



SCHOOL OF INFOCOMM TECHNOLOGY

Diploma in Information Technology

Cloud Architecture and Technologies (CAT)

**April 2021 Semester
Assignment**

30% of CAT Module – (Team: 70%, Individual 30%)

5 July – 1 August 2021 (Weeks 12 & 15)

Deadline for submission:

SOFTCOPY: Submit in MeL by 1 August 2021, 23:59

Tutorial Group:	P03	Team Number:	3
Members	Student No.	Student Name	Grade
	S10203166	Marc Lim Liang Kiat	
	S10203190	Tan Hiang Joon Gabriel	
	S10204908	Kathiravan S/O Kalidoss	
	S10206093	Nicolas Teo Zhi Yong	
	S10203273	Nicholas Ian Boey	
	S10204620	Nicholas Chng Li Kien	

Contents

Availability Zones, Auto Scaling, EC2 Instance Configuration And Relational Database Service (RDS) (Nicolas Teo)	4
Elastic Compute 2 Creation	4
Autoscaling Group	4
Relational Database Service (RDS) using MSSQL	4
Connecting Relational Database Service to EC2 Instance Web application	4
Problems Encountered and resolutions implemented	5
Tools used to create/test infrastructure implemented	7
DynamoDB Table and Lambda Function to Read and Write to DynamoDB Database (Marc Lim)	9
DynamoDB Table Creation	9
IAM Roles and Policies	9
Lambda Functions and Configuration	10
API Gateway	11
Tools used to Create/Test Infrastructure Implemented	12
Problems Encountered and Resolutions Implemented	14
Lambda Function Access Privileges Problems	14
Cross-Origin Resource Sharing Header Problems (CORS)	15
JavaScript jQuery Library Problems	15
S3 Bucket and Lambda Function to Reduce Size of the Pictures Uploaded to S3 Bucket (Nicholas Chng)	16
Description	16
Problems Encountered and Resolutions Implemented	17
Code Used for the Lambda Function	17
Tools used to Create/Test Infrastructure Implemented	18
Virtual Private Cloud, Subnets, and Load Balancer (Gabriel Tan)	18
Description	18
VPC/Subnets	18
Load Balancer	19
Problems Encountered and Resolutions Implemented	19
Tools used to Create/Test Infrastructure Implemented	20

Simple Notification Service (SNS) to SMS Administrator - S3 bucket and DynamoDB table (Kathiravan)	21
Description	21
Execution	21
S3 bucket notifications	21
DynamoDB	22
Problems Encountered and Resolutions Implemented	24
Tools used to Create/Test Infrastructure Implemented	24
Full Website Design and Contact Us Form (Nicholas Boey)	25
Description	25
HTML Contact Form	25
Customer List Page	26
Navbar and Footer	26
Problems Encountered and Resolutions Implemented	27
Tools used to Create/Test Infrastructure Implemented	27

Detailed Description of solution implemented

The following sections describe and explain the different sections of the solution infrastructure implemented by each of the respective group members

Availability Zones, Auto Scaling, EC2 Instance Configuration And Relational Database Service (RDS) (Nicolas Teo)

Elastic Compute 2 Creation

I created Elastic Compute 2 instances through the use of a launch template named "Assg2-LaunchTemplate". It uses an AMI version of a modified Microsoft Windows Server 2019 Base where server configurations have already been done and the web application is already pre installed into the AMI. It uses an instance type of t2.micro. The security group allows all inbound traffic like http, https, sql, ssh and MSSQL.

Autoscaling Group

The "Assignment-2-EC2" launch template was used in an autoscaling group named, "Assignment-2-Autoscaling". The desired group size was set to 2, minimum group set to 2 and max group size set to 4. The VPC used was 'VPC-Assign2' and 2 subnets were used to allow the servers to be live in 2 availability zones. This allowed the servers to be resilient to natural disasters. The auto scaling policy metric type is set to 10000 bytes for average network in. This will scale the server up when average network usage is above 10000 bytes and scale the servers in when average network usage is below 10000 bytes.

Relational Database Service (RDS) using MSSQL

A relational database service was also used in conjunction with the Elastic Compute 2 services. MSSQL was used and the database used a db.t2.micro instance with a general purpose SSD of 20GiB. This Database is also in the same VPC as the EC2 instances. This is to connect the database to the web application that is being hosted by the EC2 instances.

Connecting Relational Database Service to EC2 Instance Web application

The web application used was done using Microsoft Dot Net Framework Razor pages. This was done by first remotely connecting to a Microsoft Windows Server EC2 instance, and enabling the Web Server (IIS) feature. A copy of the compiled razor pages, a set up of the Microsoft server management studio for the relational database and a dotnet windows hosting bundle was also downloaded into the EC2

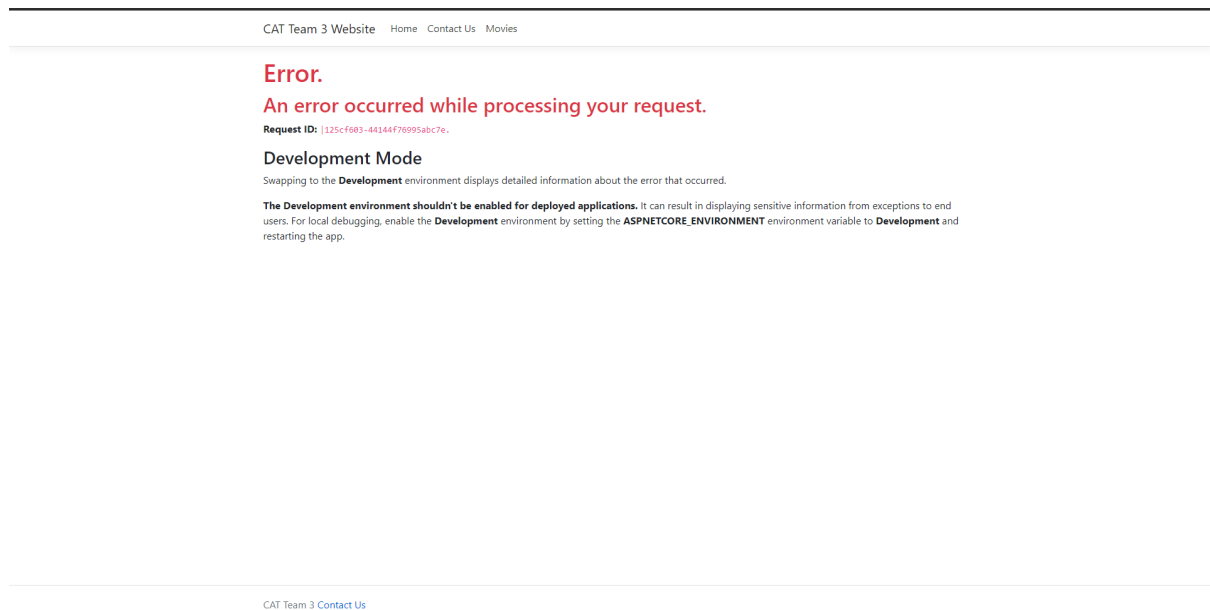
instance. The compiled razor pages were copied into the default web page directory. Next, the RDS server was connected to the razor web application through the connection string in the appsettings.json file.



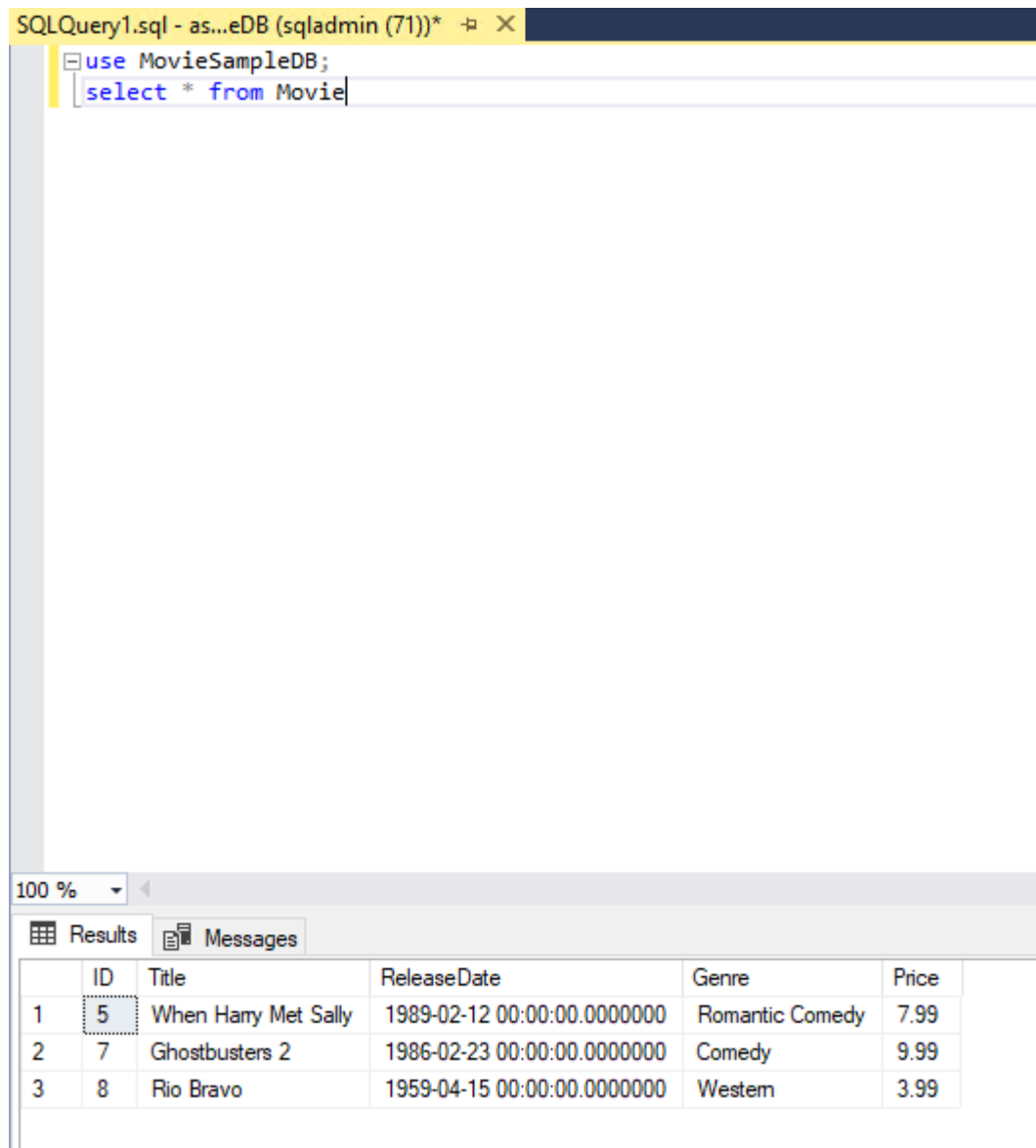
```
appsettings - Notepad
File Edit Format View Help
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "CAT_Team2_WebsiteContext": "Server=dbserver.coalczgzek.us-east-1.rds.amazonaws.com;Initial Catalog=MovieSampleDB;User ID=sqladmin;Password=Password123;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWrite;MultiSubnetFailover=False"
  }
}
```

