Cloud-Based Customer Relationship Management with AWS Lambda and DynamoDB

A Course Project Report Submitted in partial fulfillment of the course requirements for the award of grades in the subject of

CLOUD BASED AIML SPECIALITY
(22SDCS07A)

by

N. Khanishk

2210030200

Under the esteemed guidance of

Ms. P. Sree Lakshmi
Assistant Professor,
Department of Computer Science and Engineering

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

K L Deemed to be UNIVERSITY

Aziznagar, Moinabad, Hyderabad,
Telangana, Pincode: 500075

April 2025

K L Deemed to be UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Certificate

This is Certified that the project entitled **“**Cloud-Based Customer Relationship Management (CRM) with AWS Sumerian**”** which is an Experimental work carried out by N. Khanishk (2210030200), in partial fulfilment of the course requirements for the award of grades in the subject of **CLOUD BASED AIML SPECIALITY**, during the year **2024-2025**. The project has been approved as it satisfies the academic requirements.

Ms.P.Sree Lakshmi	**Dr. Arpita Gupta**
Course Coordinator	**Head of the Department**

Ms. P. Sree Lakshmi

Course Instructor

CONTENTS

1. INTRODUCTION

The Cloud-Based CRM with AWS Sumerian is a scalable customer management system that integrates cloud computing and AI-driven virtual assistance [1]. Since AWS Sumerian is no longer available, the project compensates for its interactive capabilities by enhancing the **index.html** interface with advanced features. This system enables businesses to manage customer interactions, track engagement, and automate workflows efficiently. It provides real-time data access [3], secure authentication, and an intuitive user experience, helping organizations streamline customer relationship management.

Designed as a web-based solution, the CRM enhances customer support by automating responses and optimizing business operations. The platform ensures seamless accessibility, allowing businesses to handle inquiries, store relevant data, and improve customer satisfaction [6]. By leveraging cloud technology, the system maintains a cost-effective, scalable, and efficient architecture, making it a reliable tool for modern business environments.

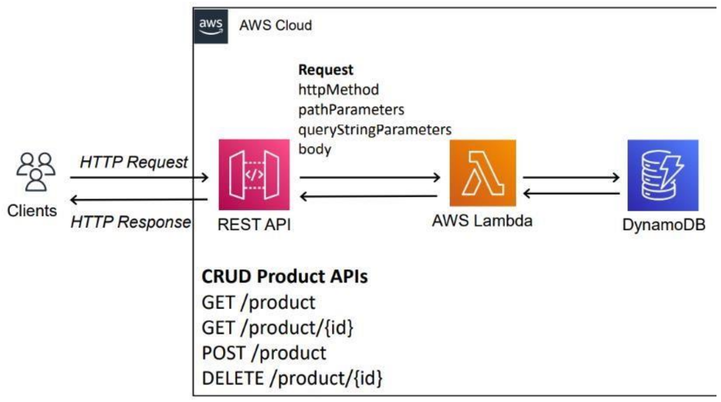
The CRM system eliminates the need for on-premise infrastructure, ensuring lower operational costs and easier scalability. It efficiently manages customer records using Amazon DynamoDB, processes requests through AWS Lambda, and secures user authentication with IAM roles [4]. Amazon CloudWatch monitors system performance [5], while API Gateway facilitates seamless communication between frontend and backend services [2].

2. AWS Services Used as part of the project

The project utilizes several AWS services to build a serverless architecture:

1. AWS Lambda:
   - Hosts the backend logic for customer management.
   - Handles all API requests, including user interactions, data processing, and automated workflows.
   - Provides a serverless execution environment, scaling automatically based on demand.

2. Amazon API Gateway:
   - Acts as the entry point for all HTTP requests from the frontend.
   - Exposes the Lambda function as RESTful endpoints (e.g., /add-customer, /getcustomers).
   - Manages CORS settings for cross-origin access from the web application.



3. Amazon DynamoDB:
   - A NoSQL database used to store customer records and interaction history.
   - Configured with customerID as the partition key.
   - Ensures low-latency, hight-performance storage for seamless operations.

4. AWS Identity and Access Management (IAM):
    - Used to create a role (CRMRole) with permissions for Lambda to interact with DynamoDB.
    - Ensures secure access to AWS resources.

5. Amazon CloudWatch:
    - Used for logging and debugging the Lambda function.
    - Provides insights into errors (e.g., 500 Internal Server Errors, Decimal serialization issues) and application behavior.

These services work together to create a fully serverless application, minimizing operational overhead and ensuring scalability.

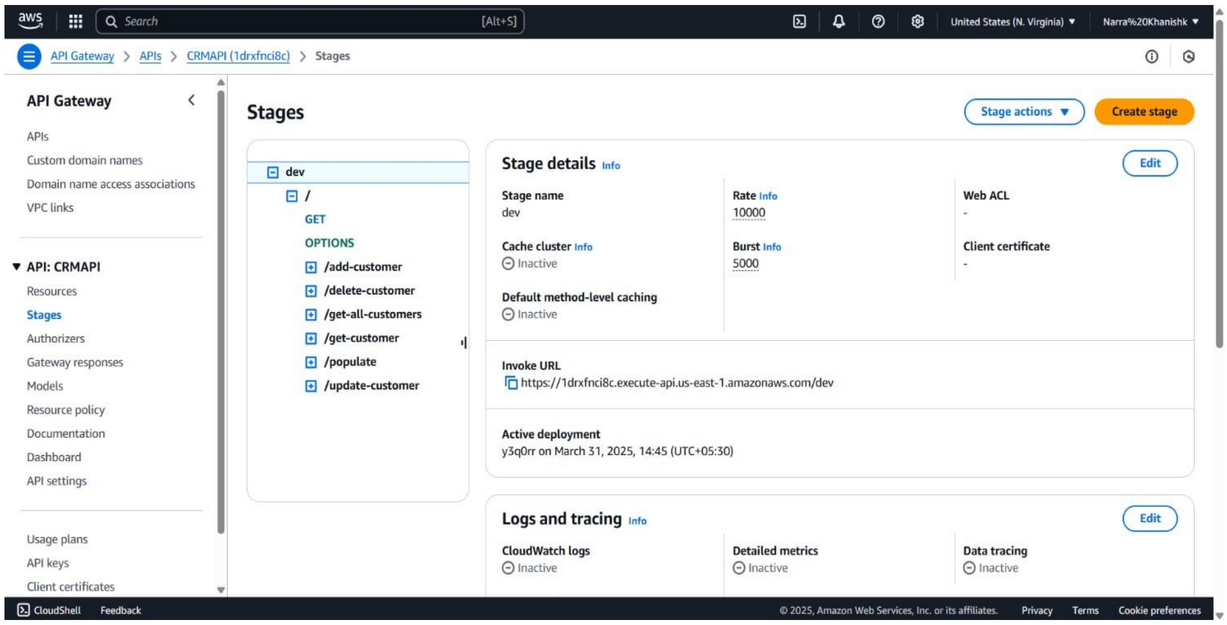
3. Steps Involved in Solving the Project Problem Statement

The project aimed to build a Cloud-Based CRM system with a serverless architecture to manage customer interactions efficiently. The following steps were taken to implement the solution:

1. Set Up the DynamoDB Table:
   - o Created a DynamoDB table named CustomerTable with customerID as the partition key.
   - o Configured on-demand capacity to handle dynamic workloads [3].
2. Created the Lambda Function:
   - o Developed CRMFunction to handle API routes (/, /add-customer, /get-customer, /get-allcustomers, /update-customer, /delete-customer) [1].
   - o Implemented logic for CRUD operations and automated workflows [6].
3. Configured API Gateway:
   - o Set up API Gateway (CRMAPI) to expose Lambda functions as RESTful endpoints.
   - o Enabled CORS for cross-origin requests from the frontend [7].
   - o Deployed the API to a dev stage, obtaining the Invoke URL (https://mo1eyckg8c.executeapi.us-east-1.amazonaws.com/dev).
4. Developed the Frontend:
   - o Created an index.html interface for customer management with form validation.
   - o Integrated Axios to make HTTP requests to the API Gateway [2].
   - o Included form validation and error handling for a better user experience.
5. Deployed the Application:
   - o Packaged lambda_function.py and index.html into a zip file (crm_lambda.zip).
   - o Uploaded the zip file to the Lambda function and deployed it.
6. Debugged and Fixed Issues:
   - o 500 Internal Server Error: Resolved by validating API requests.
   - o CORS Issues: Configured API Gateway to allow browser requests.
   - o Data Formatting Issues: Ensured correct serialization for API responses.
7. Tested the Application:
   - o Populated the database and verified CRUD operations via the frontend.
   - o Checked DynamoDB data consistency and monitored logs in CloudWatch for any errors.

4. Stepwise Screenshots with Brief Description

Step 1: API Gateway Resource Setup



Stage details Info

Stage name
dev

Rate Info
10000

Web ACL
-

Cache cluster Info
Inactive

Burst Info
5000

Client certificate
-

Default method-level caching
Inactive

Invoke URL
https://1drxfnci8c.execute-api.us-east-1.amazonaws.com/dev

Active deployment
y3q0rr on March 31, 2025, 14:45 (UTC+05:30)

Logs and tracing Info

CloudWatch logs
Inactive

