Spring Boot Upload/Download File to/from Database example

bezkoder.com/spring-boot-upload-file-database

bezkoder Last modified: August 19, 2021

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

This Spring Boot App works with:
– Angular 8 Client / Angular 10 Client / Angular 11 Client / Angular 12
– Angular Material 12
– Vue Client / Vuetify Client
– React Client / React Hooks Client
– Material UI Client

Related Posts:
– How to upload multiple files in Java Spring Boot
– Spring Boot: Upload/Import Excel file data into MySQL Database
– Spring Boot: Upload/Import CSV file data into MySQL Database

Deployment:
– Deploy Spring Boot App on AWS – Elastic Beanstalk
– Docker Compose: Spring Boot and MySQL example

Spring Boot Rest APIs for uploading Files to Database

Our Spring Boot Application will provide APIs for:

- uploading File to PostgreSQL/MySQL database
- downloading File database with the link
- getting list of Files' information (file name, url, type, size)

These are APIs to be exported:

Methods	Urls	Actions
POST	/upload	upload a File
GET	/files	get List of Files (name, url, type, size)
GET	/files/[fileId]	download a File

The uploaded files will be stored in PostgreSQL/MySQL Database **files** table with these fields: `id`, `name`, `type` and `data` as **BLOB** type (Binary Large Object is for storing binary data like file, image, audio, or video).

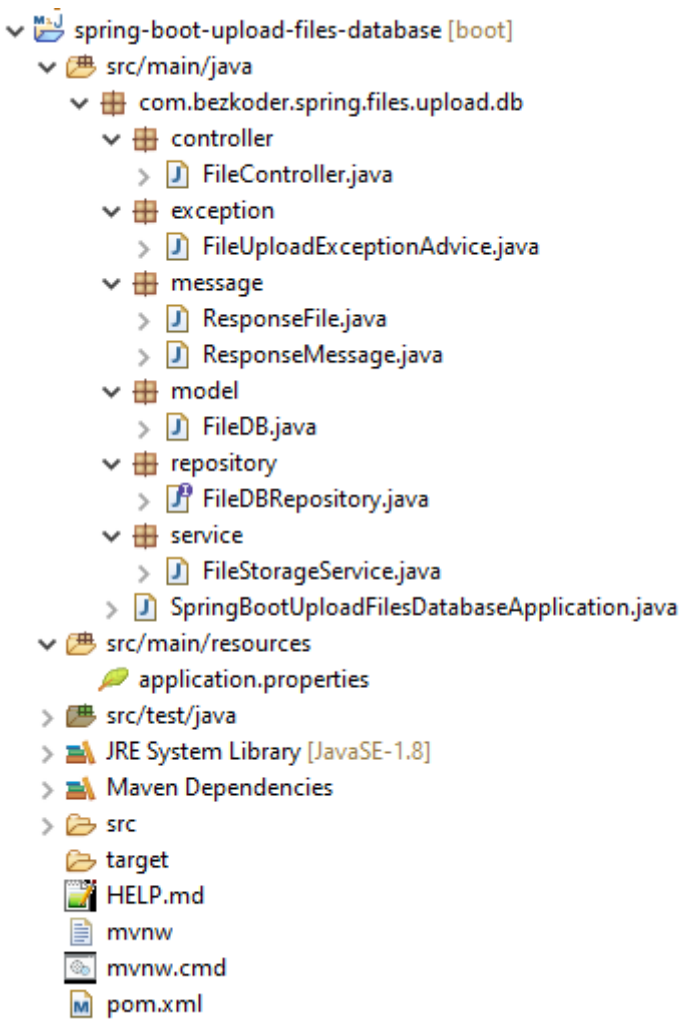
id	data	name	type
5d71322e-a954-4d7a-b0e6-7c799b5aae5f	BLOB	bezkoder.png	image/png
6ba3578c-ce22-4dd7-999e-72192bf31b53	BLOB	bezkoder.doc	application/msword
88108ee4-5354-4041-bfc6-2965fc8af4f4	BLOB	bezkoder.jpg	image/jpeg

Technology

- Java 8
- Spring Boot 2 (with Spring Web MVC)
- PostgreSQL/MySQL Database
- Maven 3.6.1

Project Structure

Let me explain it briefly.



