Giftcard

Documentation

Table of Contents

1.0 Introduction

GG Cards or short for Game Gift Cards, is a web application that offers users the chance to buy game gift cards for games that are available in the web application. This web app aims to provide a convenient and seamless way for users to purchase gift cards for their games.

With the web application being powered by AWS, the platform is able to be reliable, scalable and secure. Using AWS's microservices, the application is able to be accessed by users from their web browser and able to purchase gift cards. Some AWS services are dedicated to the functionality of the web application, such as SNS which are used to notify users of the new games available in the web app, RDS used to store data of the web application, and S3 to store images of the games. While other AWS services like CloudWatch, Elastic Beanstalk, Lambda, and API Gateway are used for the development of the web application. CloudWatch is used to monitor the real-time performance and health of the AWS resources and applications. Elastic Beanstalk is a PaaS or platform-as-a-service used to deploy and manage the web app. Lambda is to run codes without managing servers and API gateway is to deploy the APIs that are connected to the AWS services to the application.

The following documents will explain and show the implementation of AWS services to develop the system, the code snippets and explanations of the web app functions related to the AWS services, user manual to explain how the web app works, a test plan, and the system limitations. All the detailed explanations will be in the following chapters.

2.0 Implementation

2.1 Cloud Services Deployment Screenshot and Discussion

2.1.1 Amazon RDS (Relational Database Service

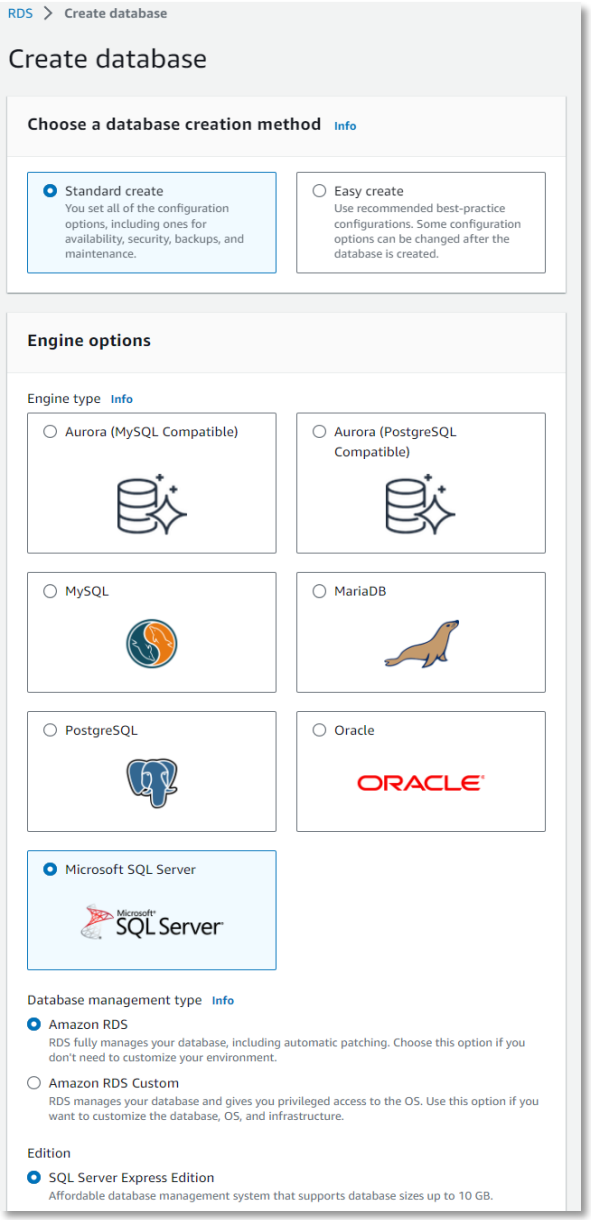

Figure 2.1.1.1 Amazon RDS Creation

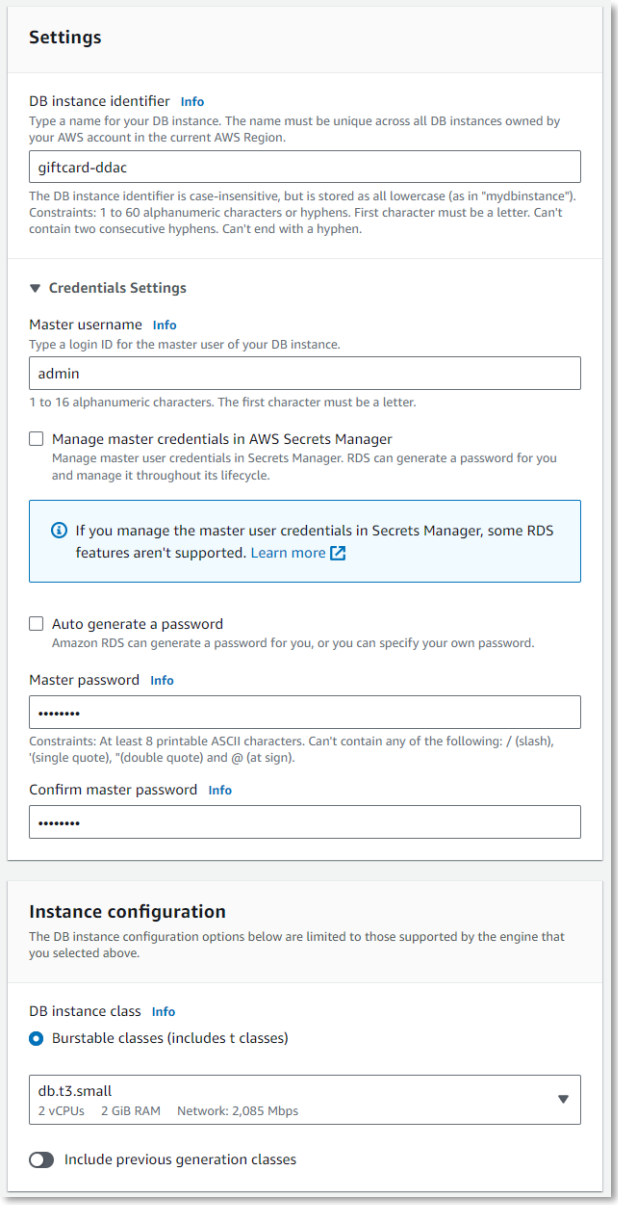

Figure 2.1.1.2 Amazon RDS Creation

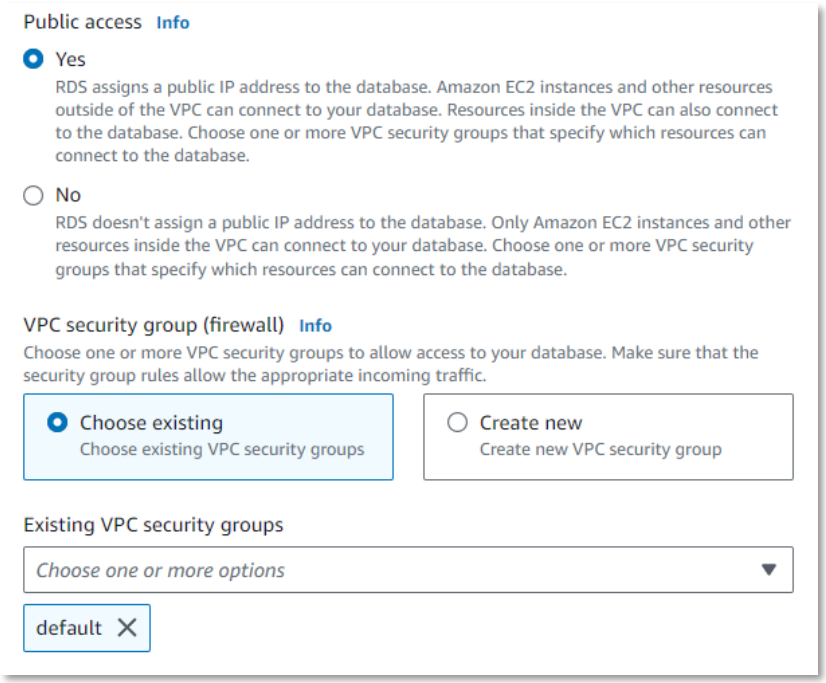

Figure 2.1.1.3 Amazon RDS Creation

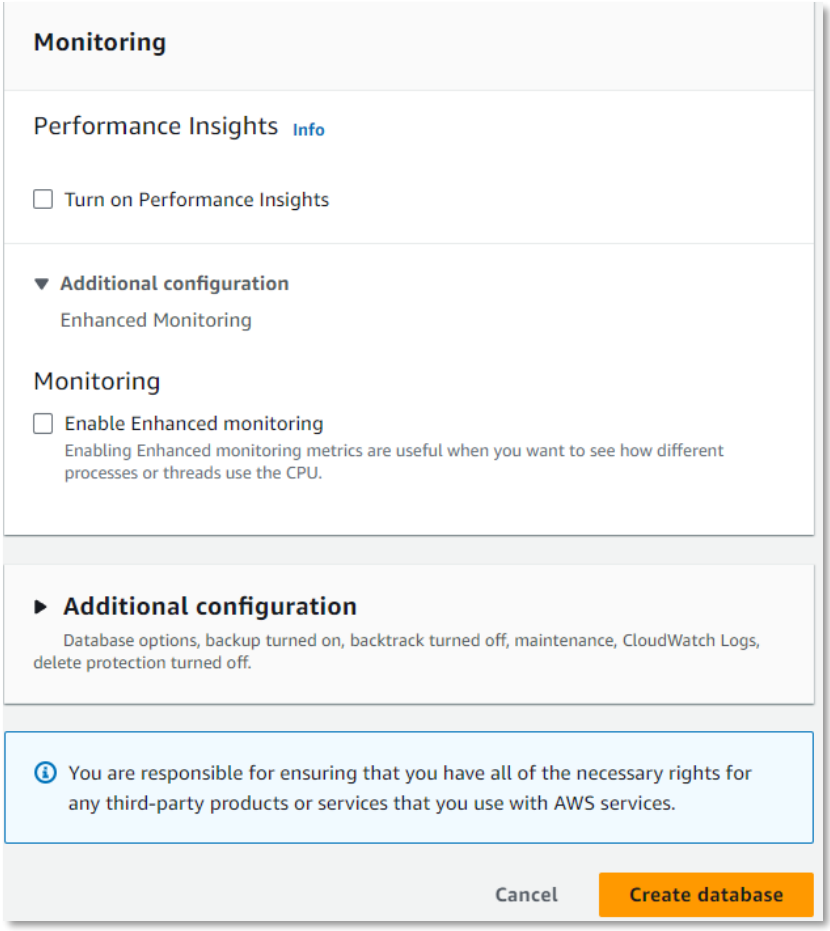

Figure 2.1.1.4 Amazon RDS Creation

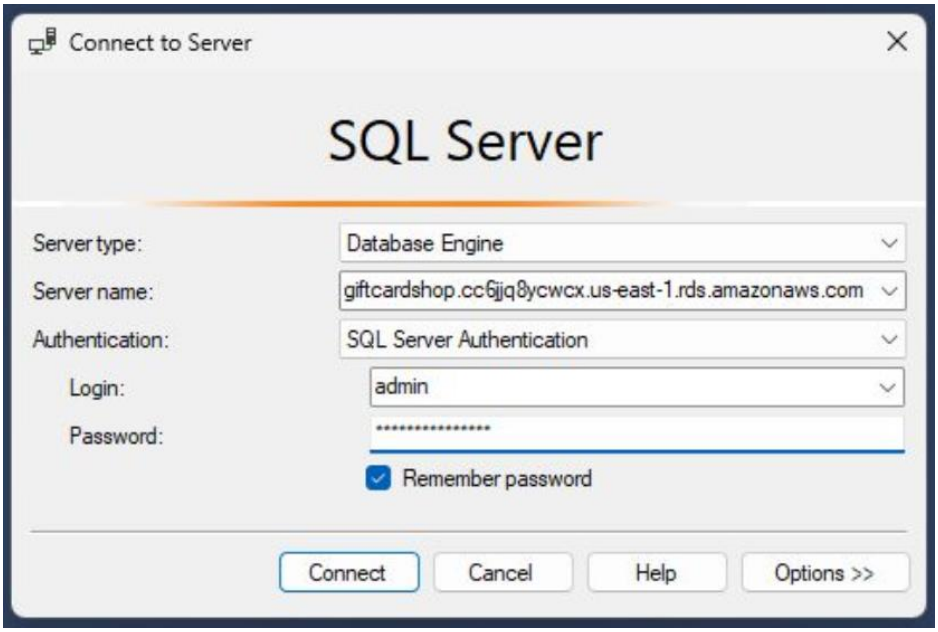

Figure 2.1.1.5 Amazon RDS Access

RDS is utilized to handle relational databases for the gift card online application, with capabilities such as built-in backup, replication, and scalability (AWS, n.d. -f). Figure 2.1.1.1 until 2.1.1.4 shows the configuration settings that are used in the creation of RDS database for this gift card project. Out of all options, the one chosen to use is Microsoft SQL Server engine due to familiarity factor. In figure 2.1.1.2, the database name must be chosen, along with credentials which are used to access the database later. The type “t3.small” was chosen because it is cost-effective for this project which current scale is not too big, so it is still fast enough to provide service for the website. Public access must be turned on for now so that it is easy to access without needing more configuration to access it. Lastly, the monitoring in this case is turned off because of the limitation of the AWS student account, which could not provide full AWS services. Figure 2.1.1.5 shows the way to access the database from Microsoft SQL Server Management Studio with the server endpoint and credentials that are used in the configuration previously.

RDS will store and manage all user account data, gift card type, values, order history, and other associated data for the gift card web application.

2.1.2 Amazon SNS (Simple Notification Service)

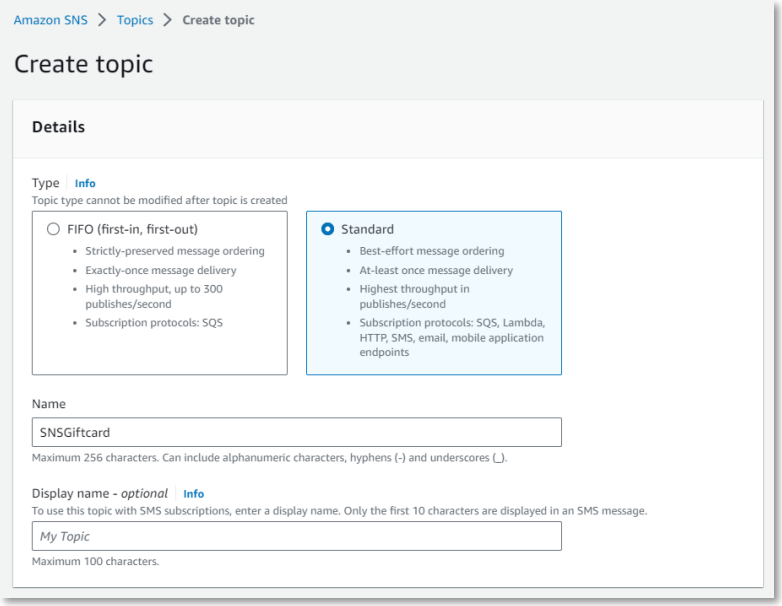

Figure 2.1.2.1 Amazon SNS Topic Creation

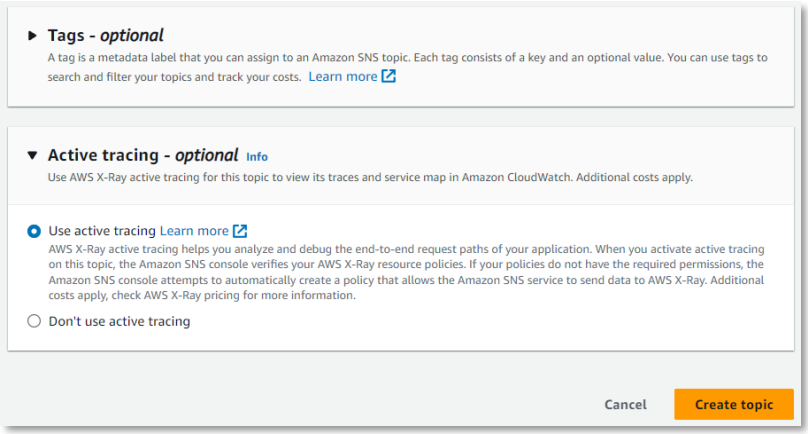

Figure 2.1.2.2 Amazon SNS Topic Creation

Amazon Simple Notification Service (Amazon SNS) is a messaging service that delivers messages to users (AWS, n.d. -g). For SNS configuration, the topic type is set to standard instead of FIFO because this topic is connected with Lambda, using email as the message, and this topic is not configured with Amazon SQS. For the optional part, everything is left as the default configuration, except the active tracing, which is set to active, so that later it shows up in the AWS CloudWatch service map. When a new gift card is introduced to the system, the SNS will alert users. This is done to promote user interaction with the online application and encourage visitors to explore and purchase gift cards.

