

CLOUD COMPUTING LAB

Serverless Feedback Collection System

ROLL NO	NAME	BATCH
2201070	SHUBHAM PANDEY	T21

AIM :- Implement a cloud-based solution that integrates atleast 2 AWS services.

1. INTRODUCTION

The purpose of this project is to create a **cloud-based feedback collection system** using AWS services, which allows users to submit feedback through a simple web form. The system integrates **AWS Lambda** for backend processing and **API Gateway** to expose the functionality as a RESTful API. This solution demonstrates how multiple AWS services can work together to build a scalable and serverless application.

2. OBJECTIVES

The primary objective of this project is to create a **functional feedback collection system** using **AWS Lambda** and **API Gateway**. By the end of the project, the system should:

- Allow users to submit feedback via a web form.
- Process the feedback data in a serverless manner using **AWS Lambda**.
- Expose the functionality via a RESTful API using **API Gateway**.
- Ensure seamless communication between the frontend and backend while considering **CORS** issues and error handling.

3. GOAL

The goal of this project is to implement a **serverless feedback collection system** on the cloud using AWS. This project aims to demonstrate the ability to:

- Leverage **AWS Lambda** for serverless backend processing.
- Use **API Gateway** to expose the backend as a RESTful API.
- Handle user inputs and provide real-time feedback to users through a simple frontend interface.

CLOUD COMPUTING LAB

Serverless Feedback Collection System

4. USE CASE SCENARIOS

- **Scenario 1:** A user fills out a feedback form with their name, email, rating, and feedback message on a webpage.
- **Scenario 2:** The user clicks the submit button, and the frontend sends the form data to the **API Gateway** endpoint.
- **Scenario 3:** The **API Gateway** triggers the **Lambda function**, which processes the data and returns a success message.
- **Scenario 4:** The frontend receives the success response and informs the user that their feedback was successfully submitted.

5. AWS SERVICES USED

The following AWS services were used in this project:

1. **AWS Lambda:** Used for processing the feedback data from the frontend.
2. **Amazon API Gateway:** Used to expose the Lambda function as a RESTful API that can be accessed from the frontend.

6. SYSTEM ARCHITECTURE

The system architecture of the project is as follows:

1. **Frontend:** A simple HTML form where users can submit their feedback. The form sends data to the backend (API Gateway).
2. **API Gateway:** Exposes the Lambda function as a RESTful API endpoint. The frontend sends feedback to this endpoint.
3. **Lambda Function:** Processes the incoming feedback data and returns a response to the API Gateway.
4. **CORS Configuration:** Ensures proper communication between the frontend (which may be running on a local server) and the API Gateway.

CLOUD COMPUTING LAB

Serverless Feedback Collection System

7. IMPLEMENTATION DETAILS

The implementation involves several steps:

- 1. Create a Lambda Function:**
 - The Lambda function is responsible for receiving the feedback data from the API Gateway, processing it (e.g., validation, formatting), and returning a response.
- 2. API Gateway Setup:**
 - A REST API is created on API Gateway to handle the /submit-feedback route. This route triggers the Lambda function upon receiving a POST request.
- 3. CORS Configuration:**
 - CORS is configured on API Gateway to allow the frontend (running on localhost for development) to communicate with the API Gateway without encountering cross-origin errors.
- 4. Frontend Setup:**
 - The frontend is a simple HTML form where users can enter their feedback. JavaScript is used to submit the form data to the API Gateway endpoint.

8. RESULT AND OBSERVATION

After integrating **AWS Lambda** and **API Gateway**, the system functions as expected. The feedback form on the frontend sends data to the API Gateway, which triggers the Lambda function. The Lambda function processes the data and returns a success response to the frontend, which then informs the user of successful submission.