– `FileDB` is the data model corresponding to **files** table in database.
– `FileDBRepository` extends Spring Data `JpaRepository` which has methods to store and retrieve files.
– `FilesStorageService` uses `FileDBRepository` to provide methods for saving new file, get file by id, get list of 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.
– `ResponseFile` contains information of the file (name, url, type, size) for HTTP response payload.
– *application.properties* contains configuration for Servlet Multipart and PostgreSQL/MySQL database connection.
– *pom.xml* for Spring Boot, Spring Data JPA and PostgreSQL/MySQL connector dependency.

Setup Spring Boot 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>
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

We also need to add one more dependency.
– If you want to use **MySQL**:

```
<dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <scope>runtime</scope>
</dependency>
```

– or **PostgreSQL**:

```
<dependency>
        <groupId>org.postgresql</groupId>
        <artifactId>postgresql</artifactId>
        <scope>runtime</scope>
</dependency>
```

Create Data Model

This Data Model is for storing File Data. There are four fields:

- `id` : automatically generated as UUID
- `name` : name of the file
- `type` : mime type
- `data` : array of bytes, map to a BLOB

model/FileDB.java

```
package com.bezkoder.spring.files.upload.db.model;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Lob;
import javax.persistence.Table;
import org.hibernate.annotations.GenericGenerator;
@Entity
@Table(name = "files")
public class FileDB {
  @Id
  @GeneratedValue(generator = "uuid")
  @GenericGenerator(name = "uuid", strategy = "uuid2")
  private String id;
  private String name;
  private String type;
  @Lob
  private byte[] data;
  public FileDB() {
  }
  public FileDB(String name, String type, byte[] data) {
    this.name = name;
    this.type = type;
    this.data = data;
  }
  public String getId() {
    return id;
  }
  public String getName() {
    return name;
  }
  public void setName(String name) {
    this.name = name;
  }
  public String getType() {
    return type;
  }
  public void setType(String type) {
    this.type = type;
  }
  public byte[] getData() {
    return data;
  }
  public void setData(byte[] data) {
    this.data = data;
  }
}
```

In the code above, `data` is annotated by `@Lob` annotation. *LOB* is datatype for storing large object data. There're two kinds of LOB: BLOB and CLOB:

- BLOB is for storing binary data
- CLOB is for storing text data

Create Repository

Under **repository** package, create `FileDBRepository` interface that extends `JpaRepository`.

repository/FileDBRepository.java

```
package com.bezkoder.spring.files.upload.db.repository;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import com.bezkoder.spring.files.upload.db.model.FileDB;
@Repository
public interface FileDBRepository extends JpaRepository<FileDB, String> {
}
```

Now we can use `FileDBRepository` with `JpaRepository`'s methods such as: `save(FileDB)`, `findById(id)`, `findAll()`.

Create Service for File Storage

The File Storage Service will use `FileDBRepository` to provide following methods:

- `store(file)`: receives `MultipartFile` object, transform to `FileDB` object and save it to Database
- `getFile(id)`: returns a `FileDB` object by provided Id
- `getAllFiles()`: returns all stored files as list of code>FileDB objects

service/FileStorageService.java

```
package com.bezkoder.spring.files.upload.db.service;
import java.io.IOException;
import java.util.stream.Stream;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.util.StringUtils;
import org.springframework.web.multipart.MultipartFile;
import com.bezkoder.spring.files.upload.db.model.FileDB;
import com.bezkoder.spring.files.upload.db.repository.FileDBRepository;
@Service
public class FileStorageService {
  @Autowired
  private FileDBRepository fileDBRepository;
  public FileDB store(MultipartFile file) throws IOException {
    String fileName = StringUtils.cleanPath(file.getOriginalFilename());
    FileDB FileDB = new FileDB(fileName, file.getContentType(), file.getBytes());
    return fileDBRepository.save(FileDB);
  }
  public FileDB getFile(String id) {
    return fileDBRepository.findById(id).get();
  }

  public Stream<FileDB> getAllFiles() {
    return fileDBRepository.findAll().stream();
  }
}
```

Define Response Information Classes

Let's create two classes in **message** package. The controller will use these classes for sending message via HTTP responses.