2.1.3 Amazon S3

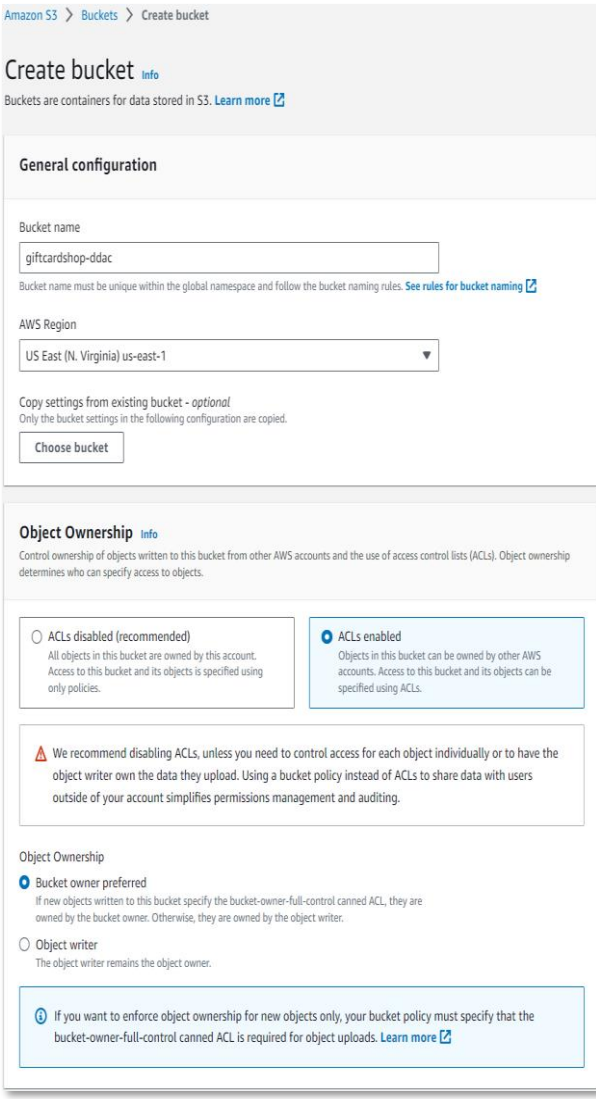

Figure 2.1.3.1 Amazon S3 Creation

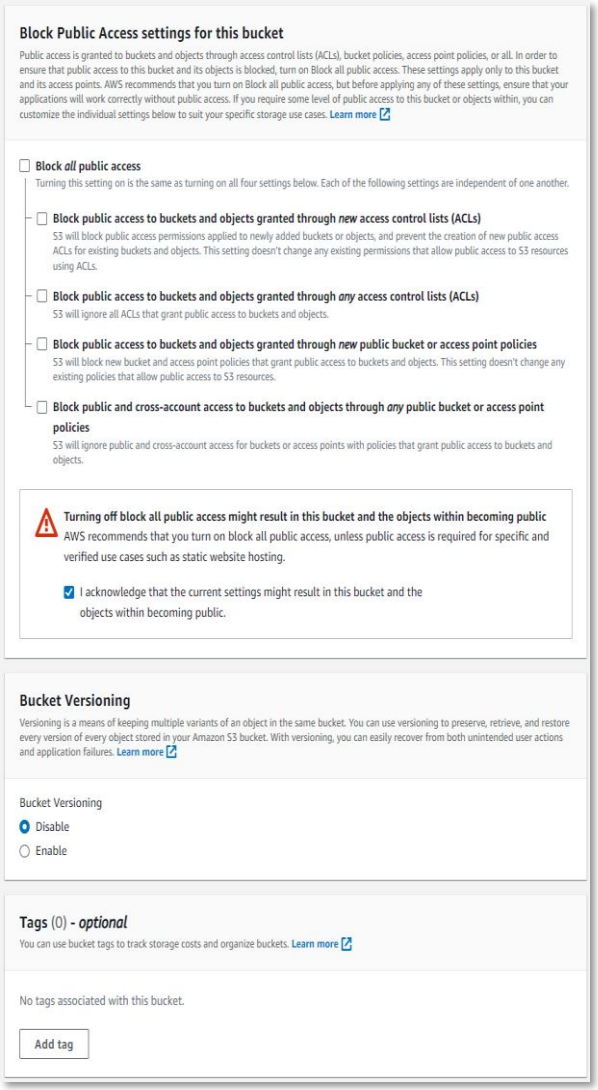

Figure 2.1.3.2 Amazon S3 Creation

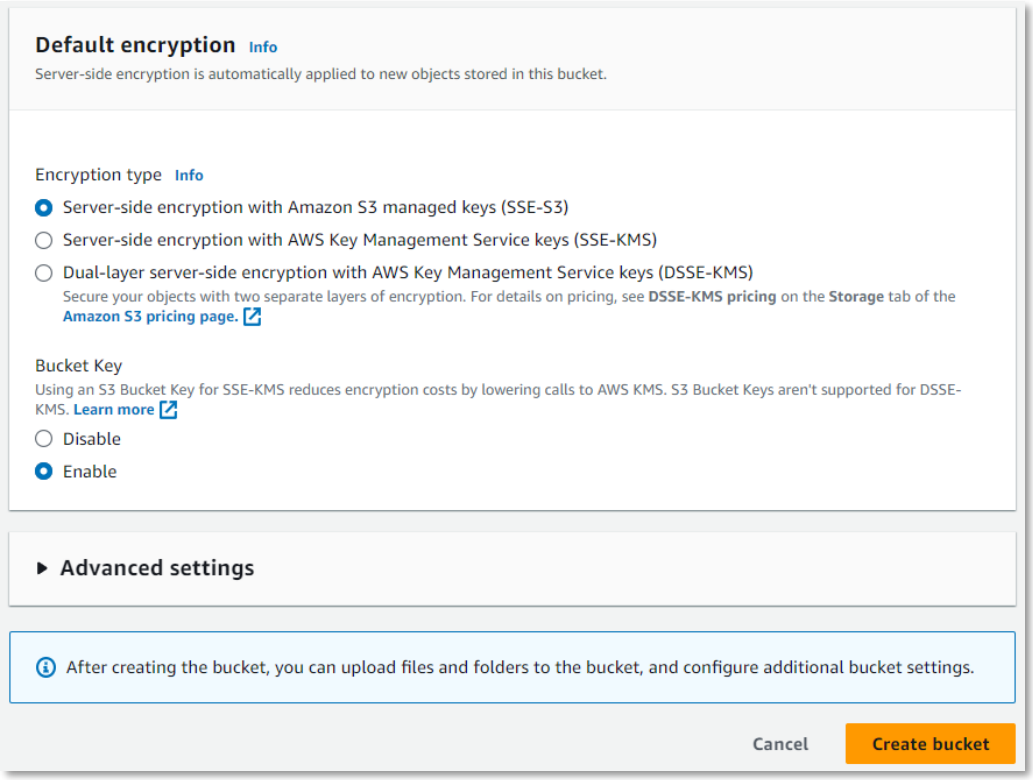

Figure 2.1.3.3 Amazon S3 Creation

Amazon S3 (Amazon Simple Storage Service) is an object storage service that provides scalability, data availability, security, and performance (AWS, n.d. -a). For the configuration, the bucket name must be unique. The ACL by default is disabled, but in this case is enabled for easy accessing, so there is no need for additional policies configuration in accessing the images. The block public access should also be turned off for the same purpose of easy access. As for bucket versioning, it is better to be disabled because enabling it will charge more (AWS, n.d. -b), and it is currently providing no business value for the gift card website to recover previous version of the bucket.

This S3 service will store and manage gift card pictures. These picture data will be utilized in the webapp's frontend. The UI will show the image of a gift card in this manner so that users can quickly recognize the gift card.

2.1.4 AWS Lambda

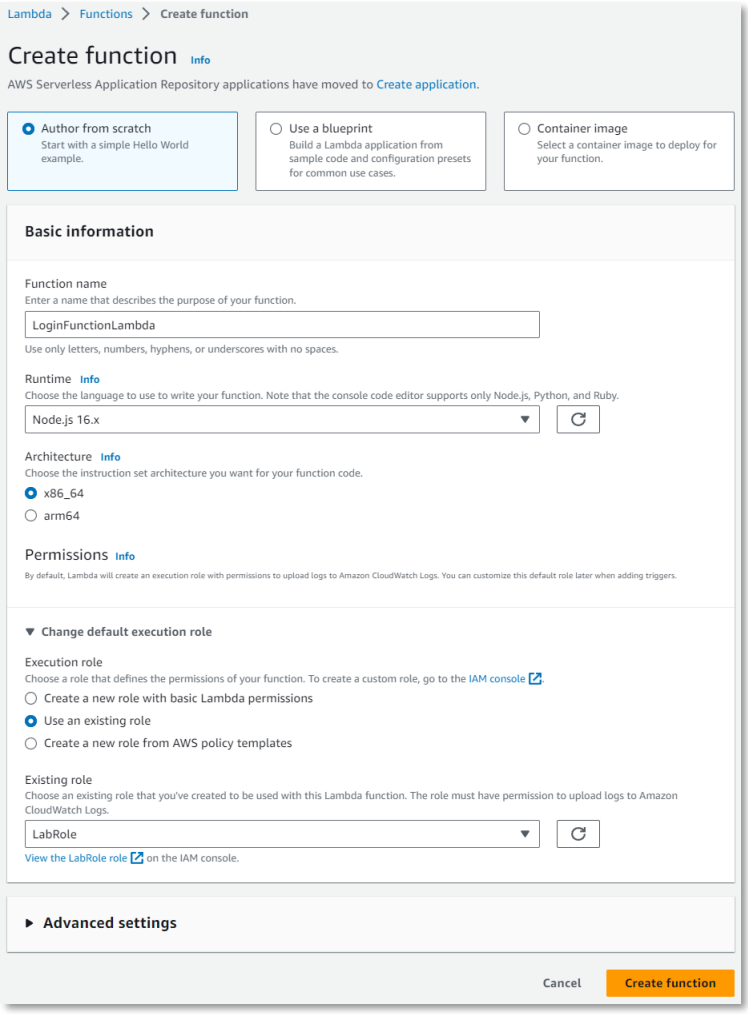

Figure 2.1.4.1 AWS Lambda Creation

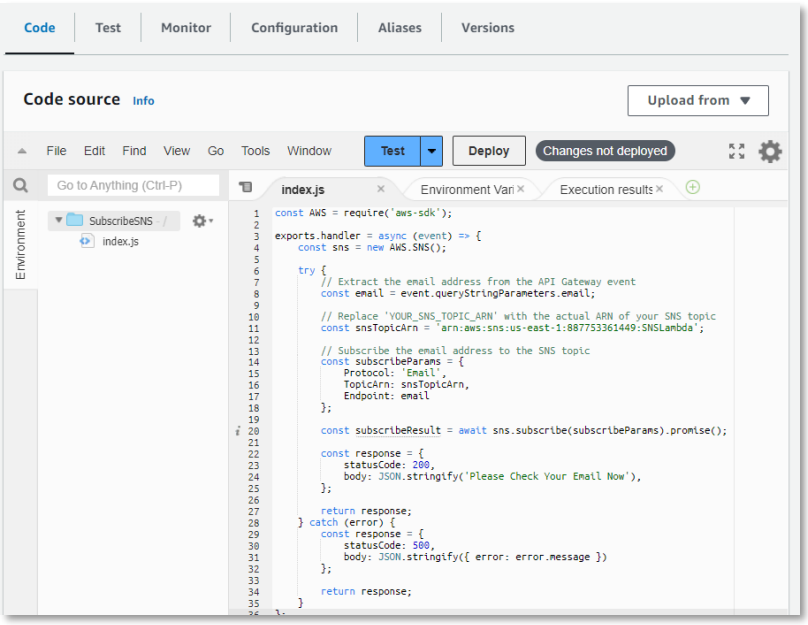

Figure 2.1.4.2 AWS Lambda Creation

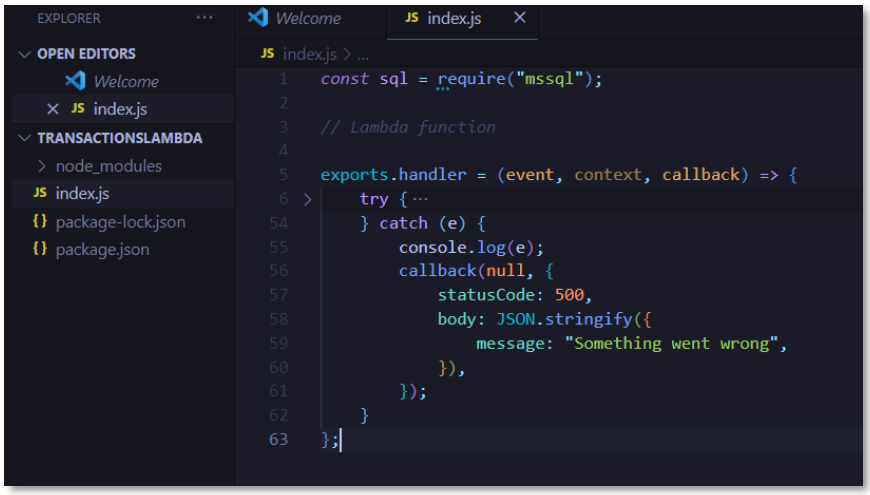

Figure 2.1.4.3 AWS Lambda Creation

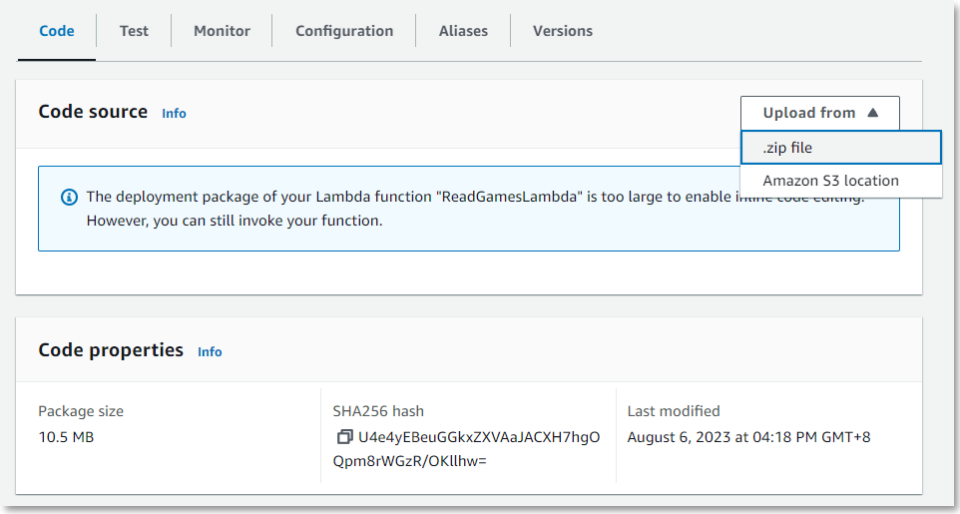

Figure 2.1.4.4 AWS Lambda Creation

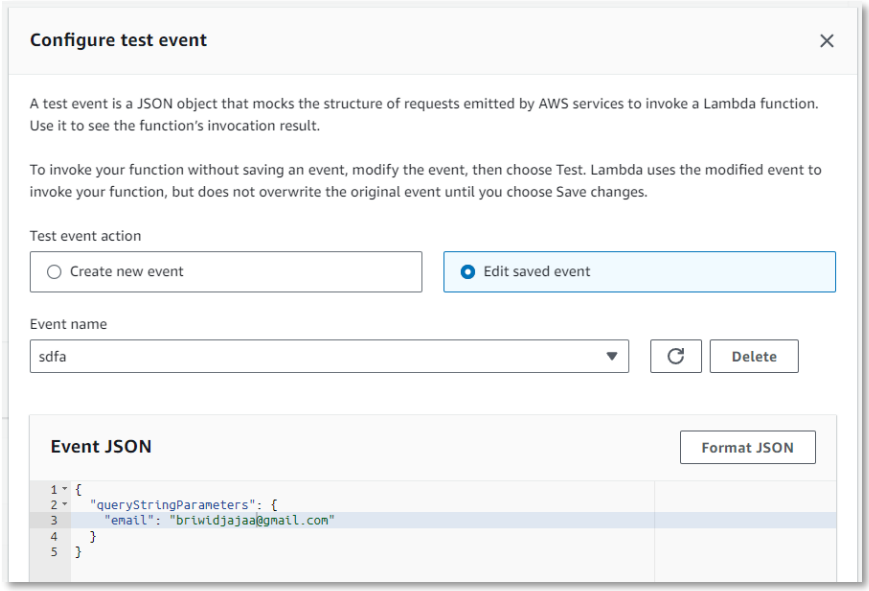

Figure 2.1.4.5 AWS Lambda Creation

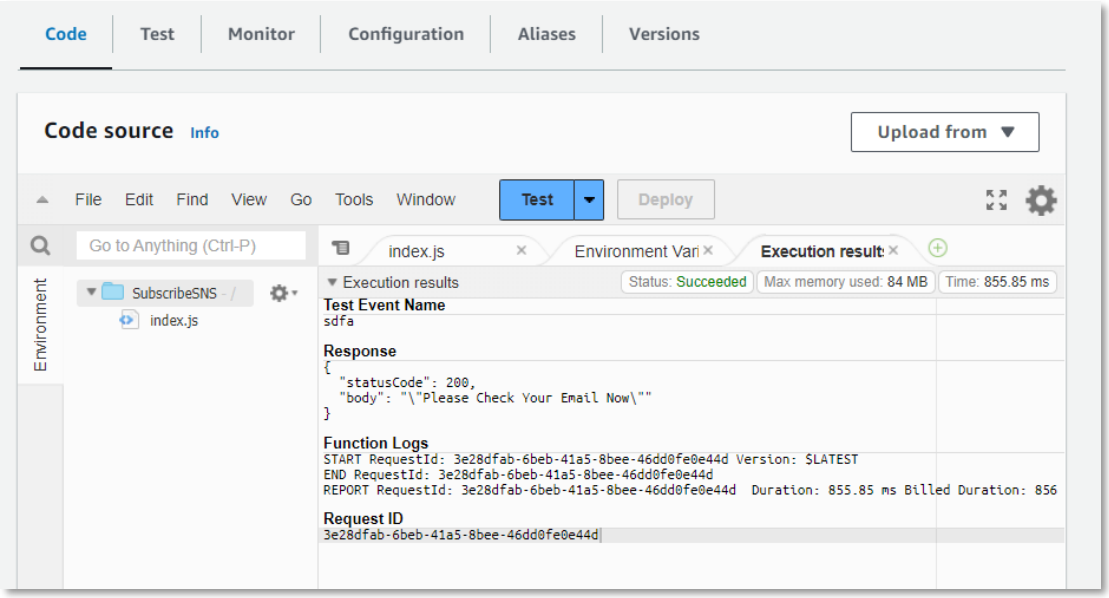

Figure 2.1.4.6 AWS Lambda Creation

AWS Lambda is used for serverless computing, which allows apps to invoke code without having to monitor or manage servers, resulting in cost effectiveness and scalability (Taylor, 2023). For the Lambda function creation, this project uses Node.js 16.x as the programming language runtime for all functions and uses x86_64 as the architecture because it is intended for desktop or laptop usage. The role is set to LabRole since this is a limited student account. For the function deployment, there are 2 methods, which use the built-in code editor like shown on figure 2.1.4.2 or uses zip file like shown on 2.1.4.3. If the "imports" node modules are readable by the built in code editor like 'aws-sdk' then there is no need to create a zip file. Contrary to that, 'mssql' library cannot be read, so it should be manually npm installed in a project directory, then zip the whole thing including index.js, package.json, and node_modules which contains the library needed to run the function. Figure 2.1.4.4 shows how to upload the function on a zip file. Figure 2.1.4.5 shows the configuration for the test event if needed, and figure 2.1.4.6 shows the test result whether the function runs or not.