This connects the website to the RDS the web application would display the contents of the Relational Database accordingly. When new records are added, deleted or modified, the changes would also be applied to the relational database.

Problems Encountered and resolutions implemented



One error I faced is connecting the RDS to the web application being hosted on the EC2 instance. Upon closer inspection of this error, it was an SQL error where there was no database found. The first thing I checked was the Relational Database and if the server was operational. I did this by connecting to the Microsoft SQL (MSSQL) server through Microsoft server management studio. From there I ran SQL queries to check if the server is online. The SQL Query ran and below is the resulting output.



This shows that the RDS server is online and working. Thus , I managed to isolate the issue and realise that the issue lied in my connection string in appsettings.json for the razor pages. This prompted me to check the connection string and I found an error in the security settings of the connection string. The username was capitalised when it was not necessary. After performing this fix and restarting the windows server, my razor pages ran and the website works as shown below.

Index

[Create New](#)

Title	ReleaseDate	Genre	Price	
When Harry Met Sally	2/12/1989	Romantic Comedy	7.99	Edit Details Delete
Ghostbusters 2	2/23/1986	Comedy	9.99	Edit Details Delete
Rio Bravo	4/15/1959	Western	3.99	Edit Details Delete

Tools used to create/test infrastructure implemented

One test I used to test my EC2 windows server instance was to remotely connect to it. By remotely connecting to it, it told me that the internet gateway, routing table and subnets were functional and that the EC2 was public.

Another test I did was to test the functionality of my Relational Database Microsoft SQL server. I did this by using the Microsoft server management studio and logging into my RDS server. From there, I ran a script to create tables and data values for the tables. The code ran with no issue and when I performed a simple select query on the resulting table, it returned me the data in the Relational database and this shows that the database is functional.

Another test I did was to test the functionality of the website. I did this by connecting to the website on my local machine through the ip address of the EC2 instance. This brought up the web application. To test that the web application and the Relational Database are connected, I added data into the web application which was then added into the RDS database.

← → 🔒 Not secure | 54.80.55.179/Movies

CAT Team 3 Website Home Contact Us Movies

Index

[Create New](#)

Title	ReleaseDate	Genre	Price	
When Harry Met Sally	2/12/1989	Romantic Comedy	7.99	Edit Details Delete
Ghostbusters 2	2/23/1986	Comedy	9.99	Edit Details Delete
Rio Bravo	4/15/1959	Western	3.99	Edit Details Delete
Test	7/8/2021	Test	2.34	Edit Details Delete

The example above shows that I added a new entry into the database with the title, “Test”. To ensure that this entry has been added into the database, I performed a simple select query on the resulting table and the result shows that the new entry has been added to the database.

SQLQuery1.sql - as...eDB (sqladmin (71))* 🔍 ✕

```
use MovieSampleDB;
select * from Movie
```

100 % 🔍

Results Messages

	ID	Title	ReleaseDate	Genre	Price
1	5	When Harry Met Sally	1989-02-12 00:00:00.0000000	Romantic Comedy	7.99
2	7	Ghostbusters 2	1986-02-23 00:00:00.0000000	Comedy	9.99
3	8	Rio Bravo	1959-04-15 00:00:00.0000000	Western	3.99
4	9	Test	2021-07-08 00:00:00.0000000	Test	2.34


DynamoDB Table and Lambda Function to Read and Write to DynamoDB Database (Marc Lim)

DynamoDB Table Creation

The DynamoDB Table I created is named Assignment_2-DynamoDB. It has 'name' as its partition key and 'email' as an added sort key for filtering customer entries in the database. DynamoDB Stream was also enabled as it was required to be used as a trigger for a lambda function which interacts with an AWS SNS topic to send an SMS to the administrator whenever an item is inserted into the database.

IAM Roles and Policies

In order for my lambda functions to read and write into the DynamoDB table, it has to be given permissions to interact and access the DynamoDB table through an AWS IAM role as the default role does not have read and write privileges. To do this, I created a new role called 'DynamoDB-Lambda' and attached a policy called 'DynamoDB-full-access' which allows any service with the role to perform all actions (including read, write, and list permissions) on all DynamoDB resources (tables) and gave the role to my lambda functions during configuration. An overview of the permissions given to the lambda functions is shown below:

Resource summary		View role document
<div> Amazon DynamoDB 11 actions, 6 resources</div>		
To view the resources and actions that your function has permission to access, choose a service.		
<div><div>By action</div><div>By resource</div></div>		
Resource	Actions	
All resources	Allow: dynamodb:ListBackups Allow: dynamodb:DescribeReservedCapacity Allow: dynamodb:ListExports Allow: dynamodb:DescribeLimits Allow: dynamodb:DescribeReservedCapacityOfferings Allow: dynamodb:ListTables Allow: dynamodb:ListStreams Allow: dynamodb:ListContributorInsights Allow: dynamodb:ListGlobalTables Allow: dynamodb:PurchaseReservedCapacityOfferings	
arn:aws:dynamodb::361624559862:global-table/*	Allow: dynamodb:*	
arn:aws:dynamodb:*:361624559862:table/*/backup/*	Allow: dynamodb:*	
arn:aws:dynamodb:*:361624559862:table/*/stream/*	Allow: dynamodb:*	
arn:aws:dynamodb:*:361624559862:table/*/export/*	Allow: dynamodb:*	
arn:aws:dynamodb:*:361624559862:table/*	Allow: dynamodb:*	

Lambda Functions and Configuration

I created two Lambda functions called 'Read-from-DynamoDB' and 'Write-to-DynamoDB' to read and write to the DynamoDB table respectively. These two functions were configured with the 'DynamoDB-Lambda' role with full DynamoDB access privileges and a Restful API (covered in the next section) as a trigger to launch them.

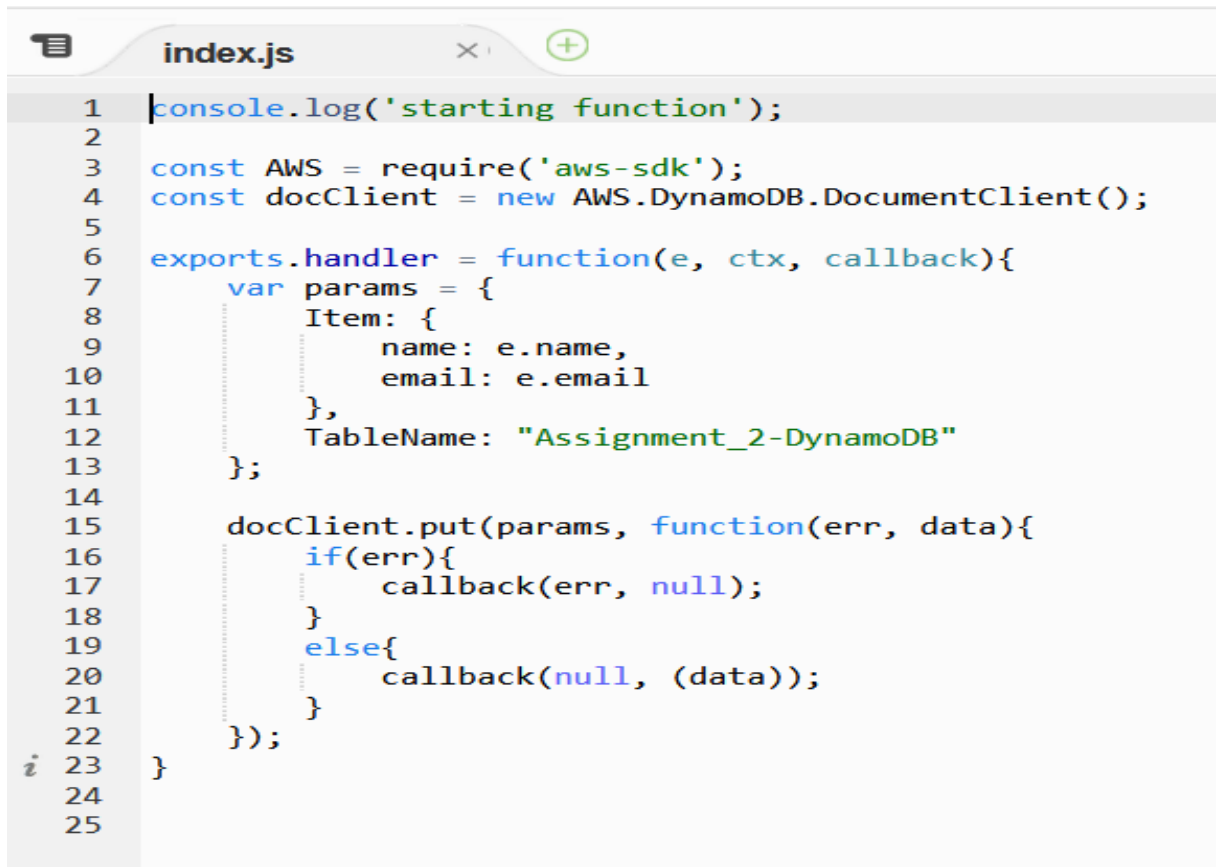
The codes for the Lambda functions are below:

- **Lambda Read from DynamoDB function**



```
index.js
1 console.log('starting function')
2
3 const AWS = require('aws-sdk');
4 const docClient = new AWS.DynamoDB.DocumentClient();
5
6 exports.handler = function(e, ctx, callback){
7     let scanningParameters = {
8         TableName: "Assignment_2-DynamoDB",
9         Limit: 100
10    };
11
12    docClient.scan(scanningParameters, function(err, data){
13        if(err){
14            callback(err, null);
15        }
16        else{
17            callback(null, data)
18        }
19    });
20 }
```

- **Lambda Write to DynamoDB Function**

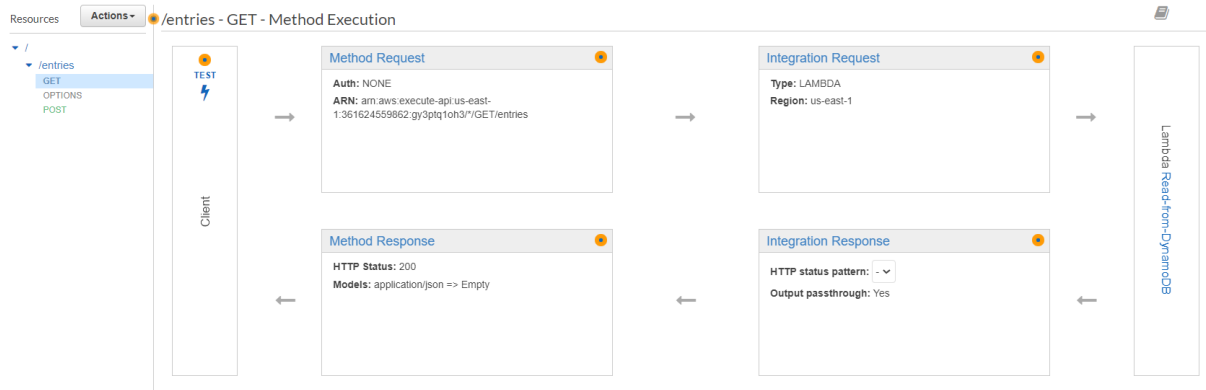
A screenshot of a code editor window titled 'index.js'. The code is written in JavaScript and implements a Lambda function handler that writes data to a DynamoDB table. The code is as follows:

```
1 console.log('starting function');
2
3 const AWS = require('aws-sdk');
4 const docClient = new AWS.DynamoDB.DocumentClient();
5
6 exports.handler = function(e, ctx, callback){
7     var params = {
8         Item: {
9             name: e.name,
10            email: e.email
11        },
12        TableName: "Assignment_2-DynamoDB"
13    };
14
15    docClient.put(params, function(err, data){
16        if(err){
17            callback(err, null);
18        }
19        else{
20            callback(null, (data));
21        }
22    });
23 }
24
25
```

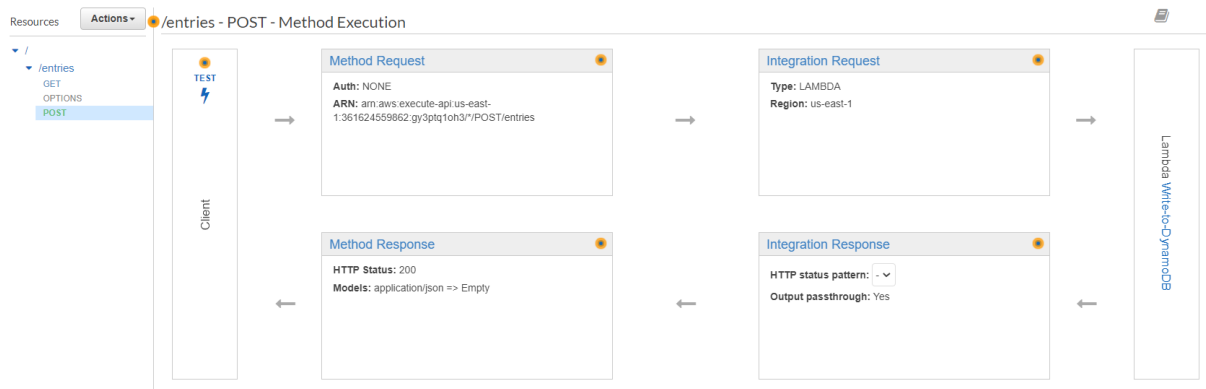
API Gateway

Additionally, since the Lambda functions to read and write into the DynamoDB table needed to be accessed through a website, I needed to use AWS API Gateway to create a RESTful API which will be used to grant our website access to AWS services in order for the captured data from the 'Contact Us' form on the website to be passed into my Lambda functions and for the entries in the DynamoDB table to be passed to the website for display to the web users. To do this, I added GET and POST Methods to the RESTful API and integrated them with the 'Read-from-DynamoDB ' and 'Write-to-DynamoDB' Lambda functions respectively. Furthermore, I had to manually add a mapping template to the Integration Request of the POST method which maps the captured data from the 'Contact Us' form on the website to the variables assigned in the 'Write-to-DynamoDB' lambda function which are mandatory for the function to work. The overviews of the API GET and API POST methods are shown below:

- API GET Method



- API POST Method



Tools used to Create/Test Infrastructure Implemented

In order to test the RESTful API and check whether the data from my group's website would be passed to and from AWS correctly when I integrated my work with the website after it was finished being developed, I coded my own mock website in HTML using Microsoft Visual Studio with a 'Contact Us' form and a 'Customer List' that would list all the entries in the 'Assignment_2_DynamoDB' table. Additionally, I also had to write code in JavaScript using the jQuery Library to GET and POST data to the DynamoDB table through the Restful API which triggers the lambda functions to read and write to the DynamoDB table. These JavaScript codes were later imported and used in the final website designed by my groupmates. The HTML code and JavaScript functions to GET and POST data to the DynamoDB table are below:

- **Visual Studio HTML Code**

```
1 <!DOCTYPE html>
2
3 <html lang="en" xmlns="http://www.w3.org/1999/xhtml">
4 <head>
5     <meta charset="utf-8" />
6     <title></title>
7 </head>
8 <body>
9     <h1>Customer List</h1>
10    <div id="entries">
11    </div>
12    <form>
13        <h1>Enter New Customer Details</h1>
14        <label for="name">name:</label>
15        <input type="text" id="name" placeholder="Your Name" />
16        <label for="email">email:</label>
17        <input type="email" id="email" placeholder="Your Email" />
18        <button id="submitButton">Submit</button>
19    </form>
```

- **JavaScript GET Function**

```

20 <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
21 <script type="text/javascript">
22     var api_url = 'https://gy3ptq1oh3.execute-api.us-east-1.amazonaws.com/Production/entries';
23     $(document).ready(function () {
24         $.ajax({
25             type: 'GET',
26             url: api_url,
27             success: function (data) {
28                 count = 1;
29                 console.log(data);
30                 $('#entries').html('');
31                 data.Items.forEach(function (customerItem) {
32                     $('#entries').append('<br>' + '<p>' + "Customer Number: " + count + '<p>');
33                     $('#entries').append('<p>' + "Customer Name: " + customerItem.name + '<p>');
34                     $('#entries').append('<p>' + "Customer Email: " + customerItem.email + '<p>');
35                     count += 1;
36                 })
37             }
38         });
39     });

```

- **JavaScript POST Function**

```
40      $('#submitButton').on('click', function () {
41          var name = $('#name').val()
42          var email = $('#email').val()
43          data = JSON.stringify({ "name": name, 'email': email });
44          console.log(name);
45          console.log(email);
46          console.log(data);
47          $.ajax({
48              type: 'POST',
49              url: api_url,
50              //headers: {
51                  //"Access-Control-Allow-Origin": "*",
52              //},
53              data: data,
54              contentType: "application/json",
55              success: function (data) {
56                  console.log('success');
57                  location.reload();
58              }
59          });
60          return false;
61      });
62
63  </script>
```

