application.properties:

```
# max file size
spring.servlet.multipart.max-file-size=10MB
# max request size
spring.servlet.multipart.max-request-size=10MB
# files storage location (stores all files uploaded via REST API)
storage.location=./uploads
```

StorageProperties.java:

```
package com.attacomsian.uploadfiles.storage;

import org.springframework.boot.context.properties.ConfigurationProperties;

@ConfigurationProperties(prefix = "storage")
public class StorageProperties {

 private String location;

 public String getLocation() {
 return location;
 }

 public void setLocation(String location) {
 this.location = location;
 }

}
```

Application.java:

```
package com.attacomsian.uploadfiles;

import com.attacomsian.uploadfiles.storage.StorageProperties;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.context.properties.EnableConfigurationProperties;

@SpringBootApplication
@EnableConfigurationProperties(StorageProperties.class)
public class Application {

 public static void main(String[] args) {
 SpringApplication.run(Application.class, args);
 }

}
```

FileController.java:

```
package com.attacomsian.uploadfiles.controllers;

import com.attacomsian.uploadfiles.commons.FileResponse;
import com.attacomsian.uploadfiles.storage.StorageService;
import org.springframework.core.io.Resource;
import org.springframework.http.HttpHeaders;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.multipart.MultipartFile;
import org.springframework.web.servlet.support.ServletUriComponentsBuilder;

import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;
```

```
@Controller
public class FileController {

 private StorageService storageService;

 public FileController(StorageService storageService) {
 this.storageService = storageService;
 }

 @GetMapping("/")
 public String listAllFiles(Model model) {

 model.addAttribute("files", storageService.loadAll().map(
 path -> ServletUriComponentsBuilder.fromCurrentContextPath()
 .path("/download/")
 .path(path.getFileName().toString())
 .toUriString())
 .collect(Collectors.toList()));

 return "listFiles";
 }

 @GetMapping("/download/{filename:.+}")
 @ResponseBody
 public ResponseEntity<Resource> downloadFile(@PathVariable String filename) {

 Resource resource = storageService.loadAsResource(filename);

 return ResponseEntity.ok()
 .header(HttpHeaders.CONTENT_DISPOSITION,
 "attachment; filename=\"" + resource.getFilename() + "\"")
 .body(resource);
 }

 @PostMapping("/upload-file")
 @ResponseBody
 public FileResponse uploadFile(@RequestParam("file") MultipartFile file) {
 String name = storageService.store(file);

 String uri = ServletUriComponentsBuilder.fromCurrentContextPath()
 .path("/download/")
 .path(name)
 .toUriString();

 return new FileResponse(name, uri, file.getContentType(), file.getSize());
 }

 @PostMapping("/upload-multiple-files")
 @ResponseBody
 public List<FileResponse> uploadMultipleFiles(@RequestParam("files") MultipartFile[] files) {
 return Arrays.stream(files)
 .map(file -> uploadFile(file))
 .collect(Collectors.toList());
 }
}
```

FileResponse.java:

```
package com.attacomsian.uploadfiles.commons;

public class FileResponse {
 private String name;
 private String uri;
 private String type;
 private long size;

 public FileResponse(String name, String uri, String type, long size) {
 this.name = name;
 this.uri = uri;
 this.type = type;
 this.size = size;
 }

 // getters and setters removed for the sake of brevity
}
```

StorageService.java:

```
package com.attacomsian.uploadfiles.storage;

import org.springframework.core.io.Resource;
import org.springframework.web.multipart.MultipartFile;

import java.nio.file.Path;
import java.util.stream.Stream;

public interface StorageService {

 void init();

 String store(MultipartFile file);

 Stream<Path> loadAll();

 Path load(String filename);

 Resource loadAsResource(String filename);
 void deleteAll();

}
```