To handle particular activities or event-driven procedures, AWS Lambda functions will be constructed and configured, allowing the gift card web application to execute code effectively. Some functions will maintain the database, run SNS, and transmit or receive photos from the S3 bucket.

2.1.5 Amazon API Gateway

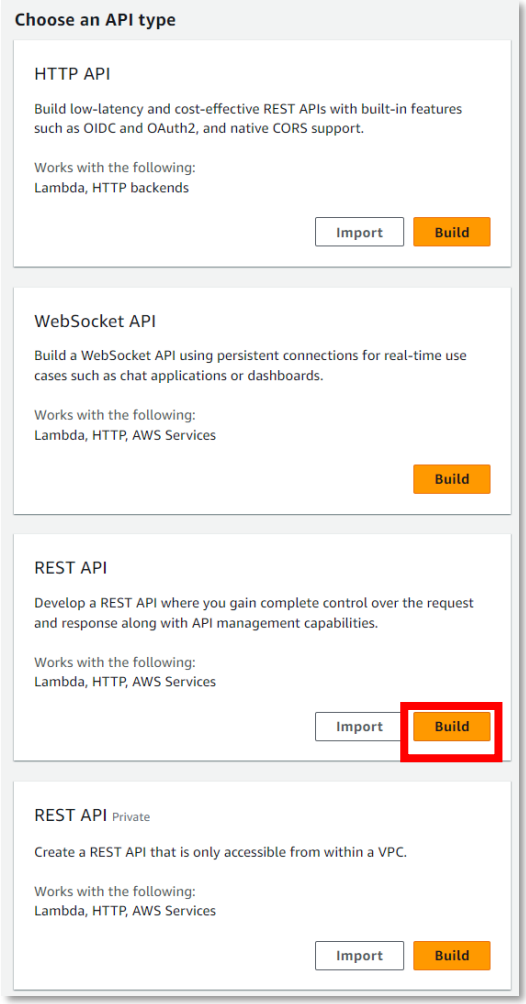

Figure 2.1.5.1 Amazon API Gateway Creation

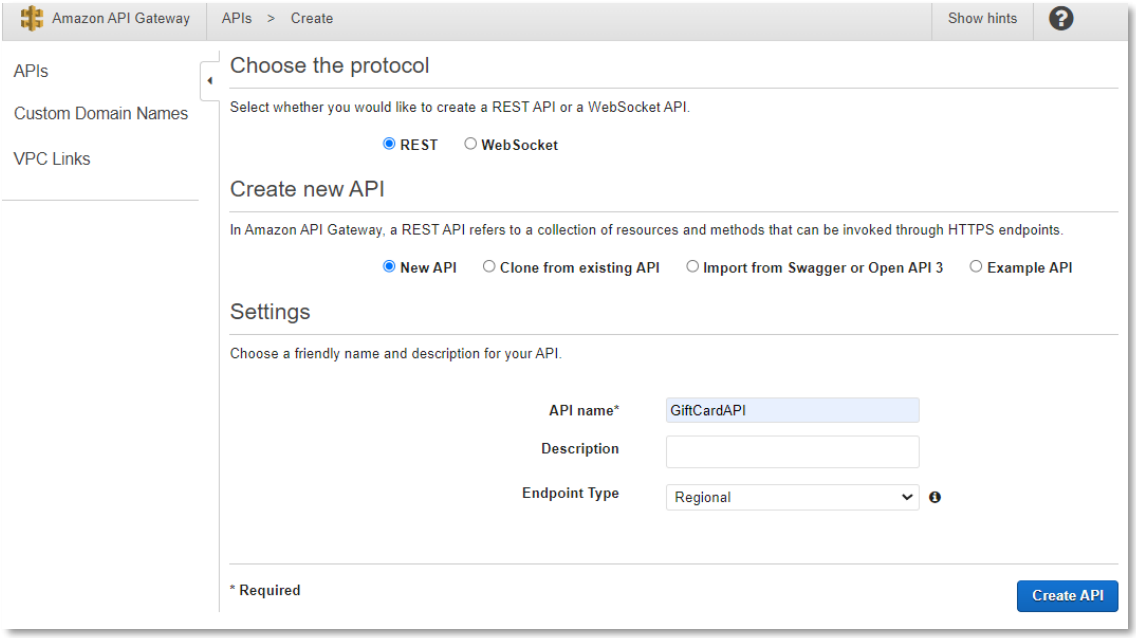

Figure 2.1.5.2 Amazon API Gateway Creation

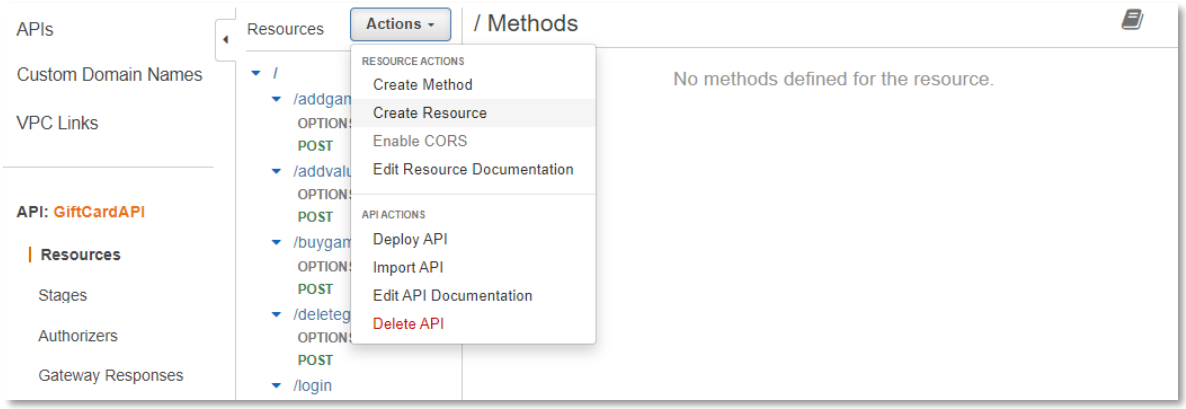

Figure 2.1.5.3 Amazon API Gateway Creation

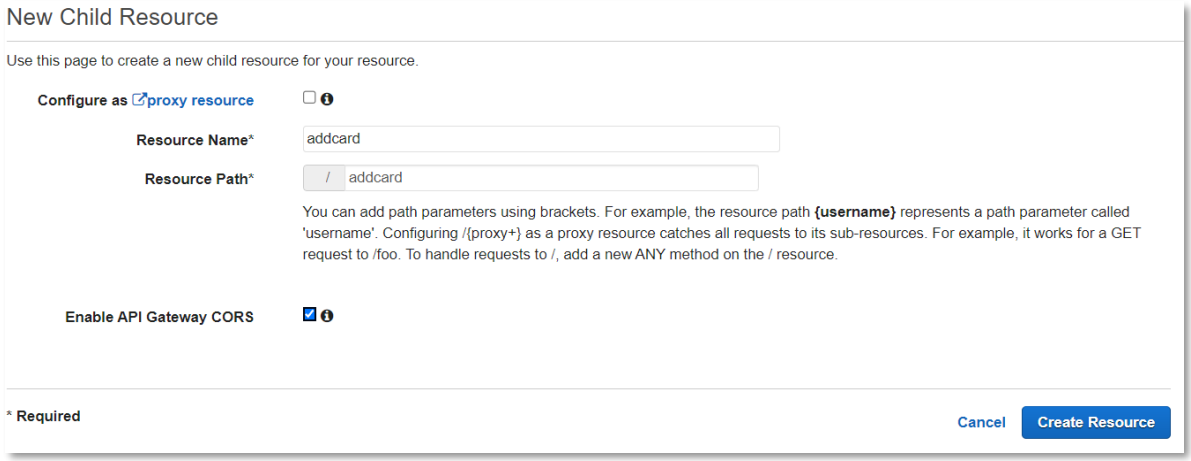

Figure 2.1.5.4 Amazon API Gateway Creation

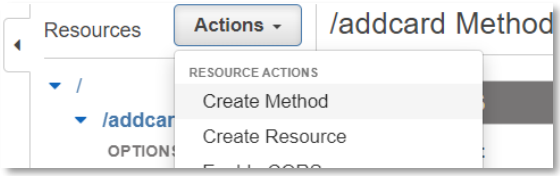

Figure 2.1.5.5 Amazon API Gateway Creation

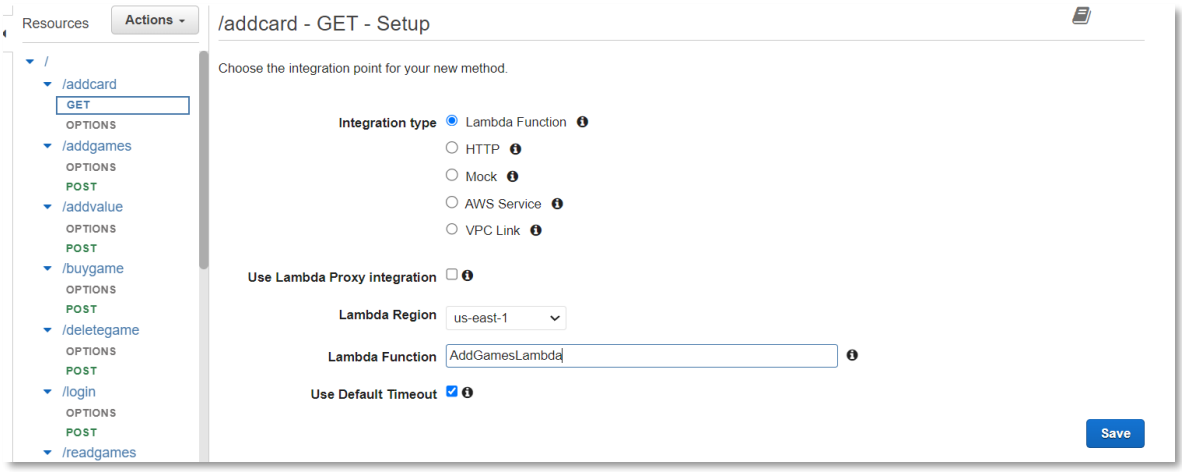

Figure 2.1.5.6 Amazon API Gateway Creation

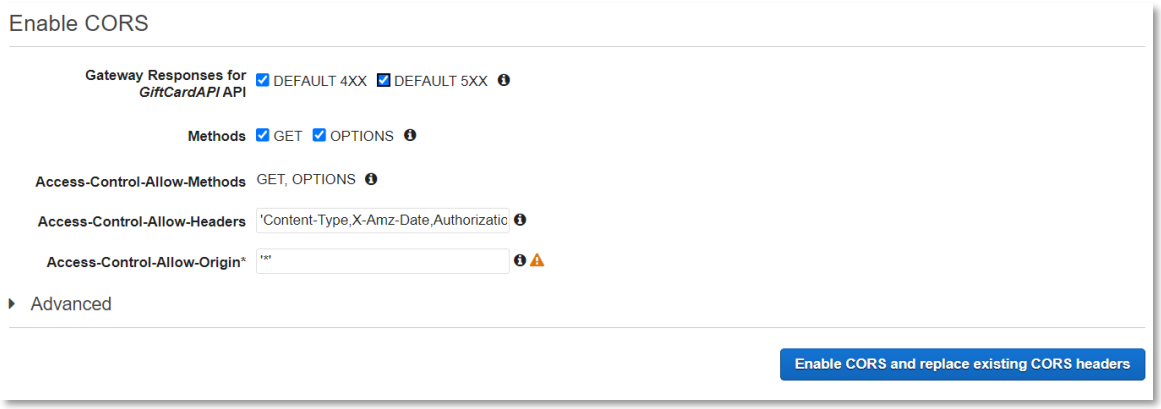

Figure 2.1.5.7 Amazon API Gateway Creation

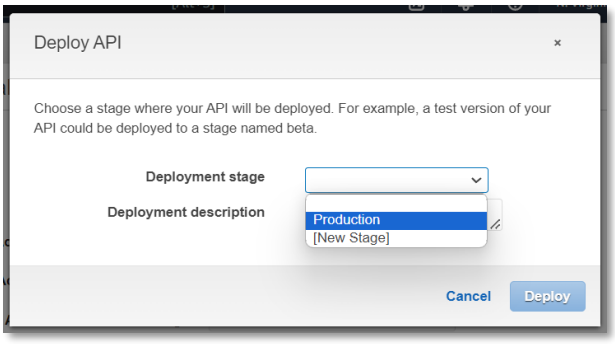

Figure 2.1.5.8 Amazon API Gateway Creation

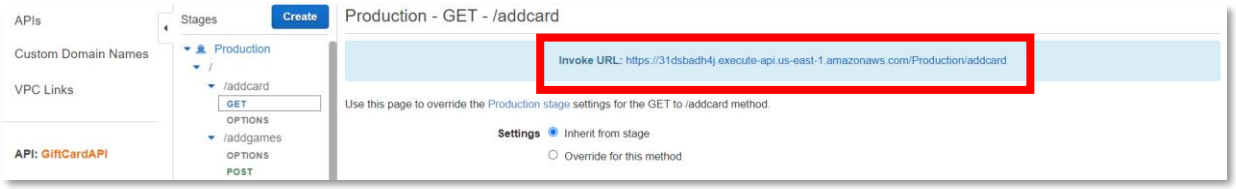

Figure 2.1.5.9 Amazon API Gateway Creation

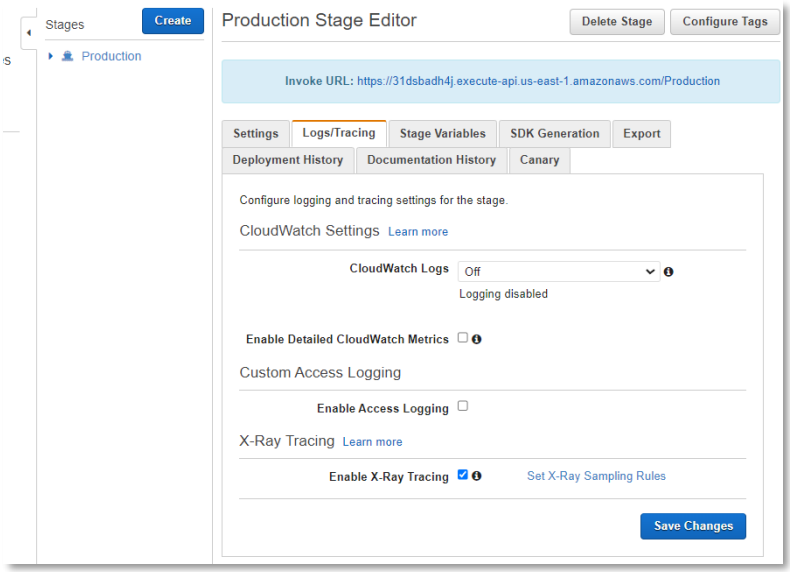

Figure 2.1.5.10 Amazon API Gateway Creation

To develop and administer APIs, AWS API Gateway is utilized. It will offer an endpoint for the application's backend so that the front end may call it (Carty, 2021). Configuration for the API Gateway is a bit lengthy. In the first choice, REST API is chosen as the structure of the API Gateway. Next configuration is to choose REST type, new API type, and give name for the API gateway.