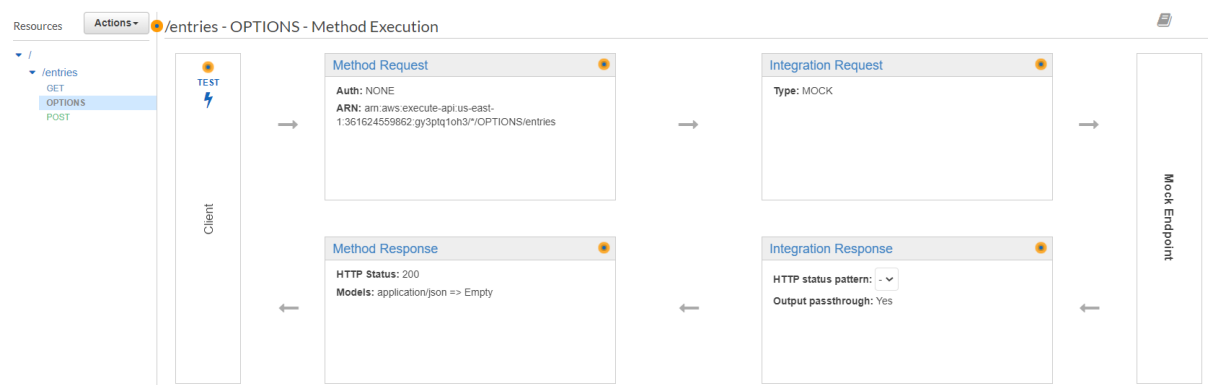
Problems Encountered and Resolutions Implemented

Lambda Function Access Privileges Problems

During the coding process of my Lambda functions, my code was not working as intended. I later learnt that the reason was that my Lambda functions were using the default role created by AWS which did not have the privileges to read or write to DynamoDB. To resolve this problem, I created a new role with policies that allowed all actions on all resources in DynamoDB as previously stated in the “IAM Roles and Policies” section.

Cross-Origin Resource Sharing Header Problems (CORS)

During the testing of the RESTful API which sends the data to my Lambda Functions, I ran into an error regarding missing CORS headers in the HTTP request sent to and from my website when sending data through the RESTful API. This error stopped the website from being able to access any of the data in the DynamoDB table through the API which stopped the website from working as intended. To resolve this problem, I had to enable and configure CORS on the AWS API Gateway console which added an OPTIONS method to the RESTful API which handles the insertion of the CORS headers into the HTTP request to and from the API. The overview of the API OPTIONS method is shown below:



JavaScript jQuery Library Problems

During the coding process of the back-end JavaScript functions used in the final website mentioned in the 'Tools Used to Create/Test Infrastructure implemented', I ran into syntax errors in my code. I later learnt that this was due to the version of the jQuery library I imported as the version I had installed was the 'slim' version that did not have the '\$.ajax' built-in function necessary for my code to work properly. To resolve this problem, I imported the correct version of jQuery from google which hosts them at <https://developers.google.com/speed/libraries>

S3 Bucket and Lambda Function to Reduce Size of the Pictures Uploaded to S3 Bucket (Nicholas Chng)

Description

An S3 bucket needs to be created that stores pictures. When pictures are uploaded to the S3 bucket, an SMS is sent to the administrator. The pictures in the bucket are resized and stored in another bucket as thumbnails.

S3 Buckets

Two S3 Buckets called “asg2bucket” and “asg2bucket-resized” are created. Pictures will be uploaded to the “asg2bucket” bucket and subsequently resized via a lambda function and placed into the second bucket “asg2bucket-resized”. The reasoning for the two buckets is to prevent recursion from happening, in the case that the lambda function is triggered for a picture that has already been resized, leading more resized pictures to be created.

S3 Configuration

Both S3 Buckets are public, with the following bucket policies:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PublicRead",
      "Effect": "Allow",
      "Principal": "*",
      "Action": [
        "s3:GetObject",
        "s3:GetObjectVersion"
      ],
      "Resource": "arn:aws:s3:::asg2bucket/*"
    }
  ]
}
```

An event notification is set up for the S3 bucket “asg2bucket”, which notifies when an object is uploaded to the bucket.

IAM Role

An IAM Role called “LambdaRole3” is created, with permission policies that are crucial for the interaction between the S3 bucket and the lambda functions. The policies attached are AmazonS3FullAccess, AWSLambdaBasicExecutionRole and AWSLambda_FullAccess. The Full Access permissions are to ensure there are no missing permissions when the lambda function is running. The policy AWSLambdaBasicExecutionRole is attached in order for the function to leave CloudWatch logs for debugging purposes.

Lambda Function

A lambda function named “thumbnail” is created. The function has the execution role “LambdaRole3”, with timeout of five seconds and 1024MB of memory to perform the lambda function. An S3 Trigger is used to trigger the lambda function, with an ObjectCreated event being the specific trigger. Python 3.7 is used for the lambda function, with the handler CreateThumbnail.handler.

Problems Encountered and Resolutions Implemented

One main problem encountered when implementing the solution was deployment packages. Since amazon uses linux OS for the servers that house the lambda function, packages that are needed in the lambda function must be compiled on a linux system. Without a linux system to compile the modules, it was difficult to make a deployment package without using virtual machines. In the end, the resolution was to find pre-compiled modules on the internet, and download it to use in the deployment package.

Another problem encountered was incorrect versioning of modules. For example, the PIL package in Python was of an incorrect version when the function was run. The resolution employed was to downgrade the version of Python that the lambda function was running, in this case 3.8 to 3.7.

Code Used for the Lambda Function

CreateThumbnail.py

```

import boto3
import os
import sys
import uuid
from PIL import Image
import PIL.Image

s3_client = boto3.client('s3')

def resize_image(image_path, resized_path):
    with Image.open(image_path) as image:
        image.thumbnail((128, 128))
        image.save(resized_path)

def handler(event, context):
    for record in event['Records']:
        bucket = record['s3']['bucket']['name']
        key = record['s3']['object']['key']
        download_path = '/tmp/{}'.format(uuid.uuid4(), key)
        upload_path = '/tmp/resized-{}'.format(key)