CLOUD COMPUTING LAB

Serverless Feedback Collection System

The screenshot shows two consecutive pages from the AWS Lambda console:

- Create function (Step 1):** This page allows users to choose how to create their function. Options include "Author from scratch" (selected), "Use a blueprint", and "Container image". It also includes sections for "Basic information" (Function name: FeedbackHandler, Runtime: Node.js 22.x, Architecture: x86_64), "Environment variables", and "AWS Lambda layers". A sidebar provides details about the x86_64 architecture.
- FeedbackHandler (Step 2):** This page shows the newly created function. A green success message states: "Successfully created the function FeedbackHandler. You can now change its code and configuration. To invoke your function with a test event, choose 'Test'." The "Function overview" section displays the function name, ARN, and configuration details like description, last modified (1 second ago), and function URL. It also shows triggers and destinations. Navigation tabs at the bottom include Code, Test, Monitor, Configuration, Aliases, and Versions.

CLOUD COMPUTING LAB

Serverless Feedback Collection System

The screenshot shows the AWS Lambda console interface. At the top, a banner indicates that the function 'FeedbackHandler' has been successfully created. The main area displays the 'Function overview' for 'FeedbackHandler'. It includes a diagram showing the function's structure, a description field, and ARN information. Below this, tabs for 'Code', 'Test', 'Monitor', 'Configuration', 'Aliases', and 'Versions' are visible. The 'Code' tab is selected, showing the code source editor. The code file 'index.mjs' contains the following JavaScript:

```
JS index.mjs
1 exports.handler = async (event) => {
2   const feedback = JSON.parse(event.body);
3
4   console.log("Received feedback:", feedback);
5
6   return {
7     statusCode: 200,
8     body: JSON.stringify({ message: "Feedback received successfully!" }),
9   };
10};
```

The AWS navigation bar at the top of the browser window also shows the Lambda service.

CLOUD COMPUTING LAB

Serverless Feedback Collection System

The screenshot shows the AWS Lambda Function Editor for a function named 'FeedbackHandler'. The code is written in JavaScript and handles an incoming event, parses its JSON body, logs the received feedback, and returns a success response with a message.

```
JS index.mjs X [Option+S]
Choose the saved event
Create new test event
1 exports.handler = async (event) => {
2   const feedback = JSON.parse(event.body);
3
4   console.log("Received feedback:", feedback);
5
6   return {
7     statusCode: 200,
8     body: JSON.stringify({ message: "Feedback received successfully!" })
9   };
10 }
```

The interface includes a sidebar for 'EXPLORER', 'TEST EVENTS [NONE SELECTED]', and 'ENVIRONMENT VARIABLES'. A modal window titled 'Select test event' is open, prompting the user to choose or create a test event. The browser address bar shows the URL for the Lambda function's configuration page.

The screenshot shows the 'Test event' configuration page for the 'FeedbackHandler' function. It allows users to invoke the function without saving an event by creating a new JSON event or editing a saved one. The current configuration is for a 'Private' event named 'TestFeedback'.

Test event [Info](#) [CloudWatch Logs Live Tail](#) [Save](#) [Test](#)

To invoke your function without saving an event, configure the JSON event, then choose Test.

Test event action Create new event Edit saved event

Event name

Maximum of 25 characters consisting of letters, numbers, dots, hyphens and underscores.

Event sharing settings

Private
This event is only available in the Lambda console and to the event creator. You can configure a total of 10. [Learn more](#)

Shareable
This event is available to IAM users within the same account who have permissions to access and use shareable events. [Learn more](#)

Template - optional [C](#)

Event JSON [Format JSON](#) [Copy](#)

```
1 * {
2   "body": "{\"name\":\"Shubham\",\"email\":\"shubham@example.com\",\"rating\":5,\"message\":\"Awesome tool!\"}"
3 }
4 }
```

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

CLOUD COMPUTING LAB

Serverless Feedback Collection System

The screenshot shows the AWS Lambda console interface. A new function named "FeedbackHandler" has been created. The "Test event" section is active, showing a JSON template for a feedback event:

```
1 {  
2   "body": "{\"name\": \"Shubham\", \"email\": \"shubham@example.com\", \"rating\": \"5\", \"message\": \"Awesome tool!\"}"  
3 }  
4
```

The screenshot shows the AWS Lambda console interface after the function has been updated. A success message is displayed:

- Successfully updated the function **FeedbackHandler**.
- The test event "TestFeedback" was successfully saved.

