>

    </ion-item-sliding>
  </ion-list>

  <ion-fab vertical="bottom" horizontal="end" slot="
    <ion-fab-button>
      <ion-icon name="add"></ion-icon>
    </ion-fab-button>

    <ion-fab-list side="top">
      <ion-fab-button (click)="createFolder()">
        <ion-icon name="folder"></ion-icon>
      </ion-fab-button>
      <ion-fab-button (click)="createFile()">
        <ion-icon name="document"></ion-icon>
      </ion-fab-button>
    </ion-fab-list>
  </ion-fab>

</ion-content>
```

The fab list at the bottom reveals the two additional buttons to **create a file or folder** in the current directory, so nothing really special in there. Most of the stuff relies on the current directory

Most of the stuff relies on the current directory you are in, and it all makes sense once we implement the real functionality now.

Listing Our Files and Folder

Now we gonna separate the functionality a bit since it would be too long for one snippet. First of all, we use the file plugin to **load a list of directories**. Because initially our `folder` is an empty string, it will use the basic `this.file.dataDirectory` (which is of course an empty list after installation).

We also implement the logic for retrieving the folder param from the `paramMap` of the activated route, which is appended to the directory that we list as well. This is the logic to **list the different directories once we navigate** to a next folder!

To start, simply change your **app/home/home.page.ts** to this so you also got already all imports that we need:

```
import { Component, OnInit } from '@angular/core';
import { File, Entry } from '@ionic-native/file/ngx'
import { Platform, AlertController, ToastController
import { FileOpener } from '@ionic-native/file-opene
import { Router, ActivatedRoute } from '@angular/rou
```

```

@Component({
  selector: 'app-home',
  templateUrl: 'home.page.html',
  styleUrls: ['home.page.scss']
})
export class HomePage implements OnInit {
  directories = [];
  folder = '';
  copyFile: Entry = null;
  shouldMove = false;

  constructor(
    private file: File,
    private plt: Platform,
    private alertCtrl: AlertController,
    private fileOpener: FileOpener,
    private router: Router,
    private route: ActivatedRoute,
    private toastCtrl: ToastController
  ) {}

  ngOnInit() {
    this.folder = this.route.snapshot.paramMap.get(
    this.loadDocuments();
  }

  loadDocuments() {
    this.plt.ready().then(() => {
      // Reset for later copy/move operations
      this.copyFile = null;
      this.shouldMove = false;

      this.file.listDir(this.file.dataDirectory, t
        this.directories = res;
      });
    });
  }
}
```

The following will all take place in this file, simply append the functionality below the current functions.

Create new Folders and Files

We can trigger the two different actions with the fab buttons, and perhaps we could have even combined it into a single function. There is basically only a difference in the function we use, either `createDir` or `writeFile` from our file plugin.

For that function, we need to supply our current path (again, appending the `folder` to the root path) and then a name for the file or folder that we want to create.

Additionally we can also write content directly into the new file (you could also create an empty file), which works great if you download images from a server and **write that blob data directly into a file**!

Go ahead and append the following functions:

```
async createFolder() {
  let alert = await this.alertCtrl.create({
    header: 'Create folder',
    message: 'Please specify the name of the new fol
    inputs: [
      {
        name: 'name',
        type: 'text',
        placeholder: 'MyDir'
      }
    ]
    ],
    buttons: [
      {
        text: 'Cancel',
        role: 'cancel',
        cssClass: 'secondary'
      },
      {
        text: 'Create',
        handler: data => {
          this.file
            .createDir(
              `${this.file.dataDirectory}/${this.fo
              data.name,
              false
            )
            .then(res => {
              this.loadDocuments();
            });
        }
      }
    ]
  });

  await alert.present();
}

async createFile() {
  let alert = await this.alertCtrl.create({
    header: 'Create file',
    message: 'Please specify the name of the new fil
    inputs: [
      {
        name: 'name',
        type: 'text',
        placeholder: 'MyFile'
      }
    ],
    buttons: [
      {

