**WEEK1 – Complete the System Design and Backend Coding**

**Monday: concept and design**

– 10 hours

I spent a few hours thinking about what kind of application I would create. And after that, I searched online for some information on how to best leverage services/products of AWS to make the implementation more efficient. Following is my finding to these two points.

1. About the Application:

To deliver an application that can extract the text from an image and then translate the English to Chinese. The user scenarios could be

* 1. When foreigners at the street want to know the meaning of a sigh.
  2. When foreigners in a supermarket want to know the introduction on the packing of commodities.

1. About the Architecture of backend (in Text now, will transfer to Graph in final report)
   1. Upload Image into S3
   2. S3 object creation trigger Lambda Function
   3. Lambda Function call Amazon Rekognition to extract the text and then call AWS translate API to translate the text
   4. Store Amazon final result as a file in S3

(Will change the ‘image upload and result showing’ from S3 to website/mobile App later after frond end implemented in phase 2)

1. About phases
   1. Phase-1: Implement the Lambda Function and complete end to end test from “manually uploading image to S3” to “get Amazon translated extracting text output”
   2. Phase-2: Create a website OR APP that can be used by Users to upload image to S3 (call AWS S3 API), and show the result of the translated extracted Text.
2. TODOs tomorrow
   1. Install ASW CLI, Maven, configure these tools On mac <https://docs.aws.amazon.com/cli/latest/userguide/install-cliv2-mac.html>
   2. Start the coding of lambda function and test the S3 event can successfully trigger the Lambda function.

**Tuesday: Env setup & Implement 1, going through lambda-s3 event trigger process**

– 10 hours

Today I tried to build up the Lambda function and go through the process that S3 object creation (by uploading an image) event triggers the Lambda function.

Prerequisite:

* Install maven and java11(lambda support java 11) on mac using brew.
* Create non-root credentials and get id account and security key
* Install AWS CLI and configure with my AWS credentials
* Create execution role for Lambda function and S3 operations

Code implementation and Lambda function creation

1. Get the code ready – I use IntelliJ as the IDE. The 2 files that are important are

***Handler.java*** – the business logical implementation.

The main Class is Handler which implement RequestHandler

public class Handler implements RequestHandler<S3Event, String>

The method is handleRequest which has event and Context as its input

public String handleRequest(S3Event s3event, Context context)

***pom.xml*** – manage the dependencies from AWS SDKs

1. Create the lambda package (after every change of your code, make sure you have the right dependencies and the compatible versions)

*mvn package*

1. Create the Lambda function

*aws lambda create-function --function-name Transv4 --zip-file fileb://s3-java-1.0-SNAPSHOT.jar --handler example.Handler::handleRequest --runtime java8 --timeout 30 --memory-size 1024 --role arn:aws:iam::561130904030:role/lambda-s3-role --debug*

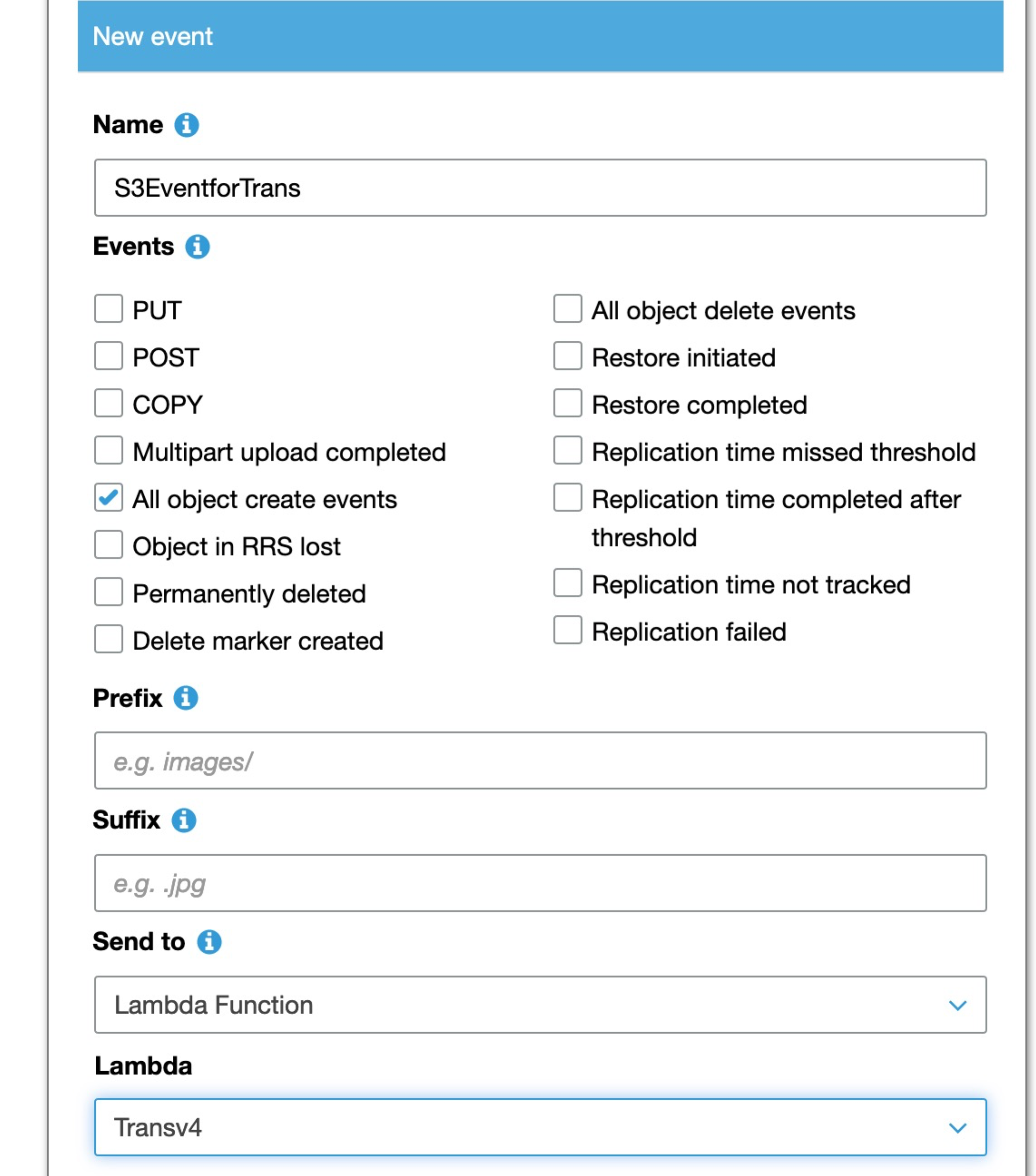
1. Test Lambda function

* By Using CLI to invoke the function (the inputFile.txt has all the event details & also need to create the bucket and upload the image beforehand)  
  aws lambda invoke --function-name *Transv4* --invocation-type Event --cli-binary-format raw-in-base64-out --payload file://inputFile.txt outputfile.txt
* By manually upload an image (firstly need to setup the function)

Create S3 bucket and setup the Lambda function

1. Create two buckets, one for image upload and one for result storing.
2. Setup the lambda function to handle the event

Bucket->properties->Event-> Add notification -> make sure the event notification sent to specific Lambda function.



**Wednesday: Implement 2, Text Extracting logical by Amazon Rekognition API**

– 6 hours

1. add related code in handler.java
2. Add dependency for aws recognition in pom
3. Add permission(**Rekognition all permission)** to the role(function) for recognition execution
4. Create package, create function
5. Trigger event

**Challenge today**

Hit some issues and fixed but cannot fix one issue about connection issue after I made some change to the pom.xml. I thought it was because of networking issue after going on-line and searching. So I would leave it to tomorrow and retry.

Thursday: bug fix & Implement 3, Add Translation logic, test the application

**- 6 hourss**

1. Yesterday’s issue fix

After a sleep, I was thinking the connection error might not because of the connection failure, but because of my package size. So I pivot to another solution and fixed the problem. Please refer Chapter **issues I hit & solution** for details.

1. Code change 1 -

RekognitionClient.detectText will extract the text for different types like WORD or LINE. So I need to restrict the Type, otherwise I will get the duplicated text in my result.

1. Code change 2 –

Add translation logic - transfer the extracted text from English to Chinese.

1. Create package, create function, Add authority for the api call execution to the role
2. Trigger event and test the logic and bug fix

**issues I hit & solution**

1. Everything looks right, but the Lambda function can not be executed.

Root cause & solution

I use a file for testing the problem. I then find the Function is not invoked at all, that is because the format of the parameter I used in the “create function” CLI is not for Java but for Python. So I search and change it to the right format and the issue was resolved

1. The test file can be generated but the file with the extracted text cannot.

Root cause:

Rekognition API execution exit because of authority issue.

Solution:

Add permission of “[AmazonRekognitionFullAccess](https://console.aws.amazon.com/iam/home?region=us-east-2#/policies/arn%3Aaws%3Aiam%3A%3Aaws%3Apolicy%2FAmazonRekognitionFullAccess) ” to the role which is used to generate the function.

1. Handler can be triggered, but the text is not extracted -> check the cloudwatch-loggroups and found following error message,

*NoSuchMethodError: com.amazonaws.transform.JsonErrorUnmarshaller*

Root cause:

The problem is in POM, change the version of all com.amazonaws dependencies to the same one.

Solution:

Like in following example, I change the version both to 1.11.268. Then the issue was gone.

<dependency>

<groupId>com.amazonaws</groupId>

<artifactId>aws-java-sdk-s3</artifactId>

<version>1.11.578</version>

</dependency>

<dependency>

<groupId>com.amazonaws</groupId>

<artifactId>aws-java-sdk-rekognition</artifactId>

<version>1.11.799</version>

</dependency>

1. Try to change some sdk version and try to package again. Then hit the timeout issue from the CLI

Connection was closed before we received a valid response from endpoint URL: "https://lambda.us-west-2.amazonaws.com/2015-03-31/functions".

Root cause:

I finally found it was because the size of my .jar package, 90M is larger than 50M as AWS CLI permitted, it is because I used aws.jdk, not aws.jdk.lambda

Solution:

It was approved to be the .jar size problem. It was over 50M which is the limit of AWS CLI for uploading function package. I then change the aws-jave-sdk to aws-java-sdk-\*\*, the problem was fixed.

Also Set the --cli-connect-timeout parameter and tried again to see the root cause

1. **Then I hit the error while invoking lambda handler:**

**NoClassDefFoundError**: **javax**/**xml**/**bind**/**JAXBException**. It is because the JAXB library (**Java** Architecture for **XML Binding**) is missing in the classpath. JAXB is included in **Java** SE 10 or older, but it is removed from **Java** SE from **Java** 11 or newer –moved to **Java** EE under Jakarta EE project.

Solution:

I changed java 11 to java 8 when create function by aws cli

**WEEK2 – Front End Coding, Documentation and TODOs**

This week, I finished the front end coding and completed the documentation work for this project with some TODOs listed for future work.

Regarding the front end implementation, I was considering a mobile App or a Web application or a kind of cross platform, which means coding once, running everywhere, either mobile or App, either IOS or Android. (refer to amazon Amplify, react)**.**

I then made a decision let me choose a easy way for the front end implementation at this moment since I only have 1 week of time and I am a totally newbie to front end. My project can be continuously improved.

Following is my journal this week.

**Day 1, Set up the front end Environment, and run through the sample simple code**

1. **Get Sample code ready** 
   1. In my case, I have a html file for the webpage layout and a java script file for S3 operations
2. **Upload the code to S3 bucket**
   1. I create the bucket for store the code. Set the permission to public for both the buckets and the files.
3. **Permission setting for the bucket where I will upload my file to**
   1. Create new identity pool. Make sure “permit unauthenticated user access”; allow the creating new roles in IAM;
   2. set the policy to the roles (in this case, the unauthenticatedrole) for S3 allowing. Make sure all actions used by your script have been listed in the policy.

*{*

*"Version": "2012-10-17",*

*"Statement": [*

*{*

*"Effect": "Allow",*

*"Action": [*

*"s3:DeleteObject",*

*"s3:GetObject",*

*"s3:ListBucket",*

*"s3:PutObject",*

*"s3:ListObject"*

*],*

*"Resource": [*

*"arn:aws:s3:::mwlibucket2/\*"*

*]*

*}*

*]*

*}*

* 1. set the permission for the source S3 bucket to permit the operations, firstly enable “public access”, then set “CORS configuration” by

*<?xml version="1.0" encoding="UTF-8"?>*

*<CORSConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">*

*<CORSRule>*

*<AllowedOrigin>\*</AllowedOrigin>*

*<AllowedMethod>POST</AllowedMethod>*

*<AllowedMethod>GET</AllowedMethod>*

*<AllowedMethod>PUT</AllowedMethod>*

*<AllowedMethod>DELETE</AllowedMethod>*

*<AllowedMethod>HEAD</AllowedMethod>*

*<ExposeHeader>ETag</ExposeHeader>*

*<AllowedHeader>\*</AllowedHeader>*

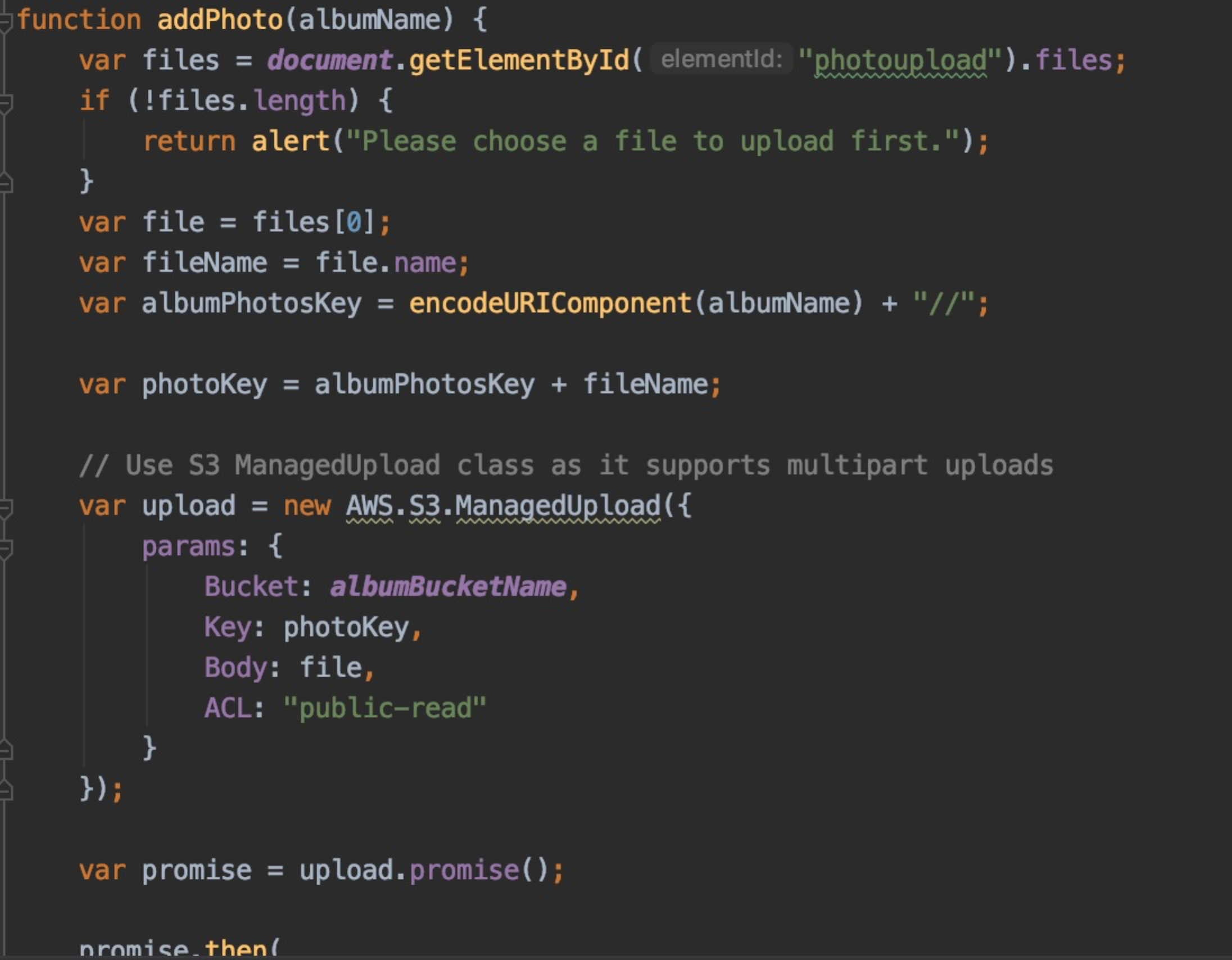
*</CORSRule>*

*</CORSConfiguration>*

1. **Configure SDK**

var ***albumBucketName*** = "mwlibucket2";  
var ***bucketRegion*** = "us-west-2";  
var ***IdentityPoolId*** = "us-west-2:15bc05f2-4595-4c85-9509-226f69f8d539"; //transitepool  
  
AWS.config.update({  
 region: ***bucketRegion***,  
 credentials: new AWS.CognitoIdentityCredentials({  
 IdentityPoolId: ***IdentityPoolId*** })  
});  
  
var ***s3*** = new AWS.S3({  
 apiVersion: "2006-03-01",  
 params: { Bucket: ***albumBucketName*** }  
});

1. **At this moment, you can start calling the SDK and create functions in the java script files.**

****

1. **Now, you can include the script file in your html and call the functions in the script file.**

****

**Day 2, completed the front end coding for image uploading to S3 and show it on website.**

This is the first time I am handing the front end coding. I use “HTML” + “Java Script”. The process was pretty nice. Nowadays, there are more than enough reference and information you can refer to. You are standing on the shoulders of giants!

1. To include Java script, you add the src in head section of html and the real script in body after the definition of the element which you will use in your script. Otherwise, you will get error like “**TypeError: document.getElementById(…) is null”.** Sometimes, it is just because the script is called too soon before the page is fully loaded
2. I would say HTML is really powerful and I was impressed that only one tag with proper parameters will generate an file upload logic on the website.
3. Another thing is about the policy setting for the bucket. Previously I use

"arn:aws:s3:::mwlibucket2/\*", then later, I found this could not work while I am calling listobjects() under mwlibucket2. So You have to add another line as “arn:aws:s3:::mwlibucket2”

**Day 3, completed the front end coding for text file showing and restructure the web page. Completed the whole business logic.**

Today is a trial day. I hit several times that my question can not be answered by google, so I have to try based on my own understanding to the APIs and their returns.

1. Show the text content of the file, (from trials)

data.Body.toString('utf-8'),

1. I also find “right click” the webpage and then see source is a good way to write html code.
2. Set charset="utf-8" for encoding problem
3. When you cannot find answers from google, just look into the SDK, carefully.

**Day 4, Finish the documentation(project introduction), Make the Video, upload to github, make future plan for continuous development.**

**TODOs**

1. configure Amazon Route53 to have a site url link

2. implement login/out,signup

3. implement image and translated text auto deletion per 12 hours

4. implement

5. Deploying A/B tests and blue/green deployments

6. Webpage beautify

7. show the final translated text according to their original position on the image

8. multiple browser support

9. mobile support (add the action of taking picture)

10. Add audio effect

11. Thinking about more AI ways that could recommended a logical explanation about the image. Like “ this is a sigh that shows the road is a only one direction” or “ this looks like the package of a kind of cracker made of cauliflower”, etc, in Chinese.

12. Multi-language support

**Links of services**

1. Cloudwatch – where I get the detail execution log

[https://us-west-1.console.aws.amazon.com/cloudwatch/home?region=us-west-1#logsV2:log-groups](https://us-west-1.console.aws.amazon.com/cloudwatch/home?region=us-west-1" \l "logsV2:log-groups)

1. S3 link – upload image and check the final result

<https://console.aws.amazon.com/s3/buckets/mwlibucket/?region=us-west-1&tab=properties>

1. IAM link – where you create account, role, and add permission

<https://console.aws.amazon.com/iam/home?region=us-east-2#/roles>

1. Cognito

<https://us-west-2.console.aws.amazon.com/cognito/federated?region=us-west-2#>

**References**

1. <https://stackoverflow.com/questions/35080393/creating-a-folder-uploading-a-file-in-amazon-s3-bucket-using-api>
2. <https://docs.aws.amazon.com/lambda/latest/dg/with-s3-example.html>
3. <https://dev.to/ramonak/how-to-upload-a-file-to-aws-s3-in-java-using-vaadin-framework-1m2m>
4. <https://docs.aws.amazon.com/rekognition/latest/dg/security_iam_service-with-iam.html#security_iam_service-with-iam-id-based-policies>
5. <https://github.com/awsdocs/aws-lambda-developer-guide/blob/master/sample-apps/s3-java/src/main/java/example/Handler.java>
6. <https://github.com/awsdocs/amazon-rekognition-developer-guide/blob/master/code_examples/java_examples/image/java-detect-text.java>
7. <https://docs.aws.amazon.com/rekognition/latest/dg/stored-video-lambda.html>
8. <https://github.com/awsdocs/amazon-rekognition-developer-guide/tree/master/code_examples/java_examples>
9. <https://github.com/aws/aws-cli/issues/3842>
10. <https://github.com/aws/aws-sdk-java/issues/930>
11. <https://docs.aws.amazon.com/rekognition>
12. <https://docs.aws.amazon.com/rekognition/latest/dg/text-detecting-text-procedure.html>
13. <https://docs.aws.amazon.com/translate/latest/dg/what-is.html#what-is-languages>
14. <https://docs.aws.amazon.com/translate/latest/dg/examples-java.html>
15. <https://aws.amazon.com/getting-started/hands-on/translate-text-between-languages-cloud/>
16. <https://docs.aws.amazon.com/AmazonS3/latest/user-guide/enable-event-notifications.html>
17. <https://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/sdk-code-samples.html>
18. [**https://aws.amazon.com/mobile/mobile-application-development/**](https://aws.amazon.com/mobile/mobile-application-development/)
19. [**https://aws.amazon.com/amplify/**](https://aws.amazon.com/amplify/)
20. [**https://aws.amazon.com/amplify/features/#Storage**](https://aws.amazon.com/amplify/features/#Storage)
21. [**https://docs.amplify.aws/lib/storage/getting-started/q/platform/js**](https://docs.amplify.aws/lib/storage/getting-started/q/platform/js)
22. [**https://aws.amazon.com/getting-started/hands-on/mobile/**](https://aws.amazon.com/getting-started/hands-on/mobile/)
23. [**https://docs.aws.amazon.com/mobile/sdkforunity/developerguide/s3.html**](https://docs.aws.amazon.com/mobile/sdkforunity/developerguide/s3.html)
24. [**https://d0.awsstatic.com/whitepapers/Storage/Building%20Static%20Websites%20on%20AWS.pdf**](https://d0.awsstatic.com/whitepapers/Storage/Building%20Static%20Websites%20on%20AWS.pdf)
25. [**https://developer.mozilla.org/en-US/docs/Web/HTML/Element**](https://developer.mozilla.org/en-US/docs/Web/HTML/Element)
26. [**https://d0.awsstatic.com/whitepapers/Storage/Building%20Static%20Websites%20on%20AWS.pdf**](https://d0.awsstatic.com/whitepapers/Storage/Building%20Static%20Websites%20on%20AWS.pdf)
27. [**https://github.com/awsdocs/aws-doc-sdk-examples/blob/master/javascript/example\_code/s3/s3\_photoExample.js**](https://github.com/awsdocs/aws-doc-sdk-examples/blob/master/javascript/example_code/s3/s3_photoExample.js)
28. [**https://aws.amazon.com/getting-started/hands-on/host-static-website/**](https://aws.amazon.com/getting-started/hands-on/host-static-website/)
29. [**https://d0.awsstatic.com/whitepapers/Storage/Building%20Static%20Websites%20on%20AWS.pdf**](https://d0.awsstatic.com/whitepapers/Storage/Building%20Static%20Websites%20on%20AWS.pdf)
30. [**https://docs.aws.amazon.com/AWSJavaScriptSDK/latest/AWS/S3.html#listObjectsV2-property**](https://docs.aws.amazon.com/AWSJavaScriptSDK/latest/AWS/S3.html#listObjectsV2-property)