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

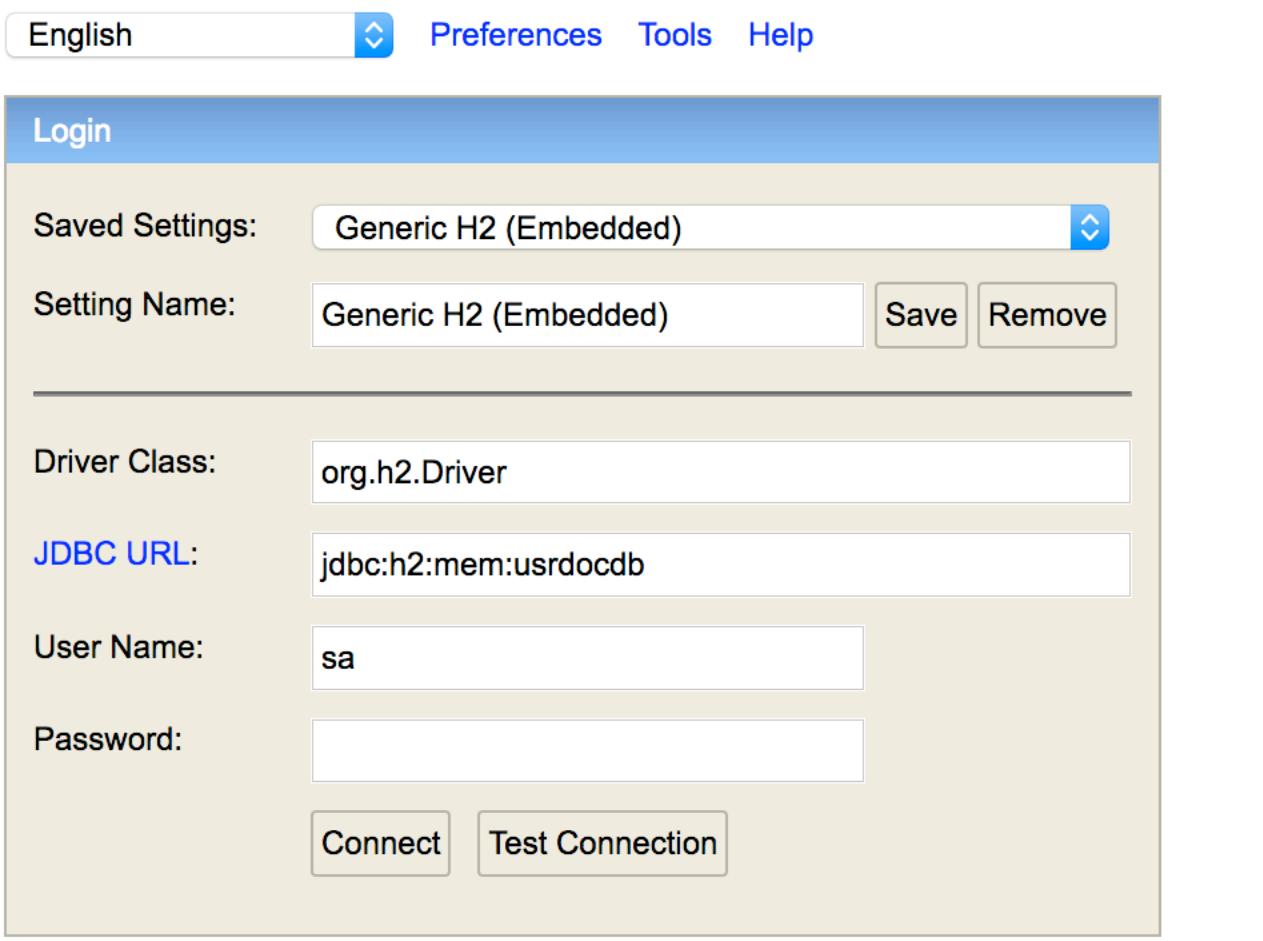
```
chmod +rx run.sh
./run.sh
```

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-0.0.1-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