        text: 'Cancel',
        role: 'cancel',
        cssClass: 'secondary'
      },
      {
        text: 'Create'
        text: 'Create',
        handler: data => {
          this.file
            .writeFile(
              `${this.file.dataDirectory}/${this.fo
              `${data.name}.txt`,
              `My custom text - ${new Date().getTim
            )
            .then(res => {
              this.loadDocuments();
            });
        }
      }
    ]
  });

  await alert.present();
}
```

Now you are already able to test the basic functionality of our Ionic file explorer. Go ahead and create some files and folders, but right now we are not yet able to navigate or perform our other operations.

Delete Files & Start Copy/Move Process

This part is pretty fast – for the deletion of a file or folder we just need the path to the object and the name of it, which we can easily get from the

information that is initially returned for the directory and stored locally.

To perform a copy or move operation, we first

To perform a copy or move operation, we first need to select a file that we want to move. In this function, we simply save a reference to it so later when we select the new destination, we know what to copy. This also changes how our header looks, and a click on an item should then have a different effect.

The two functions go as well into our current file:

```
deleteFile(file: Entry) {
  let path = this.file.dataDirectory + this.folder
  this.file.removeFile(path, file.name).then(() =>
    this.loadDocuments();
  });
}

startCopy(file: Entry, moveFile = false) {
  this.copyFile = file;
  this.shouldMove = moveFile;
}
```

Now there is just one more piece missing…

Open Files, Perform Copy & Move Operations

The click event on an item can mean a few things, based on different conditions:

- If it's a **file**, we can open it using the second

- If it's a **file**, we can open it using the second package we installed in the beginngin
- If it's a **folder**, we want to navigate into the folder by using the current path, appending the folder name and encoding everything so we **don't mess up the URL with additional slashes**
- If we selected a file for copy/move before, the now selected object needs to be a folder to which we can copy the file and finish our operation

This logic is reflected by the first function below, and the second one looks kinda strange but is just an if/else of two different conditions.

Either we want to move a file or copy it, and either it's a directory or a file.

That's why the function is pretty long, but as you can see it's only a change of the function that you use from the file plugin!

```
async itemClicked(file: Entry) {
  if (this.copyFile) {

    // Copy is in action!
    if (!file.isDirectory) {
      let toast = await this.toastCtrl.create({
        message: 'Please select a folder for your op
      });
      await toast.present();
      await toast.present();
      return;
    }
    // Finish the ongoing operation
    this.finishCopyFile(file);
  } else {
    // Open the file or folder
    if (file.isFile) {
      this.fileOpener.open(file.nativeURL, 'text/pl
    } else {
      let pathToOpen =
        this.folder != '' ? this.folder + '/' + fi
      let folder = encodeURIComponent(pathToOpen);
      this.router.navigateByUrl(`/home/${folder}`);
    }
  }
}

finishCopyFile(file: Entry) {
  let path = this.file.dataDirectory + this.folder
  let newPath = this.file.dataDirectory + this.fol

  if (this.shouldMove) {
    if (this.copyFile.isDirectory) {
      this.file
        .moveDir(path, this.copyFile.name, newPath,
        .then(() => {
          this.loadDocuments();
        });
    } else {
      this.file
        .moveFile(path, this.copyFile.name, newPath
        .then(() => {
          this.loadDocuments();
        });
    }
  } else {
    if (this.copyFile.isDirectory) {
      this.file
        .copyDir(path, this.copyFile.name, newPath,

        .then(() => {
          this.loadDocuments();
        });
    } else {
      this.file
        .copyFile(path, this.copyFile.name, newPath
        .copyFile(path, this.copyFile.name, newPath
        .then(() => {
          this.loadDocuments();
        });
    }
  }
}
```

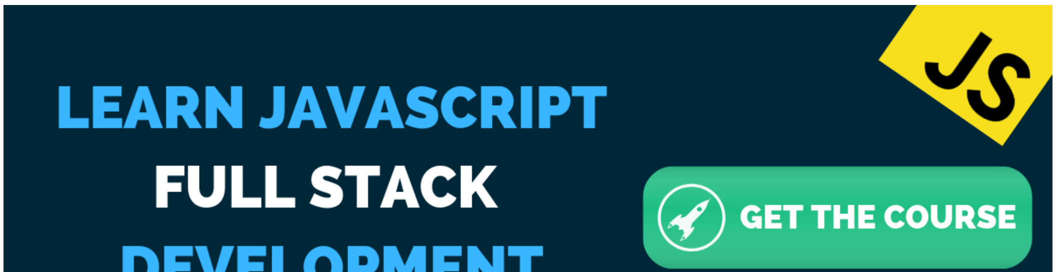
Now with this logic in place, **your Ionic file explorer is fully functional**!

Conclusion

The trickiest element when working with files in Ionic can be the native path, which is not always working as expected or doesn’t show images for example.

Hopefully this file explorer gives you a good overview about what you could do with the underlying file system inside your Ionic app, including the reuse logic for the working path in our app!

You can also find a video version of this tutorial below.



(https://courses.devdactic.com/p/software-startup-manual) ionic

CRASH COURSE

Get the free **7 day Ionic 4 Crash Course** to learn how to:

- Get started with Ionic
- Build a Tab Bar navigation
- Make HTTP calls to a REST API
- Store Data inside your app
- Use Cordova plugins
- Style your Ionic app

The course is free, so there's nothing you can lose!

First Name

Email Address

Powered by ConvertKit (http://mbsy.co/convertkit/23139740)

About Simon

Ionic Framework Expert • Educator, Consultant & Speaker • Creator of the Ionic Academy

Comments Community 1 Login

Recommend Tweet Share Sort by Best

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Shreya Goyal • 10 days ago • edited

Thanks Simon! for the great tutorial. Everything is working properly without any error.

Reply • Share ›

Simon Grimm Mod → Shreya Goyal • 5 days ago

Awesome Shreya, glad to hear!

Reply • Share ›

manunoly • 23 days ago • edited

hey Simon, great tutorial, thanks a lot for your work, one question, can i use the readFile() and formData.append('file', imgBlob, file.name) to upload PDF file to server?, i have working with image but I need to upload PDF using http.

Reply • Share ›

Simon Grimm Mod → manunoly
• 20 days ago

readFile is a general function to read in any file, so a PDF should work as well. After that, any file is just a blob that you can upload!

Reply • Share ›

Satheeshkumar C K • a month ago

Hi,
Is there any way we can save files in desktop using ionic 4 electron build?
Thanks

Reply • Share ›

Simon Grimm Mod → Satheeshkumar C K
• a month ago

I'm sure that this is possible, but I'm not an Electron expert :/

1 • Reply • Share ›

Jarra Fetene Birru • 3 months ago

Hello, thanks for the share. Can i build the app, so it can explore my PC files over wifi? would that be difficult? thanks

Reply • Share ›

Simon Grimm Mod → Jarra Fetene Birru
• 3 months ago

The file plugin allows access to the local files inside your app on a device - accessing your own PC over wifi is a completely different story!

Reply • Share ›

Mat Born • 3 months ago

can we sorting the directory list by name or date using ionic 4

^ | ˅ • Reply • Share ›

Simon Grimm Mod ➦ Mat Born
• 3 months ago

Sure, just apply a filter or sort the array by hand before assigning it!

^ | ˅ • Reply • Share ›

ALSO ON **DEVDACTIC**

How ...
2 comments • a month ago
Simon Grimm — Thanks a lot Ali :)

How ...
14 comments • 2 months ago
andyweber — I think i did it exactly in the same order and when it

Getting ...
7 comments • 6 months ago
Los Santos — Hi,how can i set the authorization bearer

Getting ...
12 comments • 7 months ago
Melanie Carr — I've used Angular material tables with ionic but love

FOLLOW ME ON TWITTER

ionic
CRASH COURSE

Get the free **7 day Ionic 4 Crash Course** to learn how to:

Get started with Ionic

Build a Tab Bar navigation

Make HTTP calls to a REST API

Store Data inside your app

Use Cordova plugins

Style your Ionic app

The course is free, so there's nothing you can lose!

First Name

Email Address

Powered by ConvertKit (http://mbsy.co/convertkit/23139740)

FEATURED ON:

The Polyglot Developer ionic joshmorony airpair Back&