After creating the API, it should be empty, so developer will need to create a new resource first, give resource name, and check the enable CORS option like shown on figure 2.1.5.4. After creating resources, the method will be created, either POST, GET, DELETE, or any other API method. The integration will be lambda function, and then key in the lambda function name that has been made previously. After connecting with lambda, make sure to enable the CORS so that it is accessible in the front-end side. The access-control-allow-origin is set to '*', meaning everyone can access for development purposes, but after deployment, it should be changed to the web URL so that is more secure.

After enabling the CORS, then the last step is to deploy the API to a stage. After deployment is done, the API endpoint now can be seen in the Stages > Stage Name > Resource Name > Method, like shown on figure 2.1.5.9. The last figure, 2.1.5.10 is where the monitoring configuration happens, where the checkbox "Enable X-Ray Tracing" is enabled which is used so that the API Gateway can be monitored later the CloudWatch Service map.

In the implementation of gift card web application, AWS API Gateway will be the interface for the web app's frontend and lambda to manage databases, call the SNS, and get image data from the S3 bucket.

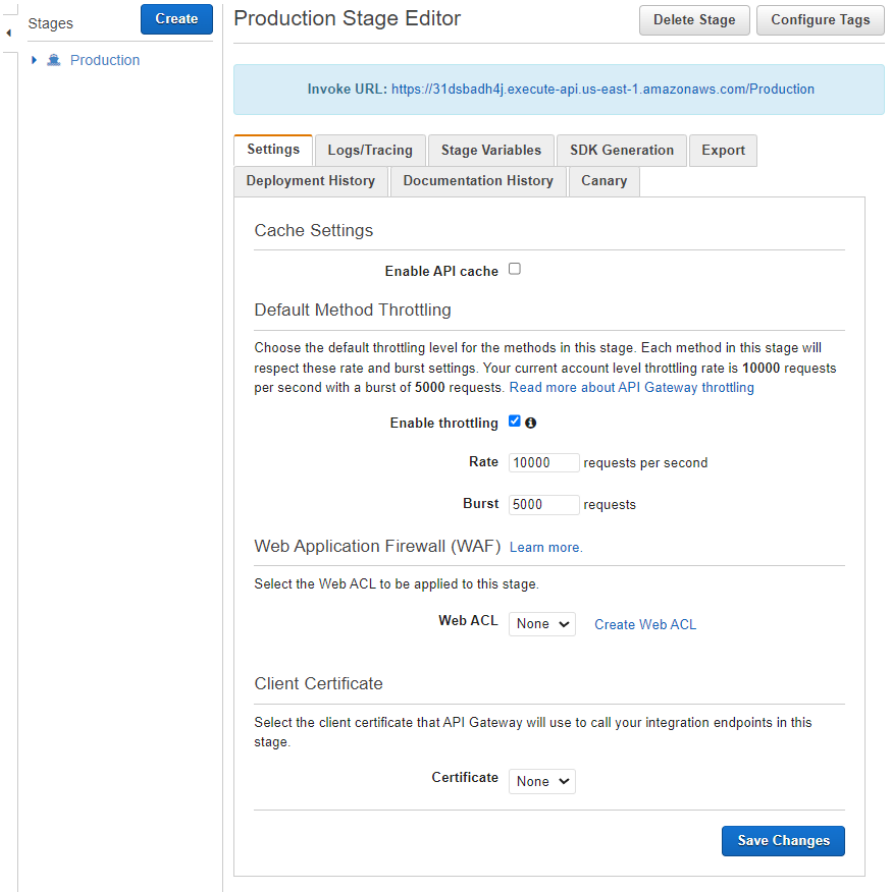

Figure 2.1.5.11 Amazon API Gateway Creation

As for this last figure in API Gateway, it is recommended to check the box for enable throttling. This will have an impact on the aspect of the availability of the API Gateway when it has a sudden spike of request. API Gateway throttles calls to the APIs to prevent them from being overloaded by too many queries. API Gateway, in particular, limits a steady-state rate and a burst of request submissions against all APIs in the developer AWS account (AWS, n.d.-c).