FileSystemStorageService.java:

```
package com.attacomsian.uploadfiles.storage;

import org.springframework.beans.factory.annotation.Autowired;
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.util.StringUtils;
import org.springframework.web.multipart.MultipartFile;

import javax.annotation.PostConstruct;
import java.io.IOException;
import java.io.InputStream;
import java.net.MalformedURLException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.StandardCopyOption;
import java.util.stream.Stream;

@Service
public class FileSystemStorageService implements StorageService {

 private final Path rootLocation;

 @Autowired
```

```
public FileSystemStorageService(StorageProperties properties) {
this.rootLocation = Paths.get(properties.getLocation());
}

@Override
@PostConstruct
public void init() {
try {
Files.createDirectories(rootLocation);
} catch (IOException e) {
throw new StorageException("Could not initialize storage location", e);
}
}

@Override
public String store(MultipartFile file) {
String filename = StringUtils.cleanPath(file.getOriginalFilename());
try {
if (file.isEmpty()) {
throw new StorageException("Failed to store empty file " + filename);
}
if (filename.contains("..")) {
// This is a security check
throw new StorageException(
"Cannot store file with relative path outside current directory "
+ filename);
}
try (InputStream inputStream = file.getInputStream()) {
Files.copy(inputStream, this.rootLocation.resolve(filename),
StandardCopyOption.REPLACE_EXISTING);
}
```

```
}
catch (IOException e) {
throw new StorageException("Failed to store file " + filename, e);
}

return filename;
}

@Override
public Stream<Path> loadAll() {
try {
return Files.walk(this.rootLocation, 1)
.filter(path -> !path.equals(this.rootLocation))
.map(this.rootLocation::relativize);
}
catch (IOException e) {
throw new StorageException("Failed to read stored files", e);
}

}

@Override
public Path load(String filename) {
return rootLocation.resolve(filename);
}

@Override
public Resource loadAsResource(String filename) {
try {
Path file = load(filename);
Resource resource = new UrlResource(file.toUri());
```

```
 if (resource.exists() || resource.isReadable()) {
 return resource;
 }
 else {
 throw new FileNotFoundException(
 "Could not read file: " + filename);
 }
 }
 catch (MalformedURLException e) {
 throw new FileNotFoundException("Could not read file: " + filename, e);
 }
 }

 @Override
 public void deleteAll() {
 FileSystemUtils.deleteRecursively(rootLocation.toFile());
 }
}
```

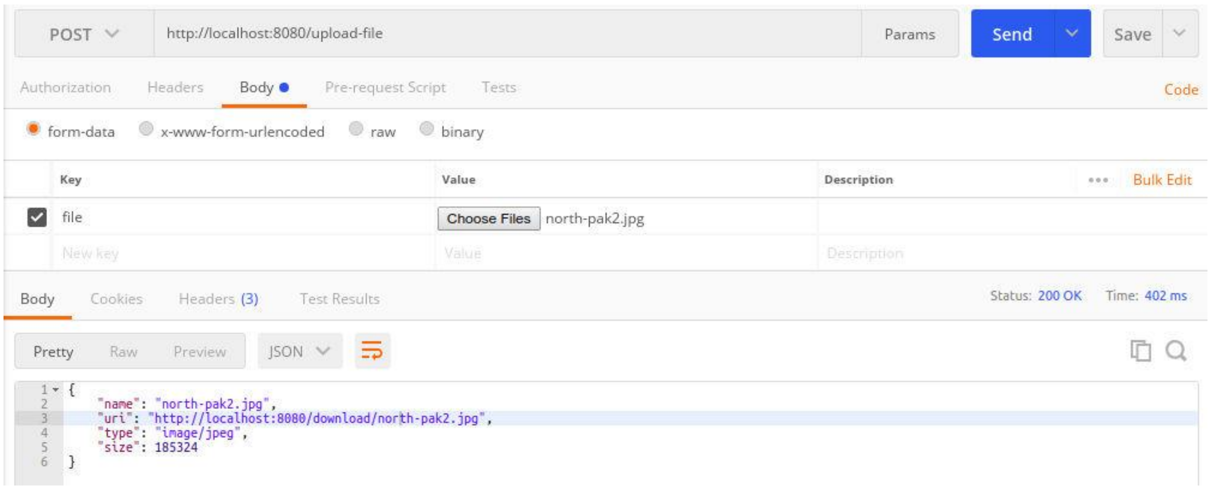
StorageException.java:

```
package com.attacomsian.uploadfiles.storage;

public class StorageException extends RuntimeException {

 public StorageException(String message) {
 super(message);
 }

 public StorageException(String message, Throwable cause) {
 super(message, cause);
 }
}
```