        s3_client.download_file(bucket, key, download_path)
        resize_image(download_path, upload_path)
        s3_client.upload_file(upload_path, '{}-resized'.format(bucket), key)

```

The deployment package including the above code was taken from <https://austinlasseter.medium.com/resize-an-image-using-aws-s3-and-lambda-fda7a6abc61c>

Tools used to Create/Test Infrastructure Implemented

Amazon CloudWatch was used to test and debug the infrastructure. The testing process included uploading images to the “asg2bucket” S3 bucket, checking the “asg2bucket-resized” S3 bucket, and going through the CloudWatch logs to find the errors.

Virtual Private Cloud, Subnets, and Load Balancer (Gabriel Tan)

Description

VPC/Subnets

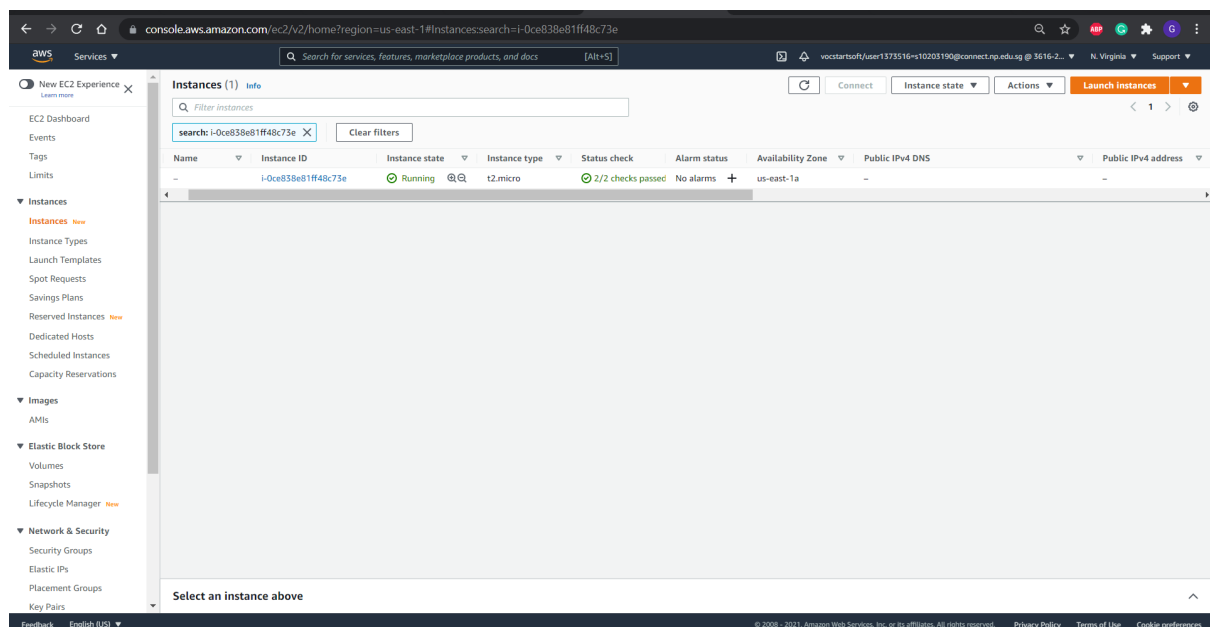
Before setting up the EC2 instances, first we need a Virtual Private Cloud(VPC) that we can use to set the EC2 instances up in. The VPC would allow our resource to be logically isolated from other VPCs in AWS. For the VPC, ‘VPC-Assign2’ is used. I used 10.0.0.0/16 for the IPv4 CIDR block. As for the subnets, i made 2 public subnets, ‘assg2 Public subnet 1’ and ‘assg2 Public subnet 2’, with the IPv4 CIDR

10.0.0.0/24 and 10.0.1.0/24 and in the Availability Zones us-east-1a and us-east-1b respectively. I then made a route table, 'assg2 route table', adding the destination 0.0.0.0/0 with the target being the internet gateway, 'assg2 igw', to the routes and the 2 subnets to the explicit subnet associations.

Load Balancer

In order to be resilient, I set up a Network Load Balancer(NLB) to distribute the network load evenly amongst the EC2 instances so that no one instance bears the entire load of the network at any one time. The NLB, 'Assignment-2-NLB', uses the internet-facing scheme and listens to TCP traffic on port 80. The NLB is connected to the VPC created earlier and all the availability zones that the subnets are in which would be used by the EC2 instances. A target group, 'Assignment-2-NLB-targetgroup', is made for the listener to route requests to the targets, the EC2 instances, using TCP as the protocol, port 80, and target type as instance. Once the instances are registered to the target group they would initialise, after which the statuses would turn healthy meaning that the NLB has been set up and is ready.

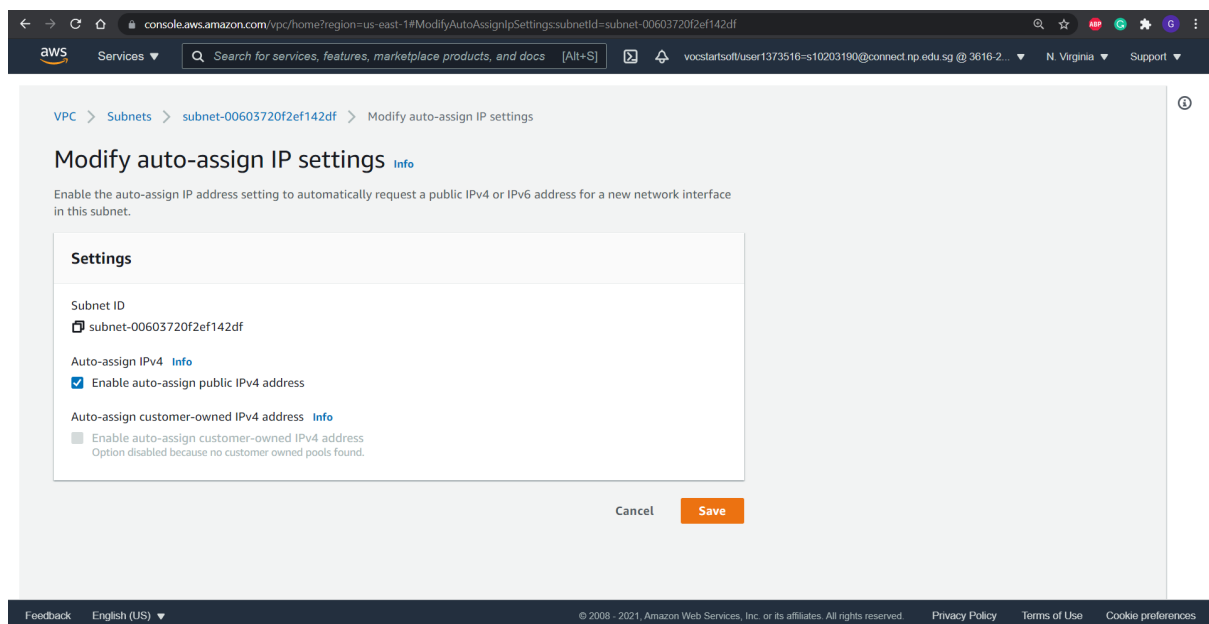
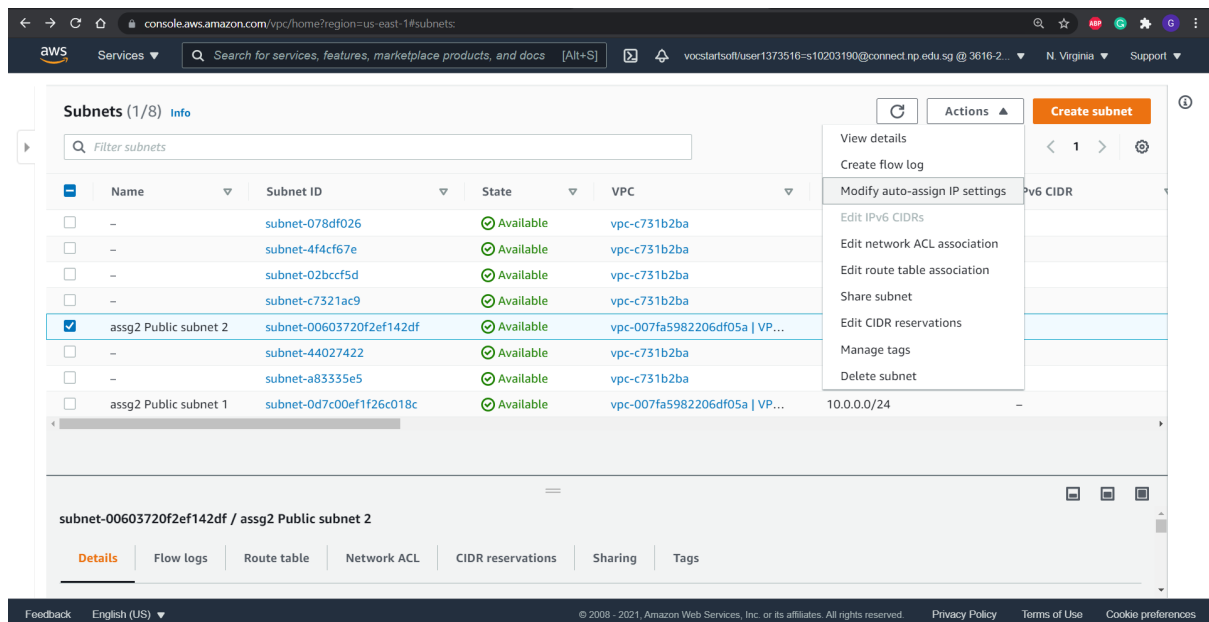
Problems Encountered and Resolutions Implemented



When setting up the EC2 instances with the VPC with Nicolas Teo, the instances did not have their Public IPv4 DNS and addresses.

After looking through AWS documentation and online forums, I found that I did not enable the auto-assignment of IPv4 addresses on the subnets.

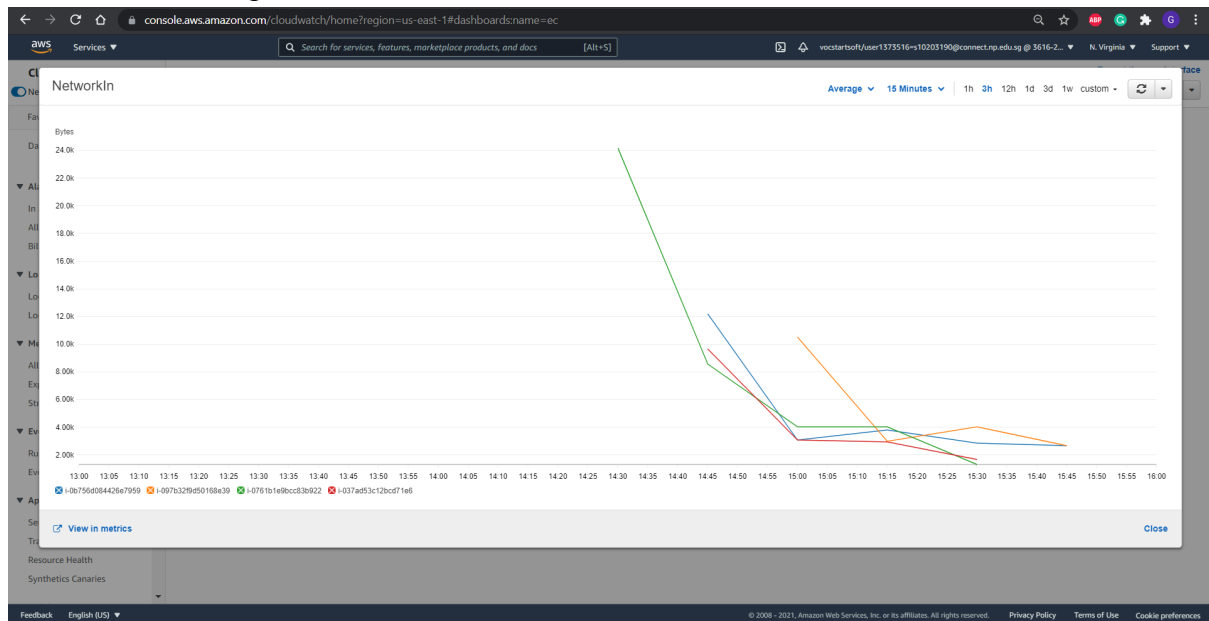
I changed the auto-assign IP settings on both subnets to enable auto-assign public IPv4 address.



Tools used to Create/Test Infrastructure Implemented

Upon successful setup, opening the DNS name, Assignment-2-NLB-be1864a53c790956.elb.us-east-1.amazonaws.com, in the load balancer's description would bring you to the default page of the website. When a load is put on the network, the load would be spread across the EC2 instances, this

can be seen using AWS Cloudwatch.



Simple Notification Service (SNS) to SMS Administrator - S3 bucket and DynamoDB table (Kathiravan)

Description

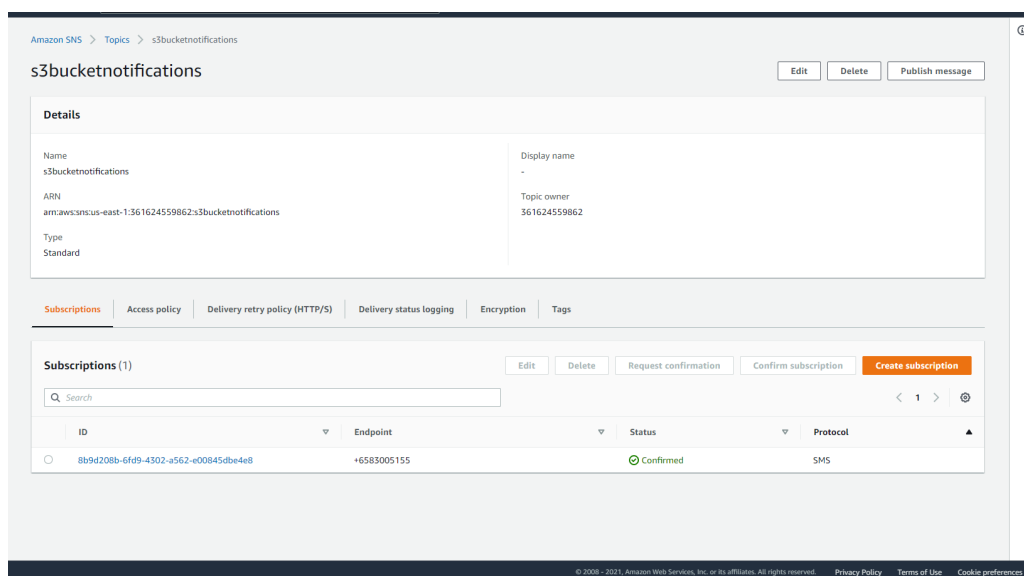
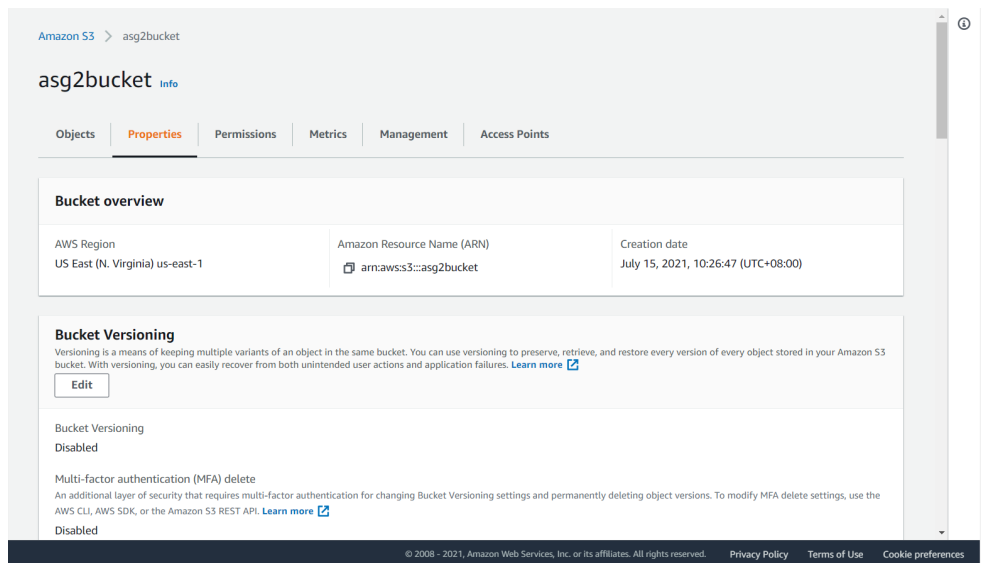
I had to come up with a configuration where an SMS will be sent to the administrator's phone number everytime a picture gets uploaded to the S3 bucket and every time a new record is created in the DynamoDB database.

Execution

S3 bucket notifications

An S3 bucket called asg2bucket was first created. Following that, an SNS topic called s3bucketnotifications was created for SNS to send a message when there were uploads to the S3 bucket. After the creation of the SNS topic, the access policy of the SNS needed to be changed. The JSON code used for the access policy is shown below.

```
{
  "Version": "2012-10-17",
  "Id": "example-ID",
  "Statement": [
    {
      "Sid": "example-statement-ID",
      "Effect": "Allow",
      "Principal": {
        "Service": "s3.amazonaws.com"
      },
      "Action": [
        "SNS:Publish"
      ],
      "Resource": "SNS-topic-ARN",
      "Condition": {
        "ArnLike": { "aws:SourceArn": "arn:aws:s3:*:*:bucket-name" },
        "StringEquals": { "aws:SourceAccount": "bucket-owner-account-id" }
      }
    }
  ]
}
```



The template for this code was taken from <https://docs.aws.amazon.com/AmazonS3/latest/userguide/ways-to-add-notification-config-to-bucket.html>. I replaced the “SNS-topic-ARN” with the ARN of the topic I am configuring the access policy on. I replaced “bucket-name” with the name of the S3 bucket linking to this topic, as well as replacing “bucket-owner-account-id” with the account id of the account this infrastructure was created on. The topic was subscribed to the administrator’s phone number with the protocol being SMS. All this information can be retrieved from the images shown above.

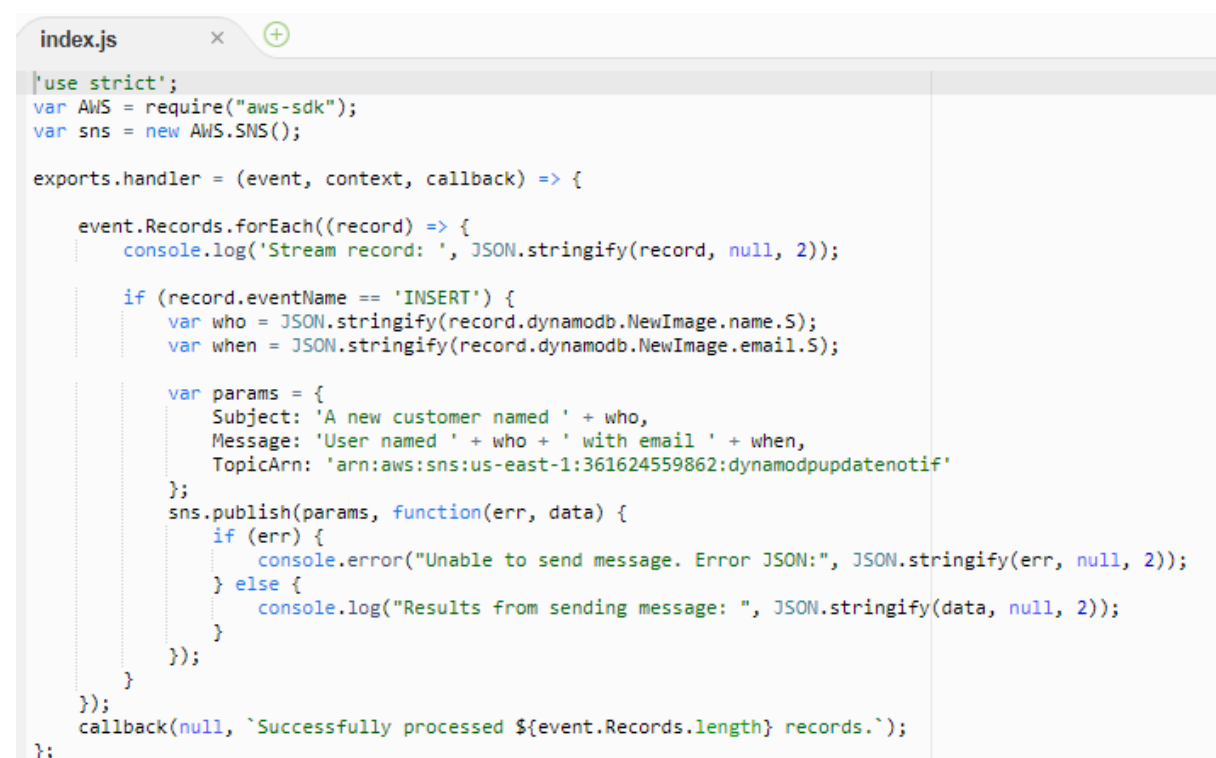
Afterwards, I had to go back to the S3 bucket, and under properties, go to Event Notifications. Under Event Notifications, I created an event notification. Under the suffix section, I entered .jpg and .png. .jpeg and .jpg are interchangeably used,

hence was not added. During testing, when a .jpeg file was added, the SMS was sent. Under event types, I selected the put method for object creation. The destination of the event was set to SNS topic, with the ARN being that of the SNS topic ARN for S3.

DynamoDB

As for DynamoDB, it was not that straightforward. Firstly, the SNS topic called dynamodpupdatenotif was to be created. Then, a subscription of the phone number was added with protocol being SMS. Following that, streams was enabled for the table in the database. The lambda function is the key ingredient in this feature. In order for the lambda function to access DynamoDB and SNS, it will require some permissions. With SNS, the access policy was edited. However for lambda, we will need to provide the permissions using an IAM role.

The IAM role is called DynamoDBxLambdaIAMRole. The policies assigned to this role are AWSLambdaDynamoDBExecutionRole, AWSLambdaInvocation-DynamoDB, AmazonSNSFullAccess, and finally AWSLambda_FullAccess. When creating the lambda function, this role was assigned to it. After creation, the code was edited to the one shown below.



