[version_1.0.4]

Errors or corrections? Contact us at https://support.aws.amazon.com/#/contacts/aws-training

Exercise: API

Story So Far

It's early, but with freshly brewed coffee ☕ at hand. All is good in the world.

You are ready to start day two of your project.

So far, you have a working website but it doesn't do anything clever (yet).

Mike walks past your desk, and gives you an approving nod. He seems happy with your progress so far.

Today, you are determined to really impress Mike, by making this thing interactive. You are going to build out a backend API.

As you know (if you did the optional stage 4 of Lab 1) your website starts up and checks for the existence of an API endpoint. If an endpoint remains as `null` in the config file. Then the site will tell you that you have no API to call. *If* however, you *do* provide an API endpoint in your config file, the website will attempt to connect with it.

As you will see in a bit, it is wired up in such a way that it will handle a specific type of response from the back end.

So this is your game plan for today.

- Setup an API gateway stage, with three (3) distinct API routes. That's one for each site function `get_reviews`, `get_ratings` and `create_report`. Each of these functions will return hard coded MOCK data that the front end is geared up to handle.
  - 📓 Regardless of what product Id is passed in the request, it's going to get the same canned response.
- You will then confirm that all 3 routes work, via the API GW console testing feature.
- You will need to set up CORS on one of the API routes (`create_report`) because the website is CROSS domain from the API endpoint.

  🤷 *For the simple GET requests, you do not need to configure all the OPTIONS settings to make AJAX calls. You only need to enable CORS on the more "involved" POST request, where we are ~~going to be~~ handling Cognito Authentication.*
- Once you publish the API stage, you will test that the website is displaying the mock data correctly.
- ~~Finish strong, with a frappe on your way home.~~

What you will learn in this lab:

1. How to create an API GW resource with 3 routes using the SDK of your choosing.
2. Learn how to test your API in the API GW test console, and learn how to navigate around the AWS API GW console. Check for settings such as the method execution and any integration response configuration.
3. You will learn how to enable CORS (Cross Origin Resource Sharing) on a route (again via the SDK).

Before you start

Let's take a moment to think about how you are going to set up all this mock data.

Each of these API's will need to be provided with mock data in a format that the website can react to.

Let's look at the type of mock data we are taking about.

If they choose "get ratings", we will have our backend API respond with this hard coded value (again regardless of the product ID passed).

Request

```
<YOUR API>/get_reviews?product_id=sdgshkdgshdgsjhdgshj //some random product id string
```

Response (*Status 200*)

This is going to be your "canned" response because the website is expecting this sort of structure.

```
{
    "product_id_str": "sdgshkdgshdgsjhdgshj", //just echoing what was sent
    "reviews_arr": [{
        "review_body_str": "Both the dropcam and nest cam have an embarrassingly bad WIFI algorithm when there are multiple access points with the same name (SID) near it.  (I have a tall house and I need multiple WIFI access points)  When you have this situation, the cameras lose connectivity all the time. The obvious workaround is to dedicate a WIFI access point specifically for the Nest Cam, which is annoying.  Why Nest can't or won't fix this is beyond me.  I know of no other WIFI enabled device that is this dumb about WIFI connectivity.  Until this is fixed it stays a 3",
        "rating_int": 3
    }, {
        "review_body_str": "It was easy to setup with a small hiccup during the scanning of the barcode on the back.  I still have issues with the software not loading correctly on my phone which customer service has said they are working on fixing.  The app hangs quite often when loading it from a push notification where I either get single spinners or double spinners.<br /><br />I do wish the monthly/yearly fees for video retention were better or there was maybe a network based solution for video storage as I would like to buy more of these and use them as a whole house system but would get quite pricy",
        "rating_int": 3
```

```
    }, {
        "review_body_str": "I've had this device for a few weeks now and I really like it.  It was easy to
setup and it's easy to use.  I already have a Nest thermostat which I love and I now use the same app (on
Android) to manage the camera.  It is really cool to be able to view the camera from my phone wherever I am.
There are some small kinks which seem to need work in the app.  For example, clicking on the notification
will open the app and infinitely try to load the image from the camera history.  If you don't pay for the
history it was just infinitely load... you could wait an hour it will never load an image.  You have to back
out of the app and open it again to see the image.  Also, the camera should come with at least one day or a
few hours of video history included for free.  It would be great to have the option to cache video history
to my own computer or network device.  Without paying the subscription fee you have ZERO video history.  You
will get a notification that the camera detected motion.... but you can't see it because it's usually over
before you can open the app.  The camera is pretty much useless without video history... but the prices for
history are not cheap.  If you don't mind paying a monthly fee... it's a great device with excellent build
quality and image quality.",
        "rating_int": 4
    }, {
        "review_body_str": "I was hoping to use this for outdoor surveillance.  Proved to be too difficult
to isolate zones where breezy plants wouldn't trigger unwanted alerts.  On one occasion, I received motion
alerts when camera was allegedly off, which made me uncomfortable about when video was/wasn't being sent to
cloud.  App had a bad habit of turning off my motion zones so my alerts were not useful.  Camera pours off
heat.  Seems overall like an unrefined product not on par with the Nest thermostat which I own and like.",
        "rating_int": 3
    }]
}
```

For the "ratings" functionality, which is going to provide the *average rating* across a product's set of reviews. You will have something more like this:

Request

```
<YOUR API>/get_av_star_rating?product_id=sdgshkdgshdgsjhdgshj //some random product id string
```

Response (*Status 200*)

This is going to be your "canned" response.

```
{
    "product_id_str": "sdgshkdgshdgsjhdgshj", //it will echo what was sent
  "average_star_review_float":  3.25
}
```

Finally for the create_report functionality.

🤷 This third API is actually really simple (for now that is ;)....evil laugh. All it does is say "got it cheers", then proceeds to do absolutely nothing else.

Like so:

Request

```
[ POST XHR ] <YOUR API>/create_report //nothing passed in the body
```

Response (*Status 200*)

This is going to be your "canned" response.

```
{
        "message_str": "report requested, check your phone shortly"
}
```

🤷 *You can ignore the check your phone thing...that's not happening. Later on, in future labs this API call will trigger a background process, and send a report out to their cell.*

Ok, so now you know what your API needs to do. It's time to actually create it all.

You will use start off by using a script called `mock` which will build out the core API. You will be using the SDK of your choice.

Accessing the AWS Management Console

1. At the top of these instructions, click **Start Lab** to launch your lab.
2. A Start Lab panel opens displaying the lab status.
3. Wait until you see the message "**Lab status: ready**", then click the **X** to close the Start Lab panel.
4. At the top of these instructions, click **AWS**

   This will open the AWS Management Console in a new browser tab. The system will automatically log you in.

> **TIP:** If a new browser tab does not open, there will typically be a banner or icon at the top of your browser indicating that your browser is preventing the site from opening pop-up windows. Click on the banner or icon and choose "Allow pop ups."

Arrange the AWS Management Console tab so that it displays along side these instructions. Ideally, you will be able to see both browser tabs at the same time, to make it easier to follow the lab steps.

Setup

1. Ensure you are in **Cloud9**. Choose **Services** and search for **Cloud9**. You should see an existing IDE called Building_2.0. Click the button **Open IDE**. Once the IDE has loaded, enter the following command into the terminal: *(This command will ensure that you are in the correct path)*

   ```
   cd /home/ec2-user/environment
   ```

2. You will need get the files that will be used for this exercise. Go to the Cloud9 **bash terminal** (at the bottom of the page) and

run the following `wget` command:

```
wget https://aws-tc-largeobjects.s3-us-west-2.amazonaws.com/DEV-AWS-MO-Building_2.0/lab-2-api.zip
```

3. Unzip:

```
unzip lab-2-api.zip
```

4. Let's cleanup:

```
rm lab-2-api.zip
```

5. Run the `resources/setup.sh` script that will grab the website contents and upload them to the S3 bucket created by our CloudFormation template.

```
chmod +x ./resources/setup.sh && ./resources/setup.sh
```

⚠️ **If you are using Java** you will also need to run the following scripts and commands:

```
chmod +x ./resources/java_setup.sh && . ./resources/java_setup.sh
```

Lab Steps

Stage 1 - Create A New API GW using the SDK

1. 🔬 Open the `mock` file in your chosen language folder. Read through it so you know what it's doing.

   ⚠️ **If you are using Java** the folder structure for java is different from for Node.js or Python. You'll need to navigate to the parent folder for the program to run. The parent folder name will correspond with the file name for the other languages. In this case, the file is named `mock` for node and python, therefore the folder you should navigate to in the java folder is the `mock` folder. Under that folder you should see a `pom.xml` file - that is how you know you are in the right spot :)

   🤷 *If you are not in the right right folder in the `CMD_LINE`, navigate to it now.*

   For example if you are using Python use `cd python_3.6.8`

   The following table shows the respective **run command** needed to run the `mock` file:

Language	Filename You Need To Edit	Command to run via `CMD_LINE`
Node.js (v10)	mock.js	`npm install aws-sdk && node mock.js`
Python (v3)	mock.py	`python3 mock.py`
Java	App.java	`mvn clean install && mvn exec:java -Dexec.mainClass=com.mycompany.app.App`

   Once you run the file in the `CMD_LINE` you should see (regardless of language) the word **done**.

   📓 A lot is happening when you run this `mock` file. So please take a moment to read though it and see what it is doing, before moving on.

   📓 One thing worth talking about in the mock file is the method and integration settings:

   How does the API handle the query string?

```
//this is a snippet from the node code but the same concepts apply for each language

        if(route_name_str === "get_reviews"){
            params.responseTemplates = {
                "application/json": JSON.stringify({
                    "product_id_str" : "$input.params('product_id')",
                    "reviews_arr": [{
                        "review_body_str": "Both the dropcam and nest cam have an embarrassingly bad WIFI algorithm when there are multiple access points with the same name (SID) near it.  (I have a tall house and I need multiple WIFI access points)  When you have this situation, the cameras lose connectivity all the time. The obvious workaround is to dedicate a WIFI access point specifically for the Nest Cam, which is annoying.  Why Nest can't or won't fix this is beyond me.  I know of no other WIFI enabled device that is this dumb about WIFI connectivity.  Until this is fixed it stays a 3",
                        "rating_int": 3
                    }///...truncated for brevity
                    }]
                })
            };
        }else if(route_name_str === "get_av_star_rating"){
            params.responseTemplates = {
                "application/json": JSON.stringify({
                    "product_id_str" : "$input.params('product_id')",
                    "average_star_review_float":  3.25
                })
            };
        }
```

   By using this as the integration response we can tell our respective GET routes to capture a "predefined" value `"$input.params('product_id')"`, . This is going to come in from the query string and map it to a variable name for the output. Namely `product_id_str`.

   When we say "predefined", we are referring to where we declare it *first* in our method request parameters. As seen below:

```
//this is a snippet from the node code but the same concepts apply for each language
```

```
requestParameters: {
            "method.request.querystring.product_id": false
}
```

That bit allows us to reference the `product_id` phrase in our integration response as above, and thus translate (or map) it to a variable that we can use later in our Lambda code. In this case `product_id_str`.

The `false` flag is just saying that we are *not* going to validate the existence of that query string key. *Otherwise you would have to set up separate validators for both GET APIs, which for this lab is probably overkill.*

Final note: For a MOCK you want to pass everything through as OK for the Integration request, like so.

```
  //this is a snippet from the node code but the same concepts apply for each language
type: "MOCK",
requestTemplates: {
        "application/json": "{'statusCode': 200}"
}
```

They are the main areas of note.

For many of you who have worked with API GW before and are perhaps used to having Lambda Proxy just send everything though to the final end point, you might be wondering "why bother doing this mapping stuff". There is no wrong or right way of approaching API Gateway with a backend, there are just different approaches. Either way I think is it useful for you to see your options and how all this integration and method stuff works.

With all that said, how do even know that our script even worked?

Stage 2 - Testing the API

Head over to the AWS console for API GW. Ensure `Fancy-APi` exists (~~we're great at naming things in this team~~), and test it, as follows.

1. Choose **AWS Cloud9** and **Go To Your Dashboard**. Choose **Services** and **API Gateway**. There should be a `Fancy-Api` that was created by that `mock` script. Choose the `Fancy-Api`. *Make sure you are in **us-west-2**.*
2. It is important that you have a browse around the API GW console, to see what your script created. Check the settings for GET Method Request URL Query String Parameters and the GET - Integration Request application/json mappings as as expected.
3. Let's test `get_reviews` first. Choose the **GET** method under `get_reviews`.
4. Choose **TEST**.

   Under `Query Strings` use:

   ```
   product_id=888888
   ```

   The id is irrelevant as this is only a mock.
5. Then select the **Test** button.

   Example output:

   Response Body (200)


```
{
  "product_id_str": "888888",
  "reviews_arr": [
    {
      "review_body_str": "Both the dropcam and nest cam have an embarrassingly bad WIFI algorithm when there are multiple access points with the same name (SID) near it.  (I have a tall house and I need multiple WIFI access points)  When you have this situation, the cameras lose connectivity all the time. The obvious workaround is to dedicate a WIFI access point specifically for the Nest Cam, which is annoying.  Why Nest can't or won't fix this is beyond me.  I know of no other WIFI enabled device that is this dumb about WIFI connectivity.  Until this is fixed it stays a 3",
      "rating_int": 3
    },
    {
      "review_body_str": "It was easy to setup with a small hiccup during the scanning of the barcode on the back.  I still have issues with the software not loading correctly on my phone which customer service has said they are working on fixing.  The app hangs quite often when loading it from a push notification where I either get single spinners or double spinners.<br /><br />I do wish the monthly/yearly fees for video retention were better or there was maybe a network based solution for video storage as I would like to buy more of these and use them as a whole house system but would get quite pricy",
      "rating_int": 3
    },
    {
      "review_body_str": "I've had this device for a few weeks now and I really like it.  It was easy to setup and it's easy to use.  I already have a Nest thermostat which I love and I now use the same app (on Android) to manage the camera.  It is really cool to be able to view the camera from my phone wherever I am.  There are some small kinks which seem to need work in the app.  For example, clicking on the notification will open the app and infinitely try to load the image from the camera history.  If you don't pay for the history it was just infinitely load... you could wait an hour it will never load an image.  You have to back out of the app and open it again to see the image.  Also, the camera should come with at least one day or a few hours of video history included for free.  It would be great to have the option to cache video history to my own computer or network device.  Without paying the subscription fee you have ZERO video history.  You will get a notification that the camera detected motion.... but you can't see it because it's usually over before you can open the app.  The camera is pretty much useless without video history... but the prices for history are not cheap.  If you don't mind paying a monthly fee... it's a great device with excellent build quality and image quality.",
      "rating_int": 4
    },
    {
      "review_body_str": "I was hoping to use this for outdoor surveillance.  Proved to be too difficult
```

```
to isolate zones where breezy plants wouldn't trigger unwanted alerts.  On one occasion, I received
motion alerts when camera was allegedly off, which made me uncomfortable about when video was/wasn't
being sent to cloud.  App had a bad habit of turning off my motion zones so my alerts were not useful.
Camera pours off heat.  Seems overall like an unrefined product not on par with the Nest thermostat which
I own and like.",
      "rating_int": 3
    }
  ]
}
```


Also scroll down to see the headers:

Response Headers

This is showing that we have set up `Access-Control-Allow-Origin` to be `"*"`, so we can make *simple* AJAX calls to this resource from a different sub-domain.


```
{"Access-Control-Allow-Origin":"*","Access-Control-Allow-Methods":"GET","Access-Control-Allow-
Headers":"Content-Type,X-Amz-Date,Authorization,X-Api-Key,X-Amz-Security-Token","Content-
Type":"application/json"}
```


One down. Two more to go.

6. Now for the ratings one. Choose the **GET** method under `get_av_star_rating`. Choose **TEST**. For **Query Strings** just like before use:

```
product_id=888888
```

Then select the **Test** button.

You should get this back:

Response Body (200)

```
{
  "product_id_str": "888888",
  "average_star_review_float": 3.25
}
```

Response Headers

Again showing that this resource is accessible from simple AJAX requests from anywhere.


```
{"Access-Control-Allow-Origin":"*","Access-Control-Allow-Methods":"GET","Access-Control-Allow-
Headers":"Content-Type,X-Amz-Date,Authorization,X-Api-Key,X-Amz-Security-Token","Content-
Type":"application/json"}
```


Perfect, you are on the home stretch. Let's test the last one for the creating report functionally.

7. Choose the **POST** method under `create_report`. We will leave the **body** empty.

Choose **Test**.

You should get this:

Response Body (200)

```
{
  "message_str": "report requested, check your phone shortly"
}
```

Response Headers

🔲 We have this set `Access-Control-Allow-Origin` to `*` for now (*essentially a placeholder*). However we are *not* going to be using *simple* requests for this particular route. Later in this Lab we will be implementing OPTIONS with non-simple requests, which will lock this down more. It is fine for now though.


```
{"Access-Control-Allow-Origin":"*","Access-Control-Allow-Methods":"POST","Access-Control-Allow-
Headers":"Content-Type,X-Amz-Date,Authorization,X-Api-Key,X-Amz-Security-Token","Content-
Type":"application/json"}
```


Magic. It's all working, albeit, its using MOCK data. Our POST route is not using our OPTIONS preflight stuff yet.

Let's deploy this API and test it on the website.

Choose the root path `/` under **Resources**. Choose **Actions** and **Deploy API**. Under **Deployment stage** choose `[New Stage]`. For **Stage name** paste in `test`. Choose **Deploy**. Copy down the **Invoke URL**.

Example endpoint (yours will be different)

```
https://c6ef6rcb46.execute-api.us-west-2.amazonaws.com/test
```

Before we can test this on the website, we need update the API URL from null to that ^ actual API endpoint.

Follow these steps to update the website's config file:

Go back to the **Cloud9** tab. In the `resources/website` folder in Cloud9. Update the `resources/website/config.js` file with the newly created (and deployed) endpoint and **save** the file.

Example:

```
var G_API_GW_URL_STR = null;
var G_COGNITO_HOSTED_URL_STR = null;
```

```
var G_API_GW_URL_STR = "https://c6ef6rcb46.execute-api.us-west-2.amazonaws.com/test";
var G_COGNITO_HOSTED_URL_STR = null;
```

🔲 Note there is no trailing end slash there

Note there is no trailing end slash there.

Run the `resources/setup2.sh` script to upload the new `config.js` file.

⚠ **If you are using Java**. You will need to change directories to be in the `/home/ec2-user/environment/java8` directory before running the following commands.

```
chmod +x ../resources/setup2.sh && ../resources/setup2.sh
```

Should give you something like this:

```
upload: resources/website/config.js to s3://c11284a122552u294891t1w617867481956-s3bucket-
rxqOuubks9hi/config.js
```

Try visiting your API GW endpoint **directly** in the browser:

Note there is no route (pathname) mentioned within that raw endpoint. So if try to hit that URL in the browser we will get this.

```
{
    "message": "Missing Authentication Token"
}
```

Now try this with one of the GET **routes** using a **querystring** like so:

Example endpoint (yours will be different, swap out the subdomain with your bucket name like before)

```
https://c6ef6rcb46.execute-api.us-west-2.amazonaws.com/test/get_av_star_rating?product_id=777777
```

You should get this. Proving your API is publicly accessible. #win

```
{
  "product_id_str": "777777",
  "average_star_review_float":  3.25
}
```

8. Time to see this working on the website. Head over to your website, who's config you have just updated.

   If you don't have your URL in hand. You can type this in the Cloud9 `CMD_LINE` to get it.

   ```
   source /home/ec2-user/.bashrc && echo $bucket_url
   ```

   Example:

   ```
   https://YOUR-BUCKET-NAME.s3-us-west-2.amazonaws.com/index.html
   ```

   If you search for an item on the website, and press "ratings" or "reviews" you should get data being returned.

   If you try and use the `create_report` button it will tell you that you are not logged in. Which is fine for now, as we have not set that up yet.

   🤷 *Hence this is why that "login to admin" hyperlink on the website also points to nowhere right now.*

Stage 3 - Setting up CORS for your POST route

Time to update your POST route, so that once we have authentication ready (next lab). This way the website can actually communicate securely with our API.

The way we can ensure this is to bypass the simple POST request and ask the browser to send a pre-flight request (OPTIONS) before sending the actual request with credentials.

The server (in this case API GW) needs to respond with which domain(s) is/are allowed to make the request, and if it is safe to send credentials or not.

In your respective folder in Cloud 9 there is a file called `cors`. You should look at that and understand what is is doing for this POST API

⚠ **If you are using Java** the folder structure for java is different from for Node.js or Python. You'll need to navigate to the parent folder for the program to run. The parent folder name will correspond with the file name for the other languages. In this case, the file is named `cors` for node and Python, therefore the folder you should navigate to in the java folder is the `cors` folder. Under that folder you should see a `pom.xml` file - that is how you know you are in the right spot :)

You will see a method called `updateIntegrationCORS`. This is where most of the magic is happening.

Essentially when a browser sends its **pre-flight OPTIONS AJAX request** we want to check its ORIGIN domain and send back a BIG GREEN OK. Only then, does the browser know that its ok to send the REAL POST request.

Run the `cors` file as per the table below. This will update your POST API and ensure it is CORS powered, and ready for integration with your Authentication system (coming soon).

🤷 *If you are not in the right right folder in the `CMD_LINE`. Navigate to it now.*

For example if you are using Python use `cd python_3.6.8`

The following table shows the respective **run command** needed to run the `cors` file:

Language	Filename You Need To Edit	Command to run via `CMD_LINE`
Node.js (v10)	cors.js	`node cors.js`
Python (v3)	cors.py	`python3 cors.py`
Java	App.java	`mvn clean install && mvn exec:java -Dexec.mainClass=com.mycompany.app.App`

		`-Dexec.mainClass=com.mycompany.app.App`

Once you run the file in the `CMD_LINE` you should see (regardless of language) the word **done**.

⚠ *If you are using Java*, and you are seeing any compilation errors. Run the following command and try running the program again.

```
export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.252.b09-2.51.amzn1.x86_64
```

Just to wrap up today, please take a few moments to navigate around the API GW console, and see how the method and integration settings from both script have been applied. Get comfortable navigating around that console.

Try testing the OPTIONS API.

- Choose `/create_report` and then **OPTIONS**. Then choose **TEST**. Then **Test** again.

  You should see this in the body `no data`, and you should see the headers similar to the following:

  ```
  {"Access-Control-Allow-Origin":"https://c11284a122552u294892t1w432556616548-s3bucket-8yc67pytw9xk.s3-us-west-2.amazonaws.com","Access-Control-Allow-Credentials":"true","Access-Control-Allow-Methods":"POST,OPTIONS","Access-Control-Allow-Headers":"Content-Type,X-Amz-Date,Authorization,X-Api-Key,X-Amz-Security-Token","Content-Type":"application/json"}
  ```

  This is better now. It means that non-simple AJAX requests across domains to this POST request will encounter this during preflight. Essentially telling the browser it is OK to send the POST with credentials but only from your website's origin.

Awesome! You have your Kiosks communicating with the backend and retuning hard coded data, and you have future proofed your POST API ready for integration with Cognito.

Tomorrow (in the next lab). You are going to be setting up all that Authentication stuff with Cognito. Once you have that in place, then you can test the `create_report` route.

A good day, today, you've done well. Get a good nights sleep as tomorrow things get a lot more involved `<evil laugh>` 😈

Lab Complete

Congratulations! You have completed the lab.

1. Click **End Lab** at the top of this page and then click **Yes** to confirm that you want to end the lab.
2. A panel will appear, indicating that "DELETE has been initiated... You may close this message box now."
3. Click the **X** in the top right corner to close the panel.

For feedback, suggestions, or corrections, please contact us at:https://support.aws.amazon.com/#/contacts/aws-training