2.1.6 AWS Elastic Beanstalk + EC2 Instance

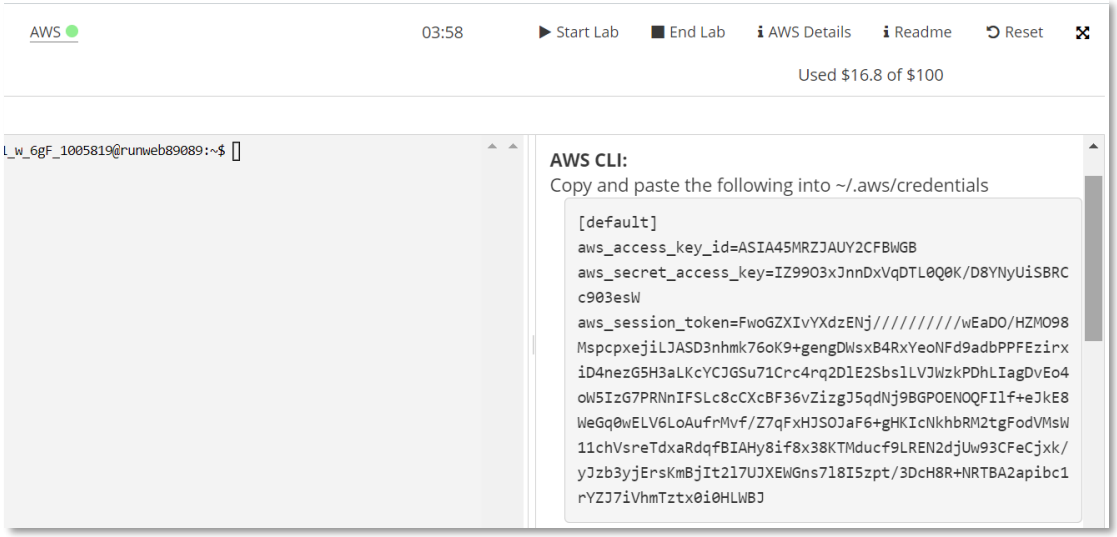

Figure 2.1.6.1 AWS Elastic Beanstalk Deployment

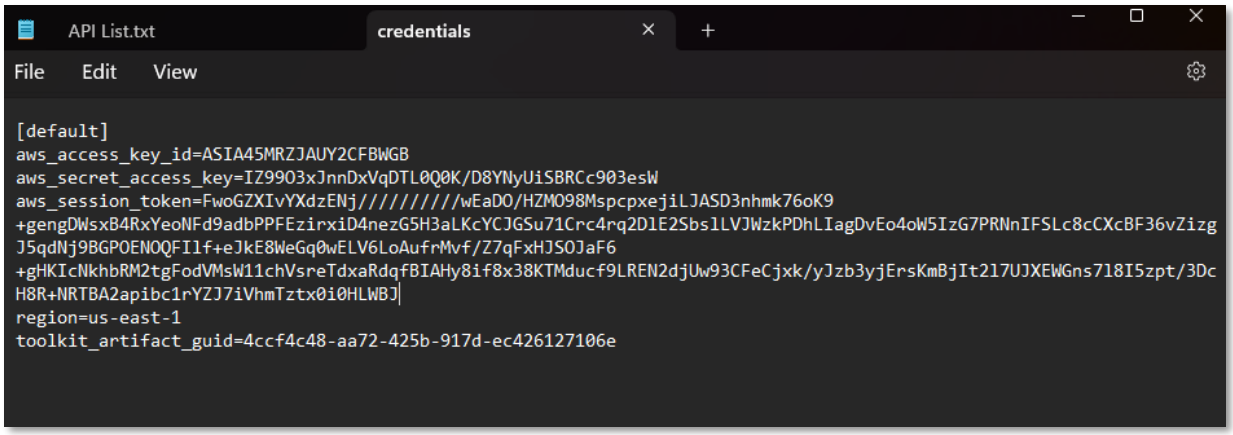

Figure 2.1.6.2 AWS Elastic Beanstalk Deployment

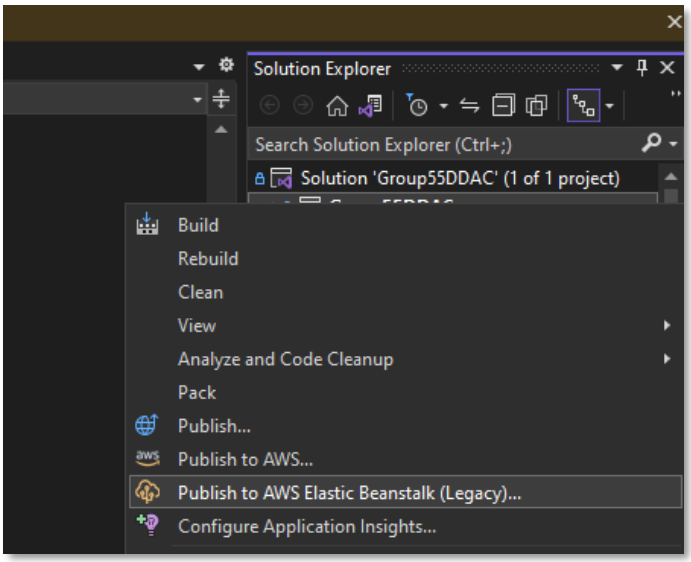

Figure 2.1.6.3 AWS Elastic Beanstalk Deployment

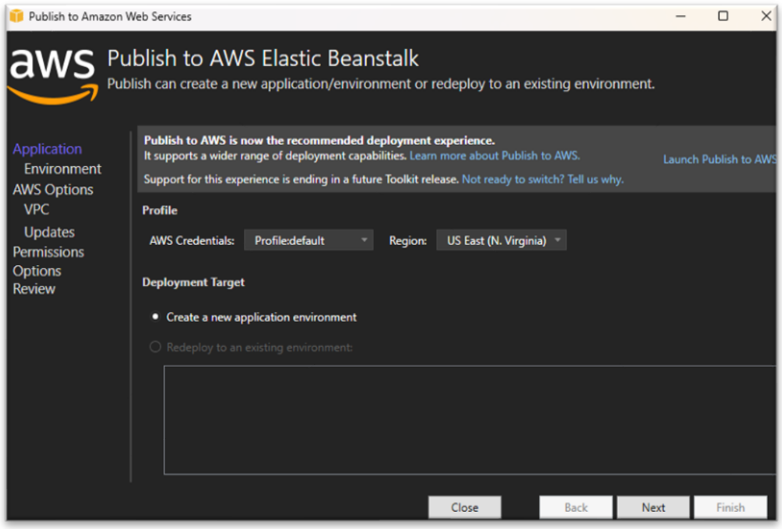

Figure 2.1.6.4 AWS Elastic Beanstalk Deployment

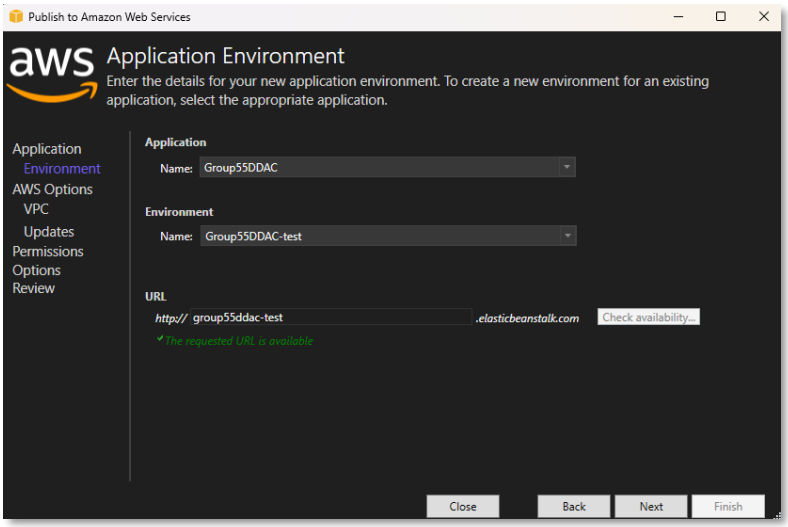

Figure 2.1.6.5 AWS Elastic Beanstalk Deployment

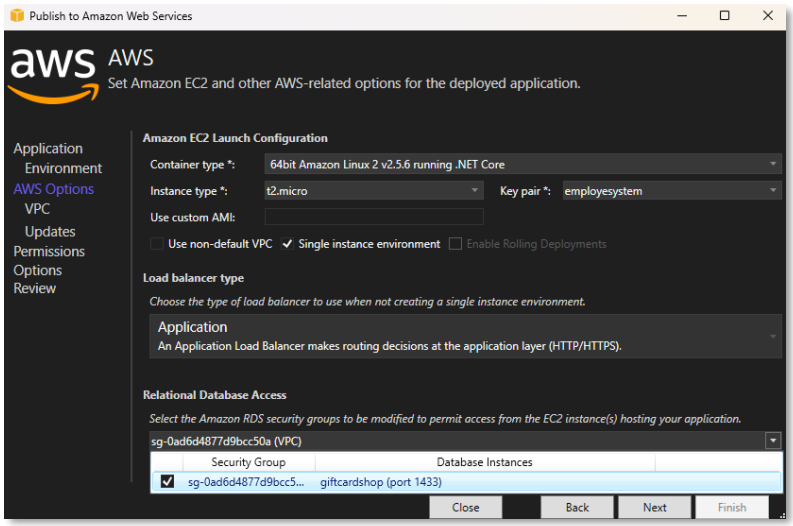

Figure 2.1.6.6 AWS Elastic Beanstalk Deployment

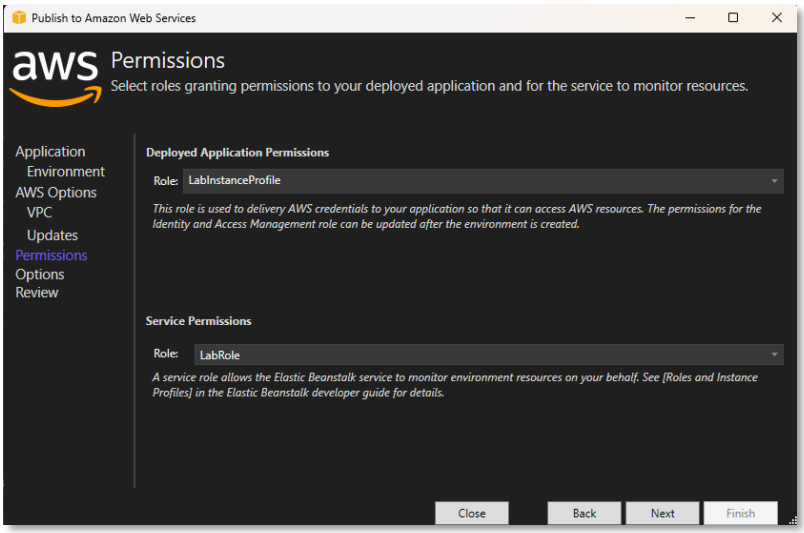

Figure 2.1.6.7 AWS Elastic Beanstalk Deployment

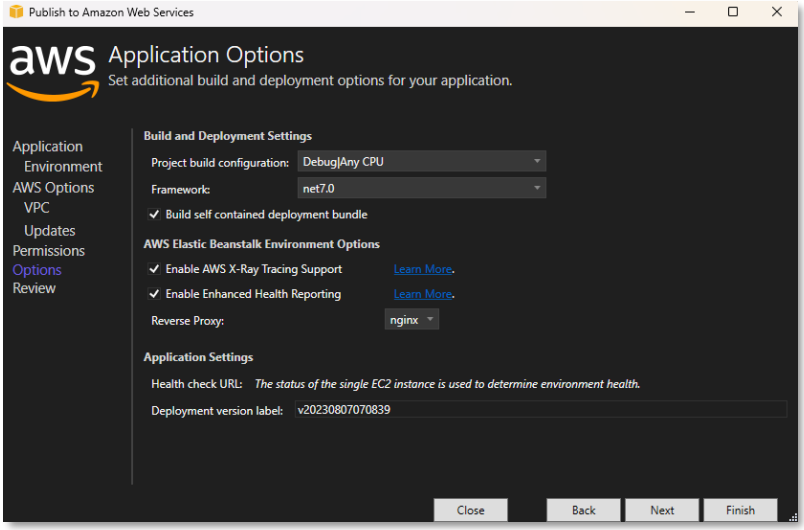

Figure 2.1.6.8 AWS Elastic Beanstalk Deployment

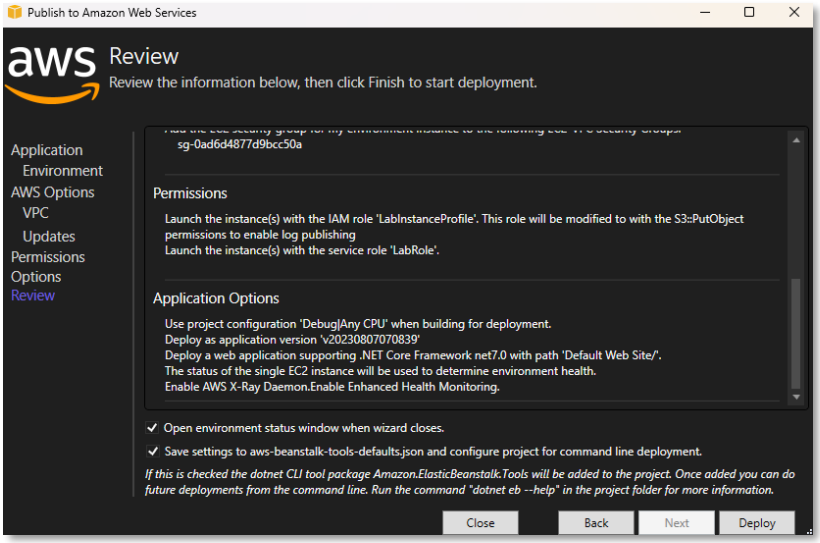

Figure 2.1.6.9 AWS Elastic Beanstalk Deployment

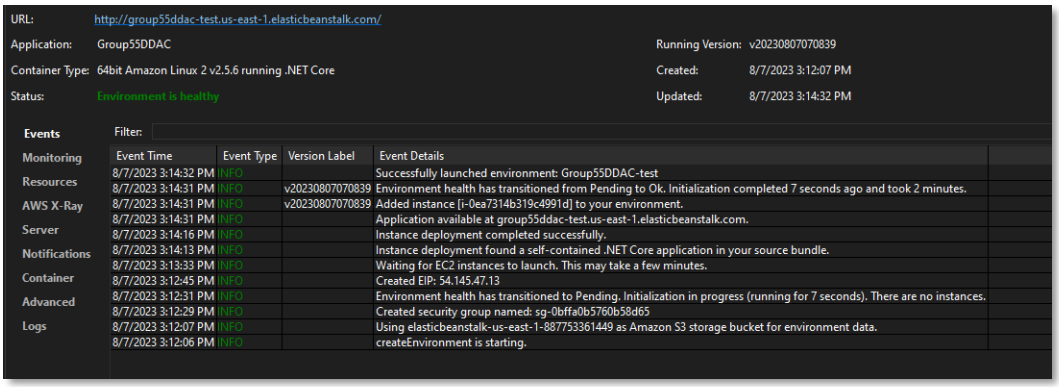

Figure 2.1.6.10 AWS Elastic Beanstalk Deployment

Amazon Elastic Compute Cloud will provide scalable processing power on-demand on the Amazon Web Services (AWS) Cloud, where the web app will be installed (AWS, n.d. -e). AWS Elastic Beanstalk is used to make web application deployment and administration easier (GeeksforGeeks, 2023). Elastic Beanstalk is used throughout the gift card web application deployment phase to swiftly launch, scale, and manage the application. In order to start the deployment using Elastic Beanstalk, the AWS lab must be started first, and then the credentials of the session must be copied into the local file in "\.aws" directory like in figure 2.1.6.2. Then from Visual Studio IDE, right click on the project solution explorer, and choose to publish to AWS Elastic Beanstalk. Then choose the app name, environment, and link availability. As for the instance type, "t2.micro" seems sufficient and accommodates the web app to be deployed without any problems. In the RDS section, it will automatically search for available DB instance. For the deployment role, since this is a student account, the role is set to "LabRole". The framework is using dotnet version 7.0. The monitoring option should be checked in this case so that it will be traced on the services map later. After all configuration is done, then click on deploy and wait for a while. Elastic Beanstalk will automatically create an EC2 instance for deployment. After the success message is shown on the log like figure 2.1.6.10, then the link URL can be accessed, and the web application is online.

In short, Elastic Beanstalk abstracts away the intricacies of infrastructure management, allowing developers to concentrate on application code. Elastic Beanstalk will be used to upload the gift card web application code, automatically manage resources such as EC2 instances and load balancers, and handle activities such as application monitoring, health checks, and resource management.

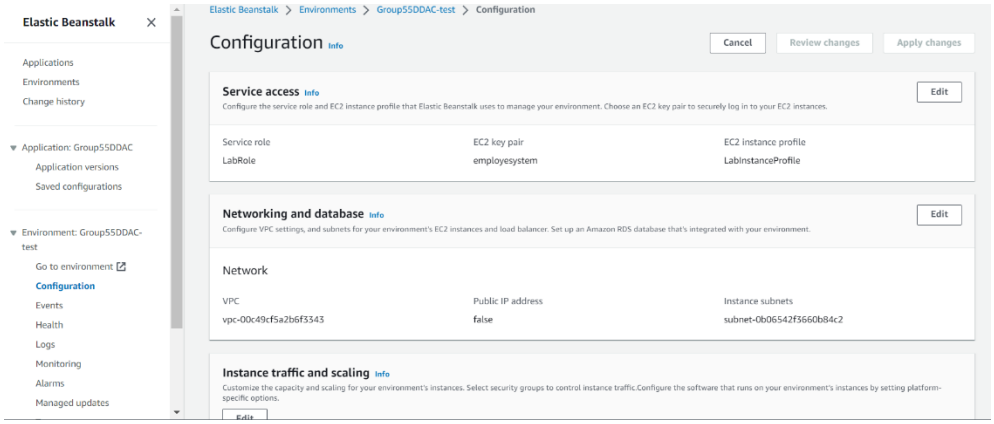

Figure 2.1.6.11 AWS Elastic Beanstalk Deployment

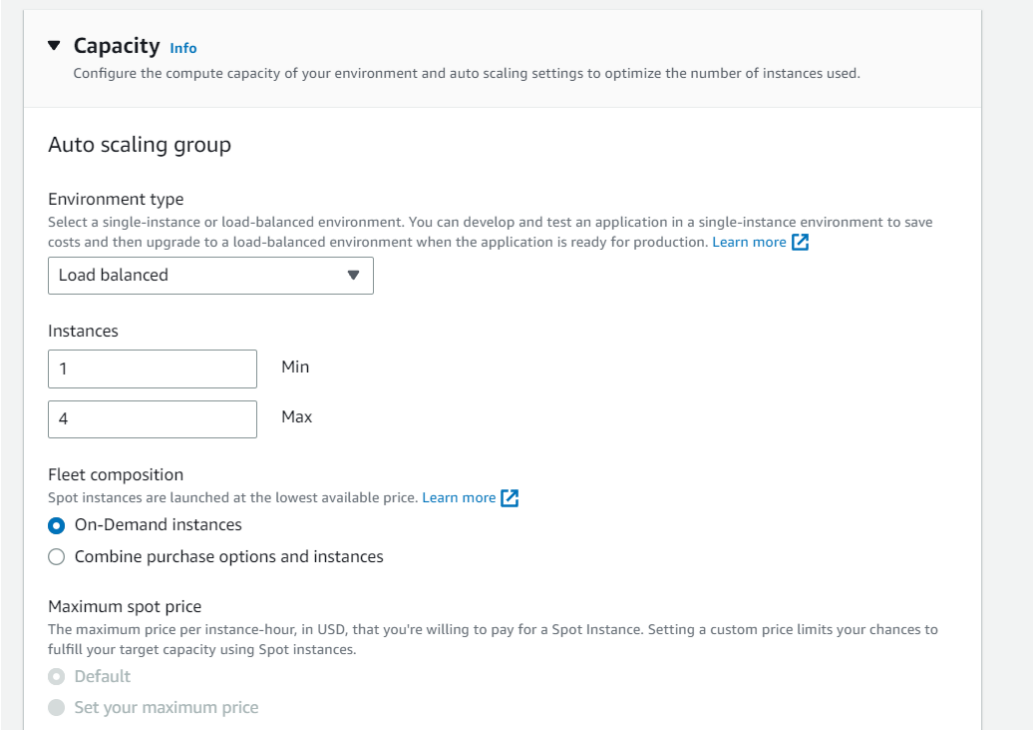

Figure 2.1.6.12 AWS Elastic Beanstalk Deployment

As for the configuration after deployment from Visual Studio, additional configuration is done in the Elastic Beanstalk. In figure 2.1.6.11, developer will go to the instance traffic and scaling part, click edit, and then configure the environment type to be load balance, with minimum 1 instance and maximum 4 instances. This is used for handling a sudden spike in request, where Elastic Beanstalk will have the capability to add more instances so that the web application will not crash. This is a good setting to ensure a smooth performance of the web application.

2.2 Code Snippets for Each Function Related to Cloud Services

2.2.1 Connect to RDS

```
1  const sql = require("mssql");
2
3  // Lambda function
4
5  exports.handler = (event, context, callback) => {
6      try {
7          sql.connect(
8              {
9                  user: "admin",
10                 password: "admin123",
11                 server: "giftcardshop.cc6jjq8ycwcx.us-east-1.rds.amazonaws.com",
12                 database: "Giftcard",
13                 port: 1433,
14             },
15             (err) => {
16                 if (err) {
17                     console.log("Fail");
18                     callback("connect failed", err);
19                 } else {
20
21                     let sqlRequest = new sql.Request();
22                     const values = event.name + "','" + event.imagelink + "');";
23                     let sqlQuery = "INSERT INTO GIFTCARDS VALUES ('" + values;
24
25                     sqlRequest.query(sqlQuery, function (err, data) {
26                         if (err) {
27                             console.log(err);
28                             sql.close();
29                             callback(null, {
30                                 statusCode: 500,
31                                 body: JSON.stringify({
32                                     message: "Error inserting data",
33                                 }),
34                             });
35                         } else {
36                             sql.close();
37                             callback(null, {
38                                 statusCode: 200,
39                                 body: JSON.stringify({
40                                     message: "Data inserted successfully",
41                                 }),
42                             });
43                         }
44                     });
45                 }
46             }
47         );
48     } catch (e) {
49         console.log(e);
50         callback(null, {
51             statusCode: 500,
52             body: JSON.stringify({
53                 message: "Something went wrong",
54             }),
55         });
56     }
57 };
```

Figure 2.2.1.1 Connect to RDS

In the code snippet above, it shows how to communicate with Amazon RDS through a lambda function. Since ‘mssql’ need to be installed first, this code cannot be deployed directly in the Lambda, rather it needed to be created in the local directory with all the node modules installed. In order to connect with RDS, command “sql.connect” is required and code from line 9 to 13 serve as the credentials, where user, password, endpoint, and database is defined. If connection success, developer can do anything with the SQL query, and the system will return the SQL result as a response if it is a GET method, and string message if its POST method such as insert or modify table.

2.2.2 Subscribe to SNS Topic

```
index.js   Environment Vari   Execution results
1  const AWS = require('aws-sdk');
2
3  exports.handler = async (event) => {
4      const sns = new AWS.SNS();
5
6      try {
7          // Extract the email address from the API Gateway event
8          const email = event.queryStringParameters.email;
9
10         // Replace 'YOUR_SNS_TOPIC_ARN' with the actual ARN of your SNS topic
11         const snsTopicArn = 'arn:aws:sns:us-east-1:887753361449:SNSLambda';
12
13         // Subscribe the email address to the SNS topic
14         const subscribeParams = {
15             Protocol: 'Email',
16             TopicArn: snsTopicArn,
17             Endpoint: email
18         };
19
20         const subscribeResult = await sns.subscribe(subscribeParams).promise();
21
22
23         const response = {
24             statusCode: 200,
25             body: JSON.stringify('Please Check Your Email Now'),
26         };
27
28         return response;
29     } catch (error) {
30         const response = {
31             statusCode: 500,
32             body: JSON.stringify({ error: error.message })
33         };
34
35         return response;
36     }
37 };
```

Figure 2.2.2.1 Subscribe to SNS Topic

Subscribing to SNS function can be directly configured in Lambda console. With 'aws-sdk' module, the function can access SNS. It takes email as a parameter, SNS Topic ARN must be configured, and the protocol chosen is email. If the SNS is successfully sent to the specified email, then the function will return status code 200. The email can be checked by subscribing to the SNS Topic specified in line 11.

2.2.3 Send Email with SNS

```
index.js   Environment Var   Execution results
1  const AWS = require('aws-sdk');
2  // Set region
3  AWS.config.update({region: 'us-east-1'});
4
5  const sns = new AWS.SNS()
6  module.exports.handler = async (event) =>
7  {
8      const params = {
9          Subject: 'New Item Is Out',
10         Message: 'Hello Customers, a new item just drop in the Gift Card Shop, Check it out, go to this link: https://aniwatch.to/home',
11         TopicArn: 'arn:aws:sns:us-east-1:887753361449:SNSLambda'
12     };
13
14     let response = {
15         statusCode: 200,
16         headers: {
17             "Access-Control-Allow-Headers" : "Content-Type",
18             "Access-Control-Allow-Origin": "*",
19             "Access-Control-Allow-Methods": "OPTIONS,POST,GET"
20         },
21         body: JSON.stringify('Hello from Lambda!'),
22     };
23
24     try {
25         const data = await sns.publish(params).promise();
26         response.messageId = data.MessageId,
27         response.result = 'Success'
28     }
29     catch (e) {
30         console.log(e.stack)
31         response.result = 'Error'
32     }
33 return response
34 };
35
```

Figure 2.2.2.1 Send Email with SNS

Now after an email is subscribed to an SNS Topic, this lambda function above is used to send the email to the subscribed email endpoint. With the same help of 'aws-sdk' module, the code can send the email. The params that needed to be configured is Subject, Message, and Topic ARN. Subject is the title of the email, Message is the content of the email, and Topic ARN is used to find out which email endpoint the message will be sent to. Then in line 25, that is the function that sends out the email.

2.2.4 Upload Image to S3

```
1   //const AWS = require('aws-sdk');
2   const XRay = require('aws-xray-sdk'); // Import the X-Ray SDK
3   const AWS = XRay.captureAWS(require('aws-sdk'));
4   const S3 = XRay.captureAWSClient(new AWS.S3());
5
6   exports.handler = async (event) => {
7       try {
8           // Extract picture data from the event (e.g., base64 encoded data)
9           const pictureData = event.pictureData.toString();
10
11          // Specify the S3 bucket and object key
12          const bucketName = 'giftcardshop-group55-ddac';
13          const objectKey = event.pictureKey + '.jpg'; // Adjust the file extension accordingly
14
15          // Convert the picture data to a buffer
16          const buffer = Buffer.from(pictureData.replace(/^data:image\/\w+; base64,/, ""), 'base64');
17
18          console.log(buffer);
19          console.log(pictureData);
20
21          await XRay.captureAsyncFunc('S3Upload', async (subsegment) => {
22              // Upload the picture to S3
23              await S3.upload({
24                  Bucket: bucketName,
25                  Key: objectKey,
26                  Body: buffer,
27                  ACL: 'public-read',
28                  ContentType: 'image/jpeg',
29                  ContentEncoding: 'base64'
30              }).promise();
31
32              subsegment.close(); // Close the subsegment when the operation is done
33          });
34
35          const s3ImageLink = 'https://giftcardshop-group55-ddac.s3.amazonaws.com/' + objectKey;
36
37          return {
38              statusCode: 200,
39              body: JSON.stringify(s3ImageLink),
40          };
41      } catch (err) {
42          console.error('Error uploading picture:', err);
43          return {
44              statusCode: 500,
45              body: JSON.stringify('Error uploading picture'),
46          };
47      }
48  };
```

Figure 2.2.4.1 Upload Image to S3

The function above shows how to upload images to S3 Bucket. The only sdk required is actually ‘aws-sdk’. The module ‘aws-xray-sdk’ will be covered in the next part. The process of uploading is to accept the image in the form of Base64 code. This will be handled by the frontend, so the Lambda function will only receive it as a request body upon API call. pictureData variable will hold the Base64 code, bucketName is to specify which bucket will be used, and objectKey is the name of the picture, also provided upon API call from the frontend. The buffer is used to transform the Base64 into desired data type suitable for uploading to the S3 Bucket. Line 23 is the process of uploading the image, where ACL is configured as public access so that it can be read from anywhere. Upon success uploading, it will return the JSON response with the image link as the body.

2.2.5 Connecting S3 to X-Ray

```
const XRay = require('aws-xray-sdk'); // Import the X-Ray SDK
const AWS = XRay.captureAWS(require('aws-sdk'));
const S3 = XRay.captureAWSClient(new AWS.S3());
```

Figure 2.2.5.1 Connecting S3 to X-Ray

```
await XRay.captureAsyncFunc('S3Upload', async (subsegment) => {
    // Upload the picture to S3
    await S3.upload({
        Bucket: bucketName,
        Key: objectKey,
        Body: buffer,
        ACL: 'public-read',
        ContentType: 'image/jpeg',
        ContentEncoding: 'base64'
    }).promise();

    subsegment.close(); // Close the subsegment when the operation
});
```

Figure 2.2.5.2 Connecting S3 to X-Ray

In the first development, the X-Ray service map does not show the S3 bucket. It turns out that since S3 must be configured manually if it is accessed from a lambda function to be traced. With the help of 'aws'xray-sdk' that is installed in the npm, it is possible to capture the S3 bucket in X-Ray service map. In the S3 uploading process, it is encapsulated in a function called captureAsyncFunc to get the traces.