```
index.js
'use strict';
var AWS = require("aws-sdk");
var sns = new AWS.SNS();

exports.handler = (event, context, callback) => {

  event.Records.forEach((record) => {
    console.log('Stream record: ', JSON.stringify(record, null, 2));

    if (record.eventName == 'INSERT') {
      var who = JSON.stringify(record.dynamodb.NewImage.name.S);
      var when = JSON.stringify(record.dynamodb.NewImage.email.S);

      var params = {
        Subject: 'A new customer named ' + who,
        Message: 'User named ' + who + ' with email ' + when,
        TopicArn: 'arn:aws:sns:us-east-1:361624559862:dynamodpupdatenotif'
      };
      sns.publish(params, function(err, data) {
        if (err) {
          console.error("Unable to send message. Error JSON:", JSON.stringify(err, null, 2));
        } else {
          console.log("Results from sending message: ", JSON.stringify(data, null, 2));
        }
      });
    }
  });
  callback(null, `Successfully processed ${event.Records.length} records.`);
};
```


The template for the code was taken from <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Streams.Lambda.Tutorial.html>. In this website, the Command Line Interface (CLI) was used to interact with the console. Since we do not have access to the CLI, I manually configured the SNS topic, IAM role, and enabled streams on the DynamoDB table. Then, I deployed the code and went back to DynamoDB.

In the table configurations of the DynamoDB table, I enabled the trigger by navigating to the trigger section of the table and added an existing lambda function, which was the lambda function mentioned above.

To summarise what I have done above, the lambda function uses the stream enabled in the DynamoDB table to read the updates. It will take the values entered as parameters and form a message. This message will then be sent to the SNS topic, to be sent out via the subscription as an SMS.

Problems Encountered and Resolutions Implemented

There were several problems encountered during implementation. One of which had to do with the permissions given to the lambda function. The IAM role needs the proper permissions for the lambda function to make use of the DynamoDB stream and SNS topic. This was mainly because the documentation made use of creating an IAM policy. However, we are not given permission to create policies using our accounts. Hence, there was a need to manually choose the permissions to be given to the lambda function.

During the testing phase, there were several glitches and errors encountered. After adding the trigger to the DynamoDB table, a dummy entry was added to the table. However, the record was not being processed. When I went back to the lambda page, the trigger was not shown. Upon reloading the page, the trigger was then shown. A dummy entry was then entered again in the DynamoDB table. The record was then processed and the message was sent.

Tools used to Create/Test Infrastructure Implemented

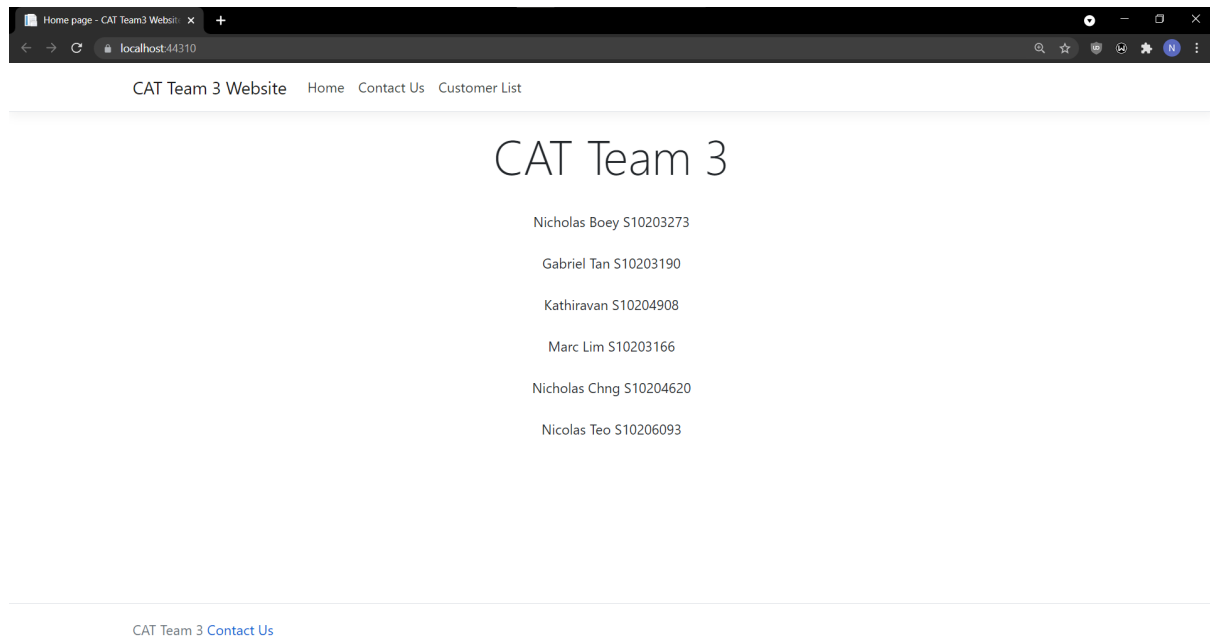
For the lambda function, Amazon Cloudwatch was used to track the progress. Once the record was entered into the table, I would first go to the triggers tab and look for the value under "Last result". If the record was processed successfully, the value will be "OK" instead of "No records processed". Following that, I would go to the lambda

function and under the monitor tab, there would be a link to the log in Cloudwatch. Using the log, I can check if there were any errors in executing the code and what those errors were. After checking all this, I would look at my phone, and the SMS would have arrived by then.

Full Website Design and Contact Us Form (Nicholas Boey)

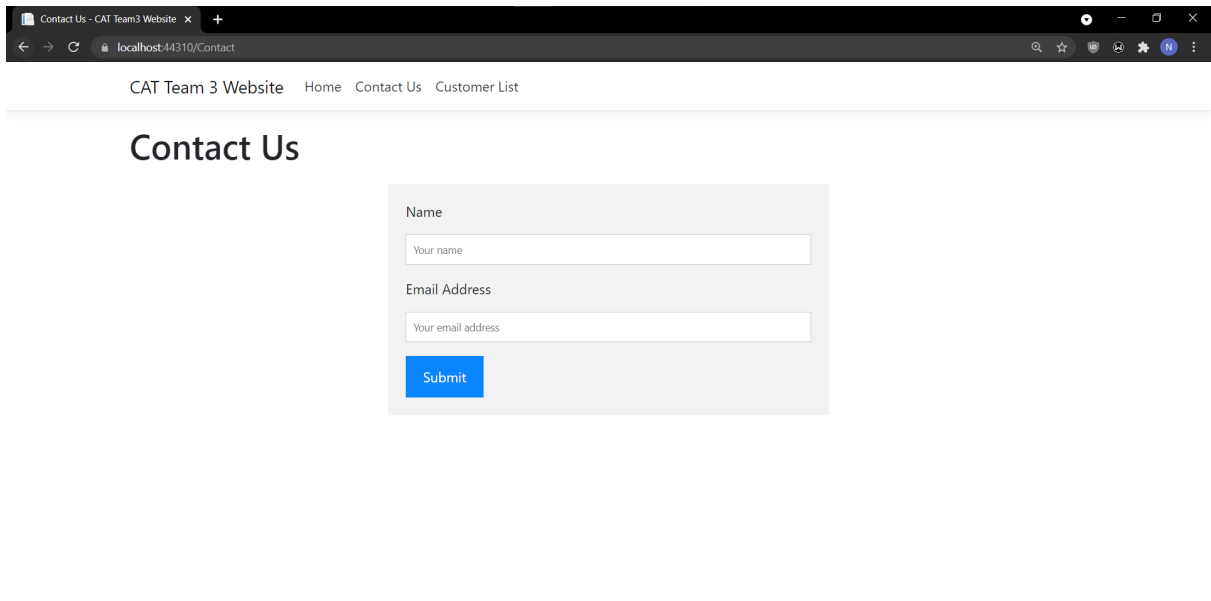
Description

I first created an ASP.NET Core Web Application using version 3.1 of the netcoreapp target framework on Microsoft Visual Studio. This created a basic Razor Pages website. I added some basic HTML code using the <p> tag to list the members of the group, as well as their corresponding Student IDs on the homepage of the website.



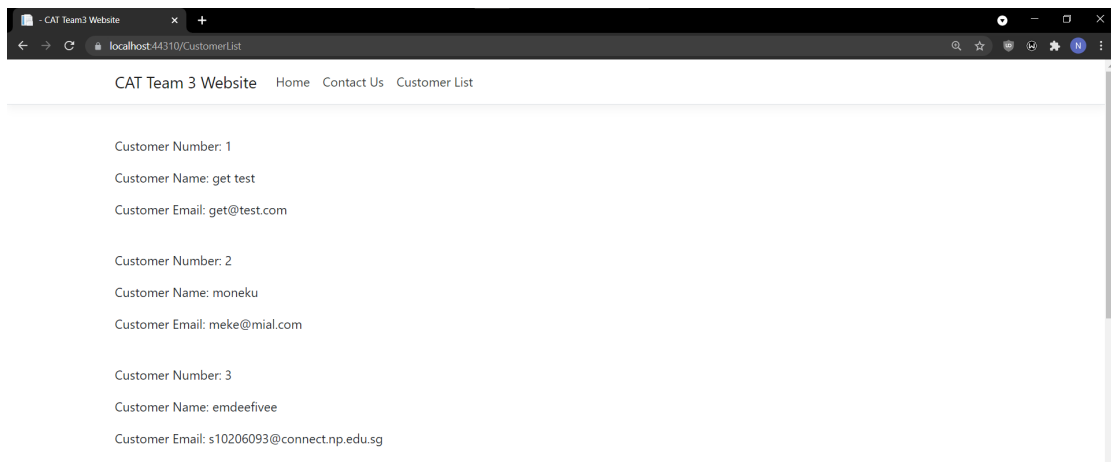
HTML Contact Form

I created a new Razor Page, named Contact.cshtml. In it, I created a HTML Form to collect a user's name and email address. It also includes a submit button. When the submit button is clicked, the JavaScript POST function is run, which sends the information to the API Gateway, which passes it through the Lambda function and finally gets written to the DynamoDB table.



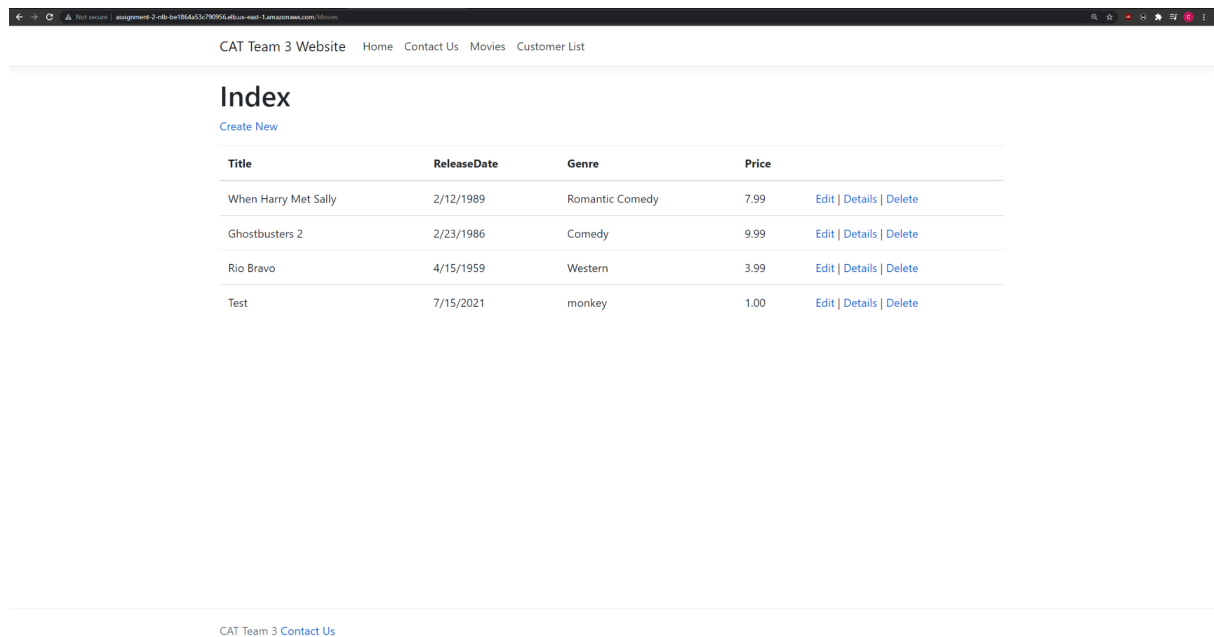
Customer List Page

This page displays the entries of the DynamoDB table using a Javascript GET function. This page is mainly for presentation purposes, to show that both the POST and GET functions are working as intended, as it would pose a security hazard by displaying sensitive contact information to the entire internet.



Movies Page

This page was created so that the website could interact with Amazon RDS. It lists a few movies along with details on each movie. When a new movie is created, the movie is added into the Amazon RDS, and will be displayed on the website until it is deleted.

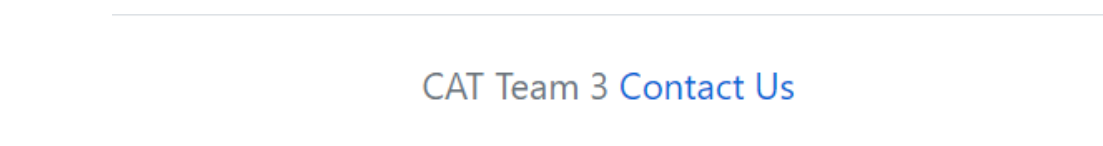


Navbar and Footer

The navbar contains 4 hyperlinks, to the home page, contact form, movies page, and the customer list page.



The footer mentions the team number, as well as including a hyperlink to the Contact page



Problems Encountered and Resolutions Implemented

Since my webpage would have to be hosted by the EC2 instance created by my teammate, Nicolas Teo, I worked closely with him to ensure minimal obstacles in the implementation of my code onto his created instance. This meant that I used Razor Pages, as he had specified that this would be the easiest method to implement the webpage code.

My teammate, Marc Lim, was responsible for the creation of the JavaScript code used to send the form data to our API Gateway, so that it can be passed into the

Lambda function which would write the data to the DynamoDB table. However, he created his JavaScript code with a regular HTML project in Visual Studio.

When first implementing the JavaScript code into the Razor Pages code, both the GET function and POST function did not work. I later found that this was due to some missing HTML tags. For example, regular HTML projects in Visual Studio come with a <body> tag by default, while Razor Pages does not. This seemed very minor when first implementing the JavaScript code, but the scripts did not work unless in a <body> tag.

Tools used to Create/Test Infrastructure Implemented

I used Visual Studio for the creation of the Razor Pages.

```
@page
@model IndexModel
@{
    ViewData["Title"] = "Home page";
}

<div class="text-center">
    <h2 class="display-4">CAT Team 3</h2>
    <p><br />Nicholas Boey S10203273<br /><br />Gabriel Tan S10203190<br />
    <br />Kathiravan S10204908<br /><br />Marc Lim S10203166<br />
    <br />Nicholas Chng S10204620<br /><br />Nicolas Teo S10206093</p>
</div>
```

html for home page

```
<div class="contactform">
    <form>
        <label for="name">Name</label>
        <input type="text" id="name" name="name" placeholder="Your name">
        <label for="email">Email Address</label>
        <input type="email" id="email" name="email" placeholder="Your email address">
        <input type="submit" id="submitButton">
    </form>
</div>
```

html form

```

72
73 input[type=text], input[type=email]{
74     width: 100%;
75     padding: 8px;
76     border: 1px solid #ccc;
77     box-sizing: border-box;
78     margin-top: 6px;
79     margin-bottom: 16px;
80     resize: vertical;
81     font-size: 12px;
82 }
83
84 input[type=submit] {
85     background-color: #0A84FF;
86     color: white;
87     padding: 12px 20px;
88     border: none;
89     cursor: pointer;
90 }
91
92 input[type=submit]:hover {
93     background-color: #0862bd;
94 }
95
96 .contactform {
97     background-color: #f2f2f2;
98     padding: 20px;
99     margin-left: 300px;
100    margin-right: 300px;
101    margin-top: 20px;
102 }
103
104 .label {
105     font-size: 13px;
106 }
107
108 ::-webkit-input-placeholder {
109     font-size: 12px;
110 }

```

css for html form

```

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
<script type="text/javascript">
    var api_url = 'https://gy3ptq1oh3.execute-api.us-east-1.amazonaws.com/Production/entries';
    $('#submitButton').on('click', function () {
        var name = $('#name').val()
        var email = $('#email').val()
        data = JSON.stringify({ "name": name, 'email': email });
        console.log(name);
        console.log(email);
        console.log(data);
        $.ajax({
            type: 'POST',
            url: api_url,
            //headers: {
            //    "Access-Control-Allow-Origin": "*",
            //},
            data: data,
            contentType: "application/json",
            success: function (data) {
                console.log('success');
                location.reload();
            }
        });
        return false;
    });
</script>

```

POST function to send data to API URL

```

<html>
<head>
    <meta charset="utf-8" />
    <title>Customer List</title>
</head>
<body>
    <div id="entries"></div>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
    <script type="text/javascript">
        var api_url = 'https://gy3ptq1oh3.execute-api.us-east-1.amazonaws.com/Production/entries';
        $(document).ready(function () {
            $.ajax({
                type: 'GET',
                url: api_url,
                success: function (data) {
                    count = 1;
                    console.log(data);
                    $('#entries').html('');
                    data.Items.forEach(function (customerItem) {
                        $('#entries').append('<br>' + '<p>' + "Customer Number: " + count + '<p>');
                        $('#entries').append('<p>' + "Customer Name: " + customerItem.name + '<p>');
                        $('#entries').append('<p>' + "Customer Email: " + customerItem.email + '<p>');
                        count += 1;
                    })
                }
            });
        });
    </script>
</body>
</html>

```

Entire code for customer list page, including GET function to list customer info


```

<header>
  <nav class="navbar navbar-expand-sm navbar-togglerable-sm navbar-light bg-white border-bottom box-shadow mb-3">
    <div class="container">
      <a class="navbar-brand" asp-area="" asp-page="/Index">CAT Team 3 Website</a>
      <button class="navbar-toggler" type="button" data-toggle="collapse" data-target=".navbar-collapse" aria-controls="navbarSupportedContent"
        aria-expanded="false" aria-label="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
      </button>
      <div class="navbar-collapse collapse d-sm-inline-flex flex-sm-row-reverse">
        <ul class="navbar-nav flex-grow-1">
          <li class="nav-item">
            <a class="nav-link text-dark" asp-area="" asp-page="/Index">Home</a>
          </li>
          <li class="nav-item">
            <a class="nav-link text-dark" asp-area="" asp-page="/Contact">Contact Us</a>
          </li>
          <li class="nav-item">
            <a class="nav-link text-dark" asp-area="" asp-page="/CustomerList">Customer List</a>
          </li>
        </ul>
      </div>
    </div>
  </nav>
</header>

```

HTML code for navbar

```

footer class="border-top footer text-muted">
  <div class="container">
    CAT Team 3 <a asp-area="" asp-page="/Contact">Contact Us</a>
  </div>
</footer>

```

HTML Code for Footer

Appendix

Appendix A

Website: [Assignment-2-NLB-be1864a53c790956.elb.us-east-1.amazonaws.com](https://assignment-2-nlb-be1864a53c790956.elb.us-east-1.amazonaws.com)

Setup on: S10203190@connect.np.edu.sg AWS account