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.

7. IMPLEMENTATION DETAILS

The implementation involves several steps:

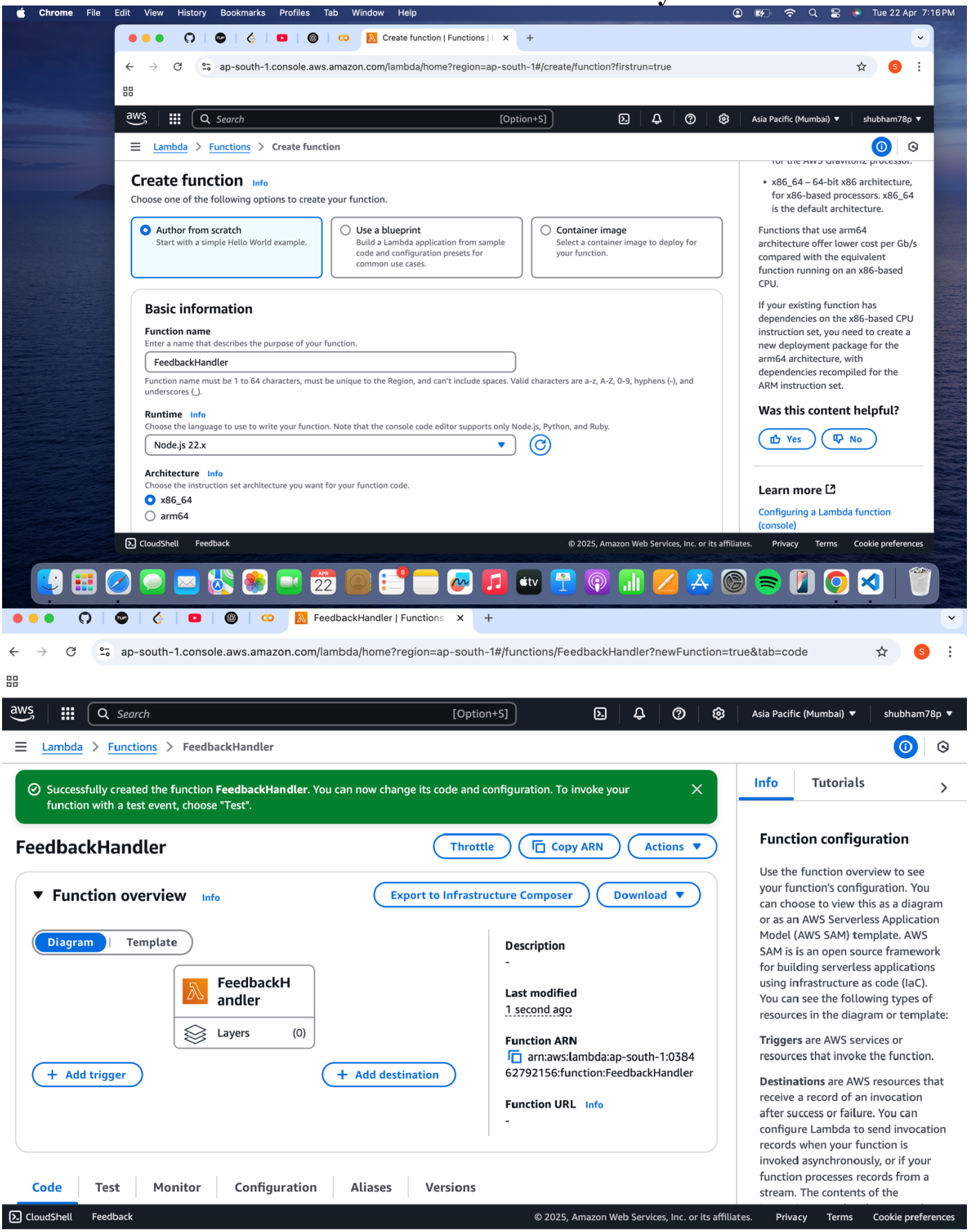
1. **Create a Lambda Function**:
   - The Lambda function is responsible for receiving the feedback data from the API Gateway, processing it (e.g., validation, formatting), and returning a response.
2. **API Gateway Setup**:
   - A REST API is created on API Gateway to handle the /submit-feedback route. 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.