2.3 User Manual

1. Login

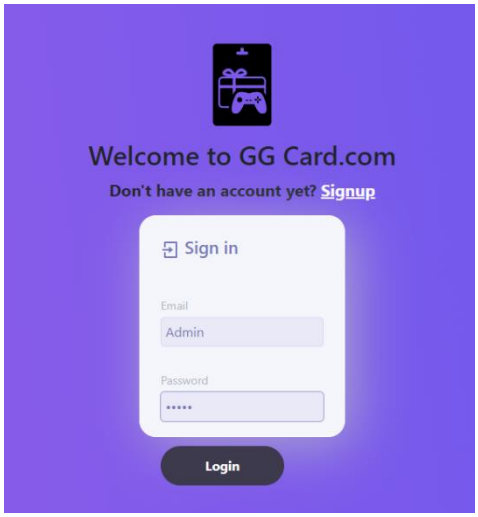

Figure 2.3.1.1: Login Page

User shall be able to log into the system by inputting the right credentials, if the email and password is correct then the user will be able to login. If they input the admin credentials, they will be directed to the admin page. If the credentials are incorrect, the system will validate and alert the user.

2. Register

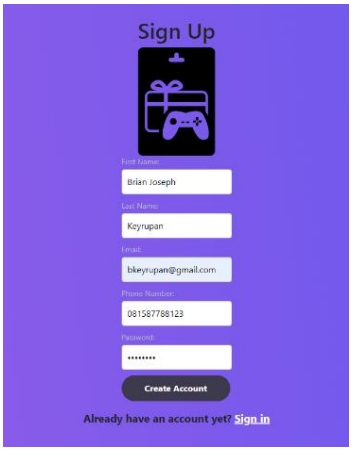

Figure 2.3.2.1: Register Page

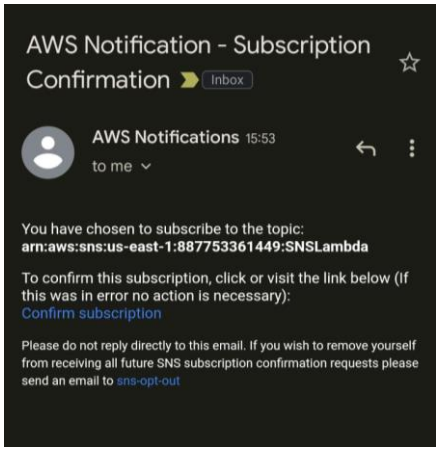

Figure 2.3.2.2: Email Notification

Users are asked to input their name, email, phone number, and password. When all inputs are given, then the user can click the register button to validate the inputs and if all is correct the n system will register the account into the system and store the data in the database. After registering the web app will be directed back to the login screen. If the user decides to go back to the login screen when in the register screen, user can use the sign in button

3. User Home

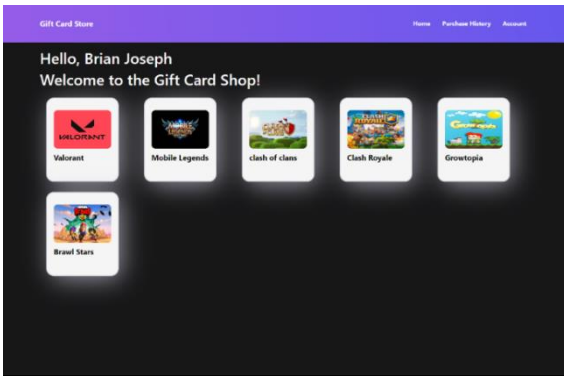

Figure 2.3.3.1: User Home Page

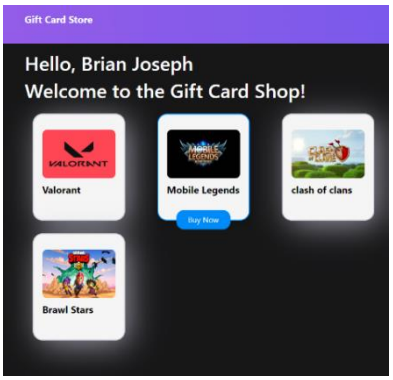

Figure 2.3.3.2: Game Cards

On top of the top of all the screen when logged in will have a top navigation bar to navigate to multiple functions of the system. The user home will have game cards where user can click to direct to the dedicated page of the game with the gift cards that user can buy to get the gift card code. When the cursor hovers on the cards, the users can see a hover design.

4. Selected Game

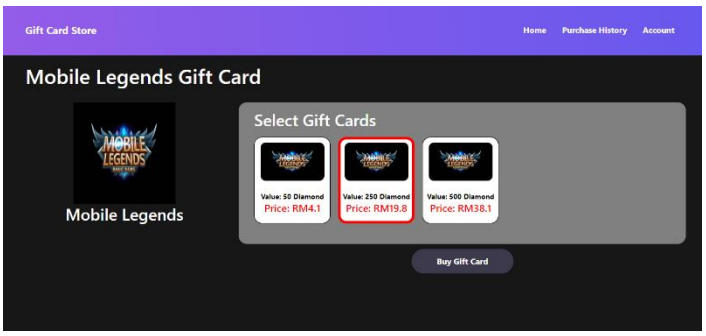

Figure 2.3.4.1: Selected Game Page

This page is where the User can select the gift cards available and click the buy gift card to direct to the payment page to buy the gift card.

5. Buy Gift Card Page

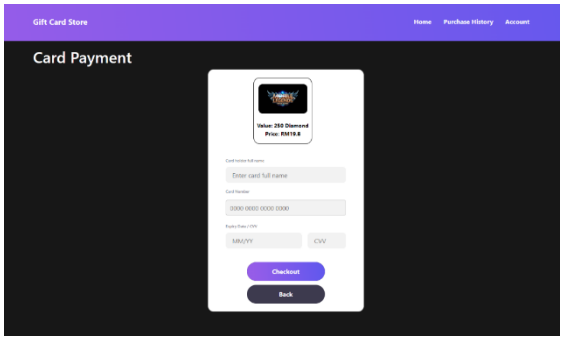

Figure 2.3.5.2: Payment Page

The user will input the payment details to make payments to check out the game gift card. The system will validate the input and if all is correct then the system will accept the payment.

6. Successful Payment with Gift Card Code Page

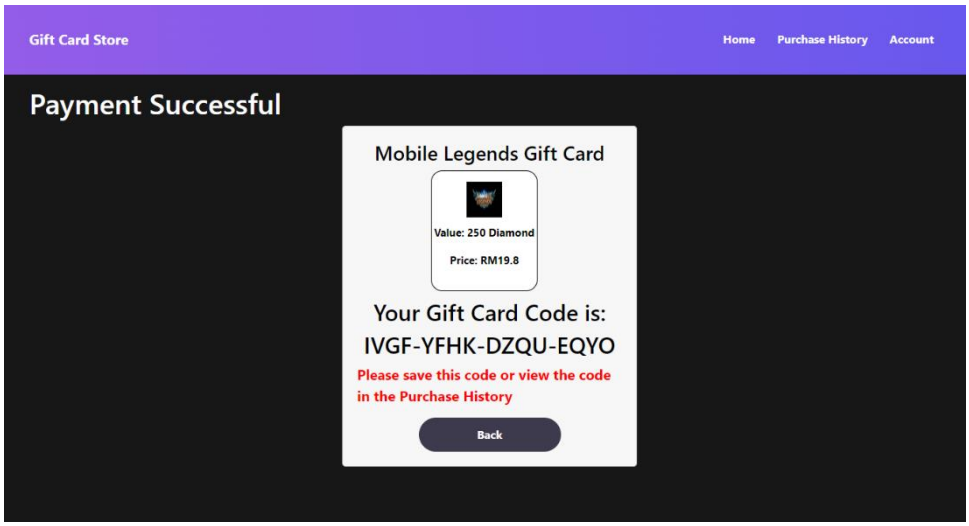

Figure 2.3.6.1: Payment Result Page

After a successful payment, the user will immediately get the gift card code for the gift card bought and has a back button to go back to the home page.

7. Purchase History

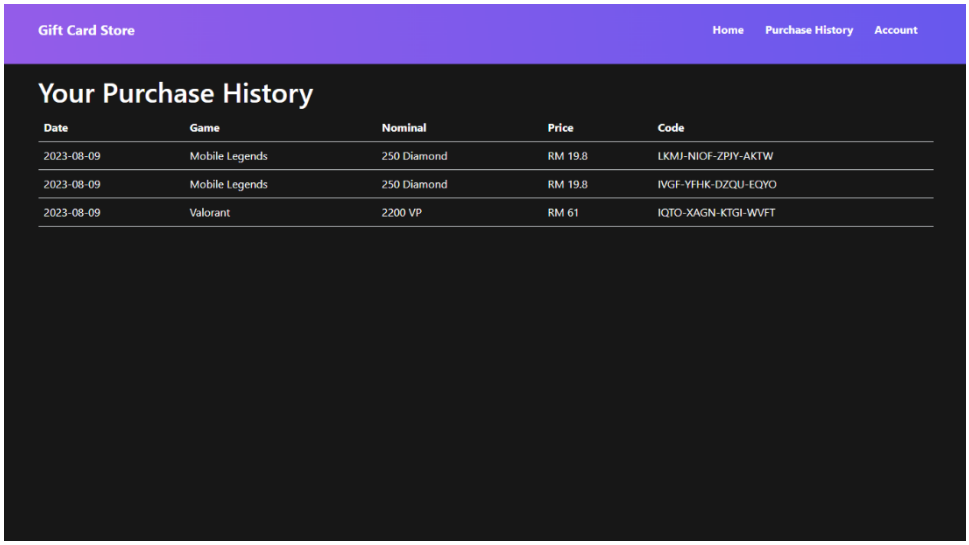

Figure 2.3.7.1: Purchase History Page

This is the user's purchase history, where the user is able to see the gift cards bought and the game code for the user to top up.

8. Account Page

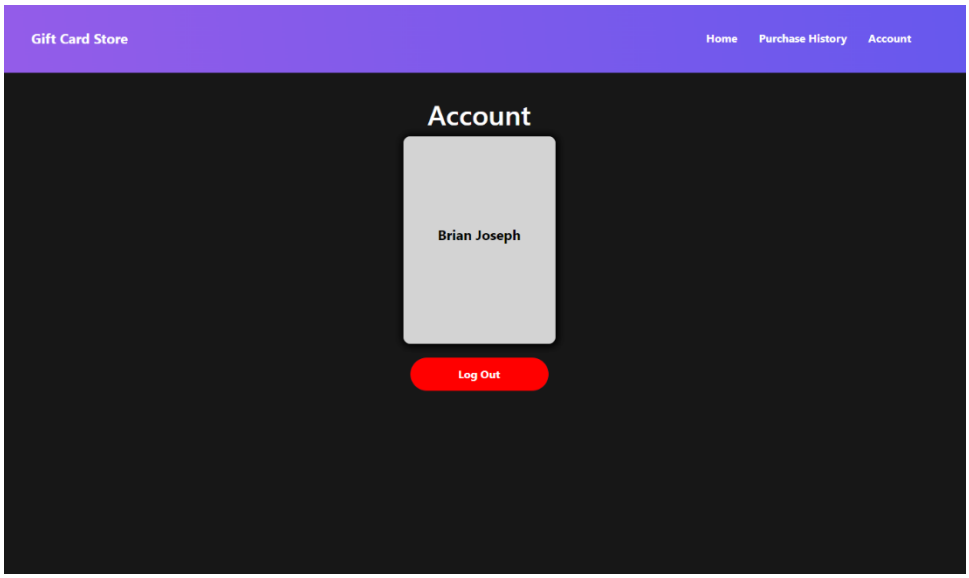

Figure 2.3.8.1: Account Page

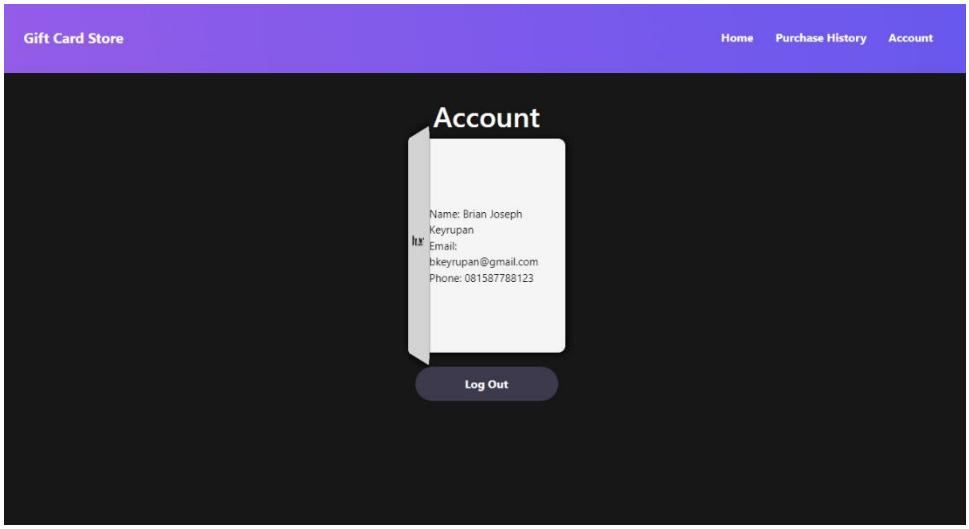

Figure 2.3.8.2: Account with opened cardPage

This is the account page of the user side. Here the user is able to see their user data like email, full name, and phone number. When user hovers on the card, they are able to see the user details. Then below that there is a log out button for the user to log out from the system.

9. Admin Home

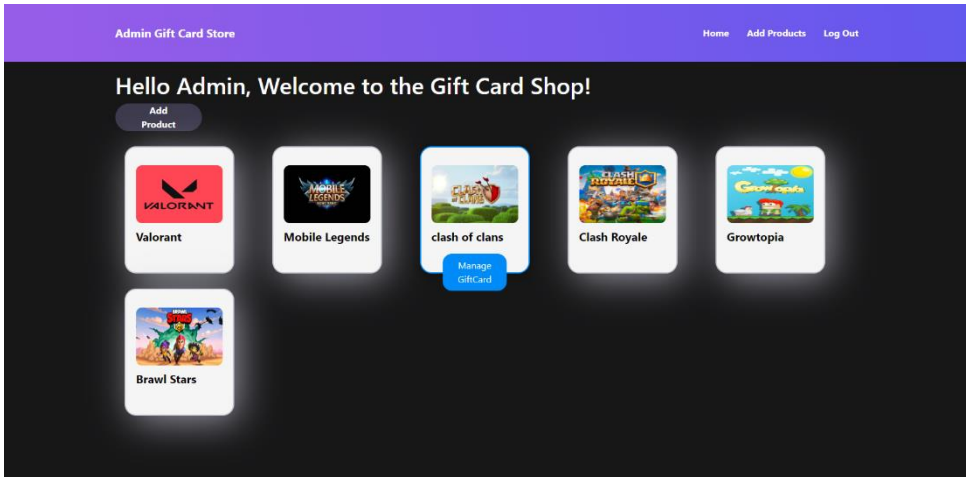

Figure 2.3.9.1: Admin Home Page

This is the admin home page, where the admin is able to see all the available games on the system and they are able to click each game to see the gift cards available. In there the admin is able to add more gift cards with different values along with the price.

10. Admin Add Game

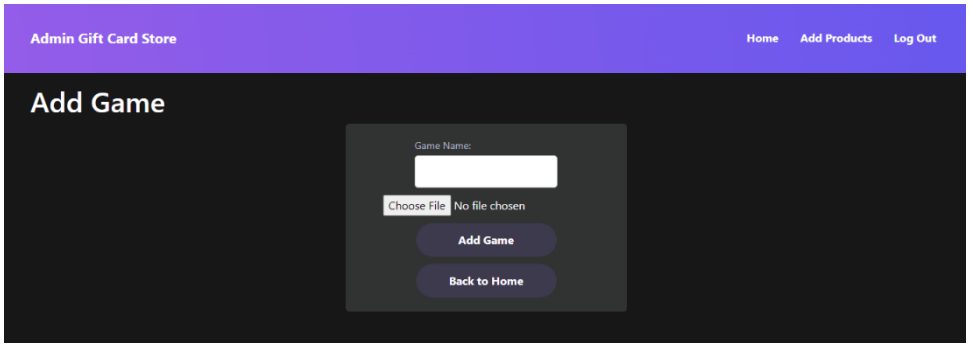

Figure 2.3.10.1: Admin Add Game Page

This is the add game page where admin is able to add new game and input the name of the game and the picture of the game.

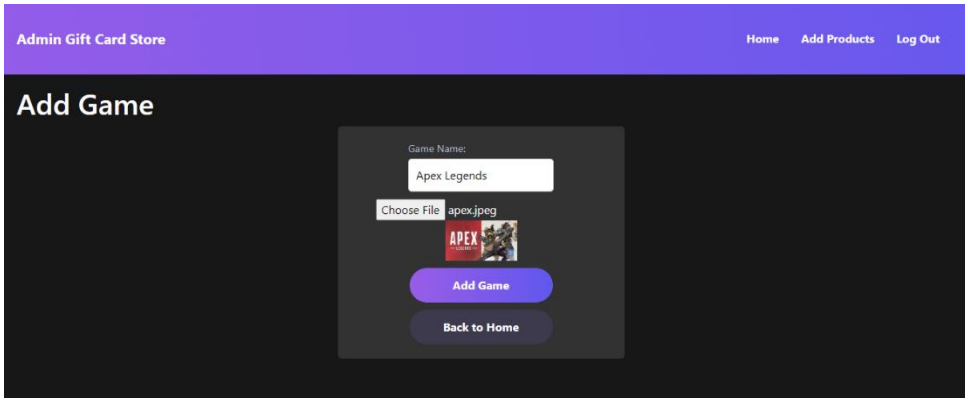

Figure 2.3.12.2: Admin Added Game Input Page

When the input is given this is what the page will look like and after inputting all the data, the admin can click the add game to store the data in the system.

