LAMDA FUNCTION (SOURCE CODE)

1. Contact.html ---→(Feed back Form)

2. lamda_function.py --→ (Boto 3,comprehend,dynamodb)

3.Success.html →(appsync and google charts)

4.lasting.html→(To display the Thank you message)

1.Contact.html

```
import json
import boto3
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Contact Form</title>
  <style>
    body {
      font-family: 'Arial', sans-serif;
            background-color: #f4f4f4;
            margin: 0;
            padding: 0;
            display: flex;
            flex-direction: column;
            align-items: center;
            height: 100vh;
          }

          form {
            background-color: #fff;
            padding: 20px;
            border-radius: 8px;
            box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
            transition: transform 0.3s ease-in-out;
          }

          h2 {
            text-align: center;
            color: #333;
            margin-bottom: 20px;
          }

          label {
            display: block;
            margin: 10px 0 5px;
            color: #555;
          }
```

```
38.
39.           input,
40.           textarea,
41.           select {
42.             width: 100%;
43.             padding: 10px;
44.             margin-bottom: 10px;
45.             box-sizing: border-box;
46.             border: 1px solid #ccc;
47.             border-radius: 4px;
48.           }
49.
50.           input[type="submit"], .chart-button {
51.             background-color: #4caf50;
52.             color: #fff;
53.             cursor: pointer;
54.             font-size: 16px;
55.             border: none;
56.             padding: 10px 20px;
57.             border-radius: 4px;
58.             text-align: center;
59.             display: block;
60.             margin: 10px auto;
61.           }
62.
63.           input[type="submit"]:hover, .chart-button:hover {
64.             background-color: #45a049;
65.           }
66.
67.           form:hover {
68.             transform: scale(1.05);
69.           }
70.
71.           .button-container {
72.             text-align: center;
73.             margin-top: 20px;
74.           }
75.         </style>
76.       </head>
77.       <body>
78.
79.         <h2>We want feedback about Healthapps</h2>
80.
81.         <form action="/dev" method="post">
82.           <h2>FeedBack </h2>
83.           <label for="Name">Name:</label>
84.           <input type="text" id="fname" name="fname" required>
85.
86.           <label for="appName">App Name:</label>
87.           <select id="appName" name="appName" required>
88.             <option value="" disabled selected>Select an app</option>
89.             <option value="Samsung Health">Samsung Health</option>
```

```
            <option value="LG Health">LG Health</option>
            <option value="Huawei Health">Huawei Health</option>
            <option value="Google Fit">Google Fit: Health and Activity Tracking</option>
            <option value="Mi Fit">Mi Fit</option>
            <option value="Teladoc Health">Teladoc Health</option>
            <option value="MDLIVE">MDLIVE</option>
            <option value="Talkspace">Talkspace</option>
            <option value="Zocdoc">Zocdoc</option>
            <option value="Mfine">Mfine</option>
            <option value="Tata 1mg">Tata 1mg</option>
            <option value="GOOD MED">GOOD MED</option>
          </select>

          <label for="email">Email ID</label>
          <input type="text" id="email" name="email" required>

          <label for="feedback">FeedBack</label>
          <textarea id="message" name="message" rows="4" cols="50" required></textarea>

          <input type="submit" value="Submit">
        </form>

        <button class="chart-button" onclick="window.location.href='https://qqainv8x5m.execute-api.ap-south-1.amazonaws.com/dev/?page=success'">
          Live Feedback Chart
        </button>

        <script>
          const form = document.querySelector('form');

          form.addEventListener('mouseover', () => {
            form.style.transform = 'scale(1.05)';
          });

          form.addEventListener('mouseout', () => {
            form.style.transform = 'scale(1)';
          });
        </script>

      </body>
      </html>
```

Lamda_function.py

```
import json
import boto3
import urllib.parse  # For decoding URL-encoded form data
comprehend = boto3.client('comprehend')