The "Code" tab is selected. In the "Execution" section, it shows:

- Executing function: succeeded ([logs](#))
- Details

The "Test event" section is also visible, showing the same JSON template as before:

```
1 {  
2   "body": "{\"name\": \"Shubham\", \"email\": \"shubham@example.com\", \"rating\": \"5\", \"message\": \"Awesome tool!\"}"  
3 }  
4
```

CLOUD COMPUTING LAB

Serverless Feedback Collection System

Screenshot of the AWS Lambda console showing the FeedbackHandler function details.

FeedbackHandler | Functions

FeedbackHandler

Successfully updated the function **FeedbackHandler**.
The test event "TestFeedback" was successfully saved.

Executing function: succeeded (logs [?])

Details

```
{ "statusCode": 200, "body": "{\"message\":\"Feedback received successfully!\"}" }
```

Summary

Code SHA-256
yhN1R+sG3rbMqUwEDSOg2yU91Z18jrWFxfngK1xN8c0=

Function version
\$LATEST

Duration
54.60 ms

Resources configured
128 MB

Init duration
145.13 ms

Log output

The area below shows the last 1 KB of the execution log. Click here [?] to view the corresponding CloudWatch logs.

Execution time
24 seconds ago

Request ID
19751a53-53cb-479d-8b0e-216f8960d12a

Billed duration
55 ms

Max memory used
71 MB

Screenshot of the AWS API Gateway console.

CloudShell **Feedback**

API Gateway

Networking & Content Delivery

API Gateway

Create and manage APIs at scale

Amazon API Gateway is a fully managed service that makes it easy for developers to create, publish, maintain, monitor, and secure APIs.

Get started

Create a new API to begin exploring API Gateway. You can also import an external definition file into API Gateway.

Create an API

How it works

API Gateway enables you to connect and access data, business logic, and functionality from backend services such as workloads running on Amazon Elastic Compute Cloud (Amazon EC2), code running on AWS Lambda, any web application, or real-time communication applications.

Pricing

With Amazon API Gateway, you only pay when your APIs are in use. There are no minimum fees or upfront commitments. For HTTP and REST APIs, you pay based on API calls received and amount of data transferred out. For WebSocket APIs, you pay based on number of messages and connection duration.

[Learn more about pricing \[?\]](#)

Resources

[Getting Started Guide](#)
[Developer Guide](#)
[API References](#)

CLOUD COMPUTING LAB

Serverless Feedback Collection System

Step 1
Configure API

Step 2 - optional
Configure routes

Step 3 - optional
Define stages

Step 4
Review and create

API details

API name
An HTTP API must have a name. The name is a non-unique value you use to identify and organize your APIs. To programmatically refer to this API, use the API ID that API Gateway generates for you.

FeedbackAPI

IP address type | Info
Select the type of IP addresses that can invoke the default endpoint for your API. You don't need to redeploy your API for the update to take effect.

IPv4
Includes only IPv4 addresses.

Dualstack
Includes IPv4 and IPv6 addresses.

Integrations (1) | Info
Specify the backend services that your API will communicate with. These are called integrations. For a Lambda integration, API Gateway invokes the Lambda function and responds with the response from the function. For an HTTP integration, API Gateway sends the request to the URL that you specify and returns the response from the URL.

Lambda

AWS Region Lambda function Version | Learn more

ap-south-1 arn:aws:lambda:ap-south-1:038462792156:function:Fee 2.0

Add integration

Cancel Review and create Next

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Step 1
Configure API

Step 2 - optional
Configure routes

Step 3 - optional
Define stages

Step 4
Review and create

Configure routes - optional

Configure routes | Info
API Gateway uses routes to expose integrations to consumers of your API. Routes for HTTP APIs consist of two parts: an HTTP method and a resource path (e.g., GET /pets). You can define specific HTTP methods for your integration (GET, POST, PUT, PATCH, HEAD, OPTIONS, and DELETE) or use the ANY method to match all methods that you haven't defined on a given resource.

Method Resource path Integration target

POST /submit-feedback → FeedbackHandler

Add route

Cancel Review and create Previous Next

CLOUD COMPUTING LAB

Serverless Feedback Collection System

The screenshot shows the 'Create API' wizard in the AWS API Gateway console. The current step is 'Step 4: Review and create'. The summary shows:

- API name and integrations:** API name is 'FeedbackAPI' and IP address type is 'IPv4'. Integration is 'FeedbackHandler (Lambda)'.
- Routes:** A single route is defined: POST /submit-feedback → FeedbackHandler (Lambda).
- Stages:** A single stage is defined: \$default (Auto-deploy: enabled).

At the bottom right, there are 'Cancel', 'Previous', and 'Create' buttons.

The screenshot shows the 'Routes' page for the 'FeedbackAPI' in the AWS API Gateway console. The left sidebar includes sections for API Gateway, Develop, Deploy, Monitor, and Protect. The main area displays:

- A green success message: "Successfully created API FeedbackAPI (9ipr1nabr3)."
- A 'Routes' section with a 'Routes for FeedbackAPI' table. It shows one route: '/submit-feedback' (POST method) with the status 'Choose a route.'
- Buttons for 'Stage: -' and 'Deploy'.

At the bottom right, there are 'Cancel', 'Previous', and 'Create' buttons.

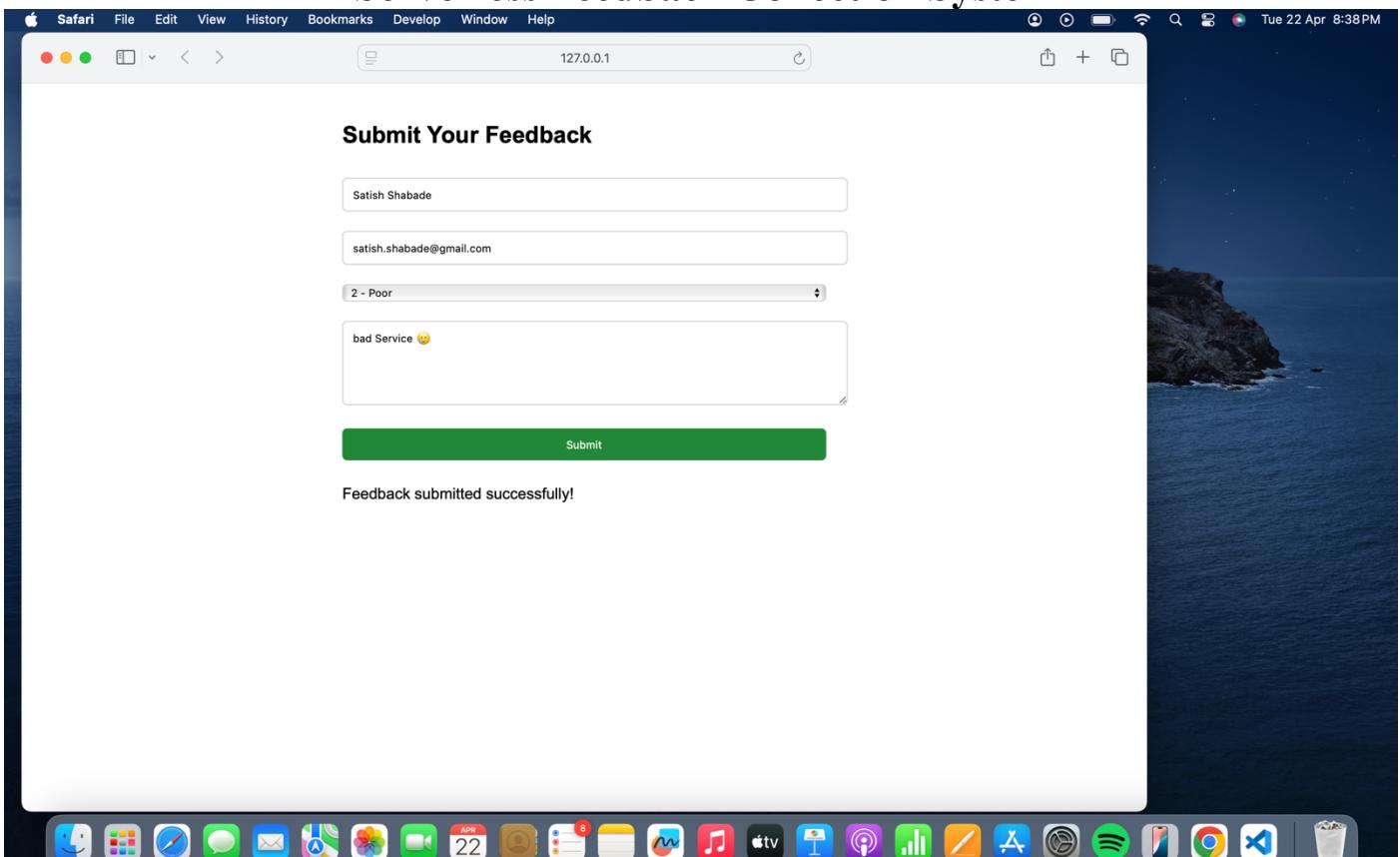
CLOUD COMPUTING LAB

Serverless Feedback Collection System

The screenshot displays two windows side-by-side. The left window is the AWS API Gateway 'Stage details' page for a 'FeedbackAPI'. It shows a green success message: 'Successfully created API FeedbackAPI (9ipr1nabr3)'. The 'Stages' section lists '\$default'. The 'Stage details' panel shows the stage was created on April 22, 2025, at 7:54 PM, with the invoke URL <https://9ipr1nabr3.execute-api.ap-south-1.amazonaws.com>. The deployment ID is 4hw0ke, and it was created on April 22, 2025, at 7:54 PM. The deployment description is 'Automatic deployment triggered by changes to the Api configuration'. The right window is a web browser showing a feedback collection form titled 'Submit Your Feedback'. The form fields include Name ('Shubham PANDEY'), Email ('shubham78p@gmail.com'), Rating ('5 - Excellent'), and a comment box ('Excellent Service!'). A green 'Submit' button is at the bottom. Below the form, a message says 'Feedback submitted successfully!'.

CLOUD COMPUTING LAB

Serverless Feedback Collection System



```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Feedback Form</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <h2>Submit Your Feedback</h2>
    <form id="feedbackForm">
        <input type="text" id="name" placeholder="Your Name" required />
        <input type="email" id="email" placeholder="Your Email" required />
        <select id="rating" required>
            <option value="">Rate us</option>
            <option value="5">5 - Excellent</option>
            <option value="4">4 - Good</option>
            <option value="3">3 - Okay</option>
            <option value="2">2 - Poor</option>
            <option value="1">1 - Terrible</option>
        </select>
        <textarea id="message" placeholder="Your feedback..." rows="5" required></textarea>
        <button type="submit">Submit</button>
    </form>

    <p id="statusMessage"></p>

<script>
    const form = document.getElementById('feedbackForm');
    const statusMessage = document.getElementById('statusMessage');

    form.addEventListener('submit', async (e) => {
        e.preventDefault();

        const data = {
            name: form.name.value,
            email: form.email.value,
            rating: form.rating.value,
            message: form.message.value
        };

        const response = await fetch('/submit', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify(data)
        });

        if (response.ok) {
            statusMessage.textContent = 'Feedback submitted successfully!';
        } else {
            statusMessage.textContent = 'Error submitting feedback';
        }
    });
</script>

```

CLOUD COMPUTING LAB

Serverless Feedback Collection System

The screenshot shows a code editor interface with two tabs open: 'index.html' and 'lambda_function.py'. The 'index.html' tab contains the following code:

```
<html lang="en">
<body>
<script>
    form.addEventListener('submit', async (e) => {
        const data = {
            name: form.name.value,
            email: form.email.value,
            rating: form.rating.value,
            message: form.message.value
        };

        try {
            const response = await fetch('https://9ipr1nabr3.execute-api.ap-south-1.amazonaws.com/submit-feedback', {
                method: 'POST',
                headers: { 'Content-Type': 'application/json' },
                body: JSON.stringify(data)
            });

            if (response.ok) {
                statusMessage.innerText = "Feedback submitted successfully!";
                // Add a delay before resetting the form (e.g., 3 seconds)
                setTimeout(() => {
                    form.reset();
                }, 3000); // Delay of 3 seconds
            } else {
                statusMessage.innerText = "Error submitting feedback.";
            }
        } catch (error) {
            statusMessage.innerText = "Network error. Try again later.";
        }
    });
</script>
</body>
</html>
```

The 'lambda_function.py' tab contains the following code:

```
def lambda_handler(event, context):
    # TODO implement
    pass
```

The screenshot shows the AWS API Gateway CORS configuration page for the 'FeedbackAPI (9ipr1nabr3)' API. The left sidebar shows navigation options like APIs, Develop, Deploy, Monitor, and Protect.

Cross-Origin Resource Sharing

Configure CORS Info

CORS allows resources from different domains to be loaded by browsers. If you configure CORS for an API, API Gateway ignores CORS headers returned from your backend integration. See our [CORS documentation](#) for more details.

Access-Control-Allow-Origin: http://127.0.0.1:5500

Access-Control-Allow-Headers: content-type

Access-Control-Allow-Methods: POST

Access-Control-Expose-Headers: No Expose Headers are allowed

Access-Control-Max-Age: 0 Seconds

Access-Control-Allow-Credentials: NO

CLOUD COMPUTING LAB

Serverless Feedback Collection System

The screenshot shows the AWS Cloud Home console with the following layout:

- Left Sidebar:** Recently visited services including API Gateway, Billing and Cost Management, Lambda, Elastic Beanstalk, Lightsail, and S3.
- Top Right Area:** Applications section showing 0 applications. It includes a "Create application" button and a search bar for finding applications.
- Middle Left Area:** Welcome to AWS section with a "Getting started with AWS" link and a "Learn the fundamentals and find valuable information to..." link.
- Middle Right Area:** Cost and usage section showing current month costs at \$0.00 and no cost data available.
- Bottom Navigation:** Includes CloudShell, Feedback, Privacy, Terms, and Cookie preferences.

9. ADVANTAGES

- Serverless:** The use of **AWS Lambda** and **API Gateway** means no server management is required, making the application scalable and cost-efficient.
- Scalable:** The solution can handle large numbers of feedback submissions as AWS automatically scales the services.
- Cost-Effective:** Both **Lambda** and **API Gateway** have pay-as-you-go pricing, meaning you only pay for the actual usage.

10. LIMITATIONS

- Limited Storage:** The system does not store feedback data persistently. For persistent storage, additional services like **DynamoDB** would need to be integrated.
- No Authentication:** The system does not have any authentication or authorization mechanisms, which could be added in the future for additional security.
- CORS Issues in Local Development:** During local testing, handling **CORS** configuration for local development could be cumbersome.

CLOUD COMPUTING LAB

Serverless Feedback Collection System

11. CHALLENGES FACED AND HOW THEY WERE OVERCOME

i) CORS Configuration

- **Challenge:** Initially, the CORS (Cross-Origin Resource Sharing) setup on **API Gateway** was not correctly configured, resulting in errors when submitting feedback from the frontend.
- **Solution:** I configured CORS correctly in the **API Gateway settings** by allowing the appropriate origins (e.g., `http://127.0.0.1:5500` for local development). This ensured the frontend could communicate with the API Gateway.

ii) Lambda Timeout

- **Challenge:** The Lambda function was experiencing occasional timeouts when processing requests, leading to failed feedback submissions.
- **Solution:** I reviewed the Lambda function logs and found that the function was not handling requests optimally. I optimized the Lambda function logic to minimize execution time, ensuring that it runs within the 15-minute limit and performs efficiently.

iii) Debugging Lambda Responses

- **Challenge:** The Lambda function wasn't returning the correct response to the API Gateway, causing errors in the frontend.
- **Solution:** I fixed the Lambda function to ensure it properly returned a **JSON response** with the correct structure (status and message). This allowed the API Gateway to forward the correct response to the frontend.

CLOUD COMPUTING LAB

Serverless Feedback Collection System

12. SETUP INSTRUCTIONS

Step-by-Step Setup for the Project:

1. Create a Lambda Function:

- Go to the **AWS Management Console** and navigate to **Lambda**.
- Click **Create function** and select **Author from Scratch**.
- Choose a runtime (e.g., Node.js, Python) and provide a name for the function (e.g., FeedbackHandler).
- In the function code, process the input (feedback) from the API Gateway and return a response.

2. Set Up API Gateway:

- Go to **API Gateway** in the AWS Management Console.
- Create a new **REST API** and define a resource (e.g., /submit-feedback).
- Set up the **POST method** for this resource and link it to the Lambda function created in step 1.
- Enable **CORS** for the method to allow communication with the frontend (e.g., local development server at <http://127.0.0.1:5500>).
- Deploy the API and take note of the **Invoke URL**.

3. Configure CORS:

- In the API Gateway, under the **POST method** settings for the /submit-feedback route, ensure **CORS** is configured correctly.
- In the **Method Response**, add the appropriate headers, such as Access-Control-Allow-Origin: * and Access-Control-Allow-Methods: POST.

4. Frontend Setup:

- Create an HTML file (index.html) with a simple feedback form that includes fields for name, email, rating, and a message.
- Add JavaScript code to handle form submission, using the **Fetch API** to send the form data to the API Gateway endpoint.
- Ensure that the fetch() method is pointing to the **Invoke URL** from the API Gateway.

5. Testing the System:

- Run the **frontend** (e.g., open index.html in a browser) and submit feedback through the form.
- Check the Lambda function logs (via **CloudWatch**) to ensure the feedback is processed correctly.
- Test the success and error cases and ensure that the correct responses are returned to the frontend.

CLOUD COMPUTING LAB

Serverless Feedback Collection System

13. CONCLUSION

The project successfully integrated **AWS Lambda** and **API Gateway** to create a serverless feedback submission system. The system allows users to submit feedback through a frontend form, and the feedback is processed in the cloud using Lambda. The solution demonstrates the use of cloud computing services in a real-world application and highlights how AWS services can be integrated to build scalable, cost-effective systems.

14. FUTURE ENHANCEMENTS

■ Integration with DynamoDB for Persistent Data Storage:

- Currently, the feedback is not stored persistently. You could integrate AWS DynamoDB to store feedback data, allowing you to retrieve and analyze feedback over time.

■ User Authentication and Authorization:

- Implement user authentication using AWS Cognito to ensure that only authenticated users can submit feedback. You could also provide role-based access to monitor or manage feedback data.

■ Real-time Feedback Analytics:

- Add an analytics dashboard using Amazon QuickSight or AWS Lambda functions to visualize feedback trends, such as ratings over time, common issues, etc.

■ Automated Email Notifications:

- Integrate AWS SES (Simple Email Service) to send automated thank-you emails to users upon feedback submission or notify administrators when new feedback is submitted.

■ Enhance Frontend with React or Angular:

- Upgrade the frontend by developing a more dynamic interface using frameworks like React or Angular to improve user experience and interactivity.

CLOUD COMPUTING LAB

Serverless Feedback Collection System

■ API Rate Limiting and Security:

- Implement API rate limiting using AWS API Gateway to prevent abuse of the feedback API. You could also enhance security with AWS WAF (Web Application Firewall) to protect against malicious traffic.

■ Machine Learning for Sentiment Analysis:

- Integrate AWS SageMaker to implement a sentiment analysis model that can automatically categorize feedback as positive, neutral, or negative based on the content.

■ Mobile App Integration:

- Extend the solution to mobile platforms by developing an Android or iOS app that interacts with the same backend for feedback collection.

■ Multi-Language Support:

- Enhance the user experience by supporting multiple languages for the feedback form. This can be done using Amazon Translate or a custom approach based on the regions you serve.

■ Data Validation and Spam Prevention:

- Implement advanced data validation and spam prevention mechanisms, such as CAPTCHA or reCAPTCHA, to ensure that the feedback collected is legitimate and not from automated bots.