How to upload multiple files in Java Spring Boot

bezkoder.com/spring-boot-upload-multiple-files

bezkoder Last modified: May 17, 2020

In this tutorial, I will show you how to upload multiple files and download with a Spring Boot Rest APIs. We also use Spring Web `MultipartFile` interface to handle HTTP multi-part requests and Postman to check the result.

Related Posts:
– Angular 8 + Spring Boot: File upload example
– React + Spring Boot: File upload example

Deployment: Deploy Spring Boot App on AWS – Elastic Beanstalk

Spring Boot Rest APIs for uploading multiple Files

Our Spring Boot Application will provide APIs for:

- uploading multiple Files to Server
- downloading File from server with the link
- getting list of Files' information (file name & url)

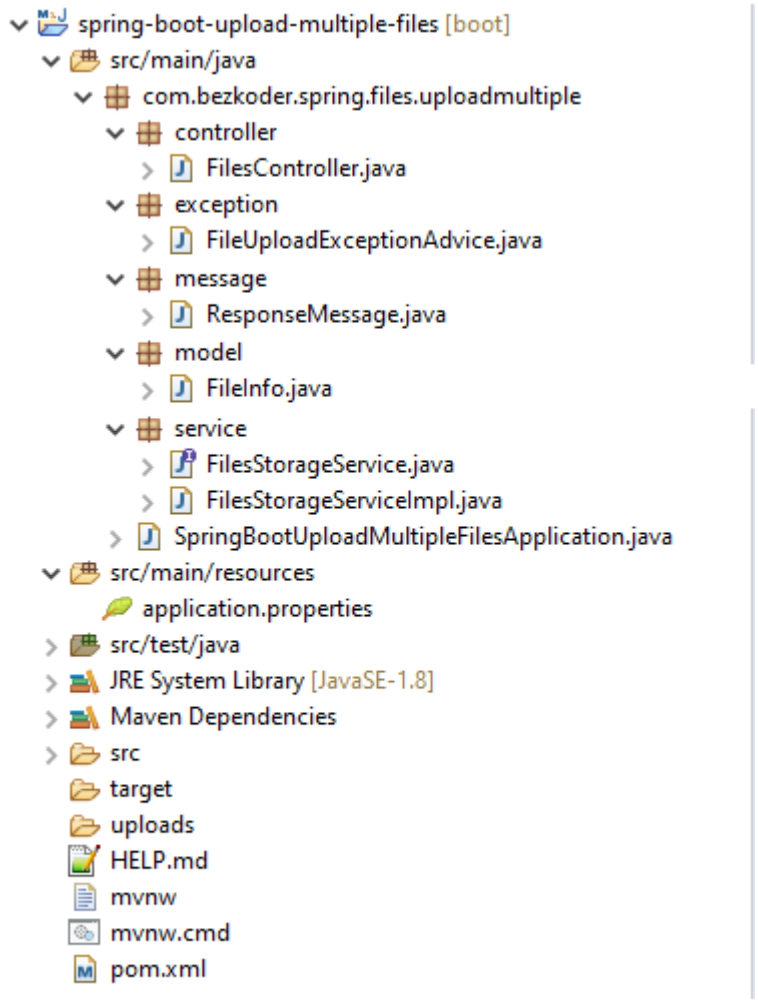
These are APIs to be exported:

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

Technology

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

Project Structure



Let me explain it briefly.

– `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 multiple files, 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.

Setup Spring Boot Multiple Files upload project

Use Spring web tool or your development tool (Spring Tool Suite, Eclipse, Intellij) to create a Spring Boot project.

Then open **pom.xml** and add these dependencies:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

Create Service for File Storage

First we need an interface that will be autowired in the Controller.
In *service* folder, create `FilesStorageService` interface like following code:

service/FilesStorageService.java

```
package com.bezkoder.spring.files.uploadmultiple.service;
import java.nio.file.Path;
import java.util.stream.Stream;
import org.springframework.core.io.Resource;
import org.springframework.web.multipart.MultipartFile;
public interface FilesStorageService {
  public void init();
  public void save(MultipartFile file);
  public Resource load(String filename);
  public void deleteAll();
  public Stream<Path> loadAll();
}
```

Now we create implementation of the interface.