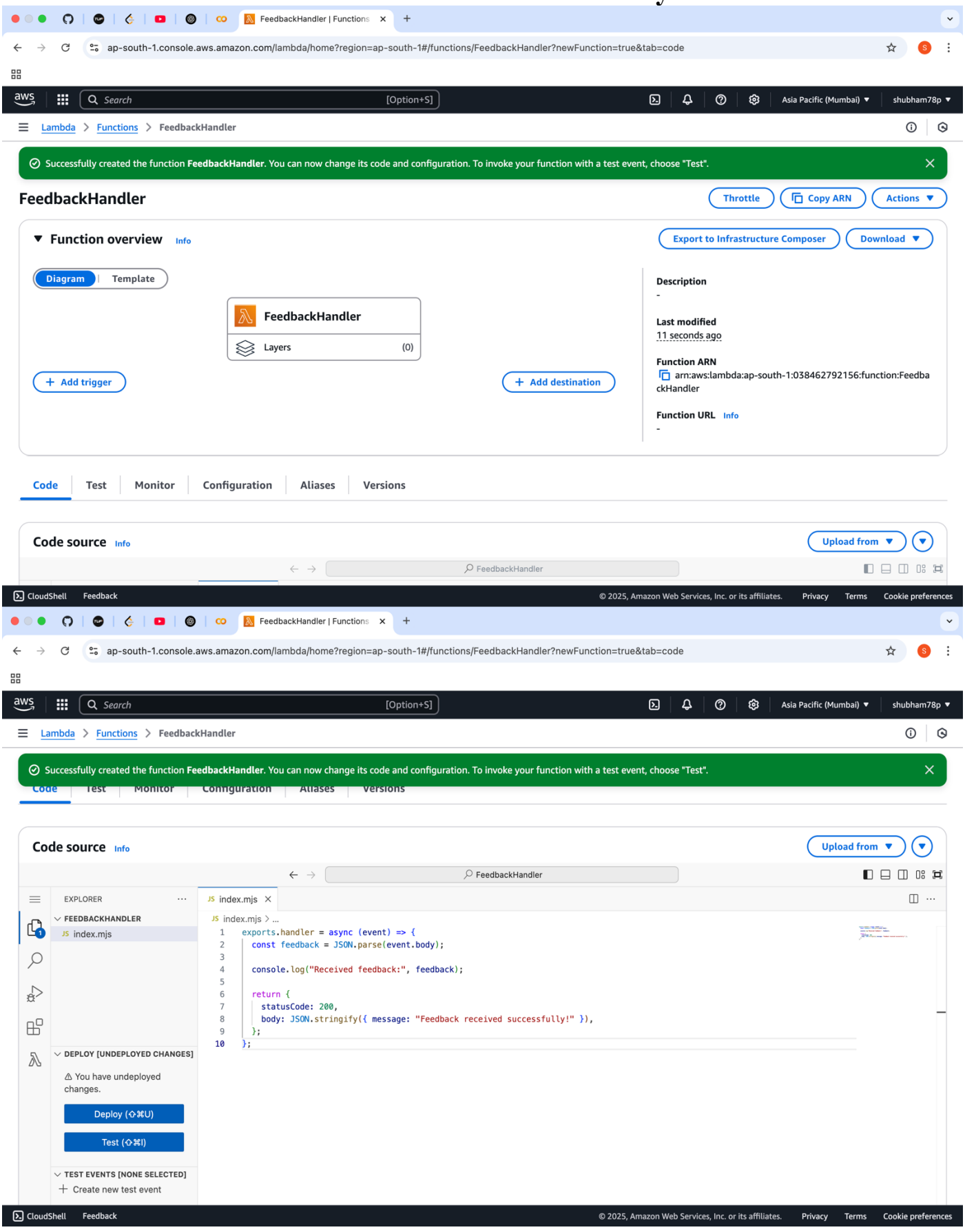
CLOUD COMPUTING LAB

Serverless Feedback Collection System



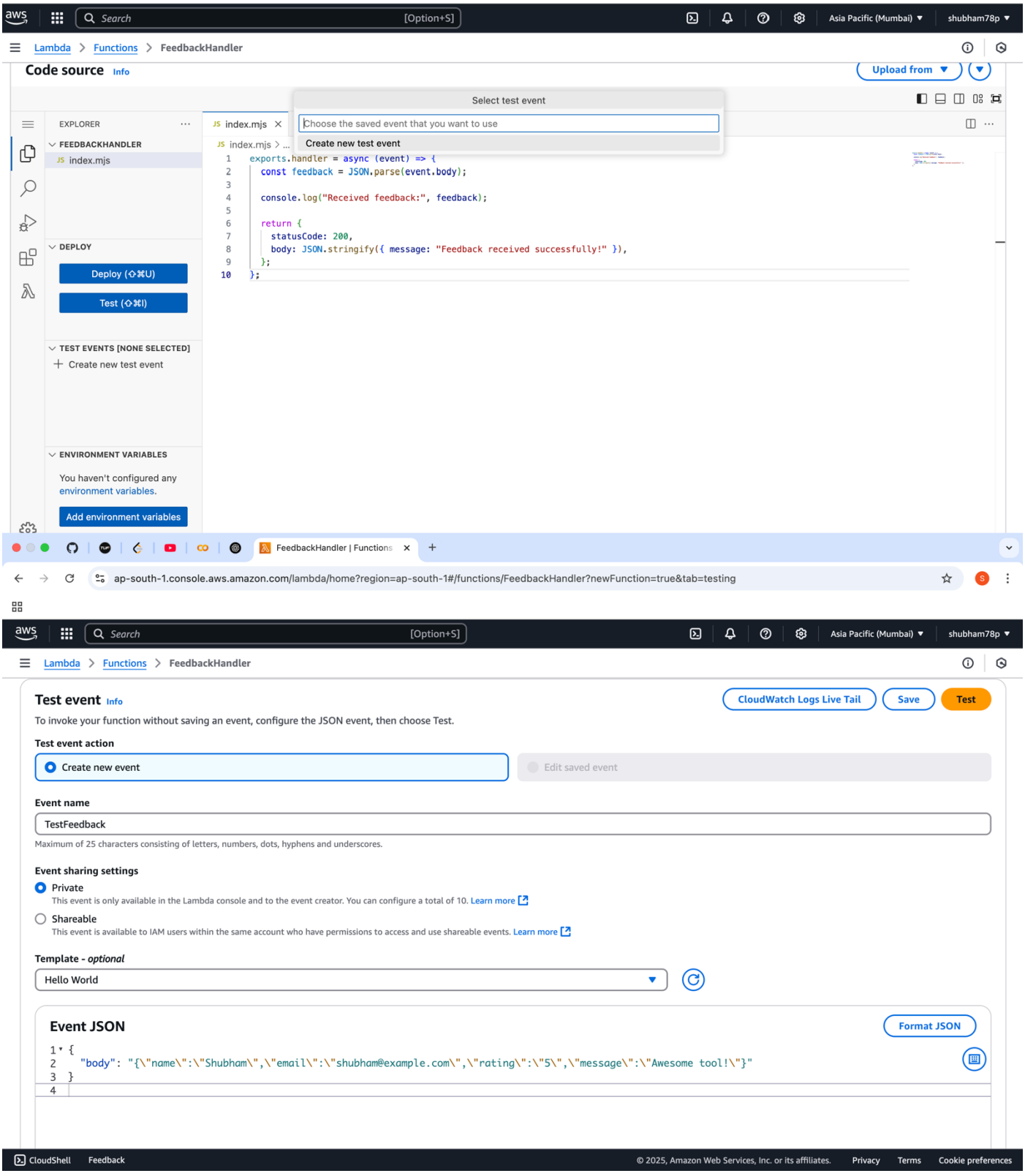
CLOUD COMPUTING LAB

Serverless Feedback Collection System



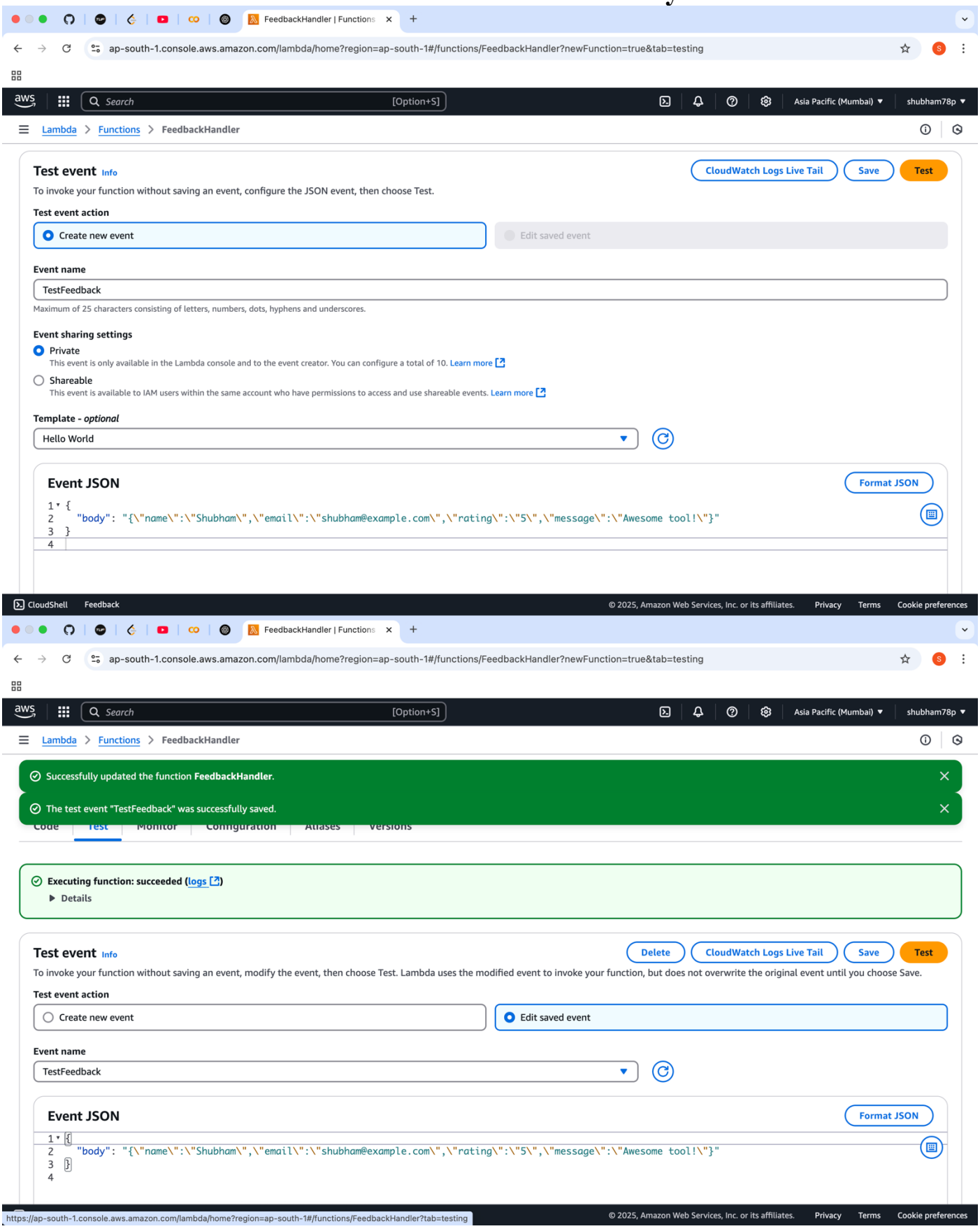