- `ResponseFile` : contains *name*, *url*, *type*, *size*
- `ResponseMessage` for notification/information message

message/ResponseFile.java

```
package com.bezkoder.spring.files.upload.db.message;
public class ResponseFile {
  private String name;
  private String url;
  private String type;
  private long size;
  public ResponseFile(String name, String url, String type, long size) {
    this.name = name;
    this.url = url;
    this.type = type;
    this.size = size;
  }
  public String getName() {
    return name;
  }
  public void setName(String name) {
    this.name = name;
  }
  public String getUrl() {
    return url;
  }
  public void setUrl(String url) {
    this.url = url;
  }
  public String getType() {
    return type;
  }
  public void setType(String type) {
    this.type = type;
  }
  public long getSize() {
    return size;
  }
  public void setSize(long size) {
    this.size = size;
  }
}
```

message/ResponseMessage.java

```
package com.bezkoder.spring.files.upload.db.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 & download Files to Database

In **controller** package, we create `FileController` class.

controller/FileController.java

```
package com.bezkoder.spring.files.upload.db.controller;
import java.util.List;
import java.util.stream.Collectors;
import org.springframework.beans.factory.annotation.Autowired;
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.support.ServletUriComponentsBuilder;
import com.bezkoder.spring.files.upload.db.service.FileStorageService;
import com.bezkoder.spring.files.upload.db.message.ResponseFile;
import com.bezkoder.spring.files.upload.db.message.ResponseMessage;
import com.bezkoder.spring.files.upload.db.model.FileDB;
@Controller
@CrossOrigin("http://localhost:8081")
public class FileController {
  @Autowired
  private FileStorageService storageService;
  @PostMapping("/upload")
  public ResponseEntity<ResponseMessage> uploadFile(@RequestParam("file")
MultipartFile file) {
    String message = "";
    try {
      storageService.store(file);
      message = "Uploaded the file successfully: " + file.getOriginalFilename();
      return ResponseEntity.status(HttpStatus.OK).body(new
ResponseMessage(message));
    } catch (Exception e) {
      message = "Could not upload the file: " + file.getOriginalFilename() + "!";
      return ResponseEntity.status(HttpStatus.EXPECTATION_FAILED).body(new
ResponseMessage(message));
    }
  }
  @GetMapping("/files")
  public ResponseEntity<List<ResponseFile>> getListFiles() {
    List<ResponseFile> files = storageService.getAllFiles().map(dbFile -> {
      String fileDownloadUri = ServletUriComponentsBuilder
          .fromCurrentContextPath()
          .path("/files/")
          .path(dbFile.getId())
          .toUriString();
      return new ResponseFile(
          dbFile.getName(),
          fileDownloadUri,
          dbFile.getType(),
          dbFile.getData().length);
    }).collect(Collectors.toList());
    return ResponseEntity.status(HttpStatus.OK).body(files);
  }
  @GetMapping("/files/{id}")
  public ResponseEntity<byte[]> getFile(@PathVariable String id) {
    FileDB fileDB = storageService.getFile(id);
    return ResponseEntity.ok()
```

```
        .header(HttpHeaders.CONTENT_DISPOSITION, "attachment; filename=\"" +
fileDB.getName() + "\"")
        .body(fileDB.getData());
  }
}
```

– `@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: `uploadFile()`
- GET /files: `getListFiles()`
- GET /files/[id]: `getFile()`

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

Configure Spring Datasource, JPA, Hibernate

Under src/main/resources folder, open application.properties and write these lines.

– For MySQL:

```
spring.datasource.url= jdbc:mysql://localhost:3306/testdb?useSSL=false
spring.datasource.username= root
spring.datasource.password= 123456
spring.jpa.properties.hibernate.dialect= org.hibernate.dialect.MySQL5InnoDBDialect
# Hibernate ddl auto (create, create-drop, validate, update)
spring.jpa.hibernate.ddl-auto= update
```

– For PostgreSQL:

```
spring.datasource.url= jdbc:postgresql://localhost:5432/testdb
spring.datasource.username= postgres
spring.datasource.password= 123
spring.jpa.properties.hibernate.jdbc.lob.non_contextual_creation= true
spring.jpa.properties.hibernate.dialect= org.hibernate.dialect.PostgreSQLDialect
# Hibernate ddl auto (create, create-drop, validate, update)
spring.jpa.hibernate.ddl-auto= update
```

- `spring.datasource.username` & `spring.datasource.password` properties are the same as your database installation.
- Spring Boot uses Hibernate for JPA implementation, we configure `MySQL5InnoDBDialect` for MySQL or `PostgreSQLDialect` for PostgreSQL
- `spring.jpa.hibernate.ddl-auto` is used for database initialization. We set the value to `update` value so that a table will be created in the database automatically corresponding to defined data model. Any change to the model will also trigger an update to the table. For production, this property should be `validate`.

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=2MB
spring.servlet.multipart.max-request-size=2MB
```

– `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

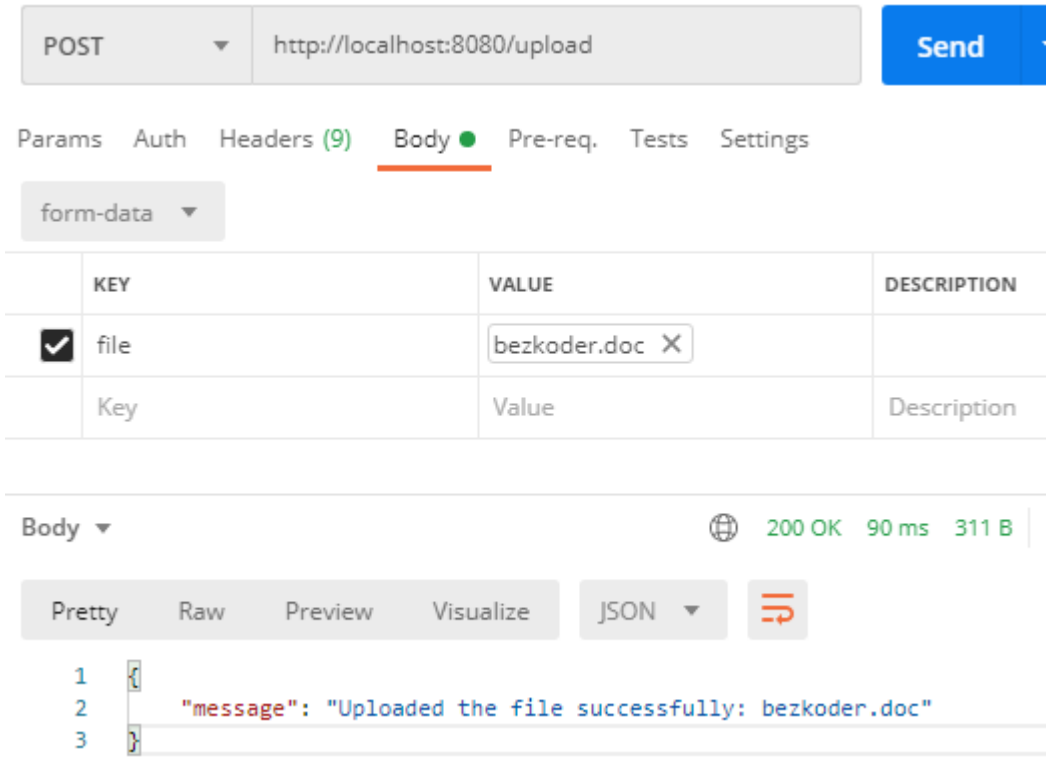
```
package com.bezkoder.spring.files.upload.db.exception;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.multipart.MaxUploadSizeExceededException;
import
org.springframework.web.servlet.mvc.method.annotation.ResponseEntityExceptionHandle

import com.bezkoder.spring.files.upload.db.message.ResponseMessage;
@ControllerAdvice
public class FileUploadExceptionAdvice extends ResponseEntityExceptionHandler {
  @ExceptionHandler(MaxUploadSizeExceededException.class)
  public ResponseEntity<ResponseMessage>
handleMaxSizeException(MaxUploadSizeExceededException exc) {
    return ResponseEntity.status(HttpStatus.EXPECTATION_FAILED).body(new
ResponseMessage("File too large!"));
  }
}
```