```

```
10.      import json
11.
12.      import json
13.
14.      def lambda_handler(event, context):
15.          try:
16.              mypage = page_router(event['httpMethod'],
 event['queryStringParameters'], event['body'])
17.              return mypage
18.          except Exception as e:
19.              return {
20.                  'statusCode': 500,
21.                  'body': json.dumps({'error': str(e)})
22.              }
23.
24.      def page_router(httpmethod, querystring, formbody):
25.          if httpmethod == 'GET':
26.              # Check if a query parameter 'page' exists to serve different pages
27.              if querystring and 'page' in querystring:
28.                  if querystring['page'] == 'success':  # Route to success.html when
 called via API Gateway inline URL
29.                      return serve_html('success.html')
30.                  else:
31.                      return {
32.                          'statusCode': 400,
33.                          'body': json.dumps({'error': 'Invalid page parameter'})
34.                      }
35.              else:
36.                  # Default GET request serves contactus.html
37.                  return serve_html('contactus.html')
38.
39.          elif httpmethod == 'POST':
40.              try:
41.                  clean_formbody = decode_and_parse_formdata(formbody)
42.                  insert_record(clean_formbody)
43.                  return serve_html('lasting.html')  # After POST request processing
44.              except Exception as e:
45.                  return {
46.                      'statusCode': 500,
47.                      'body': json.dumps({'error': str(e)})
48.                  }
49.
50.      def serve_html(filename):
51.          try:
52.              with open(filename, 'r') as htmlFile:
53.                  htmlContent = htmlFile.read()
54.              return {
55.                  'statusCode': 200,
56.                  'headers': {"Content-Type": "text/html"},
57.                  'body': htmlContent
58.              }
59.          except Exception as e:
```

```
            return {
                'statusCode': 500,
                'body': json.dumps({'error': str(e)})
            }

    def insert_record(formdata):
        """
        Inserts the parsed form data as a new item in the DynamoDB table.
        formdata: Dictionary containing the form data to insert.
        """
        dynamodb = boto3.resource('dynamodb')
        table = dynamodb.Table('pranaybuklist3')

        sentiment_response = comprehend.detect_sentiment(
            Text=formdata['message'],  # Replace with the actual key for review text
            LanguageCode='en'  # Change as needed for different languages
        )

        # Extract sentiment and add to the formdata
        formdata['sentiment'] = sentiment_response['Sentiment']
        formdata['appName'] = formdata.get('appName', 'Unknown')

        # Insert the formdata dictionary directly into the DynamoDB table
        response = table.put_item(Item=formdata)

        return response

    def decode_and_parse_formdata(formbody):
        """
        Decodes URL-encoded form data and converts it into a dictionary.
        This function also removes any non-ASCII characters from the values.
        """
        # First, decode the URL-encoded form data
        decoded_data = urllib.parse.unquote(formbody)
        decoded_data = decoded_data.replace('+',' ')

        # Parse the decoded form data into a dictionary
        formdata_pairs = decoded_data.split('&')  # Split key-value pairs
        formdata_dict = {}

        for pair in formdata_pairs:
            key, value = pair.split('=')
            # Optionally, clean the value to remove non-ASCII characters
            clean_value = ''.join([char for char in value if ord(char) < 128])
            formdata_dict[key] = clean_value

        return formdata_dict
```

****

success.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>Real-time Sentiment Chart</title>
    <script type="text/javascript"
 src="https://www.gstatic.com/charts/loader.js"></script>
    <script>
      google.charts.load('current', {'packages':['corechart']});
      google.charts.setOnLoadCallback(drawChart);

            var chart;
            var data;

            // Draw chart
            function drawChart() {
              // Initialize data
              data = google.visualization.arrayToDataTable([
                ['App Name', 'MIXED', 'POSITIVE', 'NEGATIVE', 'NEUTRAL'],
                ['Google Fit', 0, 0, 0, 0],
                ['Huawei Health', 0,0, 0, 0],
                ['Zocdoc',0,0,0,0],
                ['MDLIVE',0,0,0,0],
                ['Teladoc Health',0,0,0,0],
                ['Mi Fit',0,0,0,0],
                ['LG Health',0,0,0,0],
                ['Samsung Health',0,0,0,0],
                ['Talkspace',0,0,0,0],
                ['Mfine',0,0,0,0],
                ['GOOD MED',0,0,0,0],

                // Add more apps here
              ]);

              var options = {
                title: 'Sentiment Analysis by App',
                isStacked: true,
                hAxis: {title: 'App Name'},
                vAxis: {title: 'Sentiment Count'}
              };

              chart = new
 google.visualization.ColumnChart(document.getElementById('chart_div'));
              chart.draw(data, options);
            }

            // Function to update chart with fresh data from AppSync
            async function fetchSentiments() {
              const apiUrl = 'https://xzeajhrezvecvohp3h52ing6i4.appsync-api.ap-south-
 1.amazonaws.com/graphql'; // Update this with your AppSync API URL
              const query = `
                query GetSentiments {
                  getSentiments {
```

```
                    appName
                    sentiment
                  }
                }
              `;

              const response = await fetch(apiUrl, {
                method: 'POST',
                headers: {
                  'Content-Type': 'application/json',
                  'x-api-key': 'da2-6s2jd7clbbbpzgdcjg33pcw7qm' // Replace with your AppSync API key
                },
                body: JSON.stringify({ query })
              });

              const result = await response.json();
              const sentiments = result.data.getSentiments;

              // Reset chart data
              for (let i = 0; i < data.getNumberOfRows(); i++) {
                data.setValue(i, 1, 0); // Reset MIXED
                data.setValue(i, 2, 0); // Reset POSITIVE
                data.setValue(i, 3, 0); // Reset NEGATIVE
                data.setValue(i, 4, 0); // Reset NEUTRAL
              }