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}*

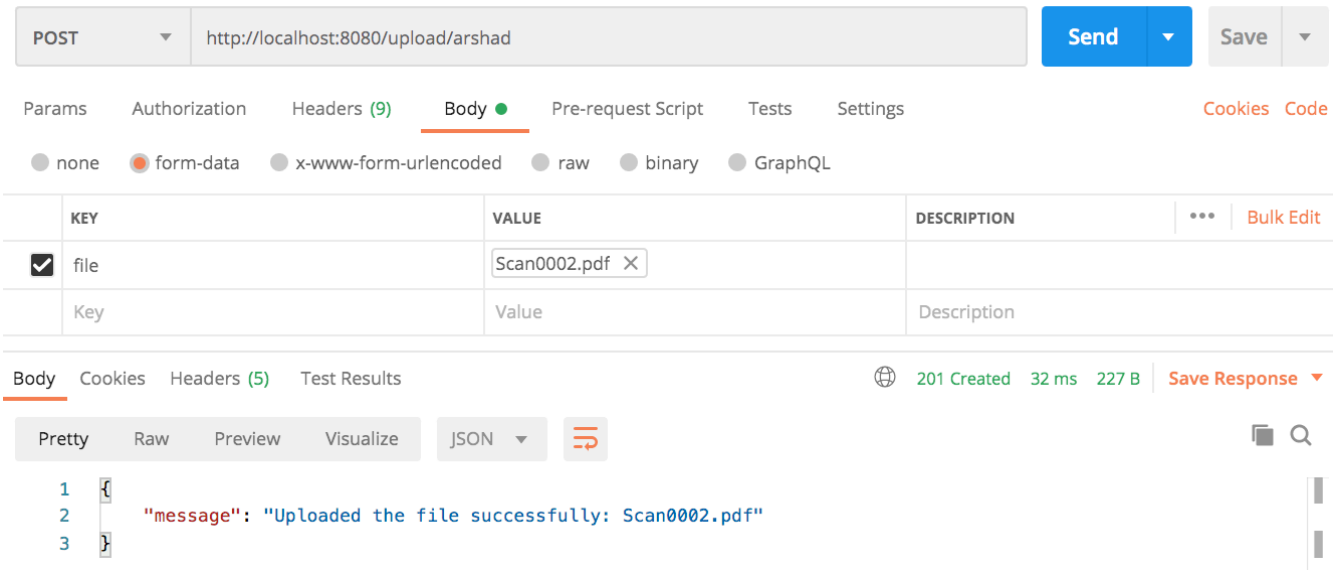
All documents uploaded are store in the following folder in the main project folder
/file-management/uploads/*{username}* *(username user specifies)*

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).

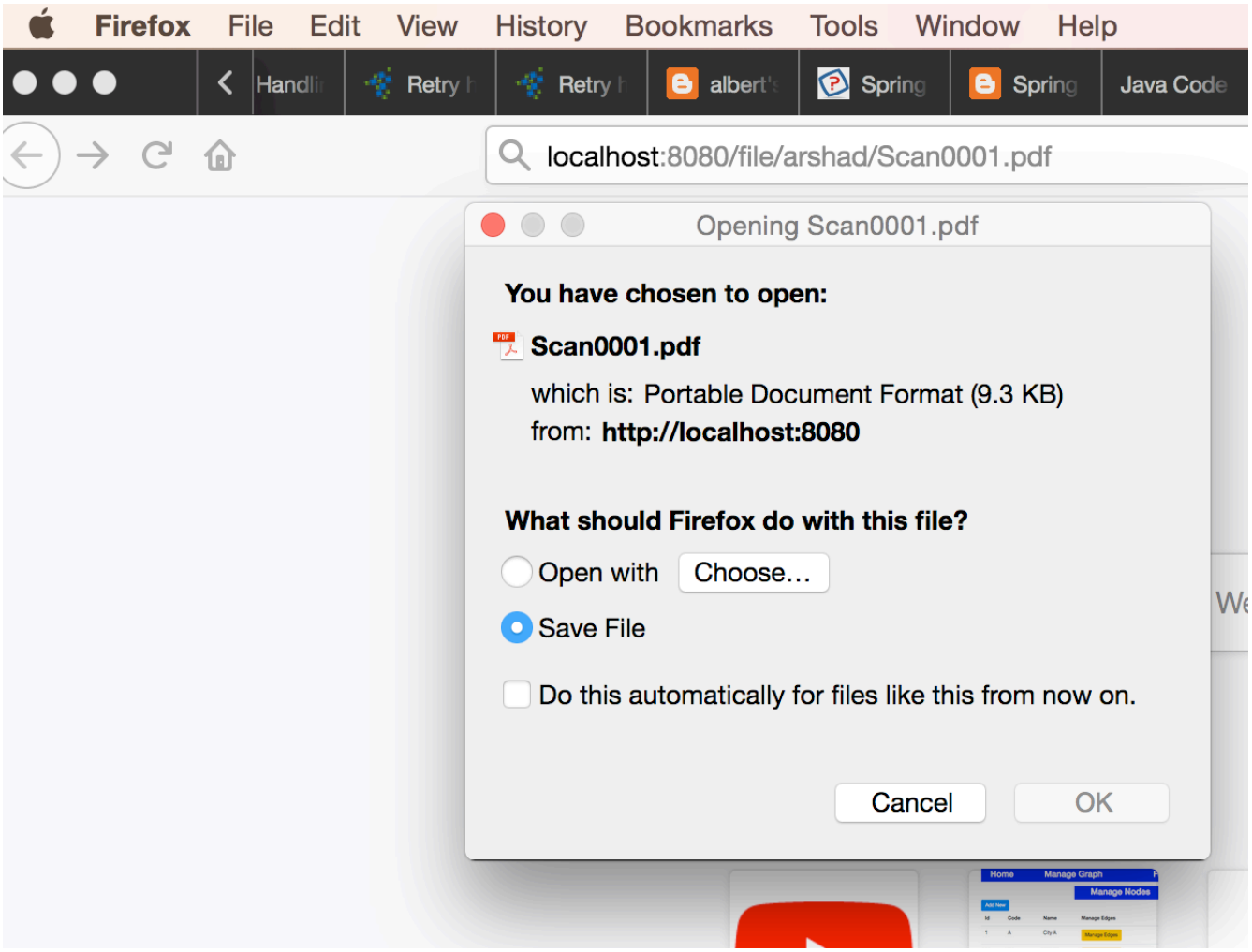
A user and related document(s) are added to the database and a folder is created with th username on the server and files are uploaded.



