CRUD Operations for File Management with Spring Boot and MongoDB

Githublink:[hacker123shiva/CRUD-Operations-for-File-Management-with-Spring-Boot-and-MongoDB: CRUD Operations for File Management with Spring Boot and MongoDB (github.com)](https://github.com/hacker123shiva/CRUD-Operations-for-File-Management-with-Spring-Boot-and-MongoDB)
LinkedId: https://www.linkedin.com/in/shivasrivastava1/
Java Dev Community: **https://www.linkedin.com/groups/14530255/**

Purpose

The primary goal of this application is to enable users to upload files, download them, and manage metadata such as file name and author (writer name). This is particularly useful in scenarios like document management systems or content repositories.

Project Structure

```
src
 └── main
     ├── java
     │   └── com
     │       └── telusko
     │           ├── controller
     │                 ├── FileController.java
     │           ├── dao
     │                 ├── FileRepository.java
     │           ├── entity
     │                 ├── FileEntity.java
     │           ├── exception
     │                 ├── DuplicateFileNameException.java
     │                 ├── ErrorResponse.java
     │                 ├── FileNotFoundException.java
     │                 ├── GlobalExceptionHandler.java
     │           ├── service
     │                 ├── FileService.java
     │           └── Application.java
     └── resources
         ├── application.properties
         └── templates
```

Dependencies

In the `pom.xml` file, we declare the necessary dependencies for our Spring Boot application. Here's a breakdown:

```xml
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-mongodb</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <scope>runtime</scope>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
```

- **spring-boot-starter-data-mongodb**: Provides support for MongoDB.
- **spring-boot-starter-web**: Contains everything needed to build web applications.
- **spring-boot-devtools**: Enables automatic restarts and other development-time features.

- **lombok**: Simplifies Java code by generating boilerplate code (getters, setters).
- **spring-boot-starter-test**: Includes testing libraries for unit and integration tests.

Configuration

In `application.properties`, we specify the application name and MongoDB configuration:

```
spring.application.name=crud-operation-big-files
spring.data.mongodb.database=filedb
spring.data.mongodb.uri=mongodb://localhost:27017/filedb
```

Core Components

1. FileEntity

This class represents the structure of the file in our database.

```
package com.telusko.entity;

import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;

@Document(collection = "files")
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
public class FileEntity {

    @Id
    private String id;
    private String fileName;
    private String writerName;
```

```
    private byte[] fileData;

    public FileEntity(String fileName, String writerName, byte[] fileData)
{
        this.fileName = fileName;
        this.writerName = writerName;
        this.fileData = fileData;
    }
}
```

- **@Document**: Indicates that this class is a MongoDB document.
- **@Id**: Specifies the primary key field.

2. FileRepository

This interface extends `MongoRepository`, providing built-in CRUD methods.

```
package com.telusko.dao;

import org.springframework.data.mongodb.repository.MongoRepository;
import org.springframework.stereotype.Repository;
import com.telusko.entity.FileEntity;

import java.util.List;
import java.util.Optional;

@Repository
public interface FileRepository extends MongoRepository<FileEntity, String>
{

    List<FileEntity> findByWriterName(String writerName);

    Optional<FileEntity> findByFileName(String fileName);
}
```

- **@Repository**: Indicates that this interface is a Spring Data repository.

3. FileService

Contains the business logic for handling file operations.

```java
package com.telusko.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.web.multipart.MultipartFile;
import com.telusko.dao.FileRepository;
import com.telusko.entity.FileEntity;
import com.telusko.exception.DuplicateFileNameException;
import com.telusko.exception.FileNotFoundException;

import java.io.IOException;
import java.util.List;
import java.util.Optional;

@Service
public class FileService {

    @Autowired
    private FileRepository fileRepository;

    public FileEntity uploadFile(String fileName, String writerName,
MultipartFile file) throws IOException {
        Optional<FileEntity> existingFile =
fileRepository.findByFileName(fileName);
        if (existingFile.isPresent()) {
            throw new DuplicateFileNameException("A file with the name '" +
fileName + "' already exists.");
        }

        FileEntity fileEntity = new FileEntity(fileName, writerName,
file.getBytes());
        return fileRepository.save(fileEntity);
    }

    public FileEntity getFileById(String id) {
        return fileRepository.findById(id).orElseThrow(() -> new
FileNotFoundException("File not found with id " + id));
    }

    public List<FileEntity> getFilesByWriterName(String writerName) {
        return fileRepository.findByWriterName(writerName);
    }
```

```
    public FileEntity getFileByFileName(String fileName) {
        return fileRepository.findByFileName(fileName).orElseThrow(() ->
new FileNotFoundException("File not found with name " + fileName));
    }

    public FileEntity updateFile(String id, String fileName, String
writerName, MultipartFile file) throws IOException {
        FileEntity fileEntity = getFileById(id);

        Optional<FileEntity> existingFile =
fileRepository.findByFileName(fileName);
        if (existingFile.isPresent() &&
!existingFile.get().getId().equals(id)) {
            throw new DuplicateFileNameException("A file with the name '" +
fileName + "' already exists.");
        }

        fileEntity.setFileName(fileName);
        fileEntity.setWriterName(writerName);
        fileEntity.setFileData(file.getBytes());
        return fileRepository.save(fileEntity);
    }

    public void deleteFile(String id) {
        FileEntity fileEntity = getFileById(id);
        fileRepository.delete(fileEntity);
    }
}
```

- **@Service**: Marks this class as a service layer component.

4. FileController

Handles HTTP requests and interacts with the `FileService`.

```
package com.telusko.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpHeaders;
```

```java
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.multipart.MultipartFile;
import com.telusko.entity.FileEntity;
import com.telusko.service.FileService;

import java.io.IOException;
import java.util.List;

@RestController
@RequestMapping("/files")
public class FileController {

    @Autowired
    private FileService fileService;

    @PostMapping("/upload")
    public FileEntity uploadFile(@RequestParam("fileName") String fileName,
                                 @RequestParam("writerName") String
writerName,
                                 @RequestParam("file") MultipartFile file)
throws IOException {
        return fileService.uploadFile(fileName, writerName, file);
    }

    @GetMapping("/{id}")
    public ResponseEntity<byte[]> downloadFile(@PathVariable String id) {
        FileEntity fileEntity = fileService.getFileById(id);
        return ResponseEntity.ok()
                .header(HttpHeaders.CONTENT_DISPOSITION, "attachment;
filename=\"" + fileEntity.getFileName() + "\"")
                .contentType(MediaType.APPLICATION_OCTET_STREAM)
                .body(fileEntity.getFileData());
    }

    @GetMapping("/writer/{writerName}")
    public List<FileEntity> getFilesByWriterName(@PathVariable String
writerName) {
        return fileService.getFilesByWriterName(writerName);
    }

    @GetMapping("/name/{fileName}")
```

```
    public FileEntity getFileByFileName(@PathVariable String fileName) {
        return fileService.getFileByFileName(fileName);
    }

    @PutMapping("/{id}")
    public FileEntity updateFile(@PathVariable String id,
                                 @RequestParam("fileName") String fileName,
                                 @RequestParam("writerName") String
writerName,
                                 @RequestParam("file") MultipartFile file)
throws IOException {
        return fileService.updateFile(id, fileName, writerName, file);
    }

    @DeleteMapping("/{id}")
    public void deleteFile(@PathVariable String id) {
        fileService.deleteFile(id);
    }
}
```

- **@RestController**: Indicates that this class handles RESTful web service requests.
- **@RequestMapping("/files")**: Maps all requests starting with `/files` to this controller.

Explanation of Every thing in Service layer

1. Class Declaration

```
@RestController
@RequestMapping("/files")
public class FileController {
```

- **@RestController**: Indicates that this class is a RESTful controller, which means it will handle HTTP requests and responses in a RESTful manner. It combines `@Controller` and `@ResponseBody`, allowing us to return data directly in the response body.
- **@RequestMapping("/files")**: Specifies the base URL for all endpoints in this controller. All requests starting with `/files` will be routed to this controller.

2. Dependency Injection

```
@Autowired
private FileService fileService;
```

- **@Autowired**: This annotation is used for dependency injection, automatically injecting an instance of `FileService` into this controller. `FileService` contains the business logic for file operations.

3. Upload File Method

```
@PostMapping("/upload")
public FileEntity uploadFile(@RequestParam("fileName") String fileName,
                             @RequestParam("writerName") String writerName,
                             @RequestParam("file") MultipartFile file)
throws IOException {
    return fileService.uploadFile(fileName, writerName, file);
}
```

- **@PostMapping("/upload")**: Maps HTTP POST requests for uploading files to this method.
- **Parameters**:
    - `@RequestParam("fileName") String fileName`: Extracts the file name from the request. It should be included in the form data when uploading a file.
    - `@RequestParam("writerName") String writerName`: Extracts the writer's name from the request.
    - `@RequestParam("file") MultipartFile file`: Accepts the actual file as a `MultipartFile`, which is a Spring representation of an uploaded file.
- **Returns**: Calls the `uploadFile` method of `FileService`, passing the extracted values. The return type is `FileEntity`, which represents the uploaded file details in the database.

4. Download File Method

```
@GetMapping("/{id}")
public ResponseEntity<byte[]> downloadFile(@PathVariable String id) {
    FileEntity fileEntity = fileService.getFileById(id);
    return ResponseEntity.ok()
            .header(HttpHeaders.CONTENT_DISPOSITION, "attachment;
filename=\"" + fileEntity.getFileName() + "\"")
            .contentType(MediaType.APPLICATION_OCTET_STREAM)
            .body(fileEntity.getFileData());
```

```
}
```

- **@GetMapping("/{id}")**: Maps HTTP GET requests with a specific file ID to this method.
- **Parameters**:
  - `@PathVariable String id`: Extracts the file ID from the URL, allowing us to retrieve the specific file.
- **Method Logic**:
  - Retrieves the `FileEntity` from the `fileService` using the provided ID.
  - Creates a `ResponseEntity` object to send the file back as a response.
- **Returns**: A `ResponseEntity<byte[]>`, which contains:
  - **Header**: Sets the `Content-Disposition` header to prompt file download with the original filename.
  - **Content Type**: Specifies that the file is of type `APPLICATION_OCTET_STREAM`, which indicates binary data.
  - **Body**: Contains the byte data of the file retrieved from `fileEntity`.

5. Get Files by Writer Name

```
@GetMapping("/writer/{writerName}")
public List<FileEntity> getFilesByWriterName(@PathVariable String
writerName) {
    return fileService.getFilesByWriterName(writerName);
}
```

- **@GetMapping("/writer/{writerName}")**: Maps HTTP GET requests for fetching files by a specific writer's name.
- **Parameters**:
  - `@PathVariable String writerName`: Extracts the writer's name from the URL.
- **Returns**: A list of `FileEntity` objects that match the provided writer's name. This list is fetched from the `fileService`.

6. Get File by Name

```
@GetMapping("/name/{fileName}")
public FileEntity getFileByFileName(@PathVariable String fileName) {
    return fileService.getFileByFileName(fileName);
}
```

- **@GetMapping("/name/{fileName}")**: Maps HTTP GET requests for fetching a specific file by its name.
- **Parameters**:
    - `@PathVariable String fileName`: Extracts the file name from the URL.
- **Returns**: A single `FileEntity` object corresponding to the provided file name, retrieved from the `fileService`.

7. Update File

```
@PutMapping("/{id}")
public FileEntity updateFile(@PathVariable String id,
                             @RequestParam("fileName") String fileName,
                             @RequestParam("writerName") String writerName,
                             @RequestParam("file") MultipartFile file)
throws IOException {
    return fileService.updateFile(id, fileName, writerName, file);
}
```

- **@PutMapping("/{id}")**: Maps HTTP PUT requests for updating a file based on its ID.
- **Parameters**:
    - `@PathVariable String id`: Extracts the file ID from the URL.
    - `@RequestParam("fileName") String fileName`: New file name to update.
    - `@RequestParam("writerName") String writerName`: New writer's name to update.
    - `@RequestParam("file") MultipartFile file`: New file data to upload.
- **Returns**: Calls the `updateFile` method of the `fileService` to update the file details. Returns the updated `FileEntity`.

8. Delete File

```
@DeleteMapping("/{id}")
public void deleteFile(@PathVariable String id) {
    fileService.deleteFile(id);
}
```

- **@DeleteMapping("/{id}")**: Maps HTTP DELETE requests for deleting a file by its ID.
- **Parameters**:
    - `@PathVariable String id`: Extracts the file ID from the URL.

- **Returns**: Calls the `deleteFile` method in `fileService` to perform the deletion. This method returns void, indicating no content is sent back in the response.

Summary of Objects Used

1. **FileService**: A service class that contains business logic for handling file operations like upload, retrieval, update, and deletion.
2. **FileEntity**: A model class representing the file data in the MongoDB database. It includes properties like `id`, `fileName`, `writerName`, and `fileData`.
3. **MultipartFile**: An interface provided by Spring that represents an uploaded file in a multipart request.
4. **ResponseEntity**: A Spring class that represents the entire HTTP response, including status code, headers, and body.

5.Exception Handling

Custom Exceptions

FileNotFoundException

```
package com.telusko.exception;

public class FileNotFoundException extends RuntimeException {
    public FileNotFoundException(String message) {
        super(message);
    }
}
```

DuplicateFileNameException

```
package com.telusko.exception;
public class DuplicateFileNameException extends RuntimeException {
   public DuplicateFileNameException(String message) {
       super(message);
   }
}
```

Global Exception Handler

We will create a global exception handler to manage exceptions thrown by our application.

```
package com.telusko.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;

@ControllerAdvice
public class GlobalExceptionHandler {

    @ExceptionHandler(DuplicateFileNameException.class)
    public ResponseEntity<String>
handleDuplicateFileNameException(DuplicateFileNameException ex) {
        return
ResponseEntity.status(HttpStatus.CONFLICT).body(ex.getMessage());
    }

    @ExceptionHandler(FileNotFoundException.class)
    public ResponseEntity<String>
handleFileNotFoundException(FileNotFoundException ex) {
        return
ResponseEntity.status(HttpStatus.NOT_FOUND).body(ex.getMessage());
    }

    @ExceptionHandler(Exception.class)
    public ResponseEntity<String> handleGeneralException(Exception ex) {
        return
ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("An unexpected
error occurred: " + ex.getMessage());
    }
}
```

- **@ControllerAdvice**: Indicates that this class provides global exception handling.
- **@ExceptionHandler**: Specifies the type of exception that this method will handle.

ErrorResponse

```
package com.telusko.exception;
import lombok.AllArgsConstructor;
import lombok.Data;
```

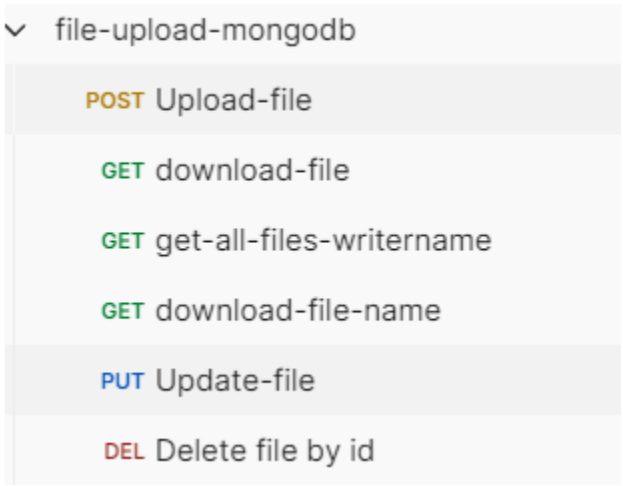
```java
import lombok.NoArgsConstructor;
@Data
@NoArgsConstructor
@AllArgsConstructor
public class ErrorResponse {
   private String message;
   private int statusCode;
}
```

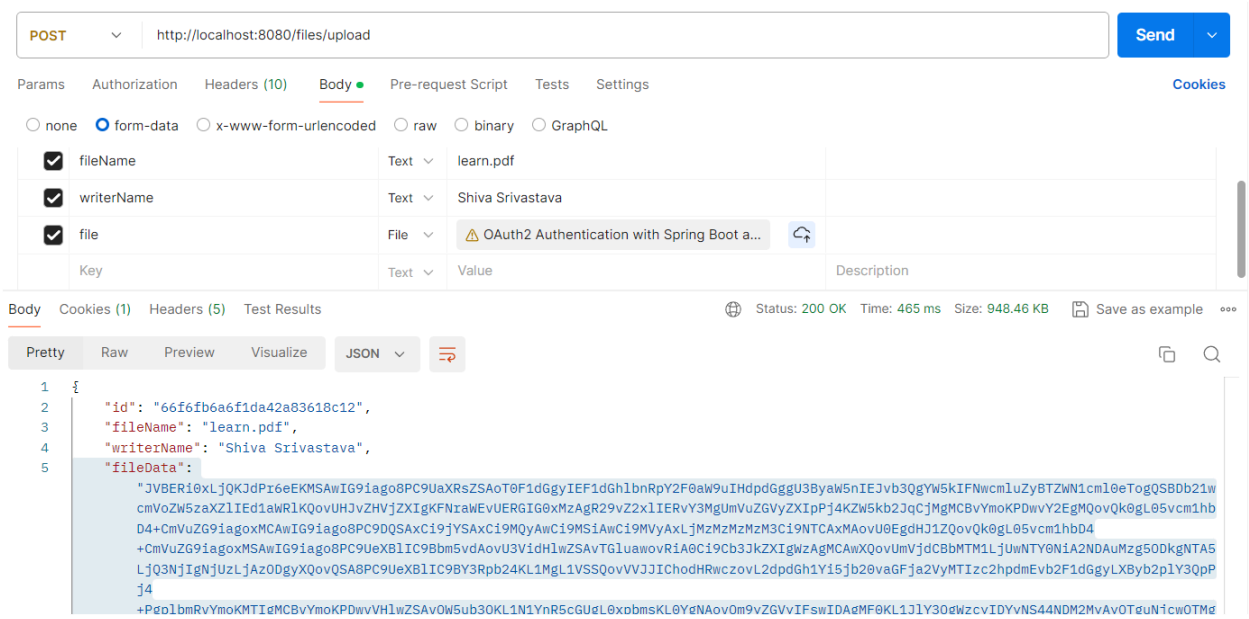
Testing the Application with Postman

Endpoints Overview

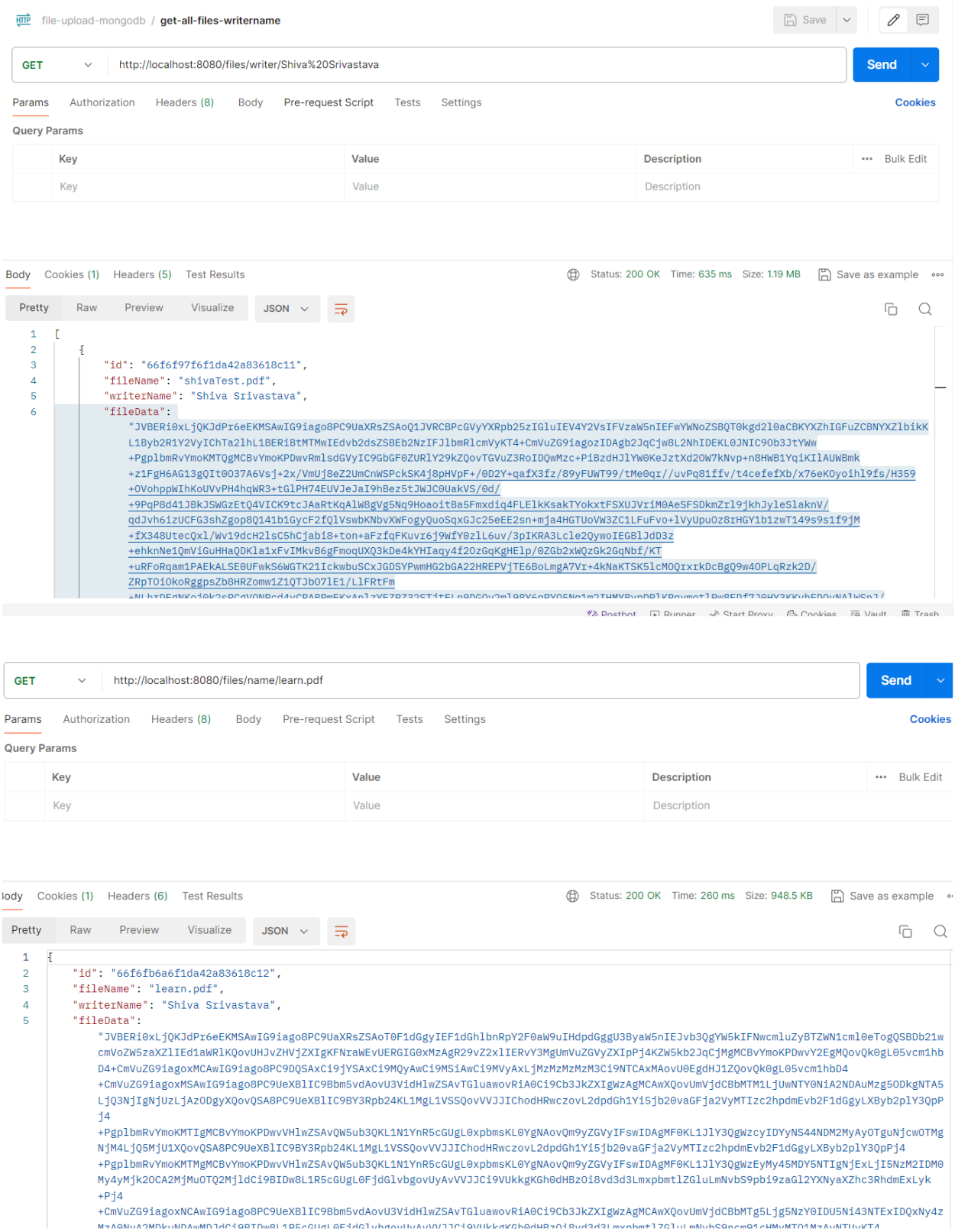
Here are the endpoints we'll be testing with Postman:

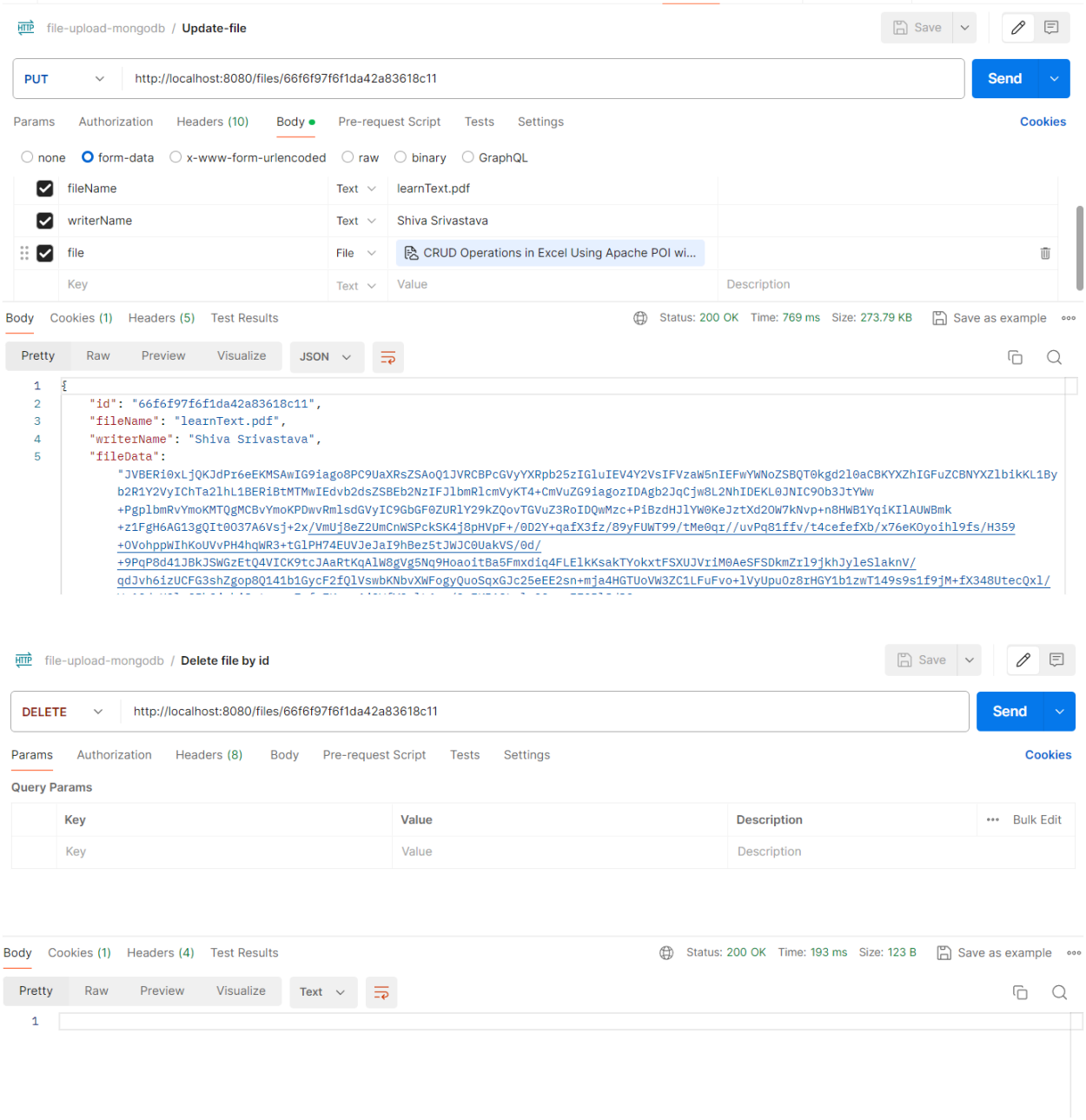
1. **Upload File**: `POST /files/upload`
2. **Download File**: `GET /files/{id}`
3. **Get Files by Writer**: `GET /files/writer/{writerName}`
4. **Get File by Name**: `GET /files/name/{fileName}`
5. **Update File**: `PUT /files/{id}`
6. **Delete File**: `DELETE /files/{id}`





Note: If you fire the same request in browser, then the file gets downloaded automatically.


file-upload-mongodb / get-all-files-writername
Save
GET
http://localhost:8080/files/writer/Shiva%20Srivastava
Send
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Key
Value
Description
Body
Cookies (1)
Headers (5)
Test Results
Status: 200 OK
Time: 635 ms
Size: 1.19 MB
Save as example
Pretty
Raw
Preview
Visualize
JSON
[
{
"id": "66f6f97f6f1da42a83618c11",
"fileName": "shivaTest.pdf",
"writerName": "Shiva Srivastava",
"fileData":
"JVBERi0xLjQKJdPr6eEKMSAwIG9iago8PC9UaXRsZSAoQ1JVRCBPcGVyYXRpb25zIGluIEV4Y2VsIFVzaW5nIEFwYWNoZSBQT0kgd2l0aCBKYXZhIGFuZCBNYXZlbikK
L1Byb2R1Y2VyIChTa2lhL1BERiBtMTMwIEdvb2dsZSBEb2NzIFJlbmRlcmVyKT4+CmVuZG9iagozIDAgb2JqCjw8L2NhIDEKL0JNIC9Ob3JtYWw
+PgplbmRvYmoKMTQgMCBvYmoKPDwvRmlsdGVyIC9GbGF0ZURlY29kZQovTGVuZ3RoIDQwMzc+PiBzdHJlYW0KeJztXd2OW7kNvp+n8HWB1YqiKIlAUWBmk
+z1FgH6AG13gQIt0O37A6Vsj+2x/VmUj8eZ2UmCnWSPckSK4j8pHVpF+/0D2Y+qafX3fz/89yFUWT99/tMe0qr//uvPq81ffv/t4cefefXb/x76eKOyoihl9fs/H359
+OVohppWIhKoUVvPH4hqWR3+tGlPH74EUVJeJaI9hBez5tJWJC0UakVS/0d/
+9PqP8d41JBkJSWGzEtQ4VICK9tcJAaRtKqAlW8gVg5Nq9HoaoitBa5Fmxdiq4FLElkSakTYokxtFSXUJVriM0AeSFSDkmZrl9jkhJyleSlaknV/
qdJvh6izUCFG3shZgop8Q141b1GycF2fQlVswbKNbvXWFogyQuoSqxGJc25eEE2sn+mja4HGTUoVW3ZC1LFuFvo+lVyUpuOz8rHGY1b1zwT149s9s1f9jM
+fX348UtecQxl/Wv19dcH2lsC5hCjabi8+ton+aFzfqFKuvr6j9WfY0zlL6uv/3pIKRA3Lcle2QywoIEGBlJdD3z
+ehknNe1QmViGuHHaQDKla1xFvIMkvB6gFmoqUXQ3kDe4kYHIaqy4f2OzGqKgHElp/0ZGb2xWQzGk2GqNbf/KT
+uRFoRqam1PAEkALSE0UFwkS6WGTK21IckwbuSCxJGDSYPwmHG2bGA22HREPVjTE6BoLmgA7Vr+4kNaKTSK5lcMOQrxrkDcBgQ9w4OPLqRzk2D/
ZRpTOiOkoRggpsZb8HRZomw1Z1QTJbO7lE1/LlFRtFm
Postbot
Runner
Start Proxy
Cookies
Vault
Trash
GET
http://localhost:8080/files/name/learn.pdf
Send
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Key
Value
Description
lody
Cookies (1)
Headers (6)
Test Results
Status: 200 OK
Time: 260 ms
Size: 948.5 KB
Save as example
Pretty
Raw
Preview
Visualize
JSON
{
"id": "66f6fb6a6f1da42a83618c12",
"fileName": "learn.pdf",
"writerName": "Shiva Srivastava",