service/FilesStorageServiceImpl.java

```
package com.bezkoder.spring.files.uploadmultiple.service;
import java.io.IOException;
import java.net.MalformedURLException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.stream.Stream;
import org.springframework.core.io.Resource;
import org.springframework.core.io.UrlResource;
import org.springframework.stereotype.Service;
import org.springframework.util.FileSystemUtils;
import org.springframework.web.multipart.MultipartFile;
@Service
public class FilesStorageServiceImpl implements FilesStorageService {
  private final Path root = Paths.get("uploads");
  @Override
  public void init() {
    try {
      Files.createDirectory(root);
    } catch (IOException e) {
      throw new RuntimeException("Could not initialize folder for upload!");
    }
  }
  @Override
  public void save(MultipartFile file) {
    try {
      Files.copy(file.getInputStream(),
this.root.resolve(file.getOriginalFilename()));
    } catch (Exception e) {
      throw new RuntimeException("Could not store the file. Error: " +
e.getMessage());
    }
  }
  @Override
  public Resource load(String filename) {
    try {
      Path file = root.resolve(filename);
      Resource resource = new UrlResource(file.toUri());
      if (resource.exists() || resource.isReadable()) {
        return resource;
      } else {
        throw new RuntimeException("Could not read the file!");
      }
    } catch (MalformedURLException e) {
      throw new RuntimeException("Error: " + e.getMessage());
    }
  }
  @Override
  public void deleteAll() {
    FileSystemUtils.deleteRecursively(root.toFile());
  }
  @Override
  public Stream<Path> loadAll() {
    try {
      return Files.walk(this.root, 1).filter(path ->
!path.equals(this.root)).map(this.root::relativize);
    } catch (IOException e) {
      throw new RuntimeException("Could not load the files!");
    }
```

```
  }
}
```

Define Data Models

In **model** package, let's create `FileInfo` which has fields: `name` & `url`.

model/FileInfo.java

```
package com.bezkoder.spring.files.uploadmultiple.model;
public class FileInfo {
  private String name;
  private String url;
  public FileInfo(String name, String url) {
    this.name = name;
    this.url = url;
  }
  public String getName() {
    return this.name;
  }
  public void setName(String name) {
    this.name = name;
  }
  public String getUrl() {
    return this.url;
  }
  public void setUrl(String url) {
    this.url = url;
  }
}
```

Define Response Message

`ResponseMessage` will be used for sending message to client in Controller and Exception Handler.

message/ResponseMessage.java

```
package com.bezkoder.spring.files.uploadmultiple.message;
public class ResponseMessage {
  private String message;
  public ResponseMessage(String message) {
    this.message = message;
  }
  public String getMessage() {
    return message;
  }
  public void setMessage(String message) {
    this.message = message;
  }
}
```

Create Controller for upload multiple Files & download

In **controller** package, we create `FilesController`.

controller/FilesController.java

```
package com.bezkoder.spring.files.uploadmultiple.controller;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.core.io.Resource;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.multipart.MultipartFile;
import
org.springframework.web.servlet.mvc.method.annotation.MvcUriComponentsBuilder;
import com.bezkoder.spring.files.uploadmultiple.model.FileInfo;
import com.bezkoder.spring.files.uploadmultiple.model.ResponseMessage;
import com.bezkoder.spring.files.uploadmultiple.service.FilesStorageService;
@Controller
@CrossOrigin("http://localhost:8081")
public class FilesController {
  @Autowired
  FilesStorageService storageService;
  @PostMapping("/upload")
  public ResponseEntity<ResponseMessage> uploadFiles(@RequestParam("files")
MultipartFile[] files) {
    String message = "";
    try {
      List<String> fileNames = new ArrayList<>();
      Arrays.asList(files).stream().forEach(file -> {
        storageService.save(file);
        fileNames.add(file.getOriginalFilename());
      });
      message = "Uploaded the files successfully: " + fileNames;
      return ResponseEntity.status(HttpStatus.OK).body(new
ResponseMessage(message));
    } catch (Exception e) {
      message = "Fail to upload files!";
      return ResponseEntity.status(HttpStatus.EXPECTATION_FAILED).body(new
ResponseMessage(message));
    }
  }
  @GetMapping("/files")
  public ResponseEntity<List<FileInfo>> getListFiles() {
    List<FileInfo> fileInfos = storageService.loadAll().map(path -> {
      String filename = path.getFileName().toString();
      String url = MvcUriComponentsBuilder
          .fromMethodName(FilesController.class, "getFile",
path.getFileName().toString()).build().toString();
      return new FileInfo(filename, url);
    }).collect(Collectors.toList());
    return ResponseEntity.status(HttpStatus.OK).body(fileInfos);
  }
  @GetMapping("/files/{filename:.+}")
  public ResponseEntity<Resource> getFile(@PathVariable String filename) {
```

```
    Resource file = storageService.load(filename);
    return ResponseEntity.ok()
        .header(HttpHeaders.CONTENT_DISPOSITION, "attachment; filename=\"" +
file.getFilename() + "\"").body(file);
  }
}
```

– `@CrossOrigin` is for configuring allowed origins.
– `@Controller` annotation is used to define a controller.
– `@GetMapping` and `@PostMapping` annotation is for mapping HTTP GET & POST requests onto specific handler methods:

- POST /upload: `uploadFiles()`
- GET /files: `getListFiles()`
- GET /files/[filename]: `getFile()`

– We use `@Autowired` to inject implementation of `FilesStorageService` bean to local variable.

The main method is `uploadFiles()` in which we use `MultipartFile[] files` as an argument, and Java 8 Stream API to work with each file in the array.

Configure Multipart File for Servlet

Let's define the maximum file size that can be uploaded in *application.properties* as following:

```
spring.servlet.multipart.max-file-size=500KB
spring.servlet.multipart.max-request-size=500KB
```

– `spring.servlet.multipart.max-file-size` : max file size for each request.
– `spring.servlet.multipart.max-request-size` : max request size for a multipart/form-data.

Handle File Upload Exception

This is where we handle the case in that a request exceeds Max Upload Size. The system will throw `MaxUploadSizeExceededException` and we're gonna use `@ControllerAdvice` with `@ExceptionHandler` annotation for handling the exceptions.

exception/FileUploadExceptionAdvice.java

```
package com.bezkoder.spring.files.uploadmultiple.exception;
import org.springframework.web.multipart.MaxUploadSizeExceededException;
import
org.springframework.web.servlet.mvc.method.annotation.ResponseEntityExceptionHandle

import com.bezkoder.spring.files.uploadmultiple.model.ResponseMessage;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
@ControllerAdvice
public class FileUploadExceptionAdvice extends ResponseEntityExceptionHandler {
  @ExceptionHandler(MaxUploadSizeExceededException.class)
  public ResponseEntity<ResponseMessage>
handleMaxSizeException(MaxUploadSizeExceededException exc) {
    return ResponseEntity
        .status(HttpStatus.EXPECTATION_FAILED)
        .body(new ResponseMessage("One or more files are too large!"));
  }
}
```

Initialize Storage

We need to run `init()` method of `FilesStorageService` (and also `deleteAll()` if necessary). So open *SpringBootUploadMultipleFilesApplication.java* and implement `CommandLineRunner` for `run()` method like this:

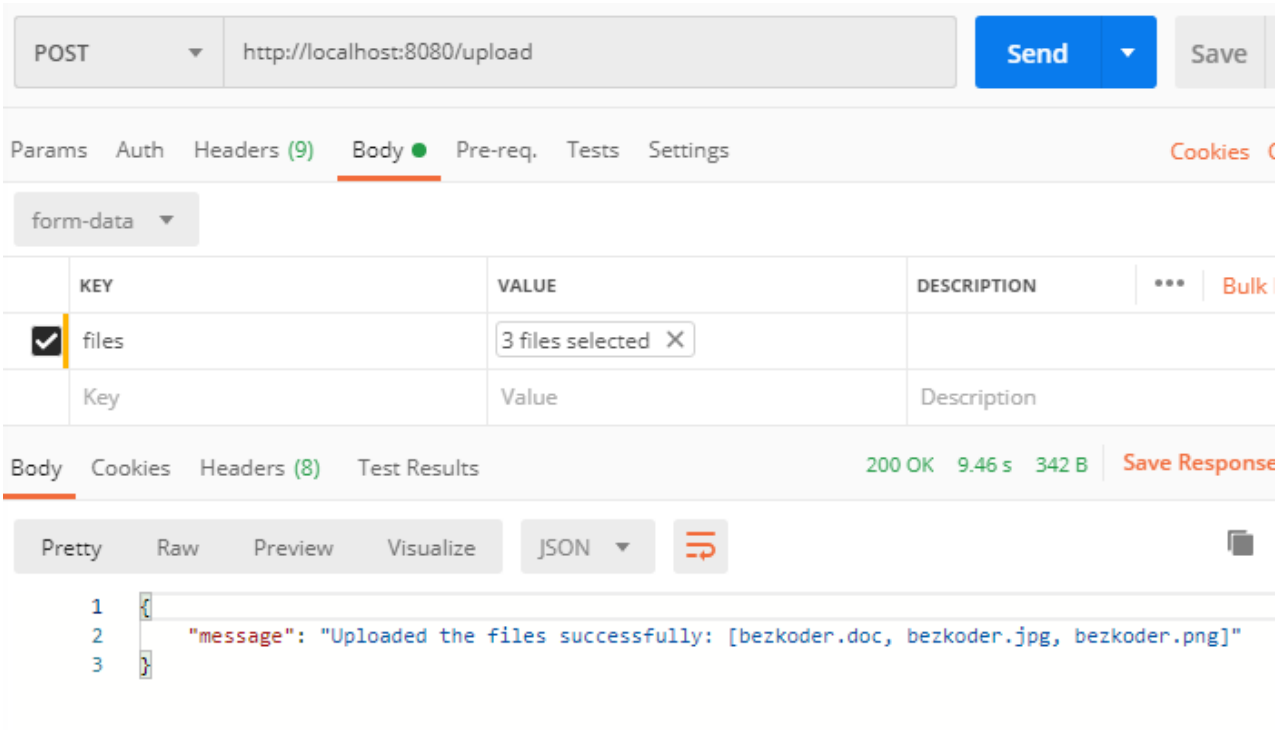
```
package com.bezkoder.spring.files.uploadmultiple;
import javax.annotation.Resource;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import com.bezkoder.spring.files.uploadmultiple.service.FilesStorageService;
@SpringBootApplication
public class SpringBootUploadMultipleFilesApplication implements CommandLineRunner
{
  @Resource
  FilesStorageService storageService;
  public static void main(String[] args) {
    SpringApplication.run(SpringBootUploadMultipleFilesApplication.class, args);
  }
  @Override
  public void run(String... arg) throws Exception {
    storageService.deleteAll();
    storageService.init();
  }
}
```