Step 2
Getting a file by username and document name

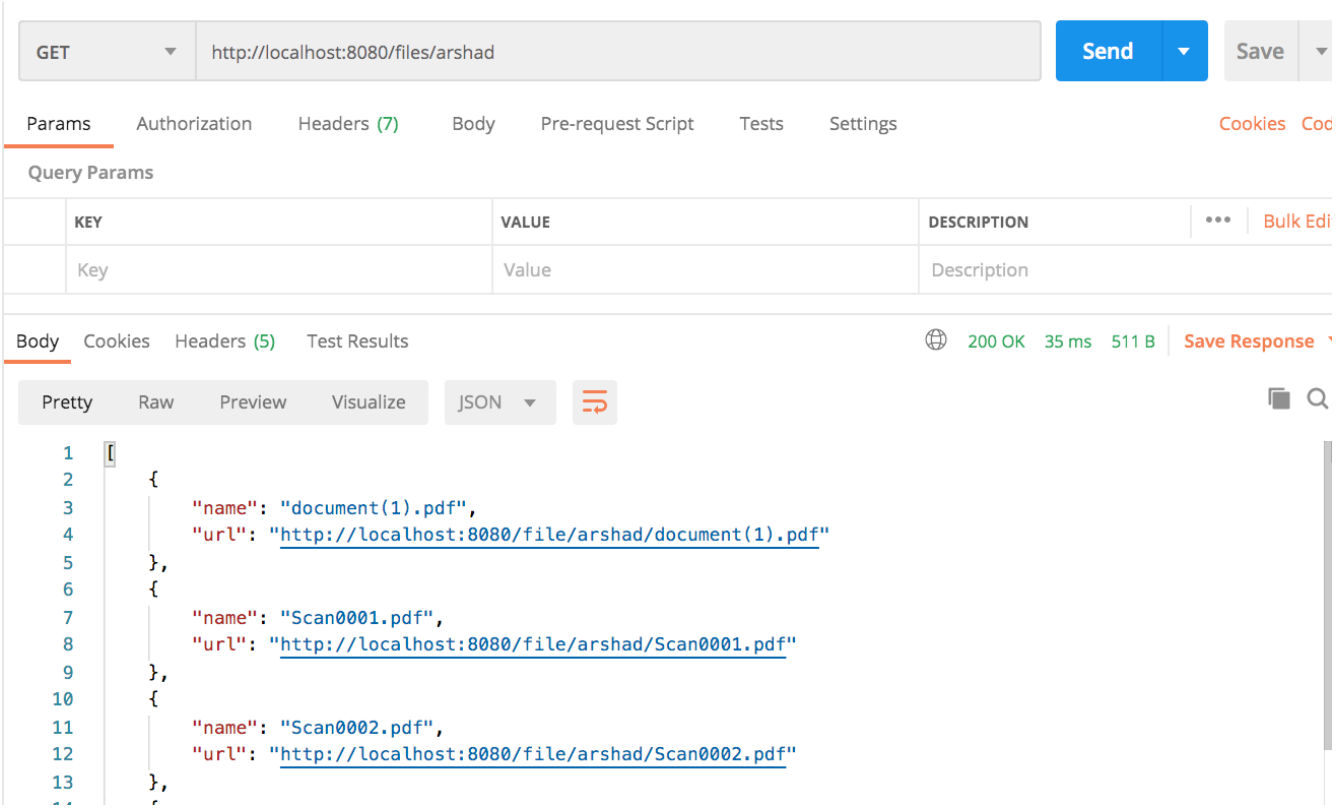
Method: GET
Url: http://localhost:8080/file/{username}/{documentName}
Path Variables: username and documentName (documentName is the full file name)
Query Parameters: None
Request body: None

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



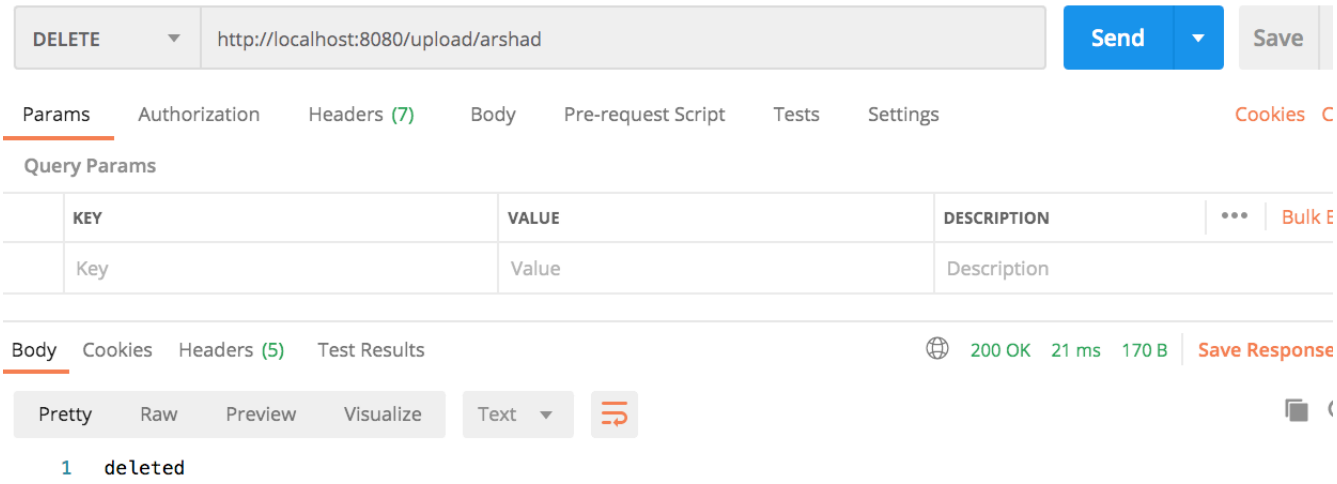
Step 5
Deleting a file need to specify username and filename:

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.

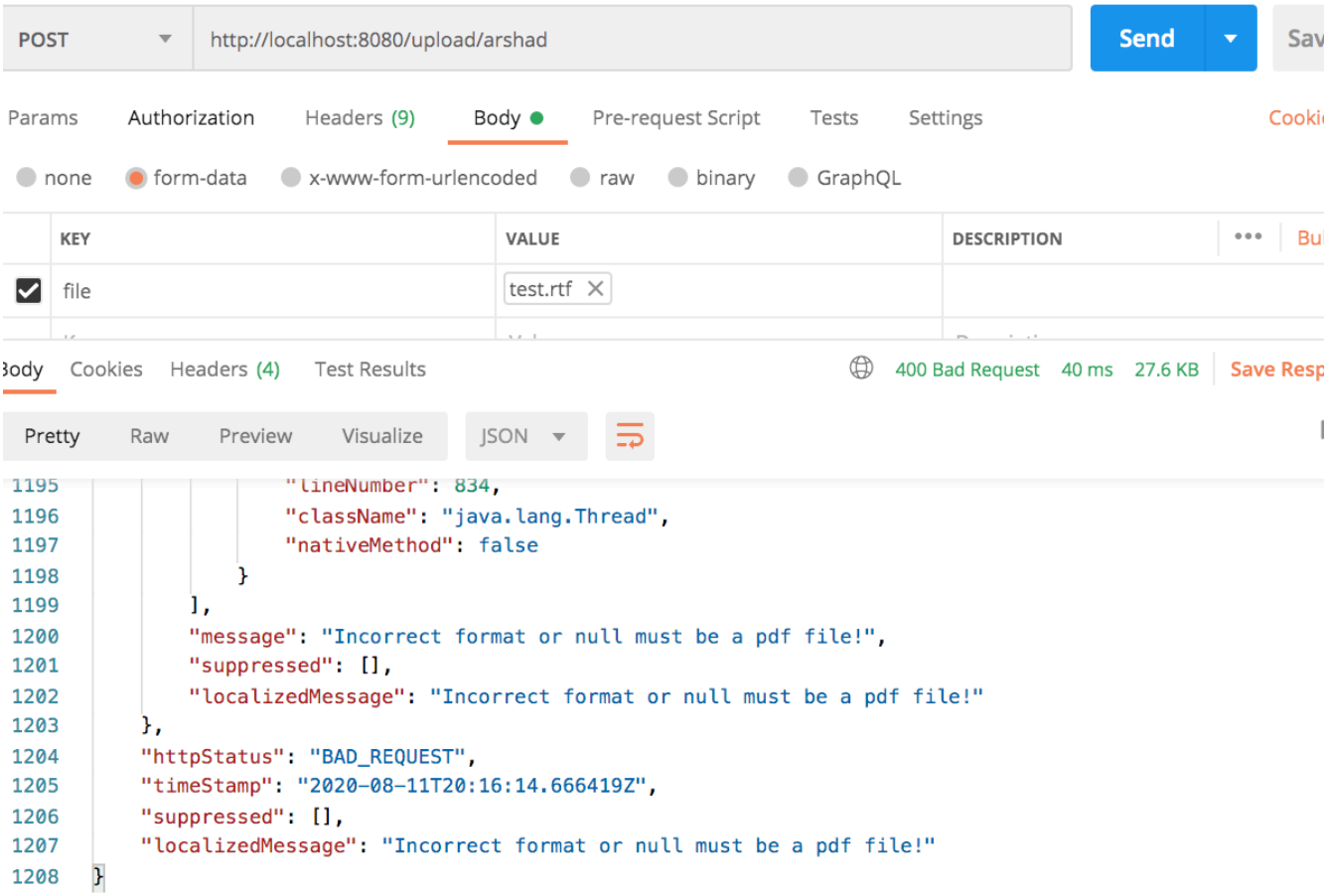
Step 5
Deleting all file by a user

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.



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



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

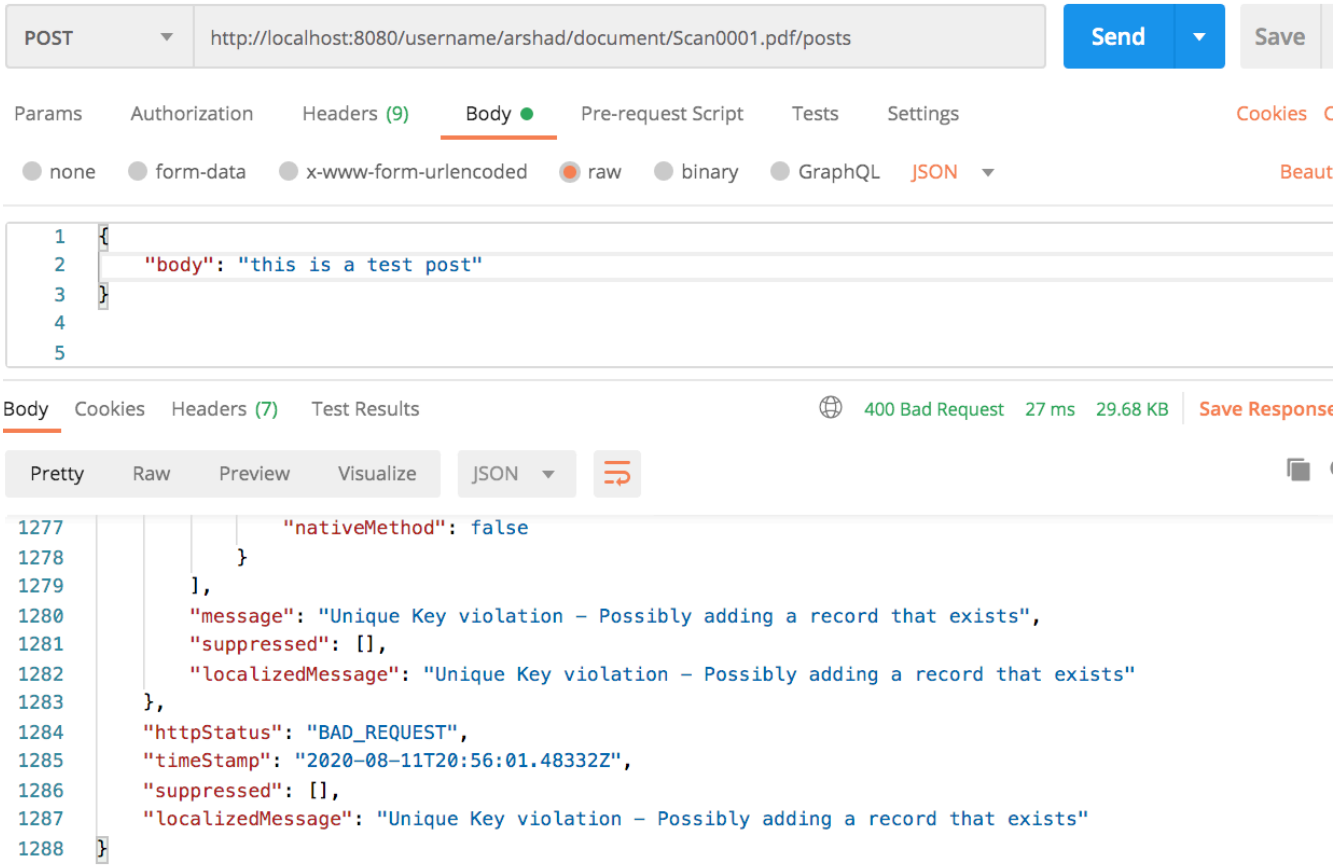
The next step covers POST and Commments

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**

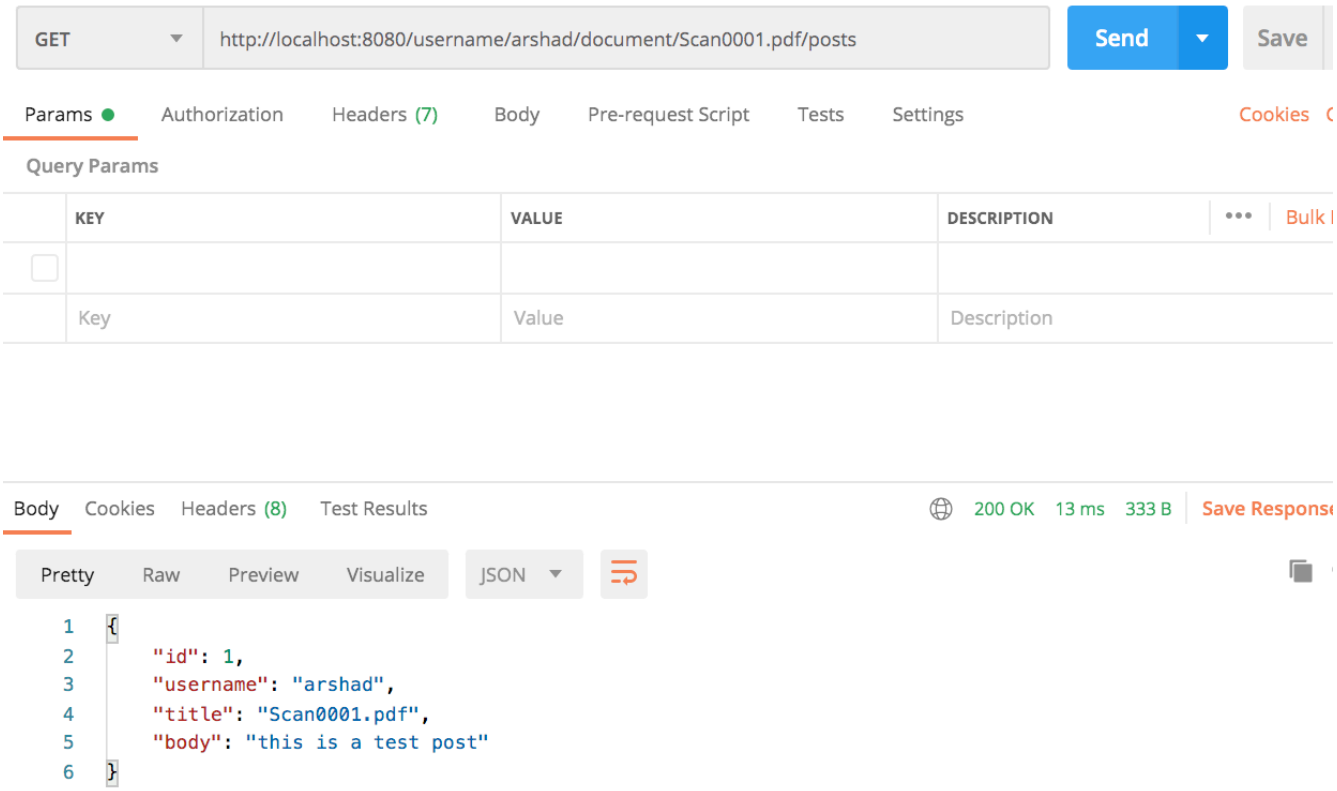
Next I would try duplicating this post by resubmitting (no duplicates allowed) our validtion kicks in and throws a custom exception return and error message “” with Http Status 400 BAD REQUEST



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)

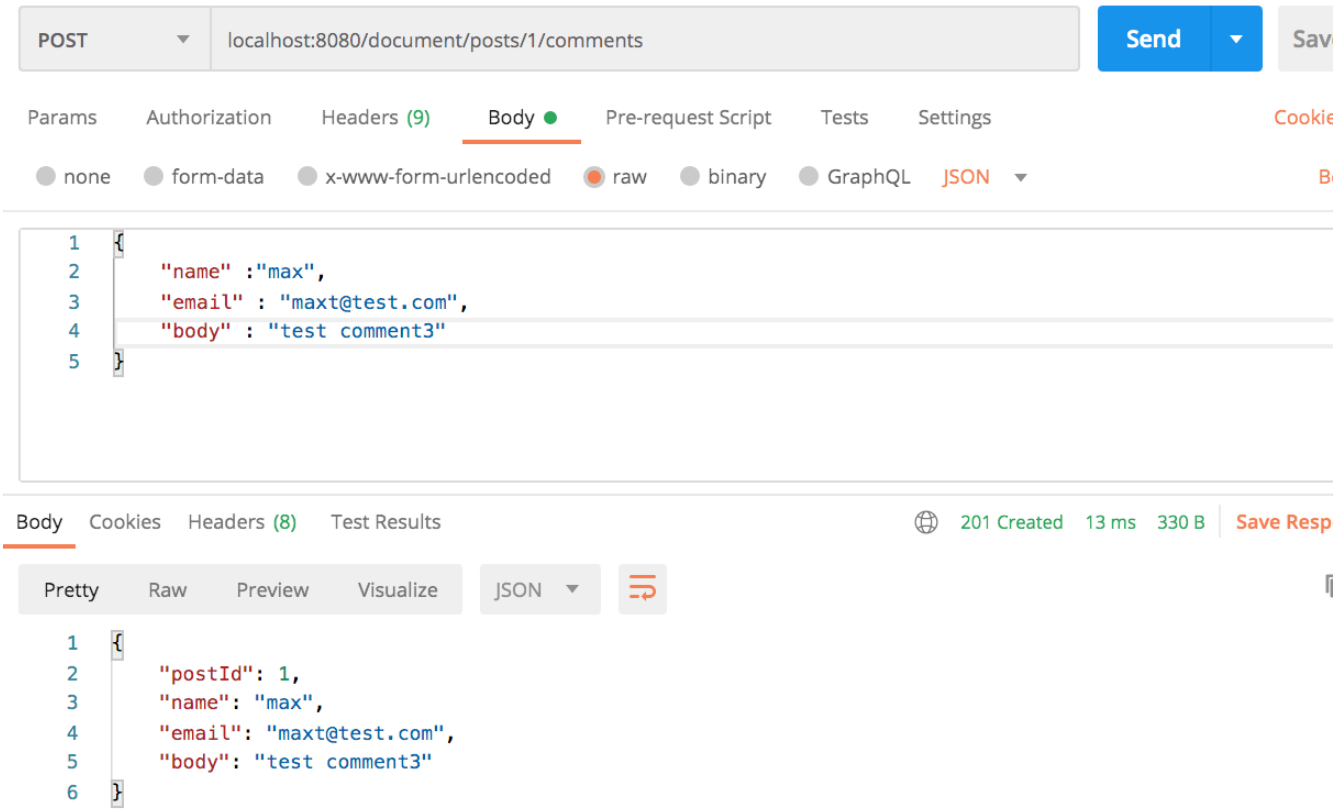
Query Parameters: None
Request body : None



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 containg 3 elements name (of the commenter), email (of the commenter) and the body of the comment

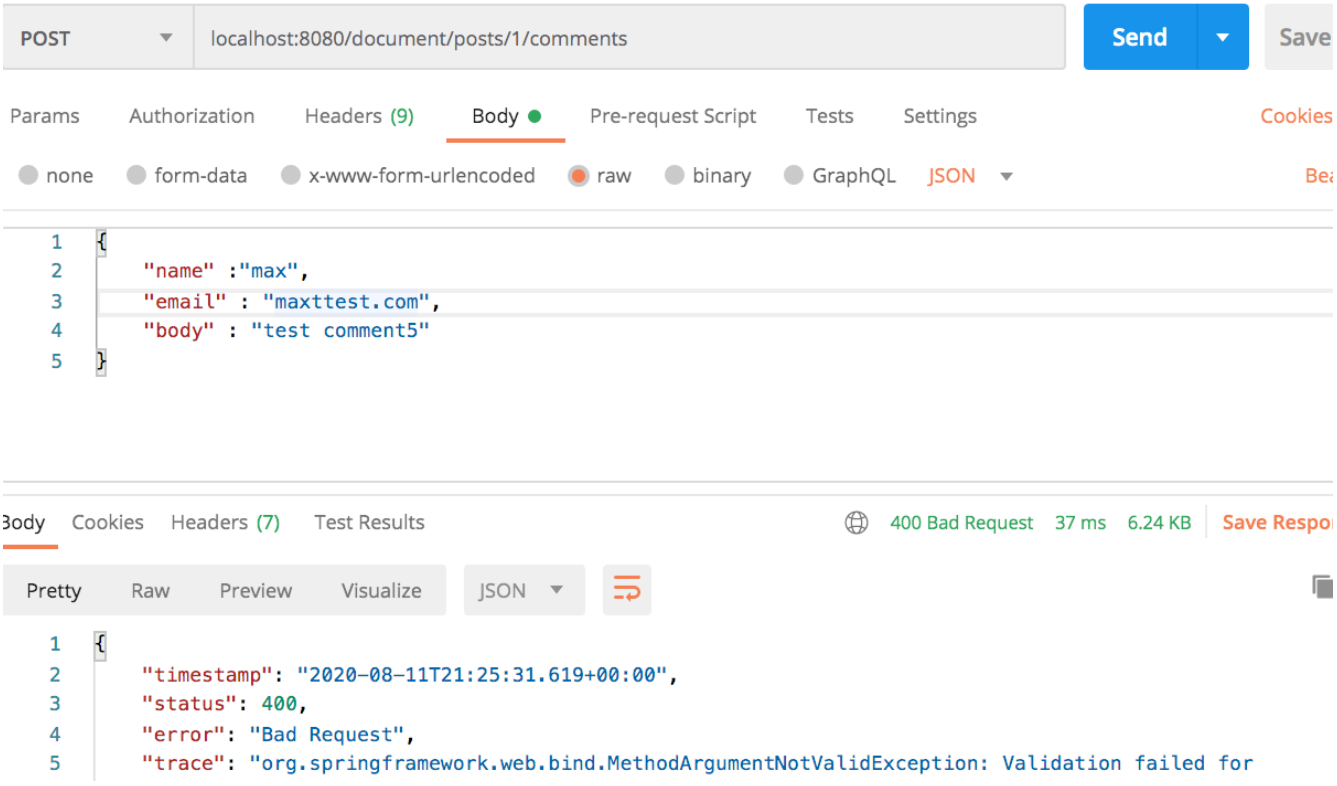


The result Http Status 201 CREATED with the the full comment in the response body is returned.

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”.

