PROGRAMMING TECHIE

Spring Boot Angular Project – Build a Youtube Clone – Part 2

0 Comments

BY SAI UPADHYAYULA | August 5, 2021

Share Tweet Pin

In this Part 2 of the Spring Boot Angular Project series, we will start building our Spring Boot backend API to upload the videos to AWS S3.

In Part 1 we discussed the functional requirements, the application architecture and created the skeleton of the project by leveraging the Maven Multi Modules. You can check out the post here

Source Code

You can find the source code of this project at Github – https://github.com/SaiUpadhyayula/youtube-clone

Video Tutorial

If you are a visual learner like me, you can check out the video tutorial I created

Feature 1: Upload Video to AWS S3

We will first configure our AWS S3 account, create an S3 Bucket to store our videos, configure our Spring Boot Project, and then implement the API to upload the video to AWS S3.

Create an AWS S3 Account

The first thing we need is an AWS account, you can create a new account at https://aws.amazon.com/free/.

When you create a new free tier account, you will be able to try out various services provided by AWS and experiment with them.

Keep in mind that AWS only provides limited access as part of the free tier, if you exceed the usage limits, then you will be billed accordingly.

Create Access Key and Secret Key

Alright, you have an account, the next step is to create an Access Key and a Secret Key for your AWS account so that we can connect our Spring Boot Application to AWS S3.

To do that, click on your username on the top right side corner of the website, and select the option **My Security Credentials**.

Now in the **Your Security Credentials** page, expand the section – **Access Keys (access key ID and secret access key)** and click on the button **Create New Access Key**.

Now click on the option **Show Access Key** and note down your Secret Access Key and keep it safe, this will be shown to you only once, you can also download the key file, which contains your Access Key ID and Secret Access Key.

Create S3 bucket

Now it's time to create a bucket inside S3, a bucket is nothing but a container that holds all your objects stored inside S3.

To create the bucket, search for the term **S3** in the search bar, and click on the button **Create bucket**.

Make sure you provide a **Bucket name**, usually the bucket name must be unique and this will be used as an identifier.

Once you have provided the name, you can click on the **Create bucket** button.

Configure S3 inside Spring Boot Project

Alright, the next step is to configure S3 credentials and details inside our Spring Boot Project, for that we are going to add two new dependencies to our **pom.xml** file.

```
<dependencies>
  <dependency>
    <groupId>io.awspring.cloud</groupId>
    <artifactId>spring-cloud-aws-context</artifactId>
    <version>{spring-cloud-aws-version}</version>
  </dependency>
  <dependency>
    <groupId>io.awspring.cloud</groupId>
    <artifactId>spring-cloud-aws-autoconfigure</artifactId>
    <version>{spring-cloud-version}</version>
  </dependency>
</dependencies>
```

Make sure you check the latest version of spring-cloud-aws here – https://github.com/awspring/spring-cloud-aws/releases

As of writing this tutorial, the latest version is **2.3.1**.

Now we need to add the Access Key ID and Secret Access Key to the **application.properties** file.

As this is sensitive information, it is not wise to store them inside source code, so we will inject these 2 properties at the time of application startup, we can do that using the **VM Options** in the **IntelliJ Run Configuration**.

IntelliJ Run Configuration – Add Properties to VM Options

Now we need to add the AWS S3 region information to our **application.properties** file and a default value to disable AWS Cloud Formation settings like below:

```
cloud.aws.region.static=eu-west-2
cloud.aws.stack.auto=false
```

Implement API to upload videos to S3

Now it's time to write some code, first, we have to create a controller which will receive the HTTP requests from the clients, let's call this class as **VideoController**

VideoController.java

```
package com.programming.techie.youtubeclone.controller;

import com.programming.techie.youtubeclone.service.VideoService;
import lombok.RequiredArgsConstructor;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.multipart.MultipartFile;

@RestController
@RequestMapping("/api/videos")
@RequiredArgsConstructor
public class VideoController {

    private final VideoService videoService;

    @PostMapping
    @ResponseStatus(HttpStatus.CREATED)
    public void uploadVideo(@RequestParam("file") MultipartFile file) {
        videoService.uploadVideo(file);
    }
}
```

- We are exposing an endpoint at URL – /api/videos which accepts POST requests from the user.
- Now the **uploadVideo** method inside the class, is accepting a **MultipartFile** as a method argument, which is passed on to the **uploadVideo** method of the **VideoService** class.
- Notice that, we are responding to this HTTP Request with HTTP Status as CREATED (201), whenever we receive a request to create any resource, we should always return the status as CREATED, according to the REST Specification.

VideoService.java

```
package com.programming.techie.youtubeclone.service;

import com.programming.techie.youtubeclone.model.Video;
import com.programming.techie.youtubeclone.repository.VideoRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;
import org.springframework.web.multipart.MultipartFile;

@Service
@RequiredArgsConstructor
public class VideoService {

    private final S3Service s3Service;
    private final VideoRepository videoRepository;

    public void uploadVideo(MultipartFile multipartFile) {
        String videoUrl = s3Service.uploadFile(multipartFile);
        var video = new Video();
        video.setVideoUrl(videoUrl);

        videoRepository.save(video);
    }
}
```

FileService.java

```
package com.programming.techie.youtubeclone.service;
```

```
2
3  import org.springframework.web.multipart.MultipartFile;
4
5  public interface FileService {
6      String uploadFile(MultipartFile file);
7  }
```

S3Service.java

```
1  package com.programming.techie.youtubeclone.service;
2
3  import com.amazonaws.services.s3.AmazonS3Client;
4  import com.amazonaws.services.s3.model.CannedAccessControlList;
5  import com.amazonaws.services.s3.model.ObjectMetadata;
6  import lombok.RequiredArgsConstructor;
7  import org.springframework.http.HttpStatus;
8  import org.springframework.stereotype.Service;
9  import org.springframework.util.StringUtils;
10 import org.springframework.web.multipart.MultipartFile;
11 import org.springframework.web.server.ResponseStatusException;
12
13 import java.io.IOException;
14 import java.util.UUID;
15
16 @Service
17 @RequiredArgsConstructor
18 public class S3Service implements FileService {
19
20     public static final String BUCKET_NAME = "youtube-demo-ptechie";
21     private final AmazonS3Client awsS3Client;
22
23     @Override
24     public String uploadFile(MultipartFile file) {
25         var filenameExtension = StringUtils.getFilenameExtension(file.getOriginalFilename());
26
27         var key = UUID.randomUUID().toString() + filenameExtension;
28
29         var metadata = new ObjectMetadata();
30         metadata.setContentLength(file.getSize());
31         metadata.setContentType(file.getContentType());
32
33         try {
34             awsS3Client.putObject(BUCKET_NAME, key, file.getInputStream(), metadata);
35         } catch (IOException ioException) {
36             throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR,
37                     "An Exception occured while uploading the file");
38         }
39
40         awsS3Client.setObjectAcl(BUCKET_NAME, key, CannedAccessControlList.PublicRead);
41
42         return awsS3Client.getResourceUrl(BUCKET_NAME, key);
43     }
44 }
```

VideoRepository.java

```
1  package com.programming.techie.youtubeclone.repository;
2
3  import com.programming.techie.youtubeclone.model.Video;
4  import org.springframework.data.mongodb.repository.MongoRepository;
5
6  public interface VideoRepository extends MongoRepository<Video, String> {
7  }
```

- We create an interface called **FileService** and a concrete implementation – **S3Service** to handle the file upload functionality to AWS S3.
- Inside the **uploadFile()** method of **S3Service** class, we are first creating a unique key for the file we are going to upload.
- After that, using the **putObject()** method of the **AmazonS3Client** class, we are uploading the file to the bucket we created in the previous section.
- Now after uploading the file, set the Access Control List value to **PublicRead**, as this file should be accessed from our Angular Application.
- Lastly, we are retrieveing the URL of the video we uploaded and storing them to the database using the **VideoRepository** Spring Data Mongo Interface.

Testing

Now let's test out the implementation, first, make sure you start the **YoutubeCloneApplication.java** class and using a REST client like Postman, make a POST request to the endpoint **/api/videos** and pass any video as Request Parameter.

Check out the below image for more details:

Conclusion

That's it for this article, in the next part we will implement the Frontend part of the feature and will start working on the next Feature.