

QDB Assessment

File-Management-API

Testing Document

This is just a document, illustrating simple steps to run and test the functionality of my File-Management API, A Spring Boot REST API application.

We will be using **Postman** to achieve this.

Dev & Build Tools used:

Java 11

Apache Maven 3.6.3

H2 In memory database

Github link to my application

<https://github.com/Arshadee/file-management.git>

Clone the application

<https://github.com/Arshadee/file-management.git>

Build and Run Scripts:

There are 2 scripts run.sh and buildRun.sh
found in the project folder **file-management**

Build / Compile and Run the application using my scripts

In the main project folder **file-management**, run the following: on macos / unix operating system

```
chmod +rx buildRun.sh
./buildRun.sh
```

Run the application only using my scripts

In the main project folder, **file-management**, run the following: on mac-os / unix operating systems

```
chmod +rx run.sh
./run.sh
```

Build / Compile the application using maven

In command-Line / terminal navigate to the project folder, **file-management**,
build the application by running : **mvn clean install**

Run the application, in command-Line / terminal navigate to the project folder, **file-management**,
and run : **java -jar target/file-management-o.o.1-SNAPSHOT.jar**

The application will start running at
<http://localhost:8080>

I have not included screen shots of the database states but it may be view at anytime at the following url:
<http://localhost:8080/h2>

English Preferences Tools Help

Login

Saved Settings: Generic H2 (Embedded)

Setting Name: Generic H2 (Embedded) Save Remove

Driver Class: org.h2.Driver

JDBC URL: jdbc:h2:mem:usrdocdb

User Name: sa

Password:

Connect Test Connection

Detail as displayed in the database login page above:

Driver Class: orge.h2.Driver

JDBC URL: jdbc:h2:mem:usrdocdb

User Name: sa
Password: {blank}

All documents uploaded are store in the following folder in the main project folder
/file-management/uploads/{username} (username user specifies)

Below I have just outline a basic Flow through the solution I presented. I have also demo'd a few error scenarios however I have not displayed every possible scenario I've catered for as this would be too much for this document. The idea is to just indicate my thought process and how I have addressed all possible scenarios.

Step 1

Create / update a User and upload a document: (using Postman)

Using our REST End Point: (Note only pdf docs are allowed to be uploaded)

Method: POST

URL: <http://localhost:8080/upload/{username}>

Path Variables: Username (I used my name as an example)

Query Parameters: None

Request body: Form data contains just a file object Key = File Type Multipart File

You may add more files this way (with separate calls).

A user and related document(s) are added to the database and a folder is created with th username on the server and files are uploaded.

The screenshot shows the Postman interface for a POST request. The URL is `http://localhost:8080/upload/arshad`. The request body is set to 'form-data' and contains a single file entry with the key 'file' and the value 'Scan0002.pdf'. The response is a 201 Created status with a response time of 32 ms and a body size of 227 B. The response body is displayed in JSON format, showing a success message: `"message": "Uploaded the file successfully: Scan0002.pdf"`.

KEY	VALUE	DESCRIPTION
file	Scan0002.pdf	
Key	Value	Description

```
{  "message": "Uploaded the file successfully: Scan0002.pdf"}
```