FileNotFoundException.java:

```
package com.attacomsian.uploadfiles.storage;


import org.springframework.http.HttpStatus;

import org.springframework.web.bind.annotation.ResponseStatus;


@ResponseStatus(HttpStatus.NOT_FOUND)

public class FileNotFoundException extends StorageException {


 public FileNotFoundException(String message) {

 super(message);

 }


 public FileNotFoundException(String message, Throwable cause) {

 super(message, cause);

 }

}
```

Upload Single File:



POST http://localhost:8080/upload-file

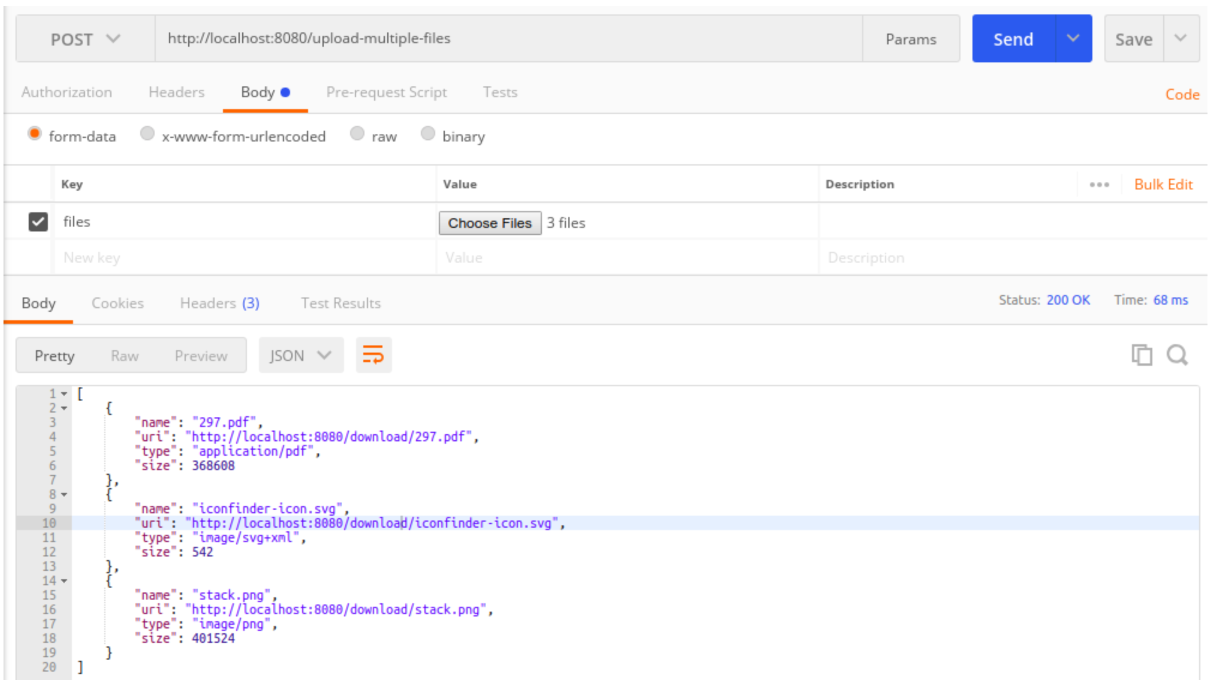
form-data

Key	Value	Description
✓ file	Choose Files north-pak2.jpg	

Status: 200 OK Time: 402 ms

```
{
    "name": "north-pak2.jpg",
    "uri": "http://localhost:8080/download/north-pak2.jpg",
    "type": "image/jpeg",
    "size": 185324
}
```

