SERVERLESS WEB APPLICATION ON AWS

Code Documentation

Submitted by-

Richard David (M24CSE019)

Vishwanath Singh (M24CSE030)

Sachin Singh (M24CSE033)

Overview

This project is a serverless web application that utilizes AWS services to create a scalable, cost-effective, and user-friendly university website. The website's front-end contains HTML, CSS, and JavaScript files to render the user interface and manage interactions. The backend is implemented using AWS Lambda and DynamoDB, supporting the collection and storage of user messages. The application leverages S3 for static file hosting, CloudFront for content distribution, and Route 53 for DNS management. Below is the detailed documentation of each file and component within this project.

Frontend Files

- **index.html:** This should be the home page, or landing page of the site, providing an overview of the university, for example in terms of welcome messages, highlights, and the key sections that would give visitors a quick glance at what the website offers. It is a navigation point to other sections that include courses, contact, and more.
- **about.html:** It may include the university's history, mission, and vision, with any values. It can outline the history of the university, such as its notable accomplishments plus maybe the administration and/or faculty members, so the users would know what the university stands for.
- **blog.html:** This blog page serves to post any news updates, articles, or announcements about the university. It might include events, academic achievement records, new courses, or relevant articles authored by either faculty or guest writers with an aim of keeping the students and the visitors engaged with fresh information.
- **contact.html:** It is basically a contact page that serves for communication between a university and its visitors. It may contain a contact form where a user could submit questions and also maybe contain phone numbers, e-mail addresses, and a location map for easy direct contact with the university.
- **course.html:** an article describing various programs and courses the university has to offer. Course catalogs could also be presented alphabetically or by category, with short descriptions, prerequisites, and even information about faculty to better assist students in searching for and choosing courses of interest or to fulfill other academic requirements.

- **style.css:** All the styles that will be used in order to give a solid structure, coloring, spacing, and layout will be put into this file in the form of CSS (Cascading Style Sheets) so that all web pages will look homogeneous.
- **script.js:** This is JavaScript files designed with the purpose of implementing interesting and dynamic functionality at a website level. It can be for form validations, animation, or other interactive elements that make the website appear more engaging and responsive.
- **Images (folder):** This folder contains all the image files of the site, like logos and banners or pictures for different pages of a website. Managing here also helps to maintain well the whole project in an organized way, provide easy access to references, and easy update any visual elements of the website.

Backend File

lambda_function.py:

A Python script for the AWS Lambda function that handles incoming requests from the contact form on contact.html.

1. **Libraries Used**:
    - json: Used to parse the incoming request data and create responses in JSON format.
    - boto3: The AWS SDK for Python, used to interact with DynamoDB.
    - datetime: Provides the current timestamp for uniquely identifying each message.
2. **Functionality:**
    - Initializes a DynamoDB resource and connects to the studentMessage table to store user-submitted messages.
    - The main function, lambda_handler, handles incoming HTTP requests and performs the following steps:

        ➢ **Parse Request Body**: Extracts user information (name, email, subject, and message) from the request body.

        ➢ **Validate Data**: Checks for required fields (name, email, subject, message). If any field is missing, it returns a 400 status code with an error message.

- **Create Item**: Constructs a dictionary (item) containing the user data and a unique identifier (id) generated using the current timestamp.
- **Store in DynamoDB**: Saves the item to the DynamoDB table (studentMessage).
- **Return Success Response**: If successful, returns a 200 status code with a success message.
- **Error Handling**: Catches any exceptions during execution, logs the error, and returns a 500 status code with an error message.

3. **Usage:**

- This Lambda function acts as an API endpoint to handle contact form submissions from the frontend.
- Through the integration with DynamoDB, it ensures that user messages are stored in a scalable and serverless manner, minimizing the need for managing infrastructure.

4. **Code:**

```
import json
import boto3
from datetime import datetime

# Initialize DynamoDB client
dynamodb = boto3.resource('dynamodb')
table_name = 'studentMessage'
table = dynamodb.Table(table_name)

def lambda_handler(event, context):
    # Remove the explicit CORS headers in code and rely on API Gateway or
Lambda URL configuration
    headers = {
        "Access-Control-Allow-Methods": "POST, OPTIONS",
        "Access-Control-Allow-Headers": "Content-Type"
    }

    try:
        # Parse the incoming request body
        body = json.loads(event['body'])
        name = body.get('name')
        email = body.get('email')
        subject = body.get('subject')
        message = body.get('message')

        # Check for required fields
        if not name or not email or not subject or not message:
            return {
                'statusCode': 400,
                'headers': headers,
                'body': json.dumps({'error': 'Missing required fields'})
            }

        # Prepare the item to store in DynamoDB
        item = {
            'id': datetime.utcnow().isoformat(),
            'name': name,
            'email': email,
            'subject': subject,
            'message': message
        }

        # Store the item in DynamoDB
        table.put_item(Item=item)

        # Return success response
        return {
            'statusCode': 200,
            'headers': headers,
            'body': json.dumps({'message': 'Data stored successfully'})
        }

    except Exception as e:
        # Log the exception and return an error response
        print(f"Error storing data: {str(e)}")
        return {
            'statusCode': 500,
            'headers': headers,
            'body': json.dumps({'error': 'Could not store data'})
        }
```

AWS Services Used

- **AWS Lambda**: Executes backend logic to handle form submissions and store data in DynamoDB without needing dedicated servers.
- **DynamoDB**: A fully managed NoSQL database used to store user messages.
- **S3**: Hosts static files (HTML, CSS, JavaScript) and image assets for the website.
- **CloudFront**: Distributes content globally with low latency by caching static assets, ensuring high availability.
- **Route 53**: Manages DNS and routes traffic to the application efficiently.

Summary

This project leverages the power of serverless architecture on AWS to create a highly scalable and cost-effective web application. The combination of frontend files with Lambda and DynamoDB enables efficient, event-driven data handling while minimizing infrastructure management. The use of S3 and CloudFront ensures fast content delivery for a seamless user experience.