Step 2

Getting a file by username and document name

Method: GET

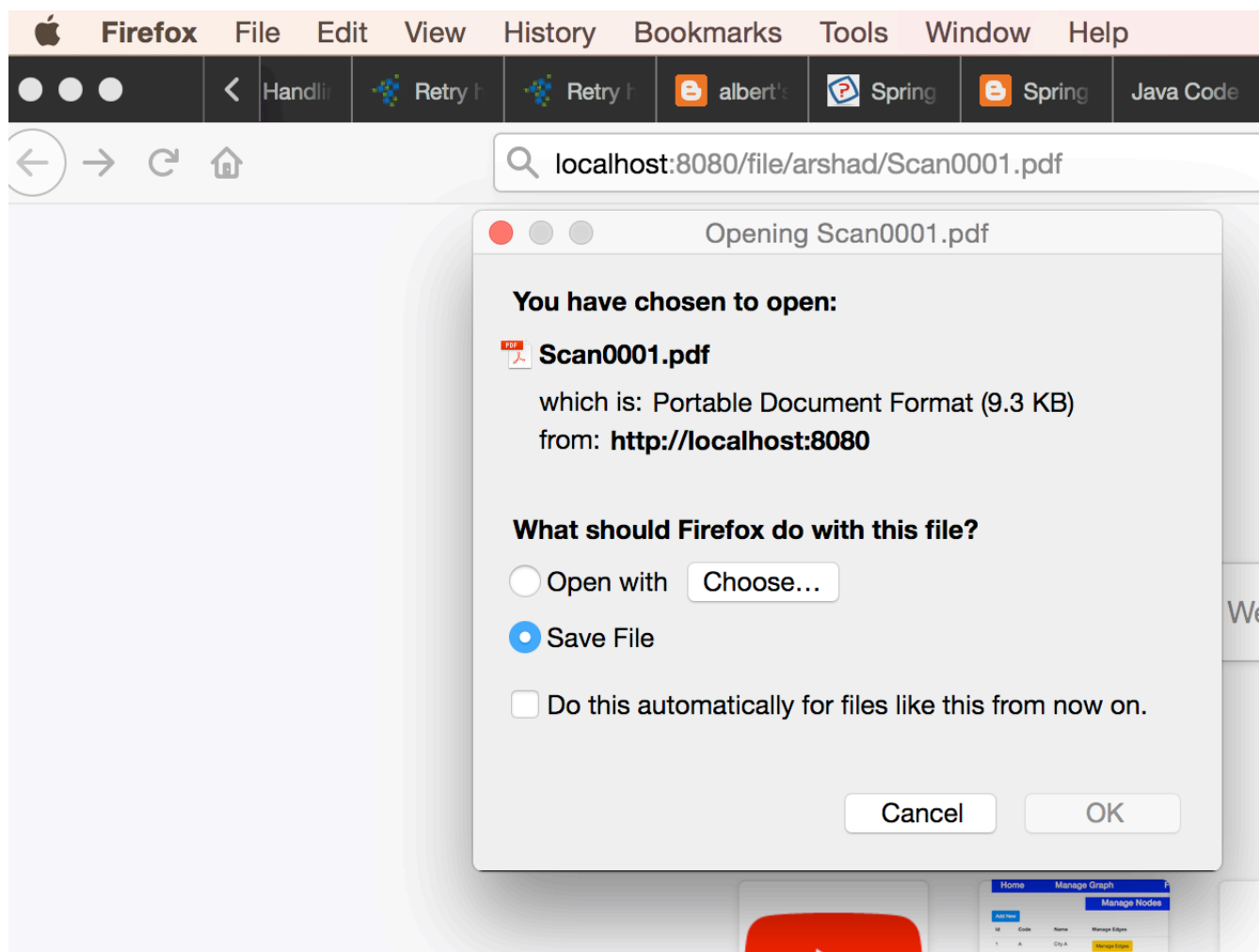
Url: <http://localhost:8080/file/{username}/{documentName}>

Path Variables: username and documentName (documentName is the full file name)

Query Parameters: None

Request body: None

This call downloads the file from localhost server, I would suggest using your fave browser for this one.



Step 4

Getting a List of all files / documents uploaded by a User.

Method: GET

Url: <http://localhost:8080/files/{username}>

Path Variable: username

Query Parameters: None

Request body: None

The screenshot shows a REST client interface with a GET request to `http://localhost:8080/files/arshad`. The response is a JSON array of three objects, each representing a file with its name and URL.

KEY	VALUE	DESCRIPTION
Key	Value	Description

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (5) Test Results 200 OK 35 ms 511 B Save Response

