Devdactic

Ionic AWS S3 Integration with NodeJS – Part 2: Ionic App

NOVEMBER 28, 2017 BY SIMON (HTTPS://DEVDACTIC.COM/AUTHOR/SIMON-REIMLER/)

For long time the Amazaon Web Services (AWS) have been around and people love to use it as a backend or simply storage engine. In this series we will see how we can build an Ionic AWS App which can upload files from our Ionic app to a S3 bucket inside AWS with a simple NodeJS server in the middle!

In the first part of this series we've built a simple NodeJS backend which acts as our gateway to AWS. Our backend has they keys and a user to access our AWS services, so our Ionic app will need to ask for permission / the signed URL to make a request to AWS. Make sure you've the first part up and running before continuing!

Part 1: Ionic AWS S3 Integration with NodeJS: Server (https://devdactic.com/ionic-aws-nodejs-1/)

In this part we will now focus on the Ionic app. We need to be able to capture images, upload them to AWS S3 and also display a list of currently hosted images along with the option to delete them.

Setting up Our Ionic AWS App

We start with a blank new Ionic app and install the camera plugin (http://ionicframework.com/docs/native/camera/) and also the file plugin (http://ionicframework.com/docs/native/file/). We need both of them to upload new images to S3, so go ahead and run:

Start our Ionic AWS app

```
ionic start devdacticAwsUpload blank
cd devdacticAwsUpload
ionic g provider aws
npm install --save @ionic-native/camera @ionic-native/file
ionic cordova plugin add cordova-plugin-camera
```

```
ionic cordova plugin add cordova-plugin-file
```

We also created a new provider which will take care of accessing our backend URLs inside our app later!

To make use of our plugins and the provider we need to add them to the **src/app/app.module.ts** so go ahead and change it to:

3 Shares

3

Load all dependencies inside app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { ErrorHandler, NgModule } from '@angular/core';
import { IonicApp, IonicErrorHandler, IonicModule } from 'ionic-angular';
import { SplashScreen } from '@ionic-native/splash-screen';
import { StatusBar } from '@ionic-native/status-bar';

import { MyApp } from './app.component';
import { HomePage } from '../pages/home/home';
import { AwsProvider } from '../providers/aws/aws';

import { HttpModule } from '@angular/http';
import { Camera } from '@ionic-native/camera';
import { File } from '@ionic-native/file';

@NgModule({
  declarations: [
    MyApp,
    HomePage
  ],
  imports: [
    BrowserModule,
    HttpModule,
    IonicModule.forRoot(MyApp)
  ],
  bootstrap: [IonicApp],
  entryComponents: [
    MyApp,
    HomePage
  ],
  providers: [
    StatusBar,
    SplashScreen,
    {provide: ErrorHandler, useClass: IonicErrorHandler},
    AwsProvider,
    Camera,
    File
  ]
})
export class AppModule {}
```

Now we are ready to dive into the connection to our server!

Developing the AWS Provider

It's a good idea to separate your HTTP calls from the rest of the app so we put all of the needed requests into our `AwsProvider`.

Remember that we have 4 URL routes of our backend (https://devdactic.com/ionic-aws-nodejs-1/), so those are the 4 first functions we implement. These routes will allow us to get signed calls to AWS or to list the files and remove files.

When we call `getFileList()` we get a quite big object but we only need the **Contents** array, so we directly map those values and only return them to the caller of the function.

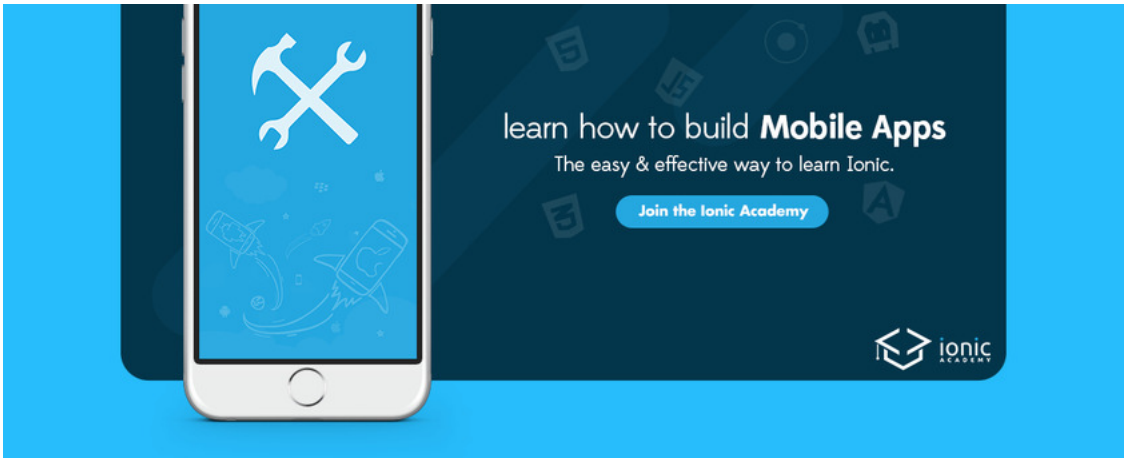
The last function of our provider is to upload our file, but as we already got the signed request at that point we just need to make a **PUT** request with the given URL and the file object (which we need to prepare before) and everything should work alright!

Go ahead and change your **src/providers/aws/aws.ts** to:

All routes for our AWS interaction

```
import { Injectable } from '@angular/core';
import { Http } from '@angular/http';
import 'rxjs/add/operator/map';
import { Observable } from 'rxjs/Observable';

@Injectable()
export class AwsProvider {
  apiUrl = 'http://127.0.0.1:5000/';

  constructor(public http: Http) { }

  getSignedUploadRequest(name, type) {
    return this.http.get(`${this.apiUrl}aws/sign?file-name=${name}&file-type=${type}`).map(res => res.json());
  }

  getFileList(): Observable<Array<any>> {
    return this.http.get(`${this.apiUrl}aws/files`)
      .map(res => res.json())
      .map(res => {
        return res['Contents'].map(val => val.Key);
      });
  }

  getSignedFileRequest(name) {
    return this.http.get(`${this.apiUrl}aws/files/${name}`).map(res => res.json());
  }

  deleteFile(name) {
    return this.http.delete(`${this.apiUrl}aws/files/${name}`).map(res => res.json());
  }

  // https://www.thepolyglotdeveloper.com/2015/03/create-a-random-nonce-string-using-javascript/
  randomString = function (length) {
    var text = "";
    var possible = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";
    for (var i = 0; i < length; i++) {
      text += possible.charAt(Math.floor(Math.random() * possible.length));
    }
    return text;
  }

  uploadFile(url, file) {
    return this.http.put(url, file);
  }
}
```

Now we only need to call our provider which will talk to the backend. Currently this points to **localhost**, so if you develop your app local it works, but if you deploy it to Heroku make sure to **check your backend URL** here!



(http://ionicacademy.com/?utm_src=devtut&utm_media=ad3)

Putting Everything Into Action

It's finally time to build our view and logic!

Our controller needs to hold an array of images which we can fill once the app has loaded. But getting the array of files is not enough at this point, because we will only get the key or name of the file like "*myfile.png*" which is not enough to access our S3 resource as it is protected from public views.

Only our backend has the rights to access these resources, therefore we make a call to `getSignedFileRequest()` for each image to get a signed URL which is a GET request then to the actual resource. We push all of those URLs to the image object and the array so we can use that URL inside our view later.

When we add an image we first present the action sheet to pick either library or the option to capture a new image.

The hard stuff in here begins after we got the image object inside the completion block of `getPicture()`, because we need to perform a few things in row:

1. Resolve the URI to a file of the filesystem
2. Read that file as **arrayBuffer** which can be added to the PUT request
3. Create a new (uniquish) name get the signed URL for the upload
4. Perform the actual upload with the provider
5. Get a new signed GET URL for the new resource and add it to our array

If you encounter any problems at this part (which is the pace where most errors can happen) leave a comment with your log below!

Working with files in iOS/Android is not always easy but hopefully everything works fine for you!

Go ahead and change your **src/pages/home/home.ts** to:

```
The controller for our upload and view actions
```

```
import { File } from '@ionic-native/file';
import { Camera, CameraOptions } from '@ionic-native/camera';
import { AwsProvider } from './../../providers/aws/aws';
import { Component } from '@angular/core';
import { NavController, ActionSheetController, ToastController, LoadingController } from 'ionic-angular';
```

```
6
7  @Component({
8    selector: 'page-home',
9    templateUrl: 'home.html'
10 })
11 export class HomePage {
12   images = [];
13
14   constructor(public navCtrl: NavController, private loadingCtrl: LoadingController, private toastCtrl: ToastController, private awsPro
15
16   ionViewWillEnter() {
17     this.loadImages();
18   }
19
20   loadImages() {
21     this.images = [];
22     this.awsProvider.getFileList().subscribe(files => {
23       for (let name of files) {
24         this.awsProvider.getSignedFileRequest(name).subscribe(res => {
25           this.images.push({key: name, url: res})
26         });
27       }
28     });
29   }
30
31   presentActionSheet() {
32     let actionSheet = this.actionSheetCtrl.create({
33       title: 'Select Image Source',
34       buttons: [
35         {
36           text: 'Load from Library',
37           handler: () => {
38             this.takePicture(this.camera.PictureSourceType.PHOTOLIBRARY);
39           }
40         },
41         {
42           text: 'Use Camera',
43           handler: () => {
44             this.takePicture(this.camera.PictureSourceType.CAMERA);
45           }
46         },
47         {
48           text: 'Cancel',
49           role: 'cancel'
50         }
51       ]
52     });
53     actionSheet.present();
54   }
55
56   takePicture(sourceType) {
57     // Create options for the Camera Dialog
58     const options: CameraOptions = {
59       quality: 100,
60       correctOrientation: true,
61       destinationType: this.camera.DestinationType.FILE_URI,
62       encodingType: this.camera.EncodingType.JPEG,
63       mediaType: this.camera.MediaType.PICTURE,
64       sourceType: sourceType
65     }
66
67     // Get the picture
68     this.camera.getPicture(options).then((imageData) => {
69
70       let loading = this.loadingCtrl.create();
71       loading.present();
72
73       // Resolve the picture URI to a file
74       this.file.resolveLocalFilesystemUrl(imageData).then(oneFile => {
75
76         // Convert the File to an ArrayBuffer for upload
77         this.file.readAsArrayBuffer(this.file.tempDirectory, oneFile.name).then(realFile => {
78           let type = 'jpg';
79           let newName = this.awsProvider.randomString(6) + new Date().getTime() + '.' + type;
80
81           // Get the URL for our PUT request
```

```
          // Get the URL for our PUT request
          this.awsProvider.getSignedUploadRequest(newName, 'image/jpeg').subscribe(data => {
            let reqUrl = data.signedRequest;

            // Finally upload the file (arrayBuffer) to AWS
            this.awsProvider.uploadFile(reqUrl, realFile).subscribe(result => {

              // Add the resolved URL of the file to our local array
              this.awsProvider.getSignedFileRequest(newName).subscribe(res => {
                this.images.push({key: newName, url: res});
                loading.dismiss();
              });
            });
          });
        });
      }, err => {
        console.log('err: ', err);
      })
    }, (err) => {
      console.log('err: ', err);
    });
  }

  deleteImage(index) {
    let toRemove = this.images.splice(index, 1);
    this.awsProvider.deleteFile(toRemove[0]['key']).subscribe(res => {
      let toast = this.toastCtrl.create({
        message: res['msg'],
        duration: 2000
      });
      toast.present();
    })
  }
}
```

The last step is to add our view, which becomes now pretty easy.

We have to iterate over the array of images and display each of them inside a card. We use the `url` parameter of the object which is the already signed GET URL to the actual file on S3!

Besides that we add a little **edit function** which will enable a delete button at the bottom of each card.

Finally, a stylish **FAB button** displays nicely the action to add and upload new images.

Overall nothing really fancy here, so finish your app by changing your **src/pages/home/home.html** to:

The view to display images and add/delete images

```
<ion-header>
  <ion-navbar color="primary">
    <ion-title>
      Ionic + AWS
    </ion-title>
```

```
    <ion-buttons end>
      <button ion-button icon-only (click)="edit = !edit">
          <ion-icon name="unlock" *ngIf="edit"></ion-icon>
          <ion-icon name="lock" *ngIf="!edit"></ion-icon>
      </button>
    </ion-buttons>
  </ion-navbar>
</ion-header>

<ion-content padding>
  <h3 [hidden]="images.length !== 0" text-center>No Images found!</h3>

  <ion-list>
    <ion-card *ngFor="let img of images; let i = index">
      <img [src]="img.url">
      <button ion-button full color="danger" [style.margin]="0" *ngIf="edit" (click)="deleteImage(i)">
              <ion-icon name="trash"></ion-icon>
            </button>
    </ion-card>
  </ion-list>

  <ion-fab right bottom>
    <button ion-fab (click)="presentActionSheet()">
       <ion-icon name="camera"></ion-icon>
      </button>
  </ion-fab>
</ion-content>
```

That's it!

Now make sure your backend is up and running and if you run the app on the simulator, you should be able to access the server directly. If you deploy to a device, perhaps just deploy the server to Heroku and use that URL inside the provider.

Conclusion

It's not super hard to **upload files to AWS S3 using Ionic**, but it involves some coding on both backend and frontend to make everything **secure**.

If you have any ideas for improvement please let me know below ☺

You can also find a video version of this article below!