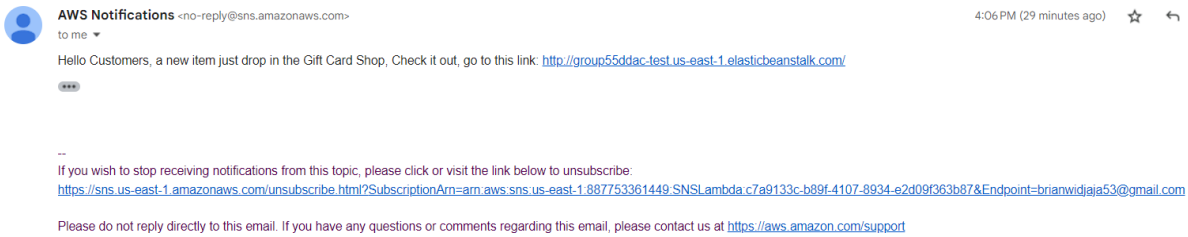

Figure 2.3.12.3: Email Notification When Game is Added

After admin adds the data into the system, the users of the system will be notified through their email using SNS, of that there is a new game added into the system. After adding the system will direct back to the home page. Showing the new game that just been added in the following figure *2.3.12.4 below*.

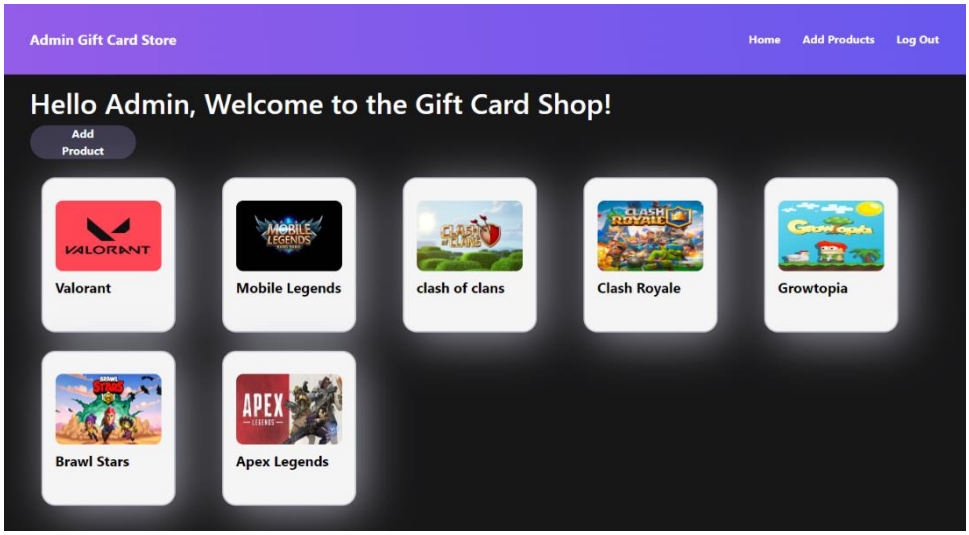

Figure 2.3.12.4: Admin Added Game Input Page

11. Admin Manage Gift Cards

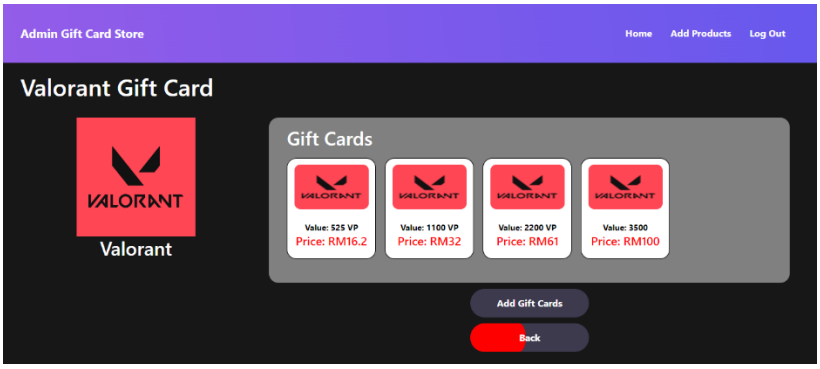

Figure 2.3.11.1: Admin Manage Gift Card Page

Here the admin is able to see all the gift cards available and a add gift card button to add a new value and price of the gift card. After clicking that button it will direct to the add gift card page.

12. Admin Add Gift Card

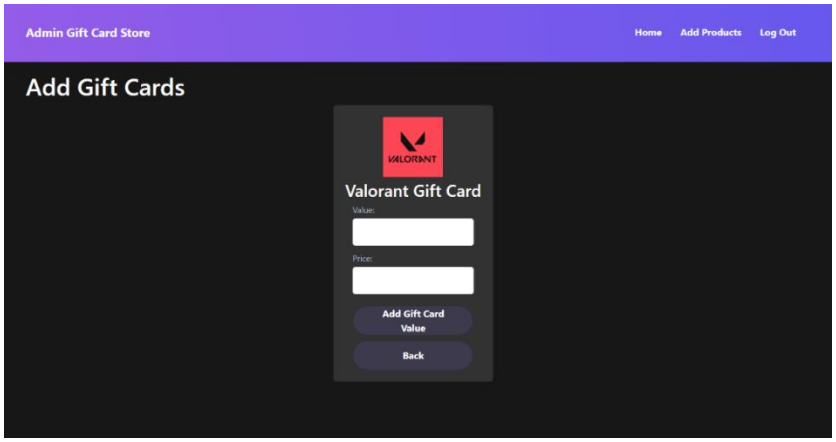

Figure 2.3.12.1: Admin Add Gift Card Page

This is the add gift card page where the admin will be asked to add the new value of the gift card and the new price. After inputting click the add gift card value button.

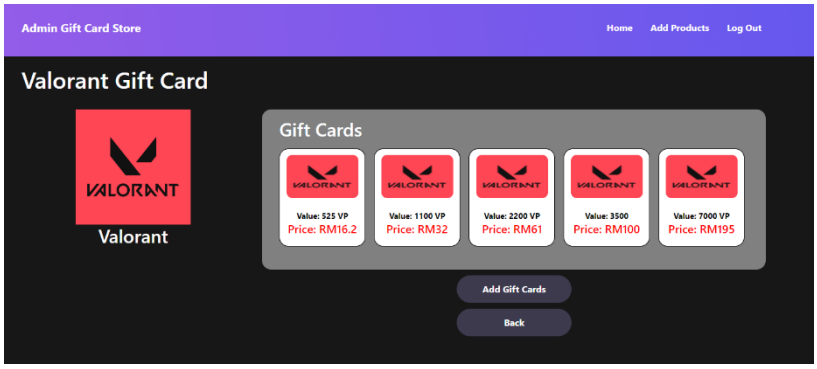

Figure 2.3.12.2: Admin Added Gift Card Page

After adding it will be directed to the add gift card page with the new value added shown on the page.

13. Logout Screen

Figure 2.3.13.1: Admin Logout Page

This page in the on the log out page of the admin side will have the log out button for admin to log out from the system.

14. Error

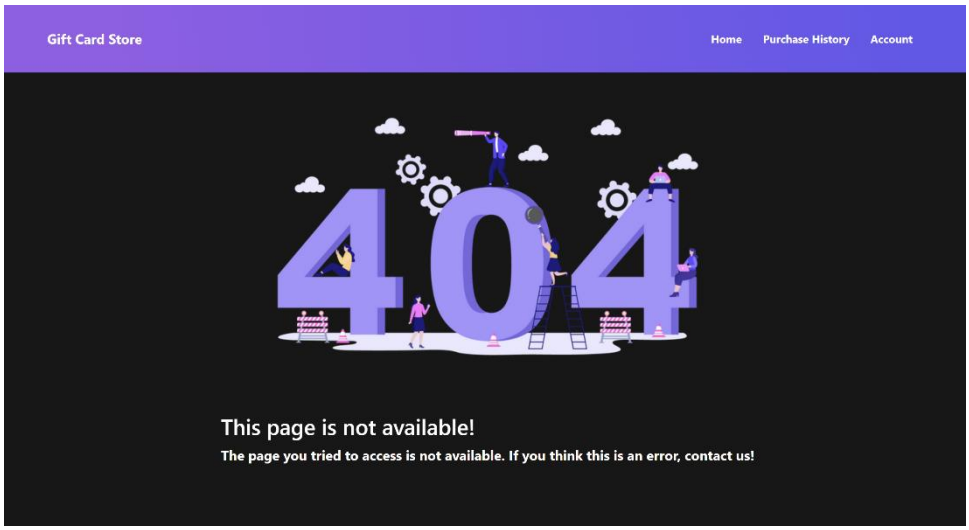

Figure 2.3.14.1: 404 Page

This page will be shown if the user tries to access a page that is not in the system by typing the link address of the system.

3.0 Test Plan

3.1 Unit and Integration Testing

Test ID	Test Description	Expected Output	Pass/Fail
T1	Login Function	If the user inputs the right credentials for the user will be directed to the user home page. If user inputs admin credentials, it directs to the admin home page. Other inputs that are not stored in the system will be alerted and notify the user that they inputted the wrong credentials.	Pass
T2	Login with wrong credential	If user inputs the wrong credential, the system will not direct to the home page	Pass
T3	Register	After user inputs all the inputs, the system will store the data into the system and directs users to the login page. Registered Email will receive an email to subscribe to the email newsletter service.	Pass
T4	Register with empty input	If user inputs with empty inputs, then the system will alert the user if there is an empty input	Pass
T5	Register with wrong email format	If the user inputs the email with the wrong email format, then the system will alert the user that the email format is incorrect.	Pass
T6	Register with short password	If user inputs the password with less than 6 letters, then system will alert the user if the password must be more than 6 letters	Pass
T7	Click Sign up button	Directs the web app to the Signup Screen.	Pass
T8	Click Sign in button	Directs the web app to the Sign in page.	Pass
T9	Admin add game	Admin will input the game name and the image of the game. If the admin inputs all the	Pass

		required inputs, then the system will store the data into the database, if some inputs are left empty it will alert the user to store the data.	
T10	Admin add product with no input	If admin has an empty input the system will have an alert in the web app.	Pass
T11	Add Gift Cards	Admin will input the value of the gift card and price of the gift card. If the admin inputs all the required inputs, then the system will store the data into the database, if some inputs are left empty it will alert the user to store the data.	Pass
T12	Add Gift Cards with empty inputs	If admin leaves an input box empty in the page then the admin will be alert the admin if there is an empty input	Pass
T13	Admin Game Cards	When admin clicks the game cards on the home page, the system should direct the page to the dedicated game page to manage the games.	Pass
T14	Admin Logout	When the admin clicks the log out page, the system should direct the user to the login screen.	Pass
T15	User Select Game	When user selects or clicks a game, the web app should direct the web to the that specific game page	Pass
T16	User Select Gift Card	When in the game page, the users can select the game gift card to buy. When the gift card is selected, the card should have a red border around it to indicate that the card is selected, then user can click the buy gift card button to proceed to check out the selected gift card.	Pass
T17	User Checkout	The checkout function is where the user will input their credit card or debit card detail to	Pass

		pay, and the system will validate the input boxes when the user clicks the checkout button. If all is ok, then the system will direct the user to the payment successful screen.	
T18	User Checkout with empty inputs	If user leaves the input payment detail empty, then system will alert the user if there is an empty input	Pass
T19	User View Purchase History	This page will display all the user's purchase history with the gift card purchase data like the date, game, nominal, price, and the gift card code.	Pass
T20	User Logout	When the user clicks the log out button, the system shall show the login screen.	Pass
T21	Not Found page	When the user manually clicks on a page link that is not in the system. The system should show an error 404 page where it means that the searched page is not available	Pass

3.2 Performance Testing

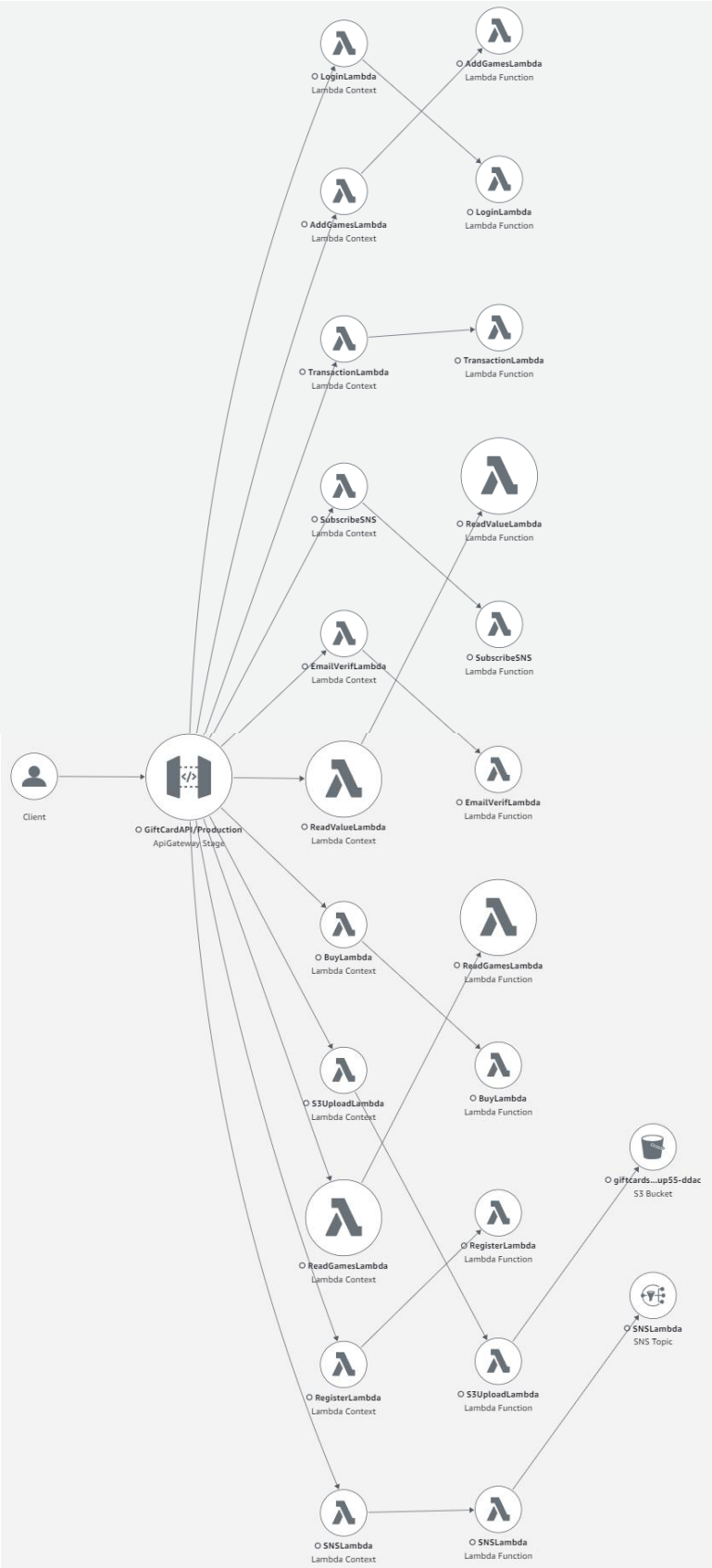

Figure 3.2.1 Amazon X-Ray Service Map

The developers will take advantage of Amazon's CloudWatch combined with X-Ray for the application's speed, functionality, and performance. AWS X-Ray enables programmers to evaluate, debug, and test the latency of their applications. By integrating this functionality, developers will be able to comprehend the application and detect difficulties that may arise after application deployment.

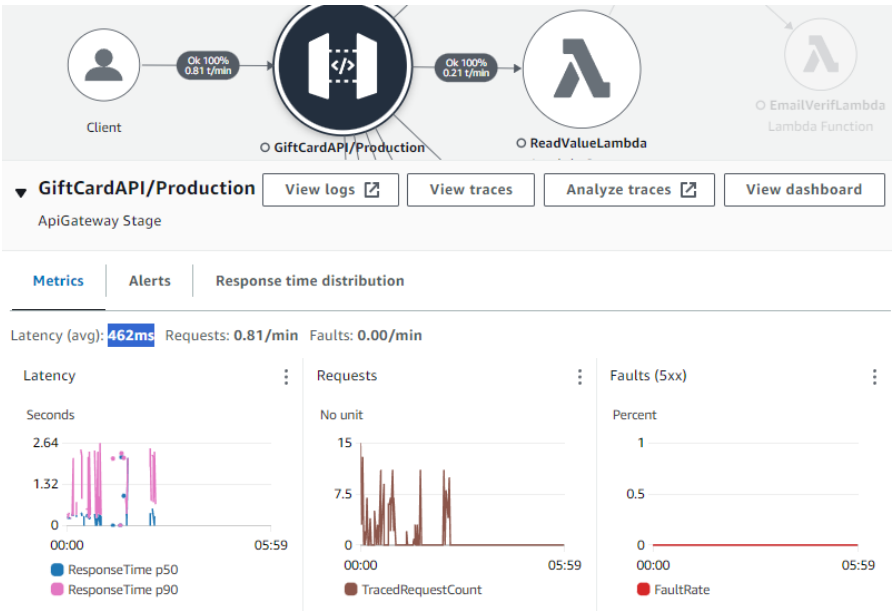

Figure 3.2.2 API Gateway Performance

As for the first and most important result, the main node that is connected to the client is the API Gateway. The average latency of it is 462ms. This can be considered slow for some people, and according to Charest (2023), the latency that is considered good is below 50ms for video conferencing and 300ms for other types of service that do not have a time-sensitive matters. Since this Gift Card Shop does not need time-sensitive things, then 462ms is generally still acceptable. A breakdown of detailed latency for each service will be described in the following paragraphs.

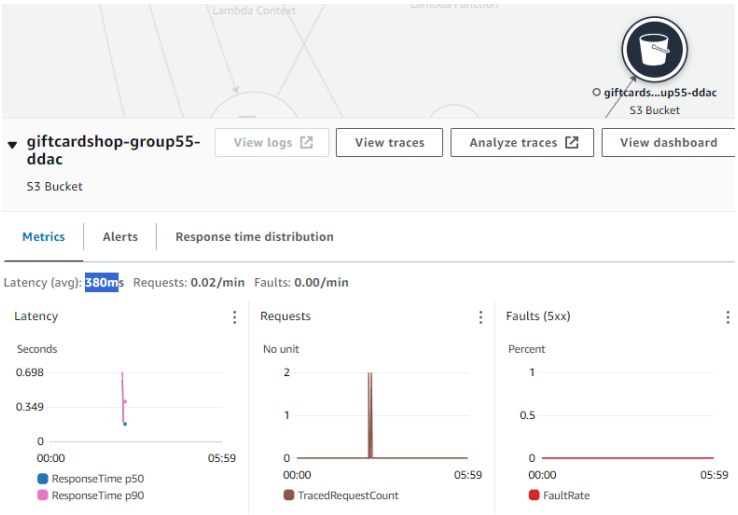

Figure 3.2.3 S3 Bucket Performance

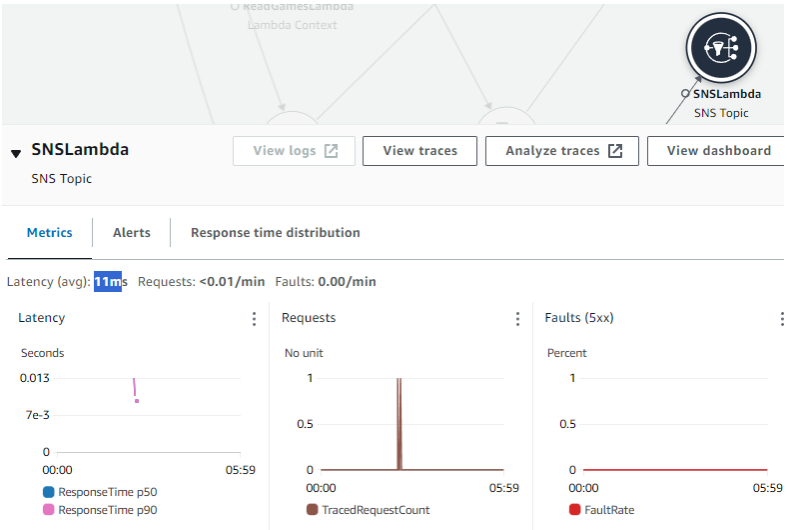

Figure 3.2.4 SNS Topic Performance

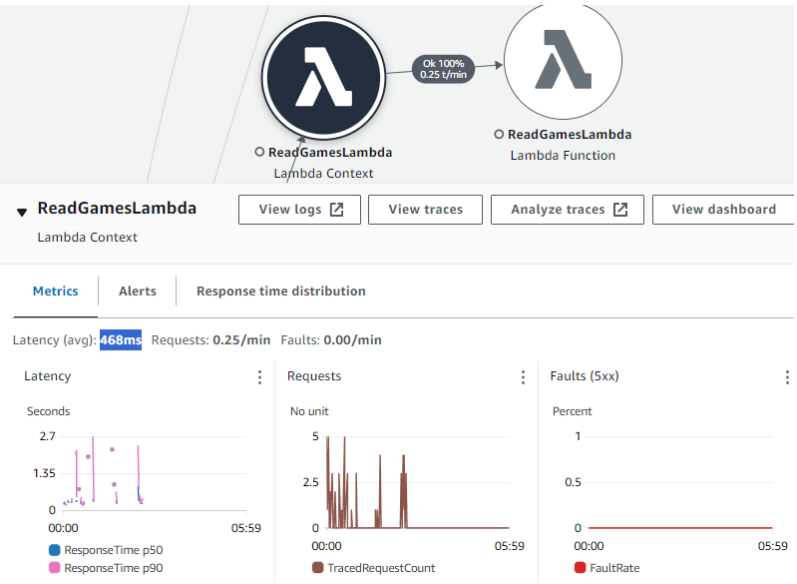

Figure 3.2.5 Lambda Context Performance

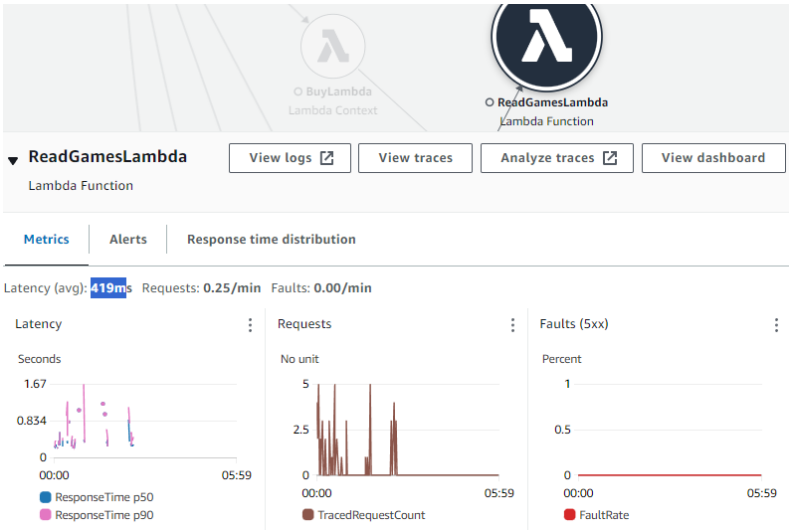

Figure 3.2.6 RDS Read Performance

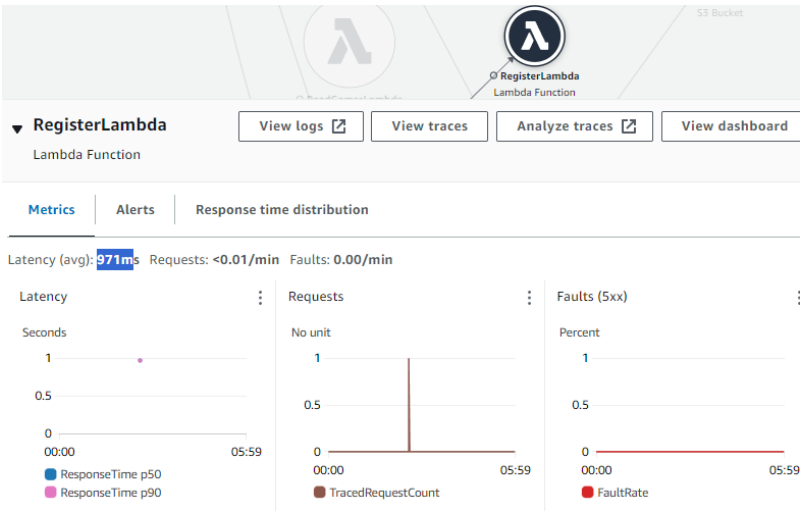

Figure 3.2.7 RDS Write Performance

The other services used in this project are Lambda, RDS, SNS, and S3. Firstly, the S3 bucket performance is pretty good, considering that the function used is for uploading images to the S3 bucket. As for the SNS topic on figure 3.2.4, it is considered excellent since the average latency is only 11ms. This exceptionally small latency helps to bring down the system's average latency. As seen on figure 3.2.1, Lambda creates 2 nodes, which is the context and function. Lambda context object provides information about the runtime environment, contains details like the AWS request ID, function name, function version, memory limits, and more (Sharma, 2021). Lambda Function is the code itself. Accessing this context information adds a minor overhead to the execution time of the Lambda function, so that is why figure 3.2.5 has a slight increase in the latency compared to the lambda function itself on figure 3.2.6.

As for the next discussion, it is about the comparison in reading and writing data into RDS. Reading data has much lower latency, which is only around 419 milliseconds, while writing into RDS takes an average of 971 milliseconds. This is due to the usage of Microsoft SQL Server, where insert statement must check current indexes and constraint, which add overhead to the INSERT operation, hence taking higher latency to complete the operations.

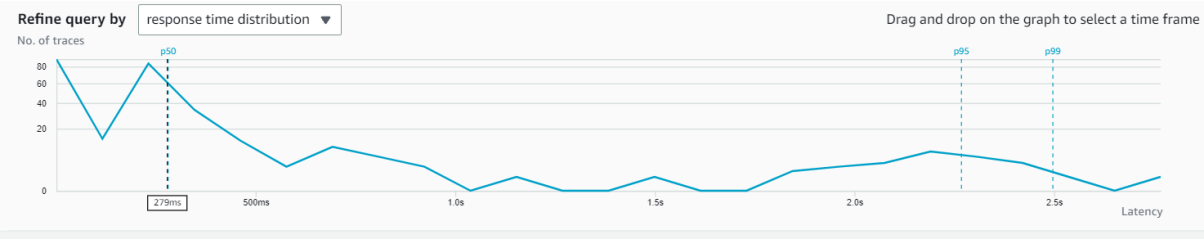

Figure 3.2.8 Response Time Distribution

Figure 3.2.8 shows the overall response time distribution of the system. P50 in this case means that the experience of 50% users will be around 279ms or less, while P95 means 95% and respectively P99 means 99%. This means only 5% of users experience response time more than 2.3 seconds, and only 1% experience response time longer than 2.5 seconds. While the median response time (P50) of 279ms suggests decent performance for the majority of users, the P95 and P99 reaction times of 2.3 seconds and 2.5 seconds, respectively, indicate that there are some anomalies or rare spikes when response times are significantly slower.

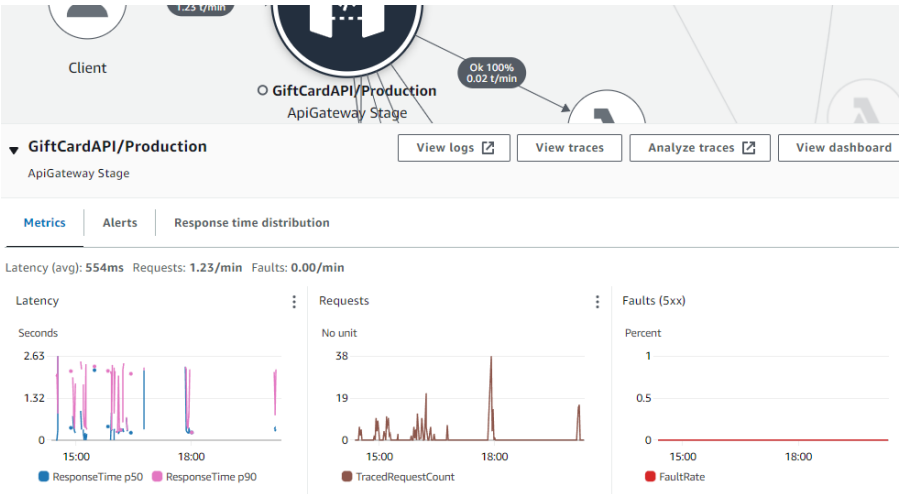

Figure 3.2.9 Peak Season Performance

In the case above, as can be seen from the request, it constantly has a high request, and at one point it reaches 38 request mark. Unlike the normal performance which is shown on figure 3.2.2, this performance testing is meant to measure and analyze how well the application handles peak request at one time. Even though the request to the web is spiking abnormally, the latency is only at 554ms without fault. It can be concluded that the web application can handle a lot of requests at the same time.

The reason why the web application can handle a lot of requests at the same time is because the usage of API Gateway and Lambda. AWS Lambda can scale automatically based on incoming traffic, which is beneficial for handling sudden spikes in demand (AWS, n.d. -d). In addition to that, API Gateway and Lambda are designed for high availability and fault tolerance. The next reason is because the configuration done on the Elastic Beanstalk on chapter 2.1.6. It is mentioned that the instance has a maximum of 4, which means that when there is a sudden spike, the instance will automatically turn on an additional one with a balanced load to maintain smooth web application experience. The last reason is because of the configuration on the API Gateway, in the throttle part on figure 2.1.5.11, which prevents the API Gateway from being overwhelmed by a sudden spike of request.

This architecture approach has its downside as well. In the performance view, Lambda has a limitation called cold start, where it could make the latency higher. While it is true that a direct ASP.NET function might be faster than this architecture, the developer still decides to go with this lambda and API or serverless architecture because it will be easier to scale and maintain in the long run.

4.0 System Limitation

One of the limitations of the system is that the deletion of game functionality is not able to be done in the system due to transactions and are the related to the game data. If deleted shall lose the game data in the transaction. Similar to this, another limitation is that the gift card values can’t be deleted as it has a relationship with the transactions and id of the game and gift card value are foreign key on the transaction. The system is also unable to delete the transactions data in order to maintain data integrity. Therefore, while this limitation might be restrictive or limit the system, instead it reinforces the need to prioritize data integrity and design applications that can handle relationships between entities.

5.0 Conclusion and Reflections

In conclusion, the development of the web app for the gift card industry, specifically the game gift card industry, using AWS services to develop the discussed product. The integration of some AWS services such as RDS, S3, Lambda, API Gateway, and SNS offers a robust and scalable web application. Amazon RDS creates a reliable and efficient database management to store the game gift card data and the user data, while Amazon S3 is used to create storage and retrieval of the game images. AWS Lambda functions helped in the development of the project, which is to create and execute serverless code enhancing the app's code to be modifiable and minimizing operational complexities. API Gateway creates a smooth interaction between the frontend and backend. Amazon SNS allows the web app to send notifications and updates to users when a new game is available in the web store which shall be used to enhance user engagement.

During the development using Elastic Beanstalk to deploy the app made it much easier to put the app online. It's like setting up a system that can automatically adjust itself to handle more people using the app without crashing. Therefore, using elastic beanstalk makes it simple to deploy and creates a scalable product. AWS X-Ray and CloudWatch helped us monitor how well the app was running, it showed us if there were any problems or any slow parts, so we can fix them and manage the app to increase the speed of the API and make sure the web app is working smoothly.

Through this project, we not only learned the services from Amazon Web Services (AWS), but we also improved our skills in developing web apps that can be deployed.

6.0 References

AWS. (n.d.-a). *Amazon S3*. Amazon Web Services, Inc. https://aws.amazon.com/pm/serv-s3/?trk=55ffcfa3-95d3-4418-9a79-62a64040b867&sc_channel=ps&ef_id=Cj0KCQjwho-lBhC_ARIsAMpgMof6f1m7__EZOZfH8vMEoRiRD3oacEenksZh4t9hoI7dGK4FHA0uDT0aAsuKEALw_wcB:G:s&s_kwcid=AL!4422!3!536397034780!e!!g!!s3%20cloud%20storage!11543056249!112002966909

AWS. (n.d.-b). *Amazon Simple Storage Service (S3) — Cloud Storage — AWS*. Amazon Web Services, Inc. https://aws.amazon.com/s3/faqs/?nc1=h_ls

AWS. (n.d.-c). *Resilience in Amazon API Gateway - Amazon API Gateway*. Amazon Web Service. https://docs.aws.amazon.com/apigateway/latest/developerguide/disaster-recovery-resiliency.html

AWS. (n.d.-d). *Scaling and concurrency in Lambda - AWS Lambda*. Amazon Web Service. https://docs.aws.amazon.com/lambda/latest/operatorguide/scaling-concurrency.html

AWS. (n.d.-e). *What is Amazon EC2? - Amazon Elastic Compute Cloud*. Amazon Web Services. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html

AWS. (n.d.-f). *What is Amazon Relational Database Service (Amazon RDS)? - Amazon Relational Database Service*. https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Welcome.html

AWS. (n.d.-g). *What is Amazon SNS? - Amazon Simple Notification Service*. https://docs.aws.amazon.com/sns/latest/dg/welcome.html

Carty, D. (2021). Amazon API Gateway. *SearchAWS*. https://www.techtarget.com/searchaws/definition/Amazon-API-Gateway

Charest, F. (2023, March 13). *What is Latency: The Hitchhiker's Guide - Obkio*. Obkio. https://obkio.com/blog/what-is-latency/

GeeksforGeeks. (2023). Introduction to AWS Elastic Beanstalk. *GeeksforGeeks*. https://www.geeksforgeeks.org/introduction-to-aws-elastic-beanstalk/

Sharma, P. (2021, December 13). Understanding the execution of AWS lambda functions. *Medium*. https://codeburst.io/understanding-the-execution-of-aws-lambda-functions-cc6145f2180b

Taylor, D. (2023). What is AWS Lambda? Lambda Function with Examples. *Guru99*. https://www.guru99.com/aws-lambda-function.html