CLOUD COMPUTING LAB

Serverless Feedback Collection System



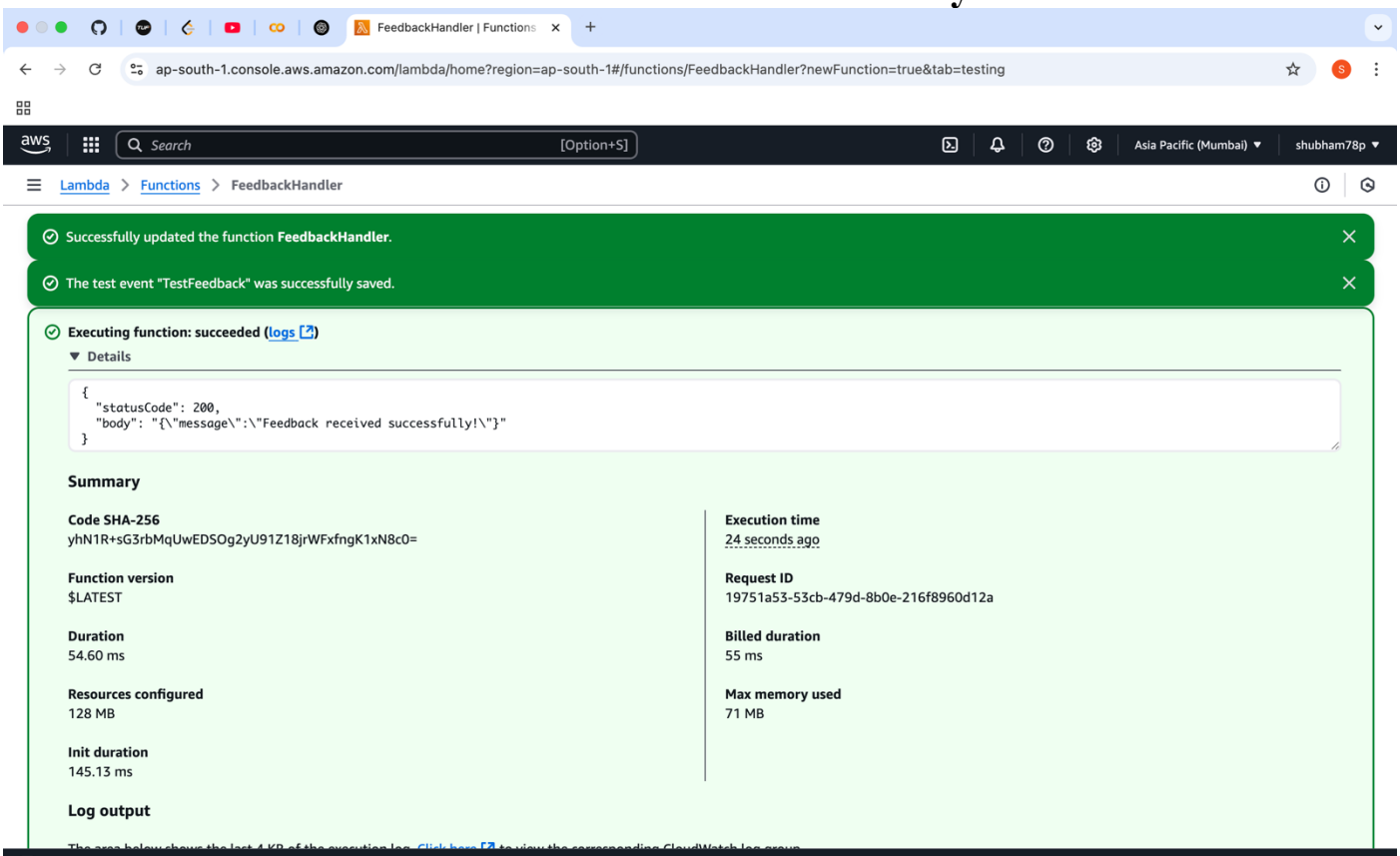
CLOUD COMPUTING LAB

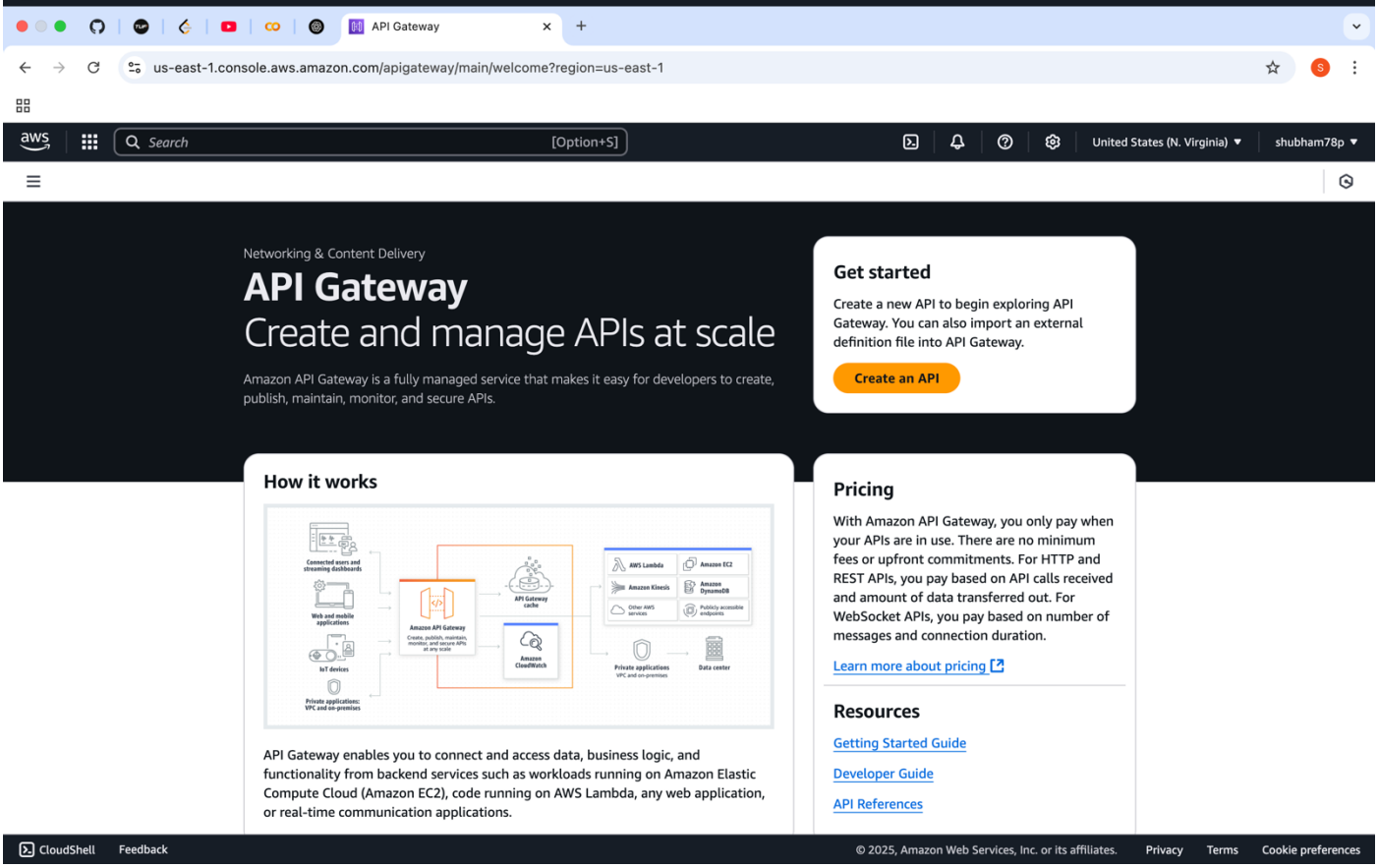
Serverless Feedback Collection System



CLOUD COMPUTING LAB

