

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

```
3. import json
4. import boto3
1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4.   <meta charset="UTF-8">
5.   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6.   <title>Contact Form</title>
7.   <style>
8.     body {
9.       font-family: 'Arial', sans-serif;
10.      background-color: #f4f4f4;
11.      margin: 0;
12.      padding: 0;
13.      display: flex;
14.      flex-direction: column;
15.      align-items: center;
16.      height: 100vh;
17.    }
18.
19.    form {
20.      background-color: #fff;
21.      padding: 20px;
22.      border-radius: 8px;
23.      box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
24.      transition: transform 0.3s ease-in-out;
25.    }
26.
27.    h2 {
28.      text-align: center;
29.      color: #333;
30.      margin-bottom: 20px;
31.    }
32.
33.    label {
34.      display: block;
35.      margin: 10px 0 5px;
36.      color: #555;
37.    }
```

```
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>
```

```

90.          <option value="LG Health">LG Health</option>
91.          <option value="Huawei Health">Huawei Health</option>
92.          <option value="Google Fit">Google Fit: Health and Activity
  Tracking</option>
93.          <option value="Mi Fit">Mi Fit</option>
94.          <option value="Teladoc Health">Teladoc Health</option>
95.          <option value="MDLIVE">MDLIVE</option>
96.          <option value="Talkspace">Talkspace</option>
97.          <option value="Zocdoc">Zocdoc</option>
98.          <option value="Mfine">Mfine</option>
99.          <option value="Tata 1mg">Tata 1mg</option>
100.         <option value="GOOD MED">GOOD MED</option>
101.        </select>
102.
103.        <label for="email">Email ID</label>
104.        <input type="text" id="email" name="email" required>
105.
106.        <label for="feedback">FeedBack</label>
107.        <textarea id="message" name="message" rows="4" cols="50"
  required></textarea>
108.
109.        <input type="submit" value="Submit">
110.      </form>
111.
112.      <button class="chart-button"
  onclick="window.location.href='https://qqainv8x5m.execute-api.ap-south-
  1.amazonaws.com/dev/?page=success'">
113.        Live Feedback Chart
114.      </button>
115.
116.      <script>
117.        const form = document.querySelector('form');
118.
119.        form.addEventListener('mouseover', () => {
120.          form.style.transform = 'scale(1.05)';
121.        });
122.
123.        form.addEventListener('mouseout', () => {
124.          form.style.transform = 'scale(1)';
125.        });
126.      </script>
127.
128.    </body>
129.  </html>

```

Lamda_function.py

```

5. import json
6. import boto3
7. import urllib.parse # For decoding URL-encoded form data
8. comprehend = boto3.client('comprehend')
9.

```

```
10.     import json
11.
12.     import json
13.
14.     def lambda_handler(event, context):
15.         try:
16.             mypage = page_router(event['httpMethod'],
17. 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:
```

```

60.         return {
61.             'statusCode': 500,
62.             'body': json.dumps({'error': str(e)})
63.         }
64.

65.     def insert_record(formdata):
66.         """
67.             Inserts the parsed form data as a new item in the DynamoDB table.
68.             formdata: Dictionary containing the form data to insert.
69.         """
70.         dynamodb = boto3.resource('dynamodb')
71.         table = dynamodb.Table('pranaybuklist3')
72.
73.         sentiment_response = comprehend.detect_sentiment(
74.             Text=formdata['message'], # Replace with the actual key for review text
75.             LanguageCode='en' # Change as needed for different languages
76.         )
77.
78.         # Extract sentiment and add to the formdata
79.         formdata['sentiment'] = sentiment_response['Sentiment']
80.         formdata['appName'] = formdata.get('appName', 'Unknown')
81.
82.         # Insert the formdata dictionary directly into the DynamoDB table
83.         response = table.put_item(Item=formdata)
84.
85.     return response
86.
87. def decode_and_parse_formdata(formbody):
88.     """
89.         Decodes URL-encoded form data and converts it into a dictionary.
90.         This function also removes any non-ASCII characters from the values.
91.     """
92.     # First, decode the URL-encoded form data
93.     decoded_data = urllib.parse.unquote(formbody)
94.     decoded_data = decoded_data.replace('+', ' ')
95.
96.     # Parse the decoded form data into a dictionary
97.     formdata_pairs = decoded_data.split('&') # Split key-value pairs
98.     formdata_dict = {}
99.
100.    for pair in formdata_pairs:
101.        key, value = pair.split('=')
102.        # Optionally, clean the value to remove non-ASCII characters
103.        clean_value = ''.join([char for char in value if ord(char) < 128])
104.        formdata_dict[key] = clean_value
105.
106.    return formdata_dict

```

success.html

```

1. <!DOCTYPE html>
2. <html>
3.   <head>
4.     <title>Real-time Sentiment Chart</title>
5.     <script type="text/javascript"
6.       src="https://www.gstatic.com/charts/loader.js"></script>
7.     <script>
8.       google.charts.load('current', {'packages':['corechart']});
9.       google.charts.setOnLoadCallback(drawChart);
10.
11.       var chart;
12.       var data;
13.
14.       // Draw chart
15.       function drawChart() {
16.         // Initialize data
17.         data = google.visualization.arrayToDataTable([
18.           ['App Name', 'MIXED', 'POSITIVE', 'NEGATIVE', 'NEUTRAL'],
19.           ['Google Fit', 0, 0, 0, 0],
20.           ['Huawei Health', 0, 0, 0, 0],
21.           ['Zocdoc', 0, 0, 0, 0],
22.           ['MDLIVE', 0, 0, 0, 0],
23.           ['Teladoc Health', 0, 0, 0, 0],
24.           ['Mi Fit', 0, 0, 0, 0],
25.           ['LG Health', 0, 0, 0, 0],
26.           ['Samsung Health', 0, 0, 0, 0],
27.           ['Talkspace', 0, 0, 0, 0],
28.           ['Mfine', 0, 0, 0, 0],
29.           ['GOOD MED', 0, 0, 0, 0],
30.
31.           // Add more apps here
32.         ]);
33.
34.         var options = {
35.           title: 'Sentiment Analysis by App',
36.           isStacked: true,
37.           hAxis: {title: 'App Name'},
38.           vAxis: {title: 'Sentiment Count'}
39.         };
40.
41.         chart = new
42.           google.visualization.ColumnChart(document.getElementById('chart_div'));
43.           chart.draw(data, options);
44.
45.           // Function to update chart with fresh data from AppSync
46.           async function fetchSentiments() {
47.             const apiUrl = 'https://xzeajhrezvecvohp3h52ing6i4.appsync-api.ap-south-
48.               1.amazonaws.com/graphql'; // Update this with your AppSync API URL
49.               const query = `
50.                 query GetSentiments {
51.                   getSentiments {

```

```

50.          appName
51.          sentiment
52.      }
53.  }
54. `;
55.
56.     const response = await fetch(apiUrl, {
57.       method: 'POST',
58.       headers: {
59.         'Content-Type': 'application/json',
60.         'x-api-key': 'da2-6s2jd7clbbbbzgdcjg33pcw7qm' // Replace with your
AppSync API key
61.       },
62.       body: JSON.stringify({ query })
63.     });
64.
65.     const result = await response.json();
66.     const sentiments = result.data.getSentiments;
67.
68.     // Reset chart data
69.     for (let i = 0; i < data.getNumberOfRows(); i++) {
70.       data.setValue(i, 1, 0); // Reset MIXED
71.       data.setValue(i, 2, 0); // Reset POSITIVE
72.       data.setValue(i, 3, 0); // Reset NEGATIVE
73.       data.setValue(i, 4, 0); // Reset NEUTRAL
74.     }
75.
76.     // Update chart with new data
77.     sentiments.forEach(sentiment => {
78.       for (let i = 0; i < data.getNumberOfRows(); i++) {
79.         if (data.getValue(i, 0) === sentiment.appName) {
80.           if (sentiment.sentiment === 'MIXED') data.setValue(i, 1,
data.getValue(i, 1) + 1);
81.           if (sentiment.sentiment === 'POSITIVE') data.setValue(i, 2,
data.getValue(i, 2) + 1);
82.           if (sentiment.sentiment === 'NEGATIVE') data.setValue(i, 3,
data.getValue(i, 3) + 1);
83.           if (sentiment.sentiment === 'NEUTRAL') data.setValue(i, 4,
data.getValue(i, 4) + 1);
84.         }
85.       }
86.     });
87.
88.     chart.draw(data); // Redraw chart with updated data
89.   }
90.
91.   // Fetch data every 10 seconds
92.   setInterval(fetchSentiments, 10000);
93. 
```

```
97.      </body>
98.    </html>
```

4.Lastng.html

```
1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4.   <meta charset="UTF-8">
5.   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6.   <title>Thank You</title>
7.   <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet">
8.   <style>
9.     body {
10.       display: flex;
11.       justify-content: center;
12.       align-items: center;
13.       height: 100vh;
14.       background-color: #f8f9fa;
15.     }
16.     .thank-you-container {
17.       text-align: center;
18.       background: white;
19.       padding: 30px;
20.       border-radius: 10px;
21.       box-shadow: 0 0 15px rgba(0, 0, 0, 0.1);
22.       transition: transform 0.3s ease-in-out;
23.     }
24.     .thank-you-container:hover {
25.       transform: scale(1.05);
26.     }
27.     .btn-home {
28.       margin-top: 20px;
29.     }
30.   </style>
31. </head>
32. <body>
33.   <div class="thank-you-container">
34.     <h2 class="text-success">Thank You!</h2>
35.     <p>Your valuable feedback has been received.</p>
36.     <p>We appreciate your time and effort.</p>
37.     <a href="https://qqainv8x5m.execute-api.ap-south-1.amazonaws.com/dev/" class="btn btn-primary btn-home">Go to Home</a>
38.   </div>
39. </body>
40. </html>
```

PROJECT (EXPLANATION)

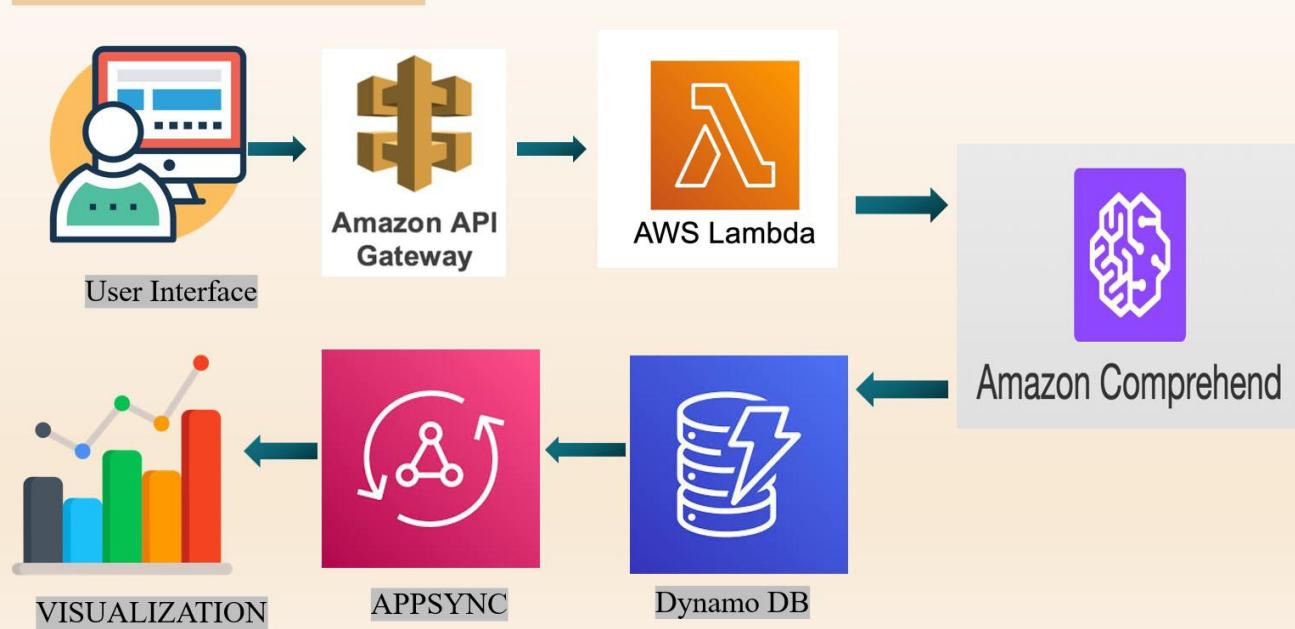
TITLE: AWS-Serverless Feedback System: Automating Analysis and Visualization

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

ARCHITECTURE



1. USER INTERFACE (Feedback form)

We want feedback about Healthapps

FeedBack

Name: CHALUMURI PRANAY KUMAR

App Name: Huawei Health

Email ID: pralakqwe@gmail.com

FeedBack
Excellent and good performance of servers

Submit

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

The screenshot shows the AWS Lambda console interface. At the top, the URL is ap-south-1.console.aws.amazon.com/lambda/home?region=ap-south-1#/functions/pranaydemofunction?subtab=triggers&tab=configure. The navigation bar includes links for New Tab, services.frompdf... (disabled), YouTube, Maps, Gmail, Become an OCI Gen..., and yna. The AWS logo and search bar are also present.

The main content area displays the function **pranaydemofunction**. The **Function overview** tab is active, showing a diagram where the function is triggered by an **API Gateway**. There are two triggers listed under the API Gateway. The **Configuration** tab is also visible, showing the **Triggers** section with two triggers listed. The right sidebar contains sections for **Description**, **Last modified** (21 hours ago), **Function ARN** (arnaws:lambda:ap-south-1:337909760489:function:pranaydemofunction), and **Function URL**.

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

The screenshot shows the AWS DynamoDB console interface. At the top, the URL is ap-south-1.console.aws.amazon.com/dynamodbv2/home?region=ap-south-1#table?name=pranaybuklist3. The navigation bar includes links for New Tab, services.frompdf... (disabled), YouTube, Maps, Gmail, Become an OCI Gen..., and yna. The AWS logo and search bar are also present.

The main content area displays the table **pranaybuklist3**. The **Overview** tab is active, showing general information about the table. The **General information** section includes fields for Partition key (email, String), Sort key (-), Capacity mode (Provisioned), and Table status (Active). It also shows Point-in-time recovery (PITR) status (Off) and Resource-based policy (Not active). The **Additional info** section includes fields for Table class (DynamoDB Standard), Indexes (0 globals, 0 locals), DynamoDB stream (Off), Time to Live (TTL) (Off), Replication Regions, Encryption, Date created, and Deletion protection.

6. APP SYNC

1. APP SYNC API DETAILS

The screenshot shows the AWS AppSync API Details page. On the left, a sidebar lists options like APIs, My AppSync API, Schema, Data sources, Functions, Queries, Caching, Settings (which is selected), Monitoring, and Custom domain names. The main content area has two tabs: 'API details' and 'API configurations'. Under 'API details', fields include API name (My AppSync API), Contact details (empty), API ID (ieplgb4ywzd35kcn6o2i4xnqie), API ARN (arn:aws:appsync:ap-south-1:337909760489:apis/ieplgb4ywzd35kcn6o2i4xnqie), GraphQL endpoint (https://xzeajhrezvecvohp3h52ing6i4.appspot.com/graphql), and API type (GraphQL API). Under 'API configurations', sections include Logging (Field resolver log level, Include verbose content, Role), Enhanced monitoring (Resolver metrics behavior, Data source metrics behavior, Operation metrics behavior), and Tracing with AWS X-Ray (Off). Other settings like Web ACL, Introspection queries, and Query depth are also listed.

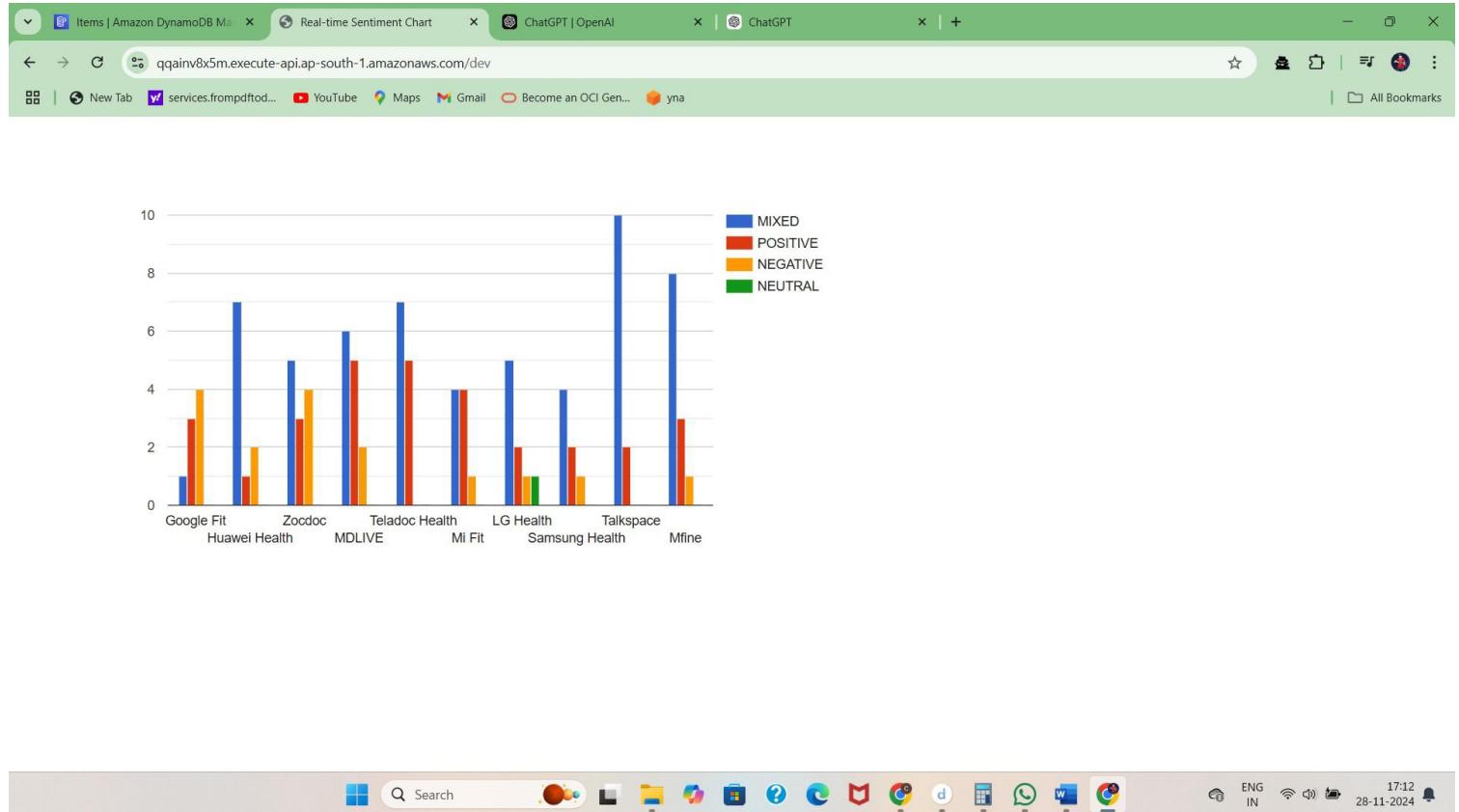
2. SCHEMA

The screenshot shows the AWS AppSync Schema editor. The left sidebar includes APIs, My AppSync API (Schema is selected), Data sources, Functions, Queries, Caching, Settings, Monitoring, and Custom domain names. The main area has tabs for 'Schema' and 'Resolvers'. The 'Schema' tab displays the following GraphQL code:

```
1 type Sentiment {
2   email: String!
3   appName: String!
4   sentiment: String!
5 }
6
7 type Query {
8   getSentiments: [Sentiment]
9 }
```

The 'Resolvers' tab shows a resolver for the 'getSentiments' field, mapping it to the 'pranayfinal' resolver. The bottom status bar indicates 'GraphQL Schema Ln 1, Col 1 Errors: 0 Warnings: 0'.

7. REAL TIME UPDATING VISUALIZATION



IAM(IDENTITY AND ACCESS MANAGEMENT)

→ ROLES AND PERMISSION I USED

Role name: **admin-lambda-role**

Permissions:

You can attach up to 10 managed policies.			
Filter by Type			
	Policy name	Type	Attached entities
<input type="checkbox"/>	AmazonDynamoDBFullAccess	AWS managed	2
<input type="checkbox"/>	AmazonS3ReadOnlyAccess	AWS managed	1
<input type="checkbox"/>	appsync-ds-ddb-yHvFBdjO0Fr4-pranaytable	Customer managed	2
<input type="checkbox"/>	AWSAppSyncAdministrator	AWS managed	1
<input type="checkbox"/>	AWSAppSyncInvokeFullAccess	AWS managed	1
<input type="checkbox"/>	AWSAppSyncSchemaAuthor	AWS managed	1
<input type="checkbox"/>	awscomprehendaccessfulltemp	Customer managed	2
<input type="checkbox"/>	AWSLambdaBasicExecutionRole	AWS managed	1
<input type="checkbox"/>	AWSLambdaInvocation-DynamoDB	AWS managed	1
<input type="checkbox"/>	ComprehendDetectSentimentPolicy	Customer inline	0