              // Update chart with new data
              sentiments.forEach(sentiment => {
                for (let i = 0; i < data.getNumberOfRows(); i++) {
                  if (data.getValue(i, 0) === sentiment.appName) {
                    if (sentiment.sentiment === 'MIXED') data.setValue(i, 1, data.getValue(i, 1) + 1);
                    if (sentiment.sentiment === 'POSITIVE') data.setValue(i, 2, data.getValue(i, 2) + 1);
                    if (sentiment.sentiment === 'NEGATIVE') data.setValue(i, 3, data.getValue(i, 3) + 1);
                    if (sentiment.sentiment === 'NEUTRAL') data.setValue(i, 4, data.getValue(i, 4) + 1);
                  }
                }
              });

              chart.draw(data); // Redraw chart with updated data
            }

            // Fetch data every 10 seconds
            setInterval(fetchSentiments, 10000);
          </script>
        </head>
        <body>
          <div id="chart_div" style="width: 1000px; height: 800px;"></div>
```

```
        </body>
      </html>
```

4.Lasting.html

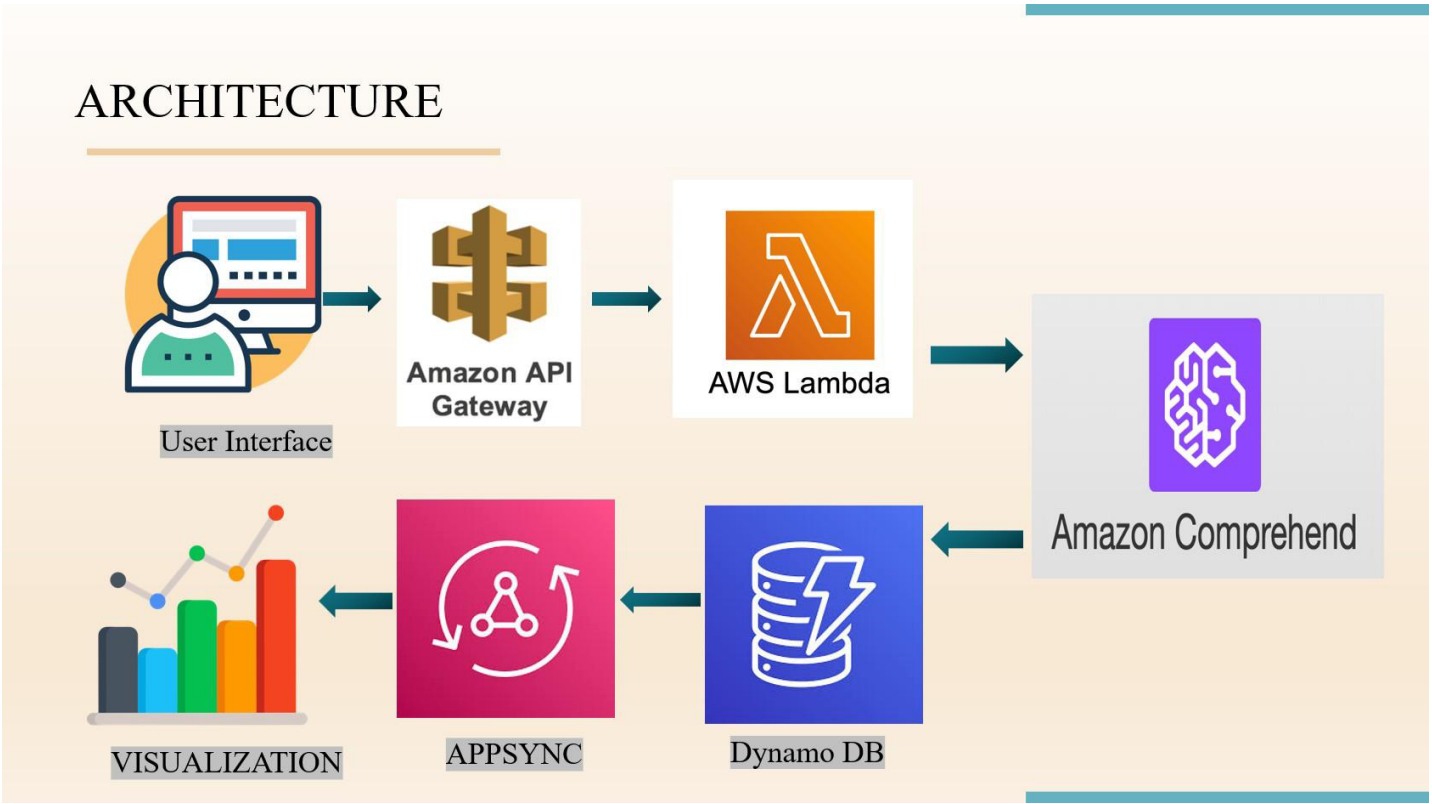
```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Thank You</title>
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet">
  <style>
    body {
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;
            background-color: #f8f9fa;
          }
          .thank-you-container {
            text-align: center;
            background: white;
            padding: 30px;
            border-radius: 10px;
            box-shadow: 0 0 15px rgba(0, 0, 0, 0.1);
            transition: transform 0.3s ease-in-out;
          }
          .thank-you-container:hover {
            transform: scale(1.05);
          }
          .btn-home {
            margin-top: 20px;
          }
        </style>
      </head>
      <body>
        <div class="thank-you-container">
          <h2 class="text-success">Thank You!</h2>
          <p>Your valuable feedback has been received.</p>
          <p>We appreciate your time and effort.</p>