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "name": "document(1).pdf",
4     "url": "http://localhost:8080/file/arshad/document(1).pdf"
5   },
6   {
7     "name": "Scan0001.pdf",
8     "url": "http://localhost:8080/file/arshad/Scan0001.pdf"
9   },
10  {
11    "name": "Scan0002.pdf",
12    "url": "http://localhost:8080/file/arshad/Scan0002.pdf"
13  },
14 ]
```

Step 5

Deleting a file need to specify username and filename:

Method: DELETE

Url: <http://localhost:8080/upload/{username}/{fileName}>

Path Variable: username and fileName

Query Parameters: None

Request body: None

The entry is removed from the database and file is removed from the database.

Step 5

Deleting all file by a user

Method: DELETE

Url: <http://localhost:8080/upload/{username}>

Path Variable: username and fileName

Query Parameters: None

Request body: None

The document entries are removed from the database and user's folder and files are removed from the server.

DELETE <http://localhost:8080/upload/arshad> Send Save

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies C

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk E
Key	Value	Description		

Body Cookies Headers (5) Test Results 200 OK 21 ms 170 B Save Response

Pretty Raw Preview Visualize Text 1 deleted

Step 6

Uploading and incorrect file type

Method: POST

URL: <http://localhost:8080/upload/{username}>

Path Variables: Username (I used my name as an example)

Query Parameters: None

Request body : Form data contains just a file object Key = File Type Multipart File

POST http://localhost:8080/upload/arshad Send Save

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings Cookies

☐ none
 ☒ form-data
 ☐ x-www-form-urlencoded
 ☐ raw
 ☐ binary
 ☐ GraphQL

	KEY	VALUE	DESCRIPTION	...	Bu
<input checked="" type="checkbox"/>	file	test.rtf X			

body Cookies Headers (4) Test Results 400 Bad Request 40 ms 27.6 KB Save Resp

Pretty Raw Preview Visualize JSON ↔

```

1195         "lineNumber": 834,
1196         "className": "java.lang.Thread",
1197         "nativeMethod": false
1198     },
1199 ],
1200     "message": "Incorrect format or null must be a pdf file!",
1201     "suppressed": [],
1202     "localizedMessage": "Incorrect format or null must be a pdf file!"
1203 },
1204     "httpStatus": "BAD_REQUEST",
1205     "timeStamp": "2020-08-11T20:16:14.666419Z",
1206     "suppressed": [],
1207     "localizedMessage": "Incorrect format or null must be a pdf file!"
1208 }
  
```

The validation throws a custom exception that returns a Http Status 400 BAD REQUEST and a custom message “Incorrect format or null must be a pdf file”

The next step covers POST and Comments

Step 7

Adding / creating a post against a document and user (document owner)

Method: POST

URL: <http://localhost:8080/username/{username}/document/{documentName}/posts>

Path Variables: Path Variables: username and documentName (documentName is the full file name)

Query Parameters: None

Request body : (raw) Json containing one element just the body of the post.

The body could be too long to make as a path variable or query variable

We received the full Json POST in the response body and Http Status 201 CREATED
Note: if we put in User or Document that did not exist we would get back an Response with and custome error message “Related sntiy/ entity does not exist”handled by a custom exception with the Http Status 400 **BAD REQUEST**

Next I would try duplicating this post by resubmitting (no duplicates allowed) our validation kicks in and throws a custom exception return and error message “” with Http Status 400 BAD REQUEST

The screenshot shows a REST client interface. At the top, the method is set to POST and the URL is `http://localhost:8080/username/arshad/document/Scan0001.pdf/posts`. The 'Body' tab is selected, showing a JSON payload: `{ "body": "this is a test post" }`. Below the request, the response is displayed. The status is '400 Bad Request' with a response time of '27 ms' and a size of '29.68 KB'. The response body is a JSON object:

```
{  "nativeMethod": false,  },  {    "message": "Unique Key violation - Possibly adding a record that exists",    "suppressed": [],    "localizedMessage": "Unique Key violation - Possibly adding a record that exists"  },  {    "httpStatus": "BAD_REQUEST",    "timeStamp": "2020-08-11T20:56:01.48332Z",    "suppressed": [],    "localizedMessage": "Unique Key violation - Possibly adding a record that exists"  } }
```

Step 8

Getting a Post against a document using the document and username.

Method: GET

URL: <http://localhost:8080/username/{username}/document/{documentName}/posts>

Path Variables: Path Variables: username and documentName (documentName is the full file name)

Query Parameters: None
Request body : None

The screenshot shows a REST client interface. At the top, the method is set to GET and the URL is `http://localhost:8080/username/arshad/document/Scan0001.pdf/posts`. Below the URL bar, there are tabs for Params, Authorization, Headers (7), Body, Pre-request Script, Tests, and Settings. The Params tab is active, showing a table with columns KEY, VALUE, and DESCRIPTION. The table is empty. Below the Params tab, there is a section for Query Params, also empty. At the bottom, the Body tab is active, showing the response body in JSON format. The response is a JSON object with the following structure: `{ "id": 1, "username": "arshad", "title": "Scan0001.pdf", "body": "this is a test post" }`. The status bar at the bottom right shows a 200 OK status, 13 ms response time, and 333 B response size.

KEY	VALUE	DESCRIPTION
-----	-------	-------------

```
{
  "id": 1,
  "username": "arshad",
  "title": "Scan0001.pdf",
  "body": "this is a test post"
}
```

Step 9

Posting Comment(s) against Post.

Just to recap the relationship between Post and Comment is one to many Post 1 → M Comments

Method: POST

URL: <http://localhost:8080/document/posts/{postId}/comments>

Path Variables: Path Variables: postId (Id of the post to be commented on)

Query Parameters: None

Request body : (Raw) JSON containing 3 elements name (of the commenter), email (of the commenter) and the body of the comment

The screenshot displays a REST client interface. At the top, a POST request is configured for the URL `localhost:8080/document/posts/1/comments`. The 'Body' tab is selected, showing a JSON payload: `{ "name": "max", "email": "maxt@test.com", "body": "test comment3" }`. Below the request, the 'Response' tab is active, showing a successful status of 201 Created. The response body is a JSON object: `{ "postId": 1, "name": "max", "email": "maxt@test.com", "body": "test comment3" }`. The interface includes tabs for Params, Authorization, Headers, Body, Pre-request Script, Tests, and Settings. The response status bar indicates '201 Created', '13 ms', and '330 B'.

The result Http Status 201 CREATED with the the full comment in the response body is returned.

Note the body field is marked as a unique key in the database therefore duplicating it would result in a Http Status 400 BAD REQUEST with a custom message “Unique Key violation – possibly adding a record that exist”.

I have used Java Bean Validation on the entities I pass as request body, for example don’t allow nulls and validate email addressed etc these are also handled by custom exception resulting in a 400 BAD REQUEST.

In the example I tried submitting a comment with an invalid email address “maxttest.com”

POST localhost:8080/document/posts/1/comments Send Save

Params Authorization Headers (9) Body ● Pre-request Script Tests Settings Cookies

● none ● form-data ● x-www-form-urlencoded ● raw ● binary ● GraphQL JSON ▼ Be:

```
1 {
2   "name" : "max",
3   "email" : "maxttest.com",
4   "body" : "test comment5"
5 }
```

Body Cookies Headers (7) Test Results 400 Bad Request 37 ms 6.24 KB Save Respo

Pretty Raw Preview Visualize JSON ▼

```
1 {
2   "timestamp": "2020-08-11T21:25:31.619+00:00",
3   "status": 400,
4   "error": "Bad Request",
5   "trace": "org.springframework.web.bind.MethodArgumentNotValidException: Validation failed for
```

Step 10

Getting all Comments for a specific Post

Method: GET

URL: <http://localhost:8080/document/posts/{postId}/comments>

Path Variables: Path Variables: postId (Id of the post to be commented on)

Query Parameters: None

Request body : None

GET localhost:8080/document/posts/1/comments Send Save

Params Authorization Headers (7) **Body** Pre-request Script Tests Settings Cookies Cod

none form-data x-www-form-urlencoded raw binary GraphQL

This request does not have a body

Body Cookies Headers (8) Test Results 200 OK 9 ms 649 B Save Response

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "id": 1,
4     "postId": 1,
5     "name": "max",
6     "email": "max@test.com",
7     "body": "test comment5"
8   },
9   {
10    "id": 2,
11    "postId": 1,
12    "name": "max",
13    "email": "max@test.com",
14    "body": "test comment1"
15  },
16  {
17    "id": 3,
18    "postId": 1,
19    "name": "max",
20    "email": "max@test.com",
21    "body": "test comment2"
```

Step 11

Getting a specific Comment

Method: GET

URL: <http://localhost:8080/document/posts/{postId}/comment/{commentId}>

Path Variables: Path Variables: **postId** (Id of the post to be commented on) and the **commentId** for the Comment we want to retrieve

Query Parameters: None

Request body : None

GET

localhost:8080/document/posts/1/comment/2

Send

Save

Params

Authorization

Headers (7)

Body

Pre-request Script

Tests

Settings

Cookies

Cod

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Body

Cookies

Headers (8)

Test Results

200 OK 15 ms 331 B

Save Response

Pretty

Raw

Preview

Visualize

JSON

```
1 {
2   "id": 2,
3   "postId": 1,
4   "name": "max",
5   "email": "max@test.com",
6   "body": "test comment1"
7 }
```