"fileData":
"JVBERi0xLjQKJdPr6eEKMSAwIG9iago8PC9UaXRsZSAoT0F1dGhlbnRpY2F0aW9uIHdpdGggU3ByaW5nIEJvb3QgYW5kIFNwcmluZyBTZWN1cml0eTogQSBDb21w
cmVoZW5zaXZlIEd1aWRlKQovUHJvZHVjZXIgKFNraWEvUERGIG0xMzAgR29vZ2xlIERvY3MgUmVuZGVyZXIpPj4KZW5kb2JqCjMgMCBvYmoKPDwvY2EgMQovQk0gL05vcm1hb
D4+CmVuZG9iagoxMCAwIG9iago8PC9DQSAxCi9jYSAxCi9MSiAwCi9MSiAwCi9MVyAxLjMzMzMzMzM3Ci9NTCAxMAovU0EgdHJ1ZQovQk0gL05vcm1hbD4
+CmVuZG9iagoxMSAwIG9iago8PC9UeXBlIC9Bbm5vdAovU3VidHlwZSAvTGluawovRiA0Ci9Cb3JkZXIgWzAgMCAwXQovUmVjdCBbMTM1LjUwNTY0NiA2NDAuMzg5ODkgNTA5
LjQ3NjIgNjUzLjAzODgyXQovQSA8PC9UeXBlIC9BY3Rpb24KL1MgL1VSSQovVVJJIChodHRwczovL2dpdGh1Yi5jb20vaGFja2VyMTIzc2hpdmEvb2F1dGgyLXByb2plY3QpP
j4
+PgplbmRvYmoKMTIgMCBvYmoKPDwvVHlwZSAvQW5ub3QKL1N1YnR5cGUgL0xpbmsKL0YgNAovQm9yZGVyIFswIDAgMF0KL1JlY3QgWzcyIDYyNS44NDM2MyAyOTguNjcwOTMg
NjM4LjQ5MjU1XQovQSA8PC9UeXBlIC9BY3Rpb24KL1MgL1VSSQovVVJJIChodHRwczovL2dpdGh1Yi5jb20vaGFja2VyMTIzc2hpdmEvb2F1dGgyLXByb2plY3QpPj4
+PgplbmRvYmoKMTMgMCBvYmoKPDwvVHlwZSAvQW5ub3QKL1N1YnR5cGUgL0xpbmsKL0YgNAovQm9yZGVyIFswIDAgMF0KL1JlY3QgWzcyIDYyNS44NDY5NTIgNjExLjI5NzM2IDM0
My4yMjk2OCA2MjMuOTQ2MjldCi9BIDw8L1R5cGUgL0FjdGlvbgovUyAvVVJJCi9VUkkgKGh0dHBzOi8vd3d3Lmxpbmtlay5nLmNvbS9pbi9zaGl2YXNyaXZhc3RhdmExLyk
+Pj4
+CmVuZG9iagoxNCAwIG9iago8PC9UeXBlIC9Bbm5vdAovU3VidHlwZSAvTGluawovRiA0Ci9Cb3JkZXIgWzAgMCAwXQovUmVjdCBbMTg5Ljg5NzY0IDU5Ni43NTExIDQxNy4z
MzA0NyA2MDkuNDAwMDJdCi9BIDw8L1R5cGUgL0FjdGlvbgovUyAvVVJJCi9VUkkgKGh0dHBzOi8vd3d3Lmxpbmtlay5nLmNvbS9pbi9zaGl2YXNyaXZhc3RhdmExLyk



Conclusion

In this blog, we have created a complete Spring Boot application that performs CRUD operations for file management with MongoDB. We explored the project structure, discussed the core components like the `FileEntity`, `FileRepository`, `FileService`, and `FileController`, and implemented exception handling to manage errors gracefully.

This application provides a robust foundation for managing files and can be extended with more features, such as file search capabilities, pagination for file listings, or integration with front-end technologies for better user experience.