Angular 13 + Spring Boot: File upload/download example

bezkoder.com/angular-13-spring-boot-file-upload

bezkoder Last modified: November 18, 2021

In this tutorial, I will show you way to build Angular 13 File upload & download example with Spring Boot server.

More Practice:
– Angular + Spring Boot: CRUD example
– Angular + Spring Boot: JWT Authentication example
– Angular + Spring Boot: Pagination example

For Multiple Files Upload at once:
Angular 13 Multiple Files upload example with Progress Bar

Serverless with Firebase:
Angular 13 File Upload with Firebase Storage example

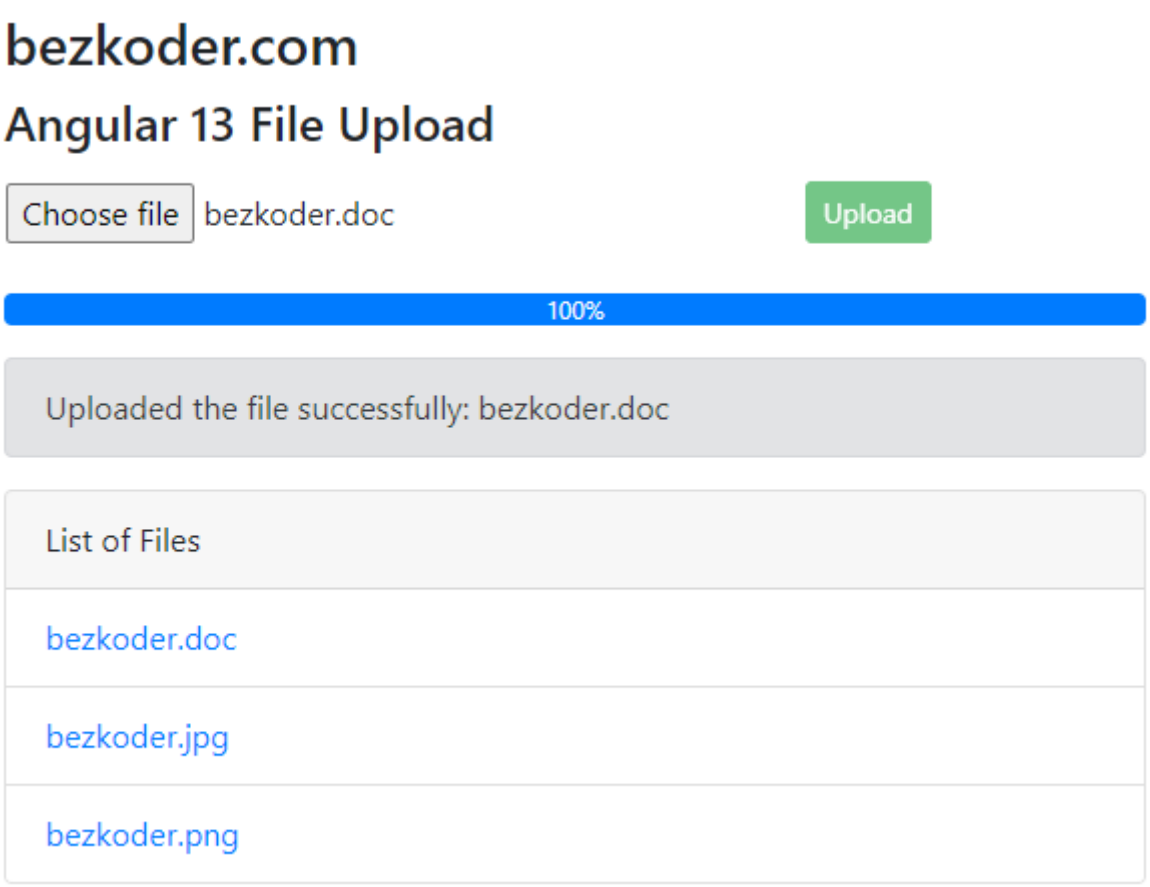
Other versions:
– Angular 8 + Spring Boot: File upload example
– Angular 10 + Spring Boot: File upload example
– Angular 11 + Spring Boot: File upload example
– Angular 12 + Spring Boot: File upload example

With Angular Material:
Angular Material File upload example with progress bar

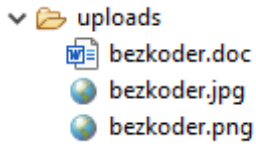
Overview

We’re gonna create a full-stack Angular 13 File upload to Spring Boot Server in that user can:

- see the upload process (percentage)
- view all uploaded files
- download by clicking on the file name

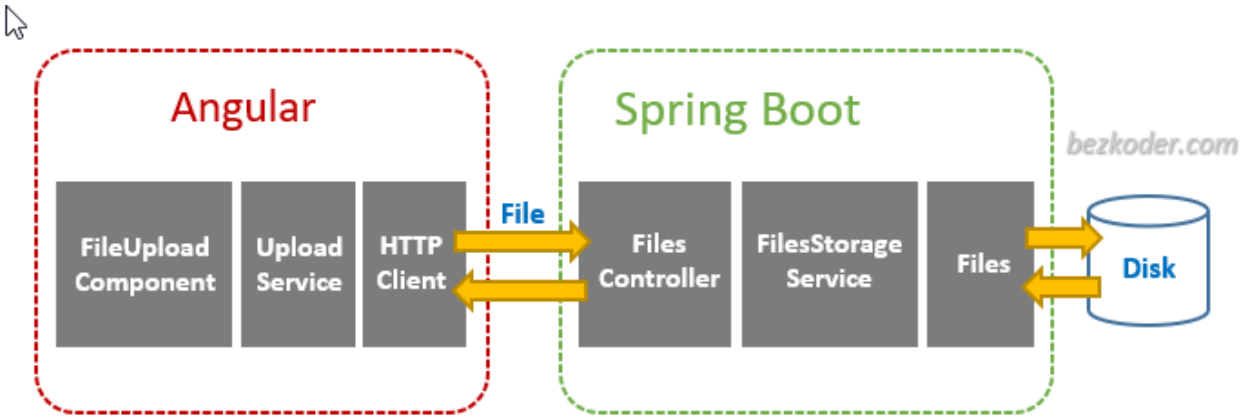


All uploaded files will be saved in **uploads** folder:



Angular 13 + Spring Boot File upload Architecture

Let's look at the architecture below:



Angular 13 Client:

- `FileUpload` Component calls `UploadService` functions for upload/download/display files
- `UploadService` uses `HttpClientModule` to make HTTP requests

Spring Boot Server:

- `FilesController` receives the HTTP requests from Client, then calls `FilesStorageService` functions for upload/download/getting files
- `FilesStorageService` implements functions for storing and retrieving file systems using Java `Files` library

Technology

Server:

- Java 8
- Spring Boot 2 (with Spring Web MVC)
- Maven 3.6.1

Client:

- Angular 13
- RxJS 7
- Bootstrap 4

Spring Boot Rest APIs for File Upload & Storage

Spring Boot App will provide APIs:

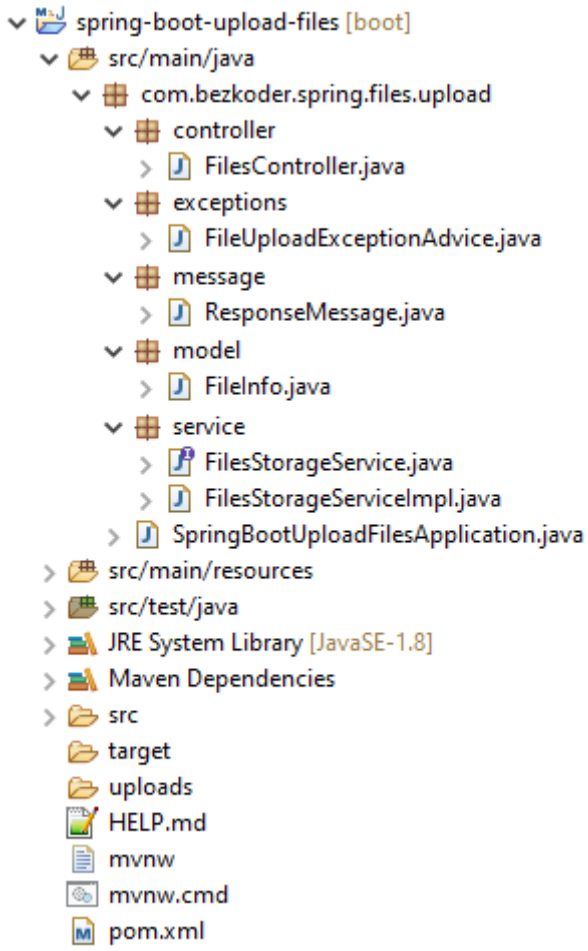
Methods	Urls	Actions
POST	/upload	upload a File
GET	/files	get List of Files (name & url)
GET	/files/[filename]	download a File

This is the project structure:

– `FileInfo` contains information of the uploaded file.
– `FilesStorageService` helps us to initialize storage, save new file, load file, get list of Files' info, delete all files.
– `FilesController` uses `FilesStorageService` to export Rest APIs: POST a file, GET all files' information, download a File.
– `FileUploadExceptionAdvice` handles exception when the controller processes file upload.
– *application.properties* contains configuration for Servlet Multipart.
– *pom.xml* for Spring Boot dependency.

You can find Step by Step to implement the Spring Boot Server (with Github) at: Spring Boot Multipart File upload (to static folder) example

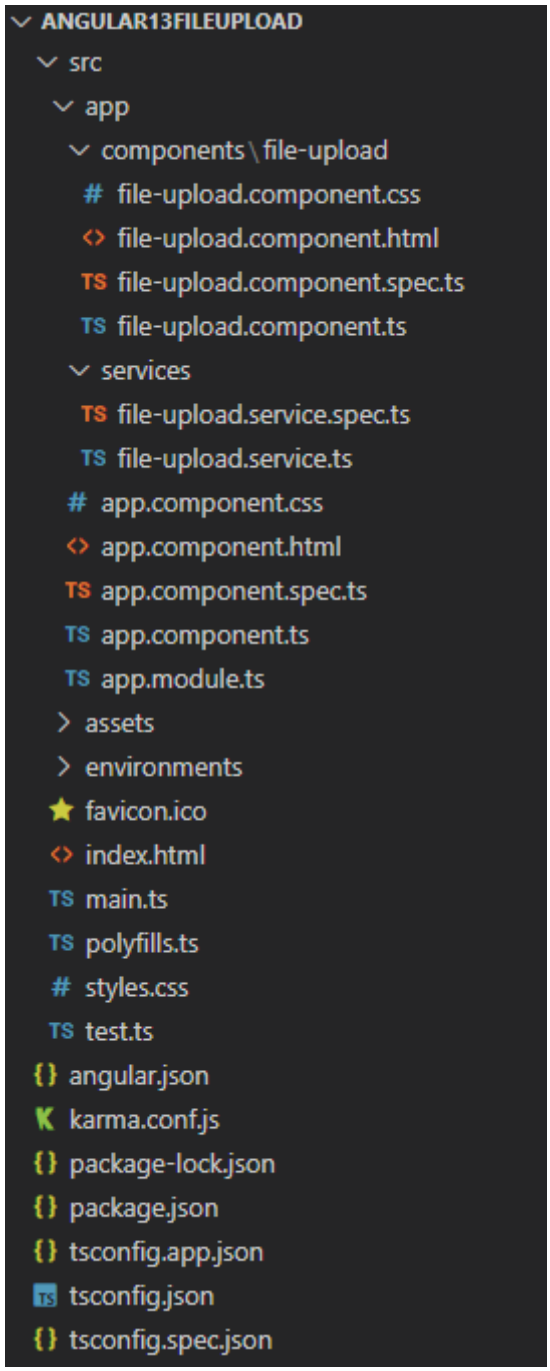
Or: Spring Boot Multipart File upload (to database) example



Angular 13 Client for file upload/download UI

This is the project structure that we're gonna build:

– We import necessary library, components in *app.module.ts*.
– *file-upload.service* provides methods to save File and get Files from Spring Boot Server.
– *file-upload.component* contains upload form, progress bar, display of list files.
– *app.component* is the container that we embed all components.
– *index.html* for importing the Bootstrap.



Setup Angular 13 Project

Let's open cmd and use Angular CLI to create a new Angular 13 Project as following command:

```
ng new Angular13FileUpload
? Would you like to add Angular routing? No
? Which stylesheet format would you like to use? CSS
```

We also need to generate some Components and Services:

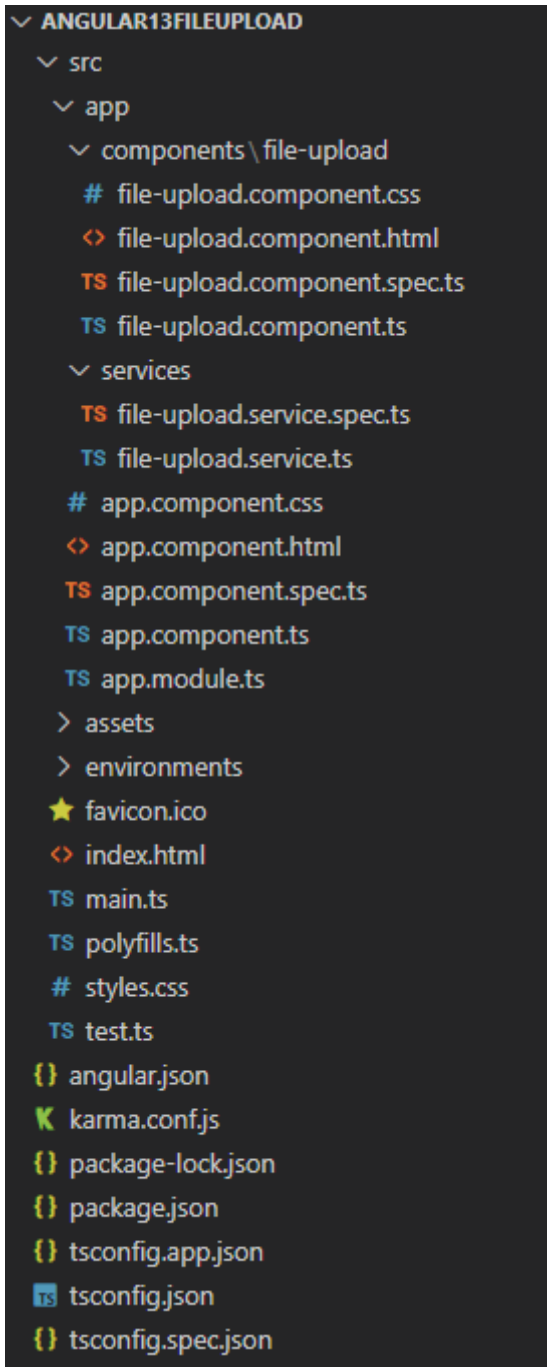
```
ng g s services/file-upload
ng g c components/file-upload
```

Now you can see that our project directory structure looks like this.

Angular 13 Project Structure

Let me explain it briefly.

– We import necessary library, components in *app.module.ts*.
– *file-upload.service* provides methods to save File and get Files from Rest API Server using `HttpClient`.
– *file-upload.component* contains upload form, progress bar, display of list files.
– *app.component* is the container that we embed all components.
– *index.html* for importing the Bootstrap.



Set up App Module for HttpClient

Open *app.module.ts* and import `HttpClientModule`:

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { HttpClientModule } from '@angular/common/http';
import { AppComponent } from './app.component';
import { FileUploadComponent } from './components/file-upload/file-
upload.component';
@NgModule({
  declarations: [
    AppComponent,
    FileUploadComponent
  ],
  imports: [
    BrowserModule,
    HttpClientModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Add Bootstrap to the project

Open *index.html* and add following line into `<head>` tag:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    ...
    <link rel="stylesheet"
href="https://unpkg.com/bootstrap@4.6.0/dist/css/bootstrap.min.css" />
  </head>
  ...
</html>
```

Create Angular 13 Service for Upload Files

This service will use Angular `HttpClient` to send HTTP requests.
There are 2 functions:

- `upload(file)` : returns `Observable<HttpEvent<any>>` that we're gonna use for tracking progress
- `getFiles()` : returns a list of Files' information as `Observable` object

services/*file-upload.service.ts*

```
import { Injectable } from '@angular/core';
import { HttpClient, HttpRequest, HttpEvent } from '@angular/common/http';
import { Observable } from 'rxjs';
@Injectable({
  providedIn: 'root'
})
export class FileUploadService {
  private baseUrl = 'http://localhost:8080';
  constructor(private http: HttpClient) { }
  upload(file: File): Observable<HttpEvent<any>> {
    const formData: FormData = new FormData();
    formData.append('file', file);
    const req = new HttpRequest('POST', `${this.baseUrl}/upload`, formData, {
      reportProgress: true,
      responseType: 'json'
    });
    return this.http.request(req);
  }
  getFiles(): Observable<any> {
    return this.http.get(`${this.baseUrl}/files`);
  }
}
```

– `FormData` is a data structure that can be used to store key-value pairs. We use it to build an object which corresponds to an HTML form with `append()` method.

– We set `reportProgress: true` to exposes progress events. Notice that this progress event are expensive (change detection for each event), so you should only use when you want to monitor it.

– We call the `request(PostRequest)` & `get()` method of `HttpClient` to send an HTTP POST & Get request to the Spring Boot File Upload server.

Create Component for Upload Files

Let's create a File Upload UI with Progress Bar, Card, Button and Message.

First we need to use the following imports:

file-upload.component.ts

```
import { Component, OnInit } from '@angular/core';
import { HttpEventType, HttpResponse } from '@angular/common/http';
import { Observable } from 'rxjs';
import { FileUploadService } from 'src/app/services/file-upload.service';
```

Then we define the some variables and inject `FileUploadService` as follows:

```
export class FileUploadComponent implements OnInit {
  selectedFiles?: FileList;
  currentFile?: File;
  progress = 0;
  message = '';
  fileInfos?: Observable<any>;
  constructor(private uploadService: FileUploadService) { }
}
```

Next we define `selectFile()` method. It helps us to get the selected Files.

```
selectFile(event: any): void {
  this.selectedFiles = event.target.files;
}
```

Next we write `upload()` method for upload file:

```
export class FileUploadComponent implements OnInit {
  selectedFiles?: FileList;
  currentFile?: File;
  progress = 0;
  message = '';
  fileInfos?: Observable<any>;
  constructor(private uploadService: FileUploadService) { }
  selectFile(event: any): void {
    this.selectedFiles = event.target.files;
  }
  upload(): void {
    this.progress = 0;
    if (this.selectedFiles) {
      const file: File | null = this.selectedFiles.item(0);
      if (file) {
        this.currentFile = file;
        this.uploadService.upload(this.currentFile).subscribe({
          next: (event: any) => {
            if (event.type === HttpEventType.UploadProgress) {
              this.progress = Math.round(100 * event.loaded / event.total);
            } else if (event instanceof HttpResponse) {
              this.message = event.body.message;
              this.fileInfos = this.uploadService.getFiles();
            }
          },
          error: (err: any) => {
            console.log(err);
            this.progress = 0;
            if (err.error && err.error.message) {
              this.message = err.error.message;
            } else {
              this.message = 'Could not upload the file!';
            }
            this.currentFile = undefined;
          }
        });
      }
      this.selectedFiles = undefined;
    }
  }
}
```

We use `selectedFiles` for accessing current File as the first Item. Then we call `uploadService.upload()` method on the `currentFile`.

The progress will be calculated basing on `event.loaded` and `event.total`.
If the transmission is done, the event will be a `HttpResponse` object. At this time, we call `uploadService.getFiles()` to get the files' information and assign the result to `fileInfos` variable.

We also need to do this work in `ngOnInit()` method:

```
ngOnInit(): void {
  this.fileInfos = this.uploadService.getFiles();
}
```

Now we create the HTML template of the Upload File UI. Add the following content to *file-upload.component.html* file:

```
<div class="row">
  <div class="col-8">
    <label class="btn btn-default p-0">
      <input type="file" (change)="selectFile($event)" />
    </label>
  </div>
  <div class="col-4">
    <button class="btn btn-success btn-sm" [disabled]="!selectedFiles"
(click)="upload()">
      Upload
    </button>
  </div>
</div>
<div *ngIf="currentFile" class="progress my-3">
  <div
    class="progress-bar progress-bar-info progress-bar-striped"
    role="progressbar"
    attr.aria-valuenow="{{ progress }}"
    aria-valuemin="0"
    aria-valuemax="100"
    [ngStyle]="{ width: progress + '%' }"
  >
    {{ progress }}%
  </div>
</div>
<div *ngIf="message" class="alert alert-secondary" role="alert">{{ message }}
</div>
<div class="card mt-3">
  <div class="card-header">List of Files</div>
  <ul
    class="list-group list-group-flush"
    *ngFor="let file of fileInfos | async"
  >
    <li class="list-group-item">
      <a href="{{ file.url }}">{{ file.name }}</a>
    </li>
  </ul>
</div>
```

Add Upload File Component to App Component

Open *app.component.html* and embed the FileUpload Component with `<app-file-upload>` tag.

```
<div class="container" style="width:600px">
  <div class="my-3">
    <h3>bezkoder.com</h3>
    <h4>{{ title }}</h4>
  </div>
  <app-file-upload></app-file-upload>
</div>
```

app.component.ts

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Angular 13 File Upload';
}
```

Run the App

Run Spring Boot Server with command: `mvn spring-boot:run`.
Refresh the project directory and you will see *uploads* folder inside it.

Because we configure CORS for origin: `http://localhost:8081`, so you need to run Angular 13 Client with command:
`ng serve --port 8081`

Open Browser with url `http://localhost:8081/` and check the result.

Further Reading

Fullstack CRUD App:
– Angular + Spring Boot + H2 example
– Angular + Spring Boot + MySQL example
– Angular + Spring Boot + PostgreSQL example
– Angular + Spring Boot + MongoDB example

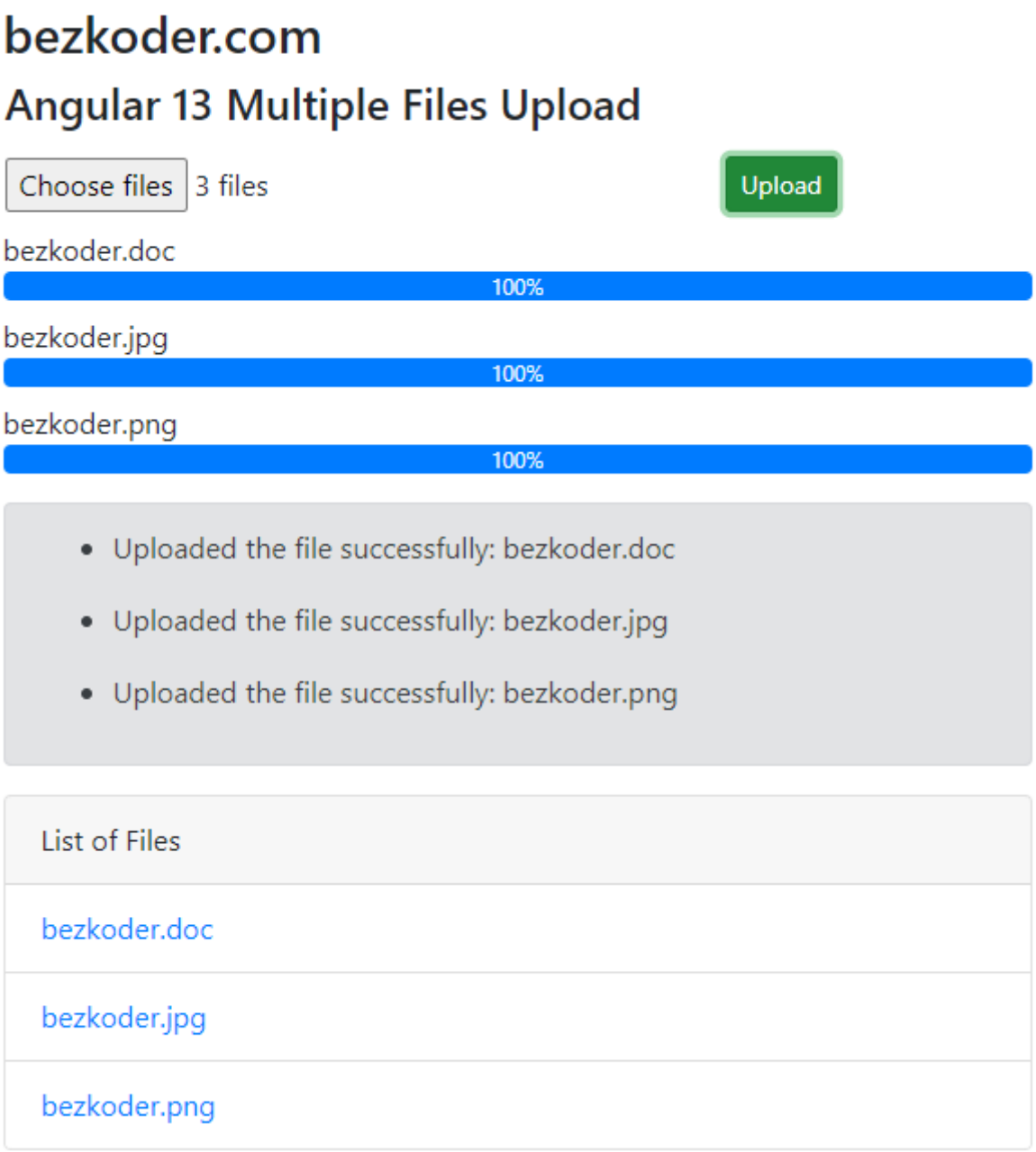
Serverless with Firebase:
Angular Upload File to Firebase Storage example

Conclusion

Today we're learned how to build an example for upload Files from Angular 13 to Spring Boot server. We also provide the ability to show list of files, upload progress using Bootstrap, and to download file from the server.

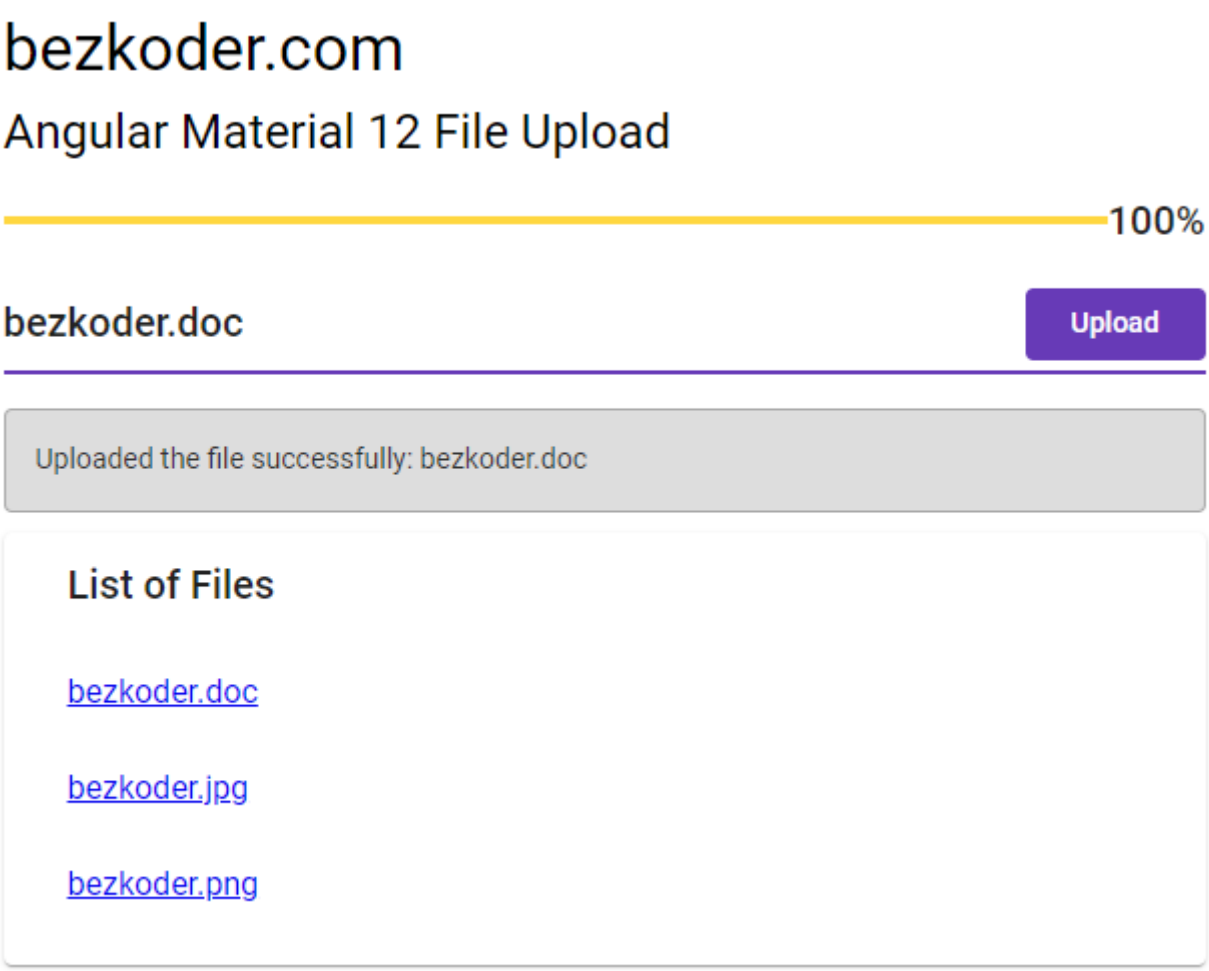
Next tutorials will show you way to implement the full-stack (with source code):
– Back-end: File Upload to Static folder / File Upload to Database
– Front-end

If you want to upload multiple files at once like this:



You can find the instruction (with Github) here:
Angular 13 Multiple Files upload example with Progress Bar

Or use Angular Material like this:



Please visit: Angular Material File upload example with progress bar

You will want to know how to run both projects in one place:
How to Integrate Angular with Spring Boot Rest API

Happy Learning! See you again.