Upload Multiple Files:



http://localhost:8080/upload-multiple-files

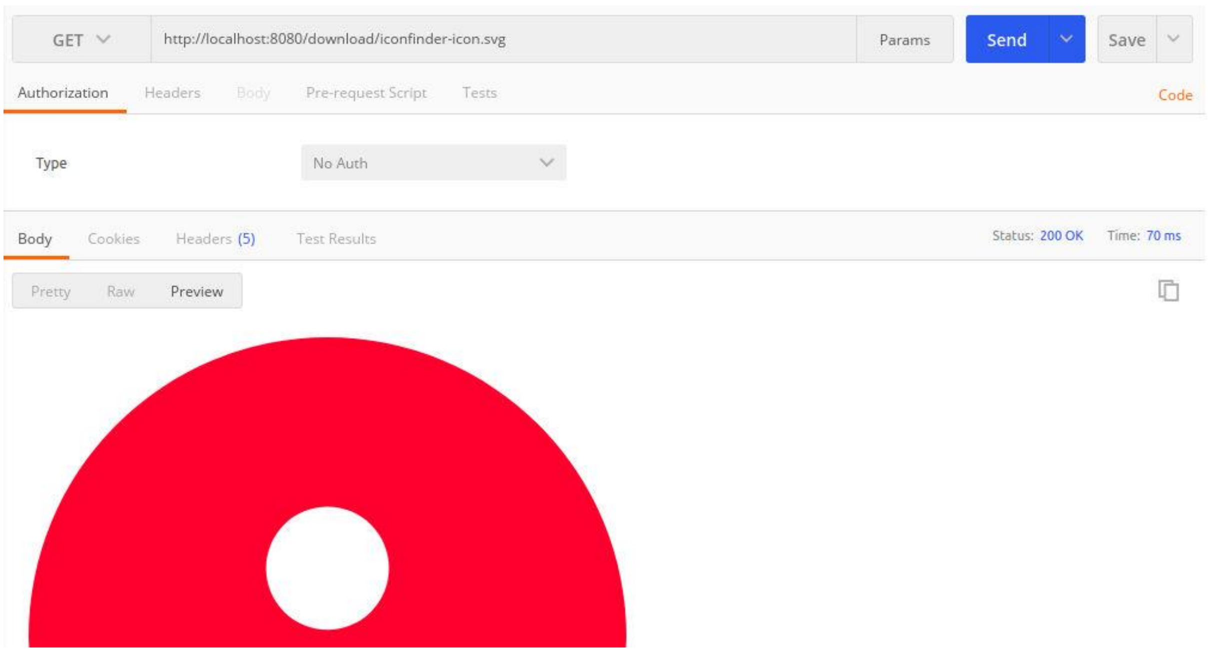
form-data x-www-form-urlencoded raw binary

Key	Value	Description
✓ files	Choose Files 3 files	
New key	Value	Description

Status: 200 OK Time: 68 ms

```
[
    {
        "name": "297.pdf",
        "uri": "http://localhost:8080/download/297.pdf",
        "type": "application/pdf",
        "size": 368608
    },
    {
        "name": "iconfinder-icon.svg",
        "uri": "http://localhost:8080/download/iconfinder-icon.svg",
        "type": "image/svg+xml",
        "size": 542
    },
    {
        "name": "stack.png",
        "uri": "http://localhost:8080/download/stack.png",
        "type": "image/png",
        "size": 401524
    }
]
```

Download File:



http://localhost:8080/download/iconfinder-icon.svg

Type No Auth

Status: 200 OK Time: 70 ms