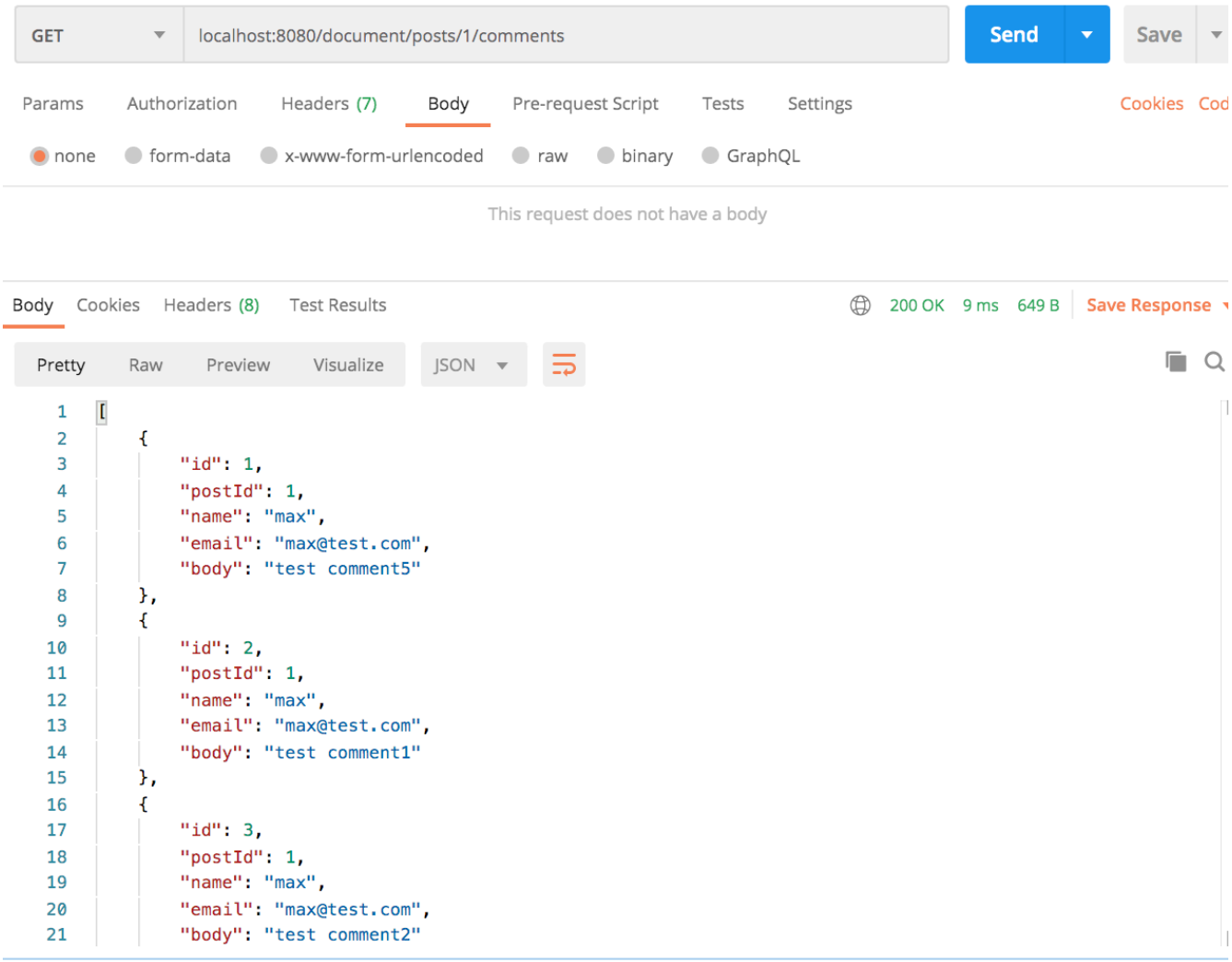
I have used Java Bean Validation on the entities I pass as request body, for example don’t allow nulls and validate email addressed etc these are also handled by custom exception resulting in a 400 BAD REQUEST.

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



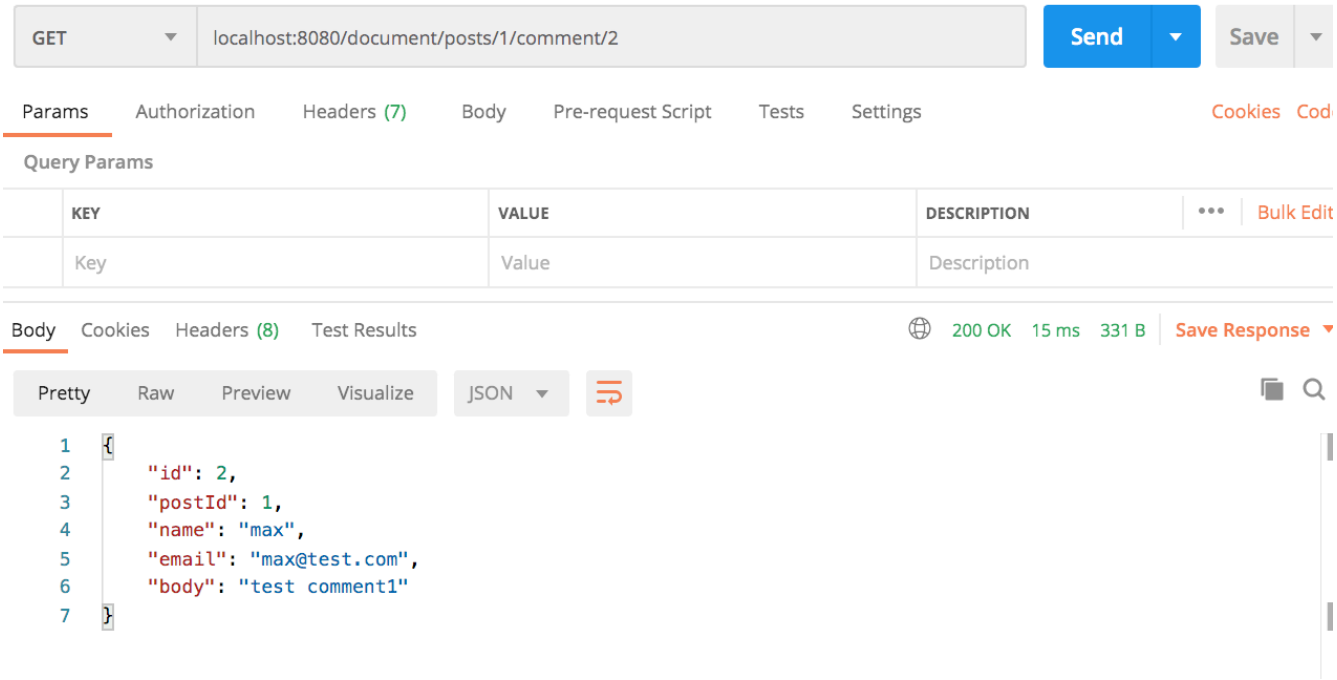
Step 10
Getting all Comments for a specific Post

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



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 rerieve
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
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
{
"id": 2,
"postId": 1,
"name": "max",
"email": "max@test.com",
"body": "test comment1"
}