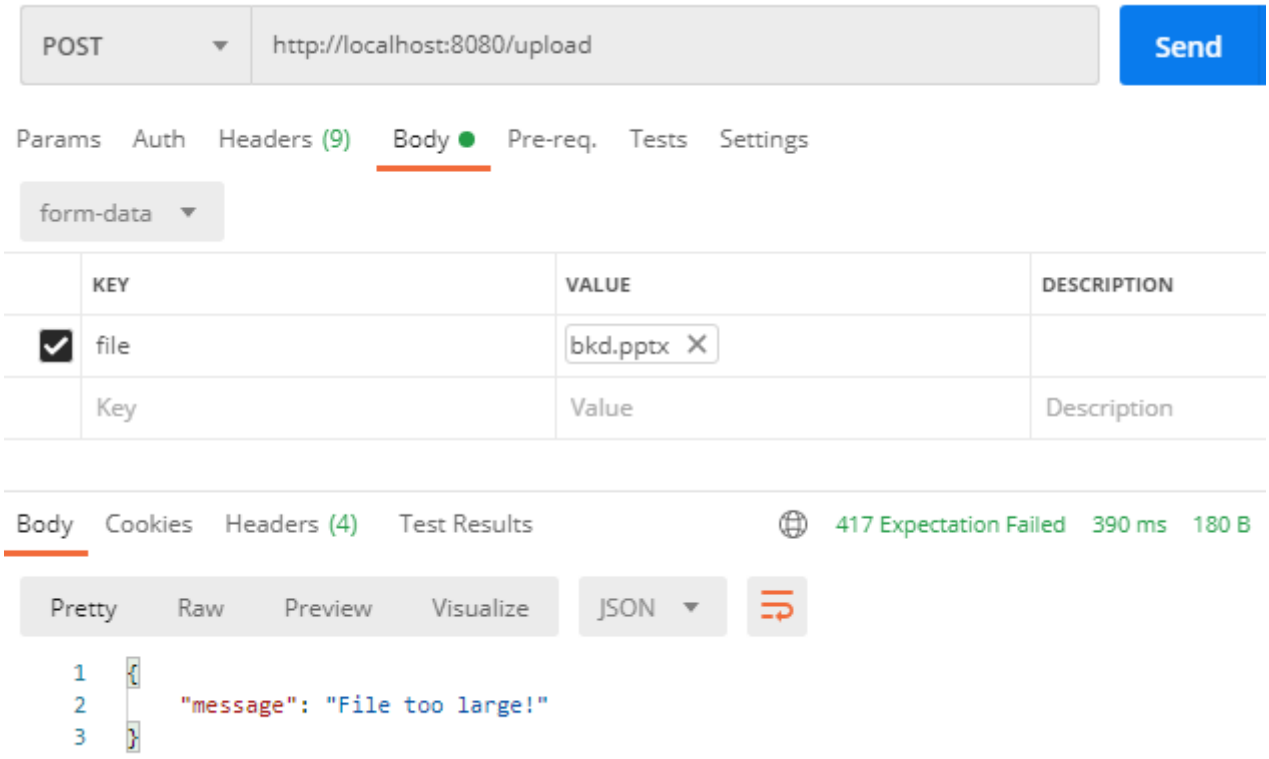
Run & Test

Run Spring Boot application with command: `mvn spring-boot:run` .
Let's use **Postman** to make some requests.

– Upload some files:



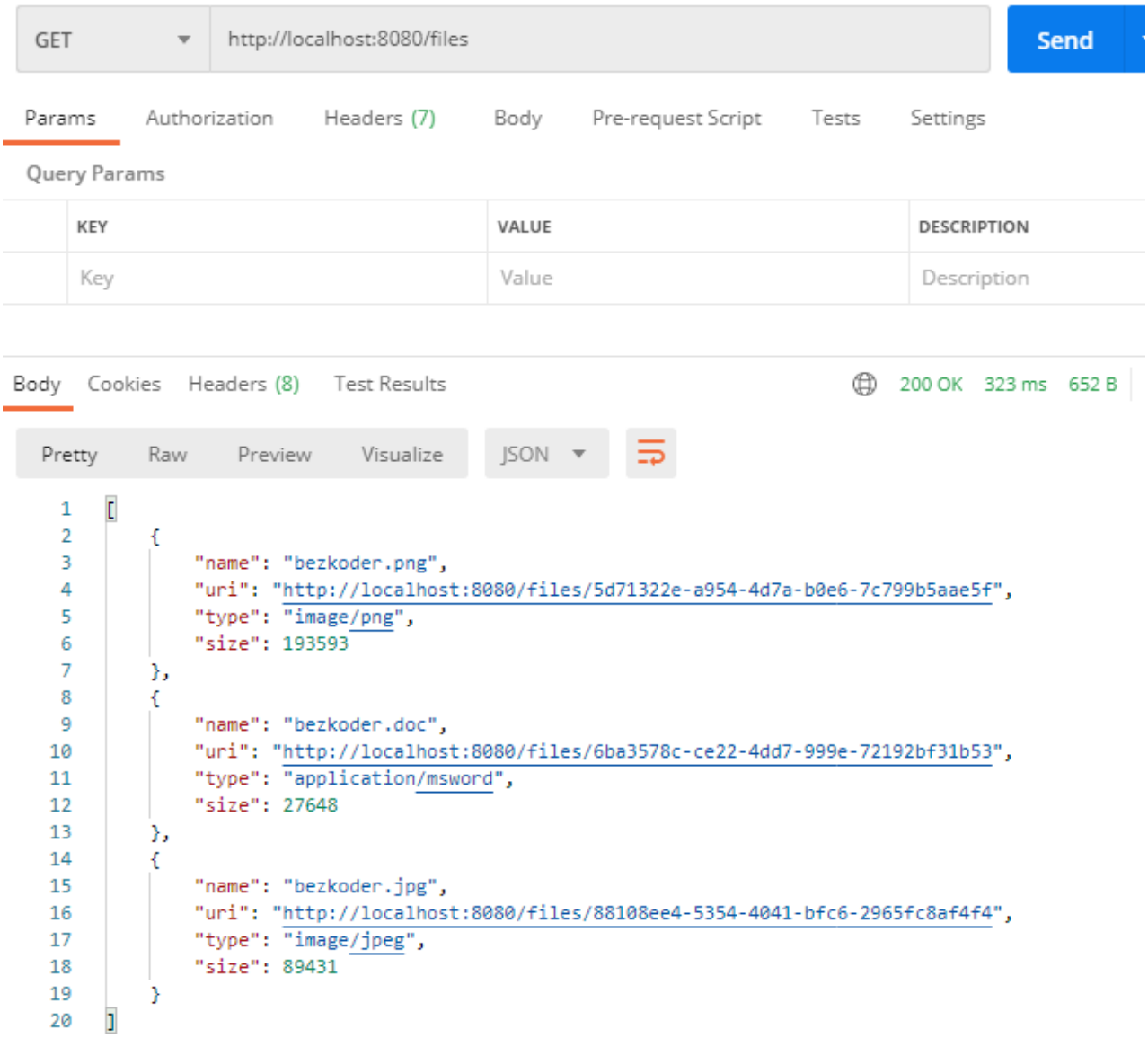
– Upload a file with size larger than max file size (2MB):



– Check **files** table in Database:

id	data	name	type
5d71322e-a954-4d7a-b0e6-7c799b5aae5f	BLOB	bezkoder.png	image/png
6ba3578c-ce22-4dd7-999e-72192bf31b53	BLOB	bezkoder.doc	application/msword
88108ee4-5354-4041-bfc6-2965fc8af4f4	BLOB	bezkoder.jpg	image/jpeg

– Retrieve list of Files' information:



– Now you can download any file from one of the paths above.
For example: `http://localhost:8080/files/6ba3578c-ce22-4dd7-999e-72192bf31b53`

Conclusion

Today we've learned how to create Spring Boot Application to upload multipart files and get files' information with static folder via Restful API.

Following tutorials explain how to build Front-end Apps to work with our Spring Boot Server:
– Angular 8 Client / Angular 10 Client / Angular 11 Client / Angular 12
– Angular Material 12
– Vue Client / Vuetify Client
– React Client / React Hooks Client
– Material UI Client

You can also know way to upload an Excel/CSV file and store the content in MySQL database with the post:
– Spring Boot: Upload/Import Excel file data into MySQL Database
– Spring Boot: Upload/Import CSV file data into MySQL Database

Happy Learning! See you again.

Further Reading

Deployment:
– Deploy Spring Boot App on AWS – Elastic Beanstalk
– Docker Compose: Spring Boot and MySQL example

Source Code

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