Spring Boot upload Multiple Files with Postman

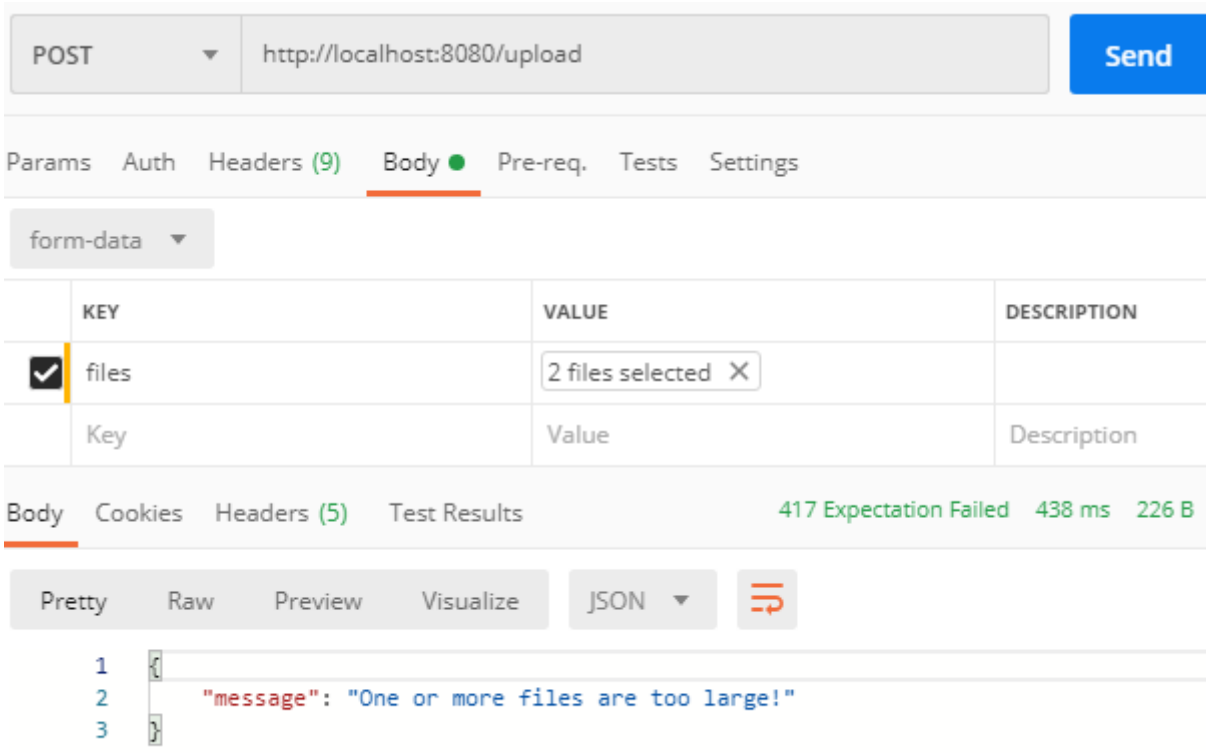
Run Spring Boot application with command: `mvn spring-boot:run`.
Refresh the project directory and you will see *uploads* folder inside it.

Let's use **Postman** to make some requests.

– Upload some files: In the **Body** tab, chose **form-data**, key **files** as *File* type. For *value* column, choose several files from your PC.

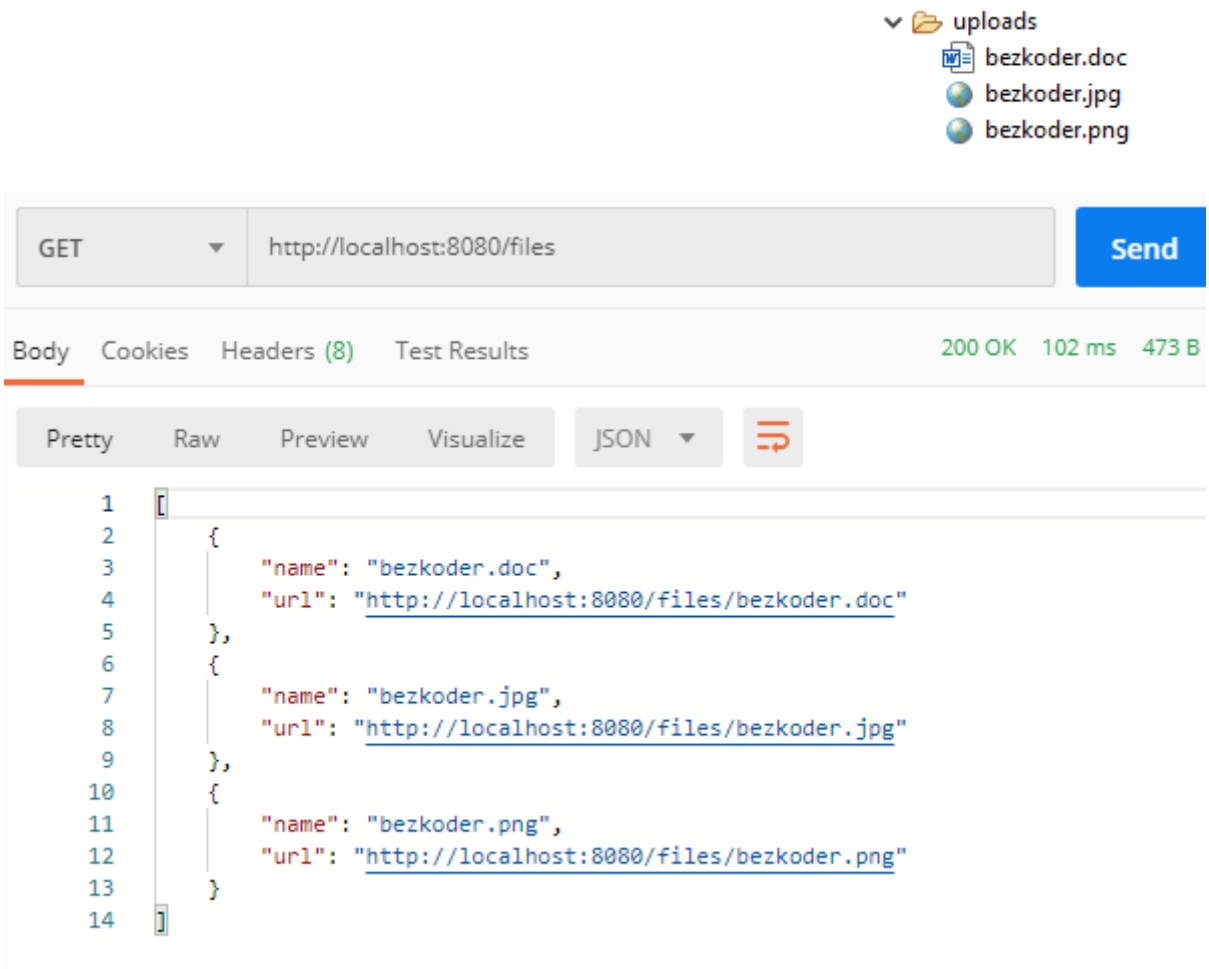


– Upload files which contains certain file with size larger than max file size (500KB):



– Check **uploads** folder:

– Retrieve list of Files' information:



– Now you can download any file from one of the paths above.
For example: `http://localhost:8080/files/bezkoder.doc`.

Conclusion

Today we've learned how to create Java Spring Boot Application to upload multiple files and get files' information via Rest API.

You can find the complete source code for this tutorial on Github.

This post explains how to build Angular Client to work with Spring Boot Server.

Happy Learning! See you again.

Further Reading

- Multipart Content-Type