          <a href="https://qqainv8x5m.execute-api.ap-south-1.amazonaws.com/dev/" class="btn btn-primary btn-home">Go to Home</a>
        </div>
      </body>
      </html>
```

PROJECT (EXPLANATION)

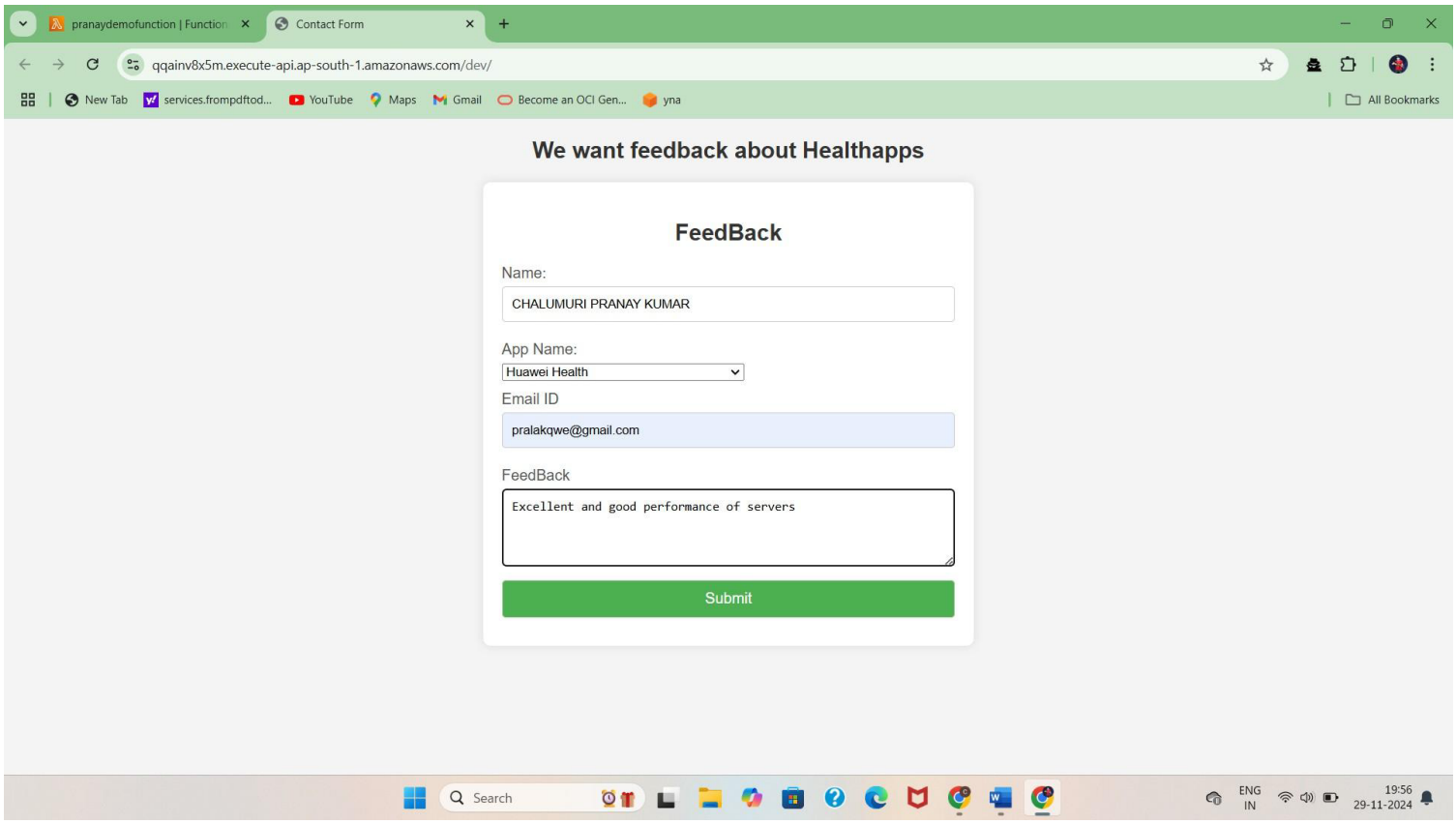
TITLE: **AWS-Serverless Feedback System: Automating Analysis and Visualization**

ABSTRACT:
Serverless Application for Feedback Storage and Sentiment Analysis Using AWS is a comprehensive project designed to automate the collection, analysis, and visualization of user feedback using cutting-edge AWS services. The application leverages AWS Lambda for serverless computing, ensuring that the system automatically scales based on demand without requiring manual intervention. AWS API Gateway enables secure and efficient handling of RESTful API methods, allowing users to submit feedback and retrieve processed data in realtime. The feedback is analysed using AWS Comprehend, which performs sentiment analysis, categorizing the feedback as positive, negative, or neutral based on natural language processing (NLP) techniques. The results are then stored in AWS DynamoDB, a NoSQL database that provides high availability and scalability, ensuring seamless handling of large data volumes.One of the key features of this system is real-time data visualization, achieved through Google Charts, which displays user sentiment trends in the form of stacked bar charts. This allows administrators to easily track shifts in customer feedback and make informed decisions based on real-time insights. The entire architecture is serverless, significantly reducing operational costs by utilizing AWS's pay-as-you-go model. AWS Lambda ensures that compute resources are only used when necessary, while AWS DynamoDB provides fast and reliable storage without the need for manual scaling.The use of AWS Comprehend's sentiment analysis adds an intelligent layer of data processing, turning raw feedback into actionable insights. Additionally, the system is designed to be highly secure, reliable, and efficient, providing businesses with a scalable solution to collect, analyze, and act on user feedback in real-time. A key feature is the integration of AWS AppSync for real-time updates. Through GraphQL subscriptions, AppSync pushes live sentiment data to connected clients, updating charts dynamically as feedback is received. This eliminates the need for constant polling and ensures immediate updates to Google Charts, which visualizes sentiment trends in stacked bar charts.

Architecture



1. USER INTERFACE (Feedback form)



2. API GATE WAYS :

API Gateway: pranaytableapi
arn:aws:execute-api:ap-south-1:337909760489:qqainv8x5m/*/GET/
API endpoint: https://qqainv8x5m.execute-api.ap-south-1.amazonaws.com/dev/

▼ **Details**

API type: **REST**
Authorization: **NONE**
isComplexStatement: **No**
Method: **GET**
Resource path: **/**
Service principal: **apigateway.amazonaws.com**
Stage: **dev**
Statement ID: **fd31a470-e5f9-54dd-b5d2-924eab0f5f9b**

GET → to display the website

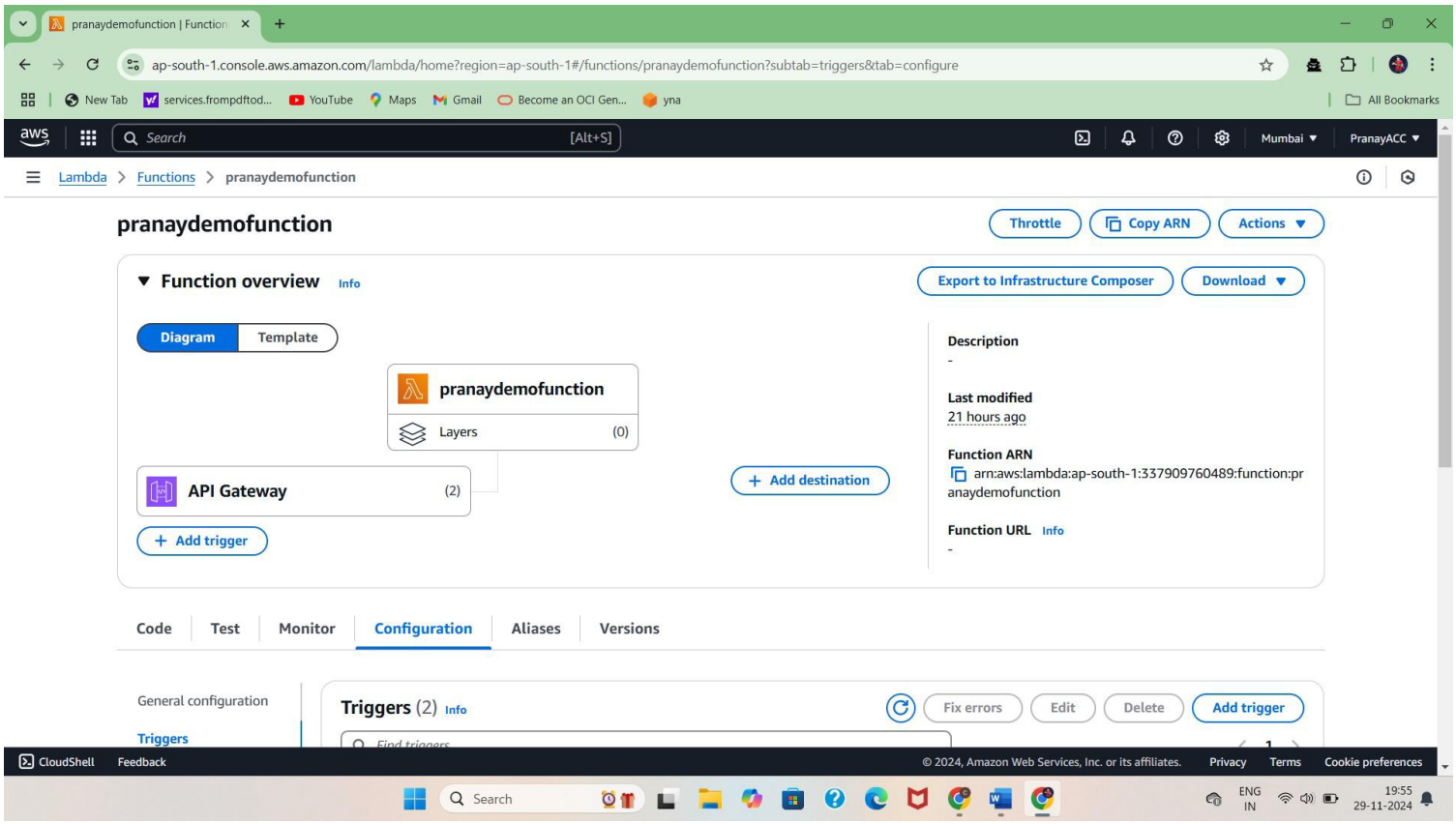
API Gateway: pranaytableapi
arn:aws:execute-api:ap-south-1:337909760489:qqainv8x5m/*/POST/
API endpoint: https://qqainv8x5m.execute-api.ap-south-1.amazonaws.com/dev/

▼ **Details**

API type: **REST**
Authorization: **NONE**
isComplexStatement: **No**
Method: **POST**
Resource path: **/**
Service principal: **apigateway.amazonaws.com**
Stage: **dev**
Statement ID: **ecf23d70-9c84-5422-ae87-dcc3bb30ec2d**

POST → post the data into dynamo table

3. LAMDA FUNCTION

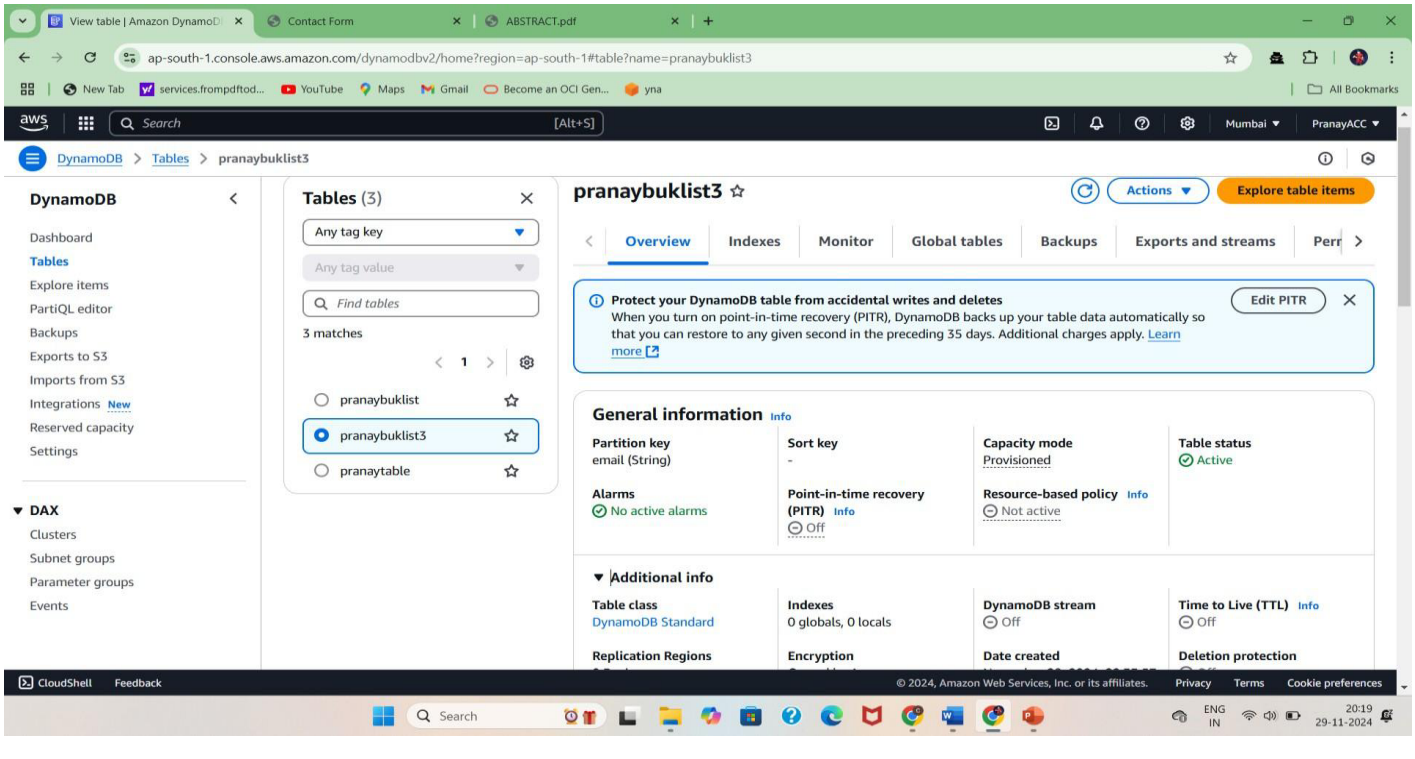


4. COMPREHEND

→I used a function directly inside the lamda_function.py (not manually used) and check the code I mentioned .

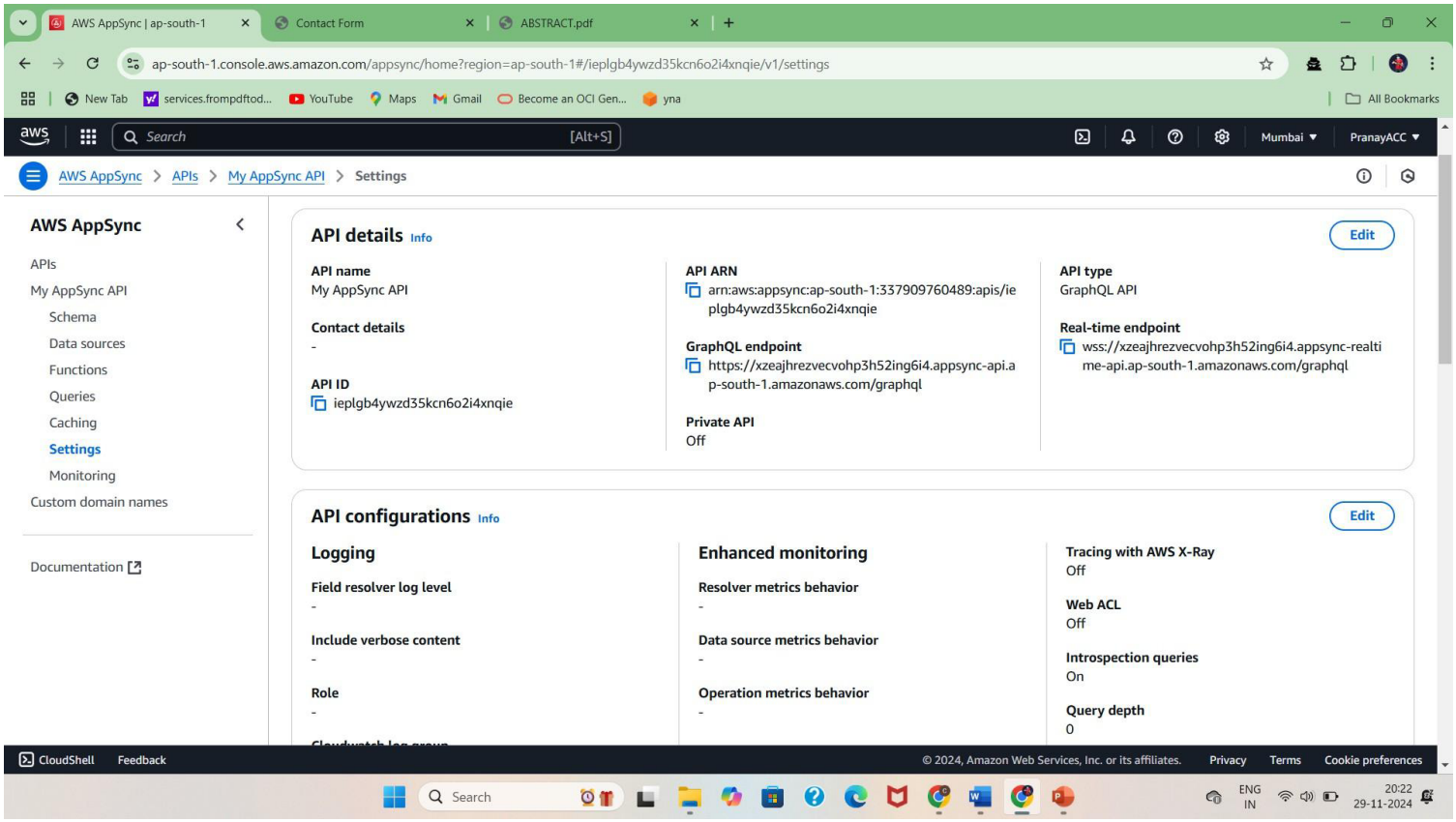
5. DYNAMO DB

Table name: pranaybuklist3

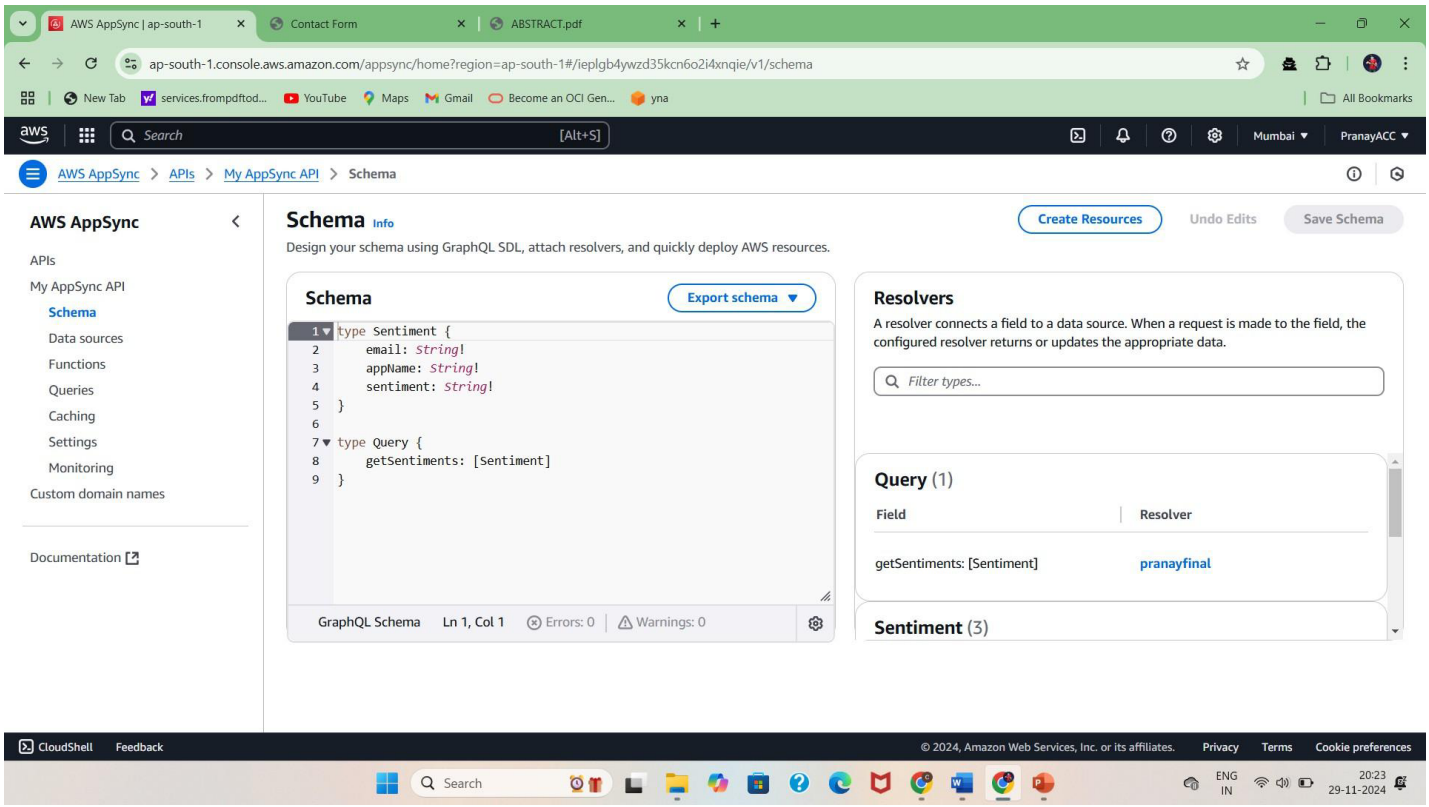


6. APP SYNC

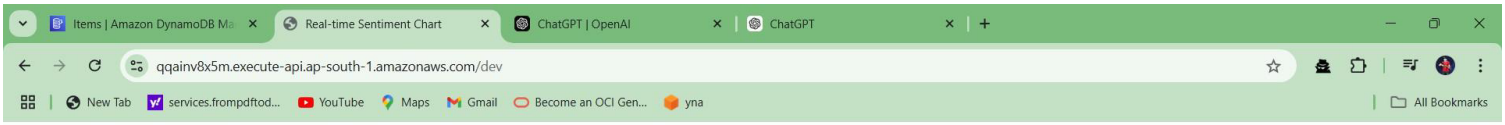
1. APP SYNC API DETAILS

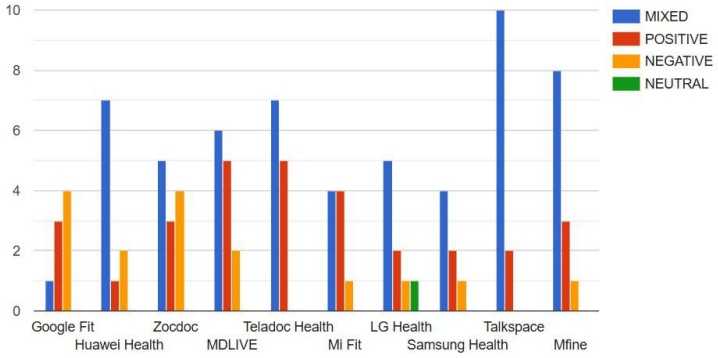


2. SCHEMA



7 . REAL TIME UPDATING VISULAIZATION





IAM(IDENTITY AND ACCESS MANEGEMENT)

→ROLES AND PERMISSION I USED

Role name: admin-lambda-role

Permissions: