API Gateway / Lambda Recitation

This guide will help you glue together API Gateway, AWS Lambda, and a frontend application. I hope it will be useful for getting started with HW 1.

Setting Up API Gateway + Lambda

Before we get started, make an AWS lambda function. For now, you can just leave the default code. Now, make a new API by going to the API Gateway console, clicking "Create API", then "Rest API" (not private). Name your api, then create it.

Then, make a new resource with "Create Resource" under "Actions". I'll name my resource "new_entry". Under that newly created resource, create a POST endpoint.

When creating the POST endpoint, let's hook up the lambda function we created that will be called when the POST endpoint is hit. Be careful not to select Lambda Proxy Integration here—we want API Gateway to do a couple things before it sends a response back to the client, namely attach the CORS headers, which the client requires. If instead you check the Lambda Proxy Integration box, the Lambda's response is passed directly to the client—therefore, the lambda is responsible for attaching the CORS headers to its response (if you want to do it this way, check out my note at the bottom).

Before we can call the endpoint, we need to deploy the API. Go ahead and do that. Now, let's test it on Postman to see that the endpoint works by making a POST request to:

```
https://{restapi_id}.execute-api.{region}.amazonaws.com/{stage_name}/new_entry.
```

Great! It works.

Generate an SDK from API Gateway

Postman is great, but we want to call this from a web application. Let's generate an SDK from API Gateway that we can use in our frontend application. Download it from in

Stages > STAGE_NAME > SDK Generation > JavaScript > Generate SDK

In your html file which will be served to the user via a CDN (ie. S3), you need to include all the scripts the SDK tells you to in its README.md by providing their relative paths in the HTML. Also include the javascript file with all the business logic, in my case, demo.js by providing its relative path. Also, don't forget to initialize the SDK somewhere. I added a short script in my HTML to do so.

I also made a textbox and a button. When the button is clicked, it should send a request to API Gateway with the text in its body.

```
<!DOCTYPE html>
<html lang="en">
<head>
    ...
    <!-- ## Adding these because SDK told me to ## -->
    <script type="text/javascript" src="./sdk/lib/axios/dist/axios.standalone.js"></script>
    <script type="text/javascript" src="./sdk/lib/CryptoJS/rollups/hmac-sha256.js"></script>
    <script type="text/javascript" src="./sdk/lib/CryptoJS/rollups/sha256.js"></script>
    <script type="text/javascript" src="./sdk/lib/CryptoJS/components/hmac.js"></script>
    <script type="text/javascript" src="./sdk/lib/CryptoJS/components/enc-base64.js"></script>
    <script type="text/javascript" src="./sdk/lib/url-template/url-template.js"></script>
    <script type="text/javascript" src="./sdk/lib/apiGatewayCore/sigV4Client.js"></script>
    <script type="text/javascript" src="./sdk/lib/apiGatewayCore/apiGatewayClient.js"></script>
    <script type="text/javascript" src="./sdk/lib/apiGatewayCore/simpleHttpClient.js"></script>
    <script type="text/javascript" src="./sdk/lib/apiGatewayCore/utils.js"></script>
    <script type="text/javascript" src="./sdk/apigClient.js"></script>
    <!-- ## ## -->
</head>
<body>
    I am demo website.
    <br>
    <br>
    <br>
    <input id="textbox" type="text">
    <button id="button">Press me to make a post</button>

    <!-- Initializing sdk before I load demo.js so it's accessible there -->
    <script>
        var sdk = apigClientFactory.newClient({});
    </script>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js"></script>
    <script src="./demo.js"></script>
</body>
</html>
```

Business Logic

In my demo.js, after the window loads, I grab the button element on the page, and make makePost() be called every time the button element is clicked. makePost() simply uses the generated SDK to make a call to the POST endpoint we made earlier.

```
$(document).ready(function() {
    var button = $('#button');
    console.log('button', button)

    button.click(function() {
        makePost()
    })

    function makePost() {
        const text = $('#textbox').val();
        console.log('text', text);
        // newEntryPost takes (params, body, additionalParams)
        sdk.newEntryPost({}, {
            entry: 'here'
        }, {})
        .then(response => console.log('response', response))
    }
})
```

Enabling CORS on API Gateway

Now, let's try it out by typing something into the textbox and clicking the button.

```
Access to XMLHttpRequest at 'https://q65uw5ss8f.execute-api.us- index.html:1
east-1.amazonaws.com/v1/new_entry' from origin 'null' has been blocked by
CORS policy: Response to preflight request doesn't pass access control
check: No 'Access-Control-Allow-Origin' header is present on the requested
resource.
```

Oh no! What happened? The Chrome browser expects certain headers to be in the response, but API Gateway isn't providing them. Let's change that by going back to the

API Gateway console, clicking /new_entry, then clicking Enable CORS under Actions. Check Default 4XX and Default 5XX, then do "Enable CORS and replace existing CORS headers". You'll notice an OPTIONS method was created. Deploy, then try again.

Great! It now works from the frontend as well.

Note:

If you use Lambda Proxy-Integration, you will manually have to attached the CORS headers on your lambda function response. You do not need to enable CORS on API Gateway.

This can be done like so:

```
return {
    'headers': {
        'Access-Control-Allow-Headers': 'Content-Type',
        'Access-Control-Allow-Origin': '*',
        'Access-Control-Allow-Methods': 'OPTIONS,POST,GET'
    },
    'statusCode': 200,
    'body': json.dumps(body)
}
```