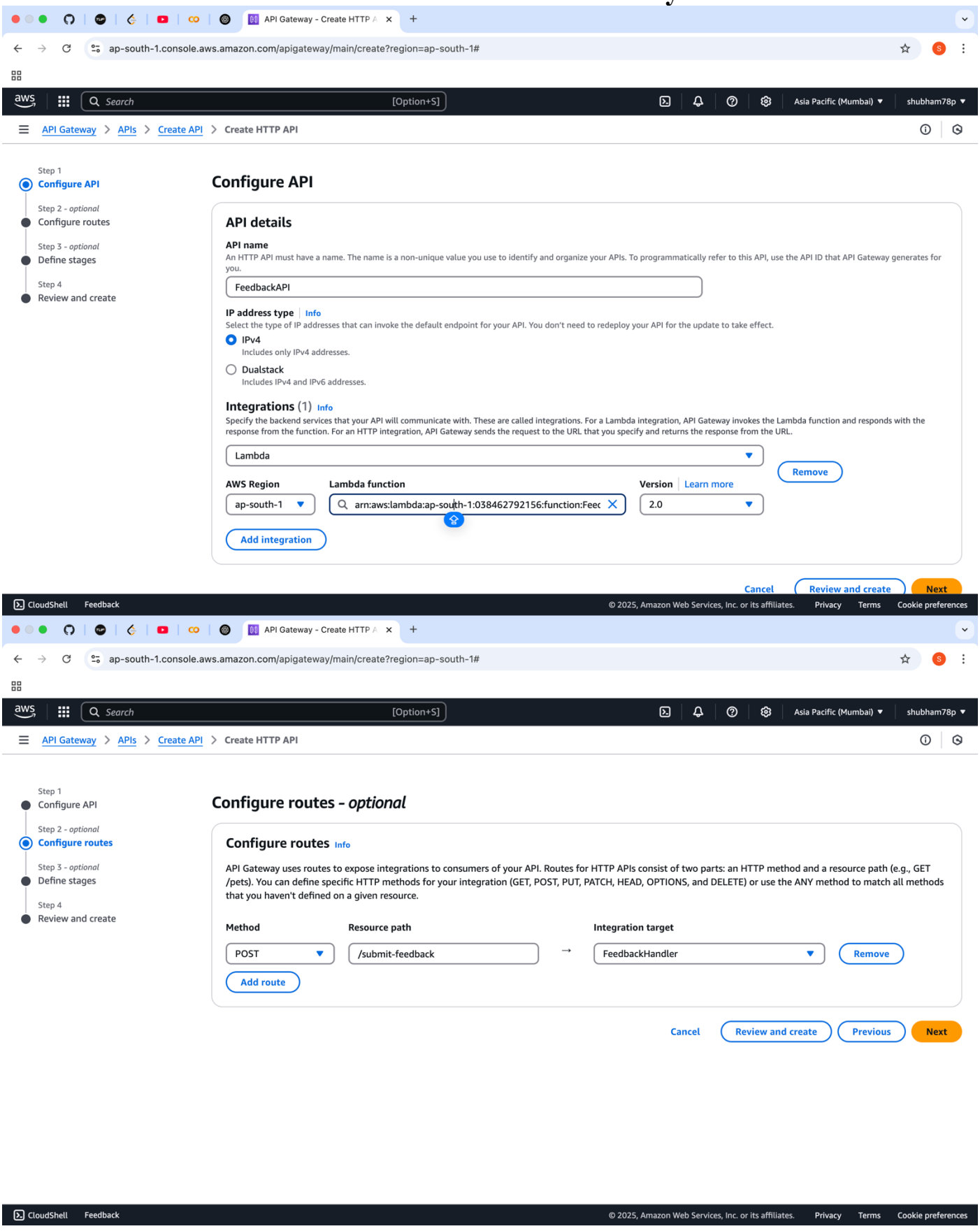
Serverless Feedback Collection System





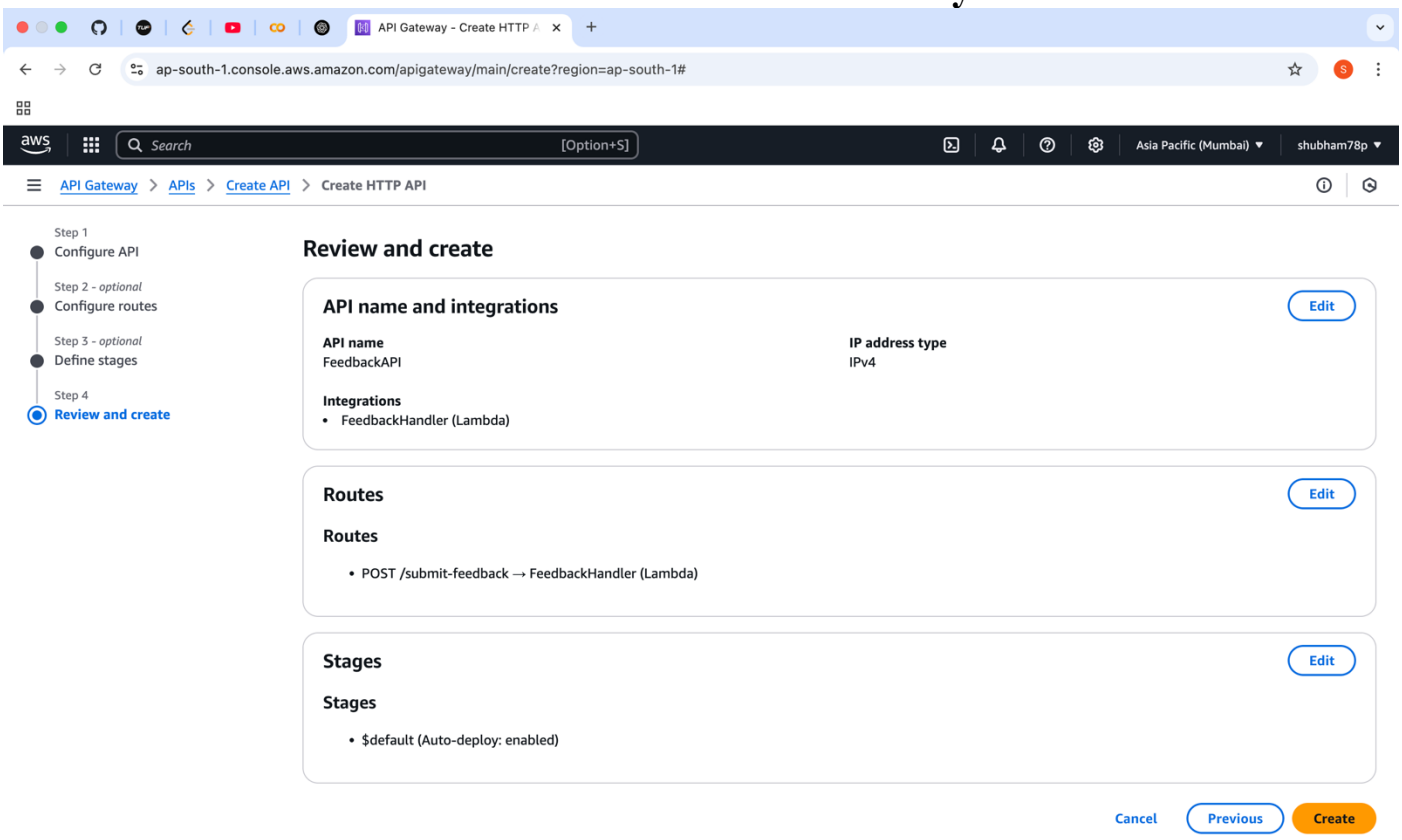
CLOUD COMPUTING LAB

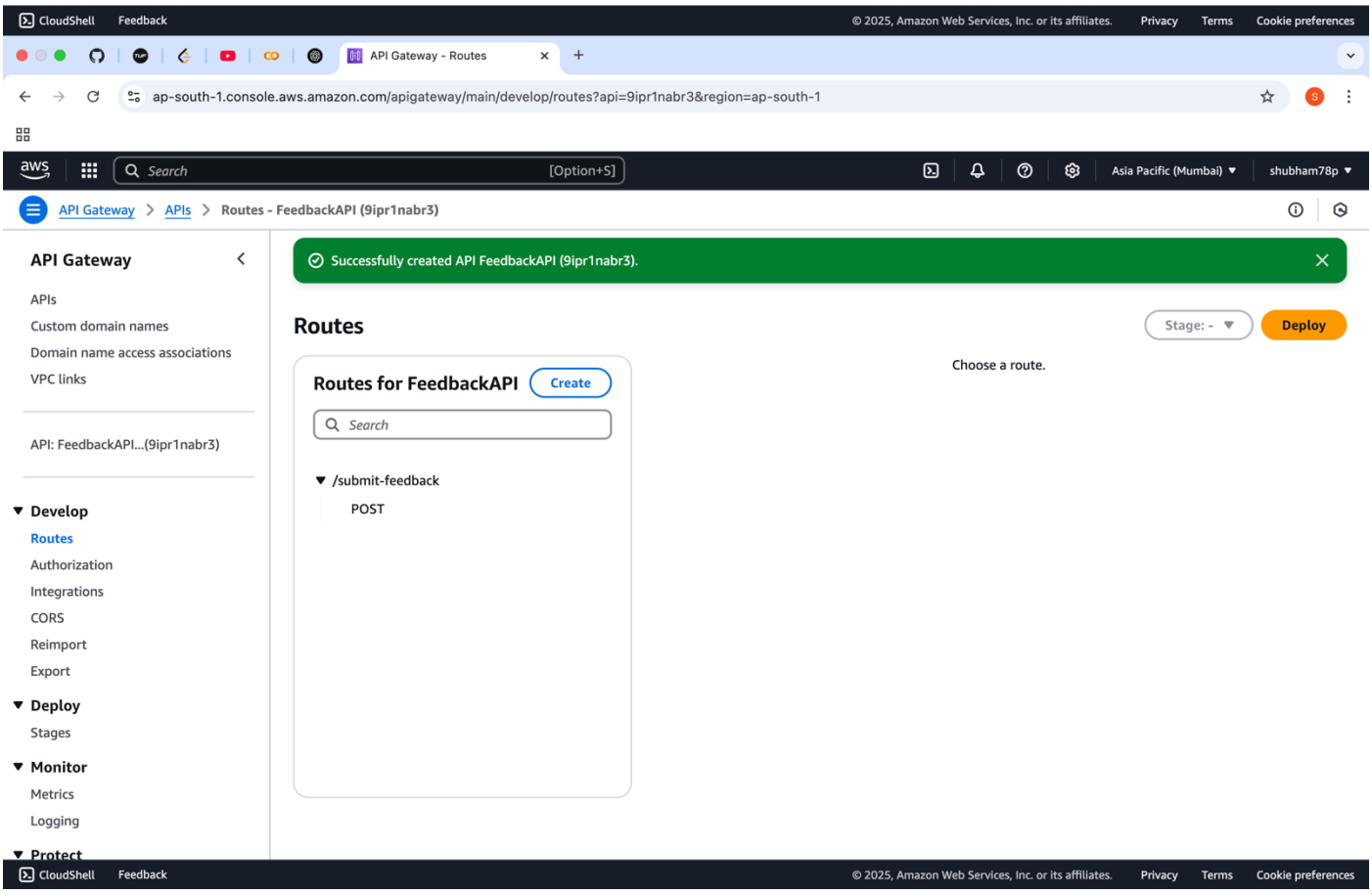
Serverless Feedback Collection System



CLOUD COMPUTING LAB

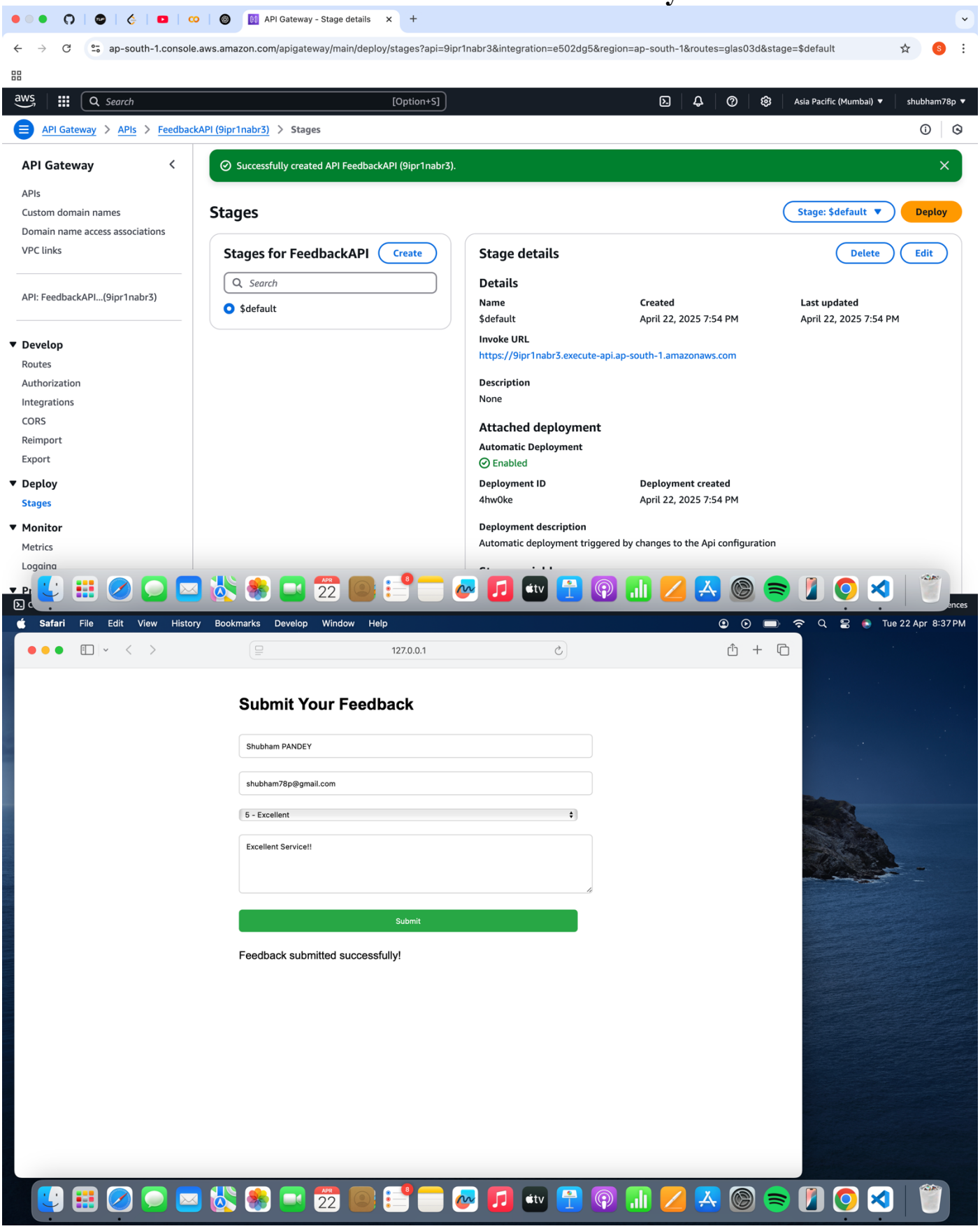
Serverless Feedback Collection System





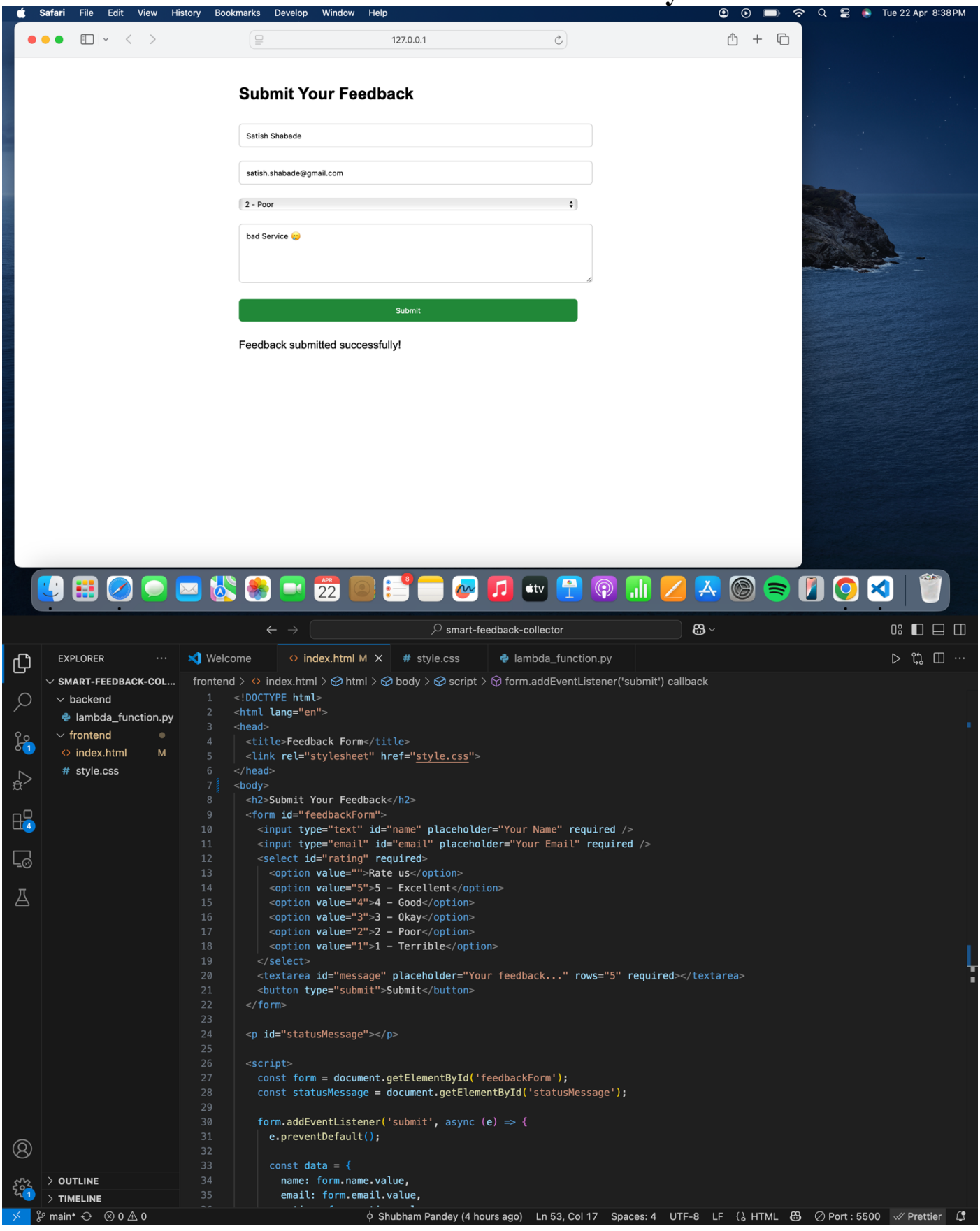
CLOUD COMPUTING LAB

Serverless Feedback Collection System



CLOUD COMPUTING LAB

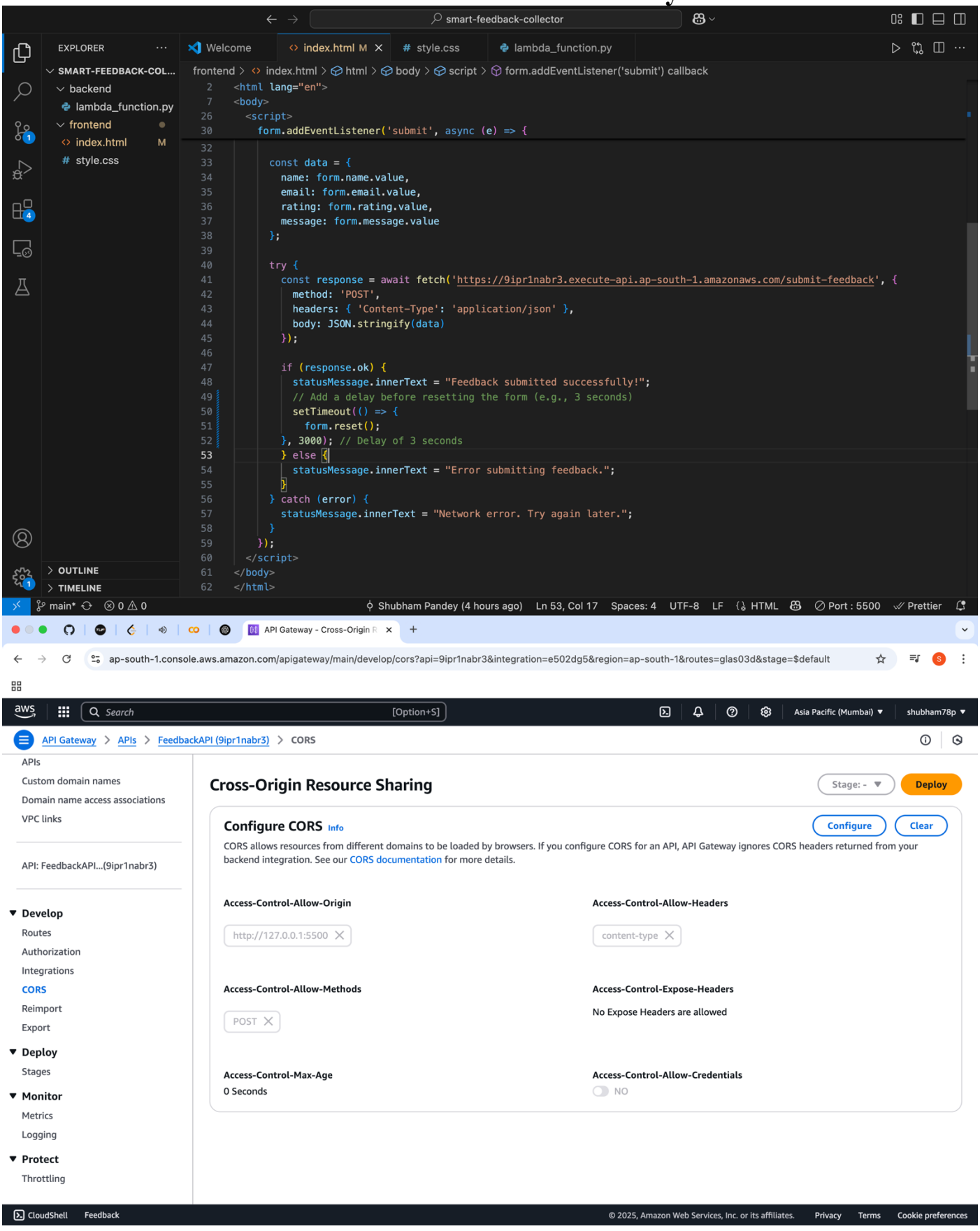
Serverless Feedback Collection System



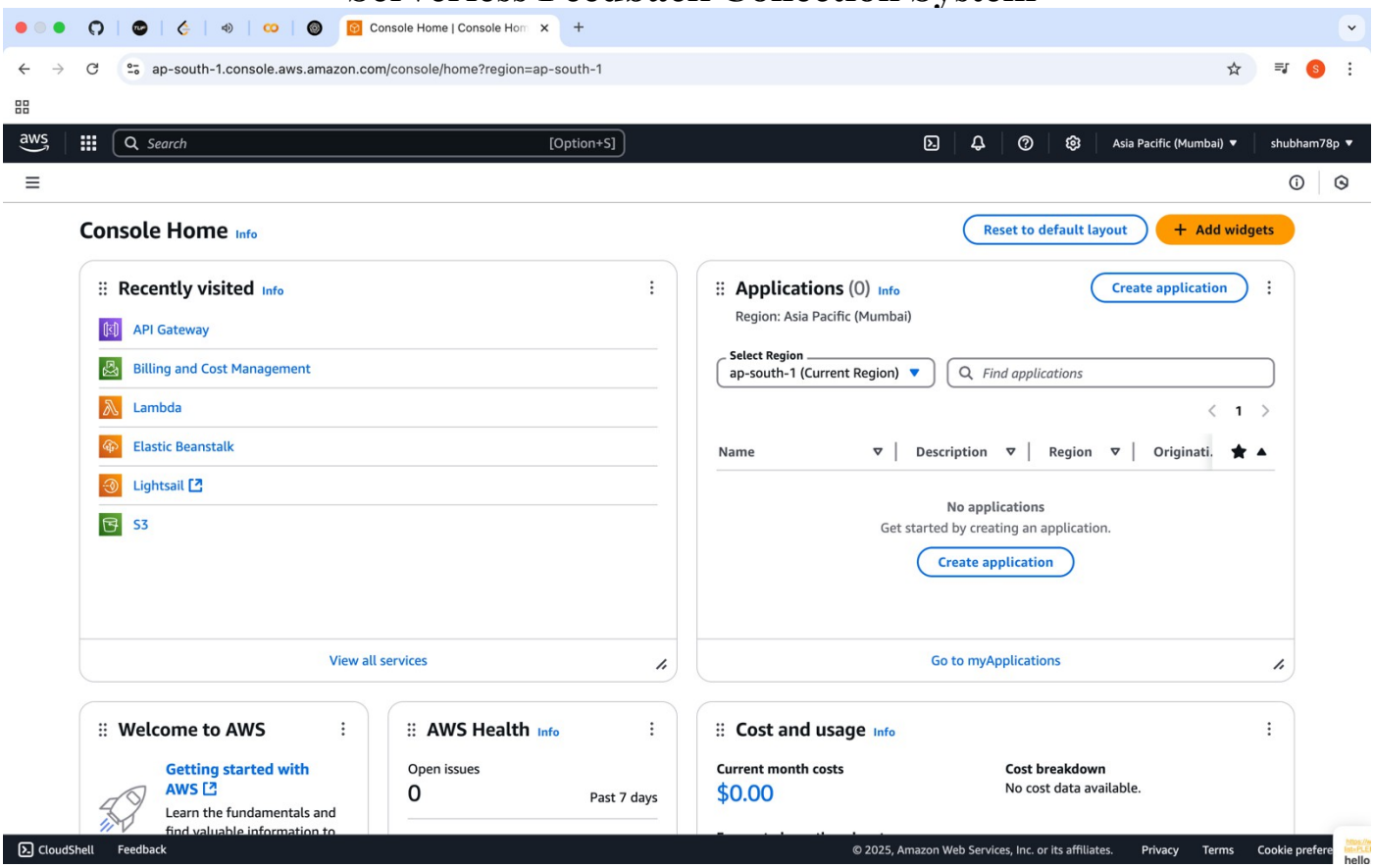

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Feedback Form</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <h2>Submit Your Feedback</h2>
  <form id="feedbackForm">
    <input type="text" id="name" placeholder="Your Name" required />
    <input type="email" id="email" placeholder="Your 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,
```

CLOUD COMPUTING LAB

Serverless Feedback Collection System



Serverless Feedback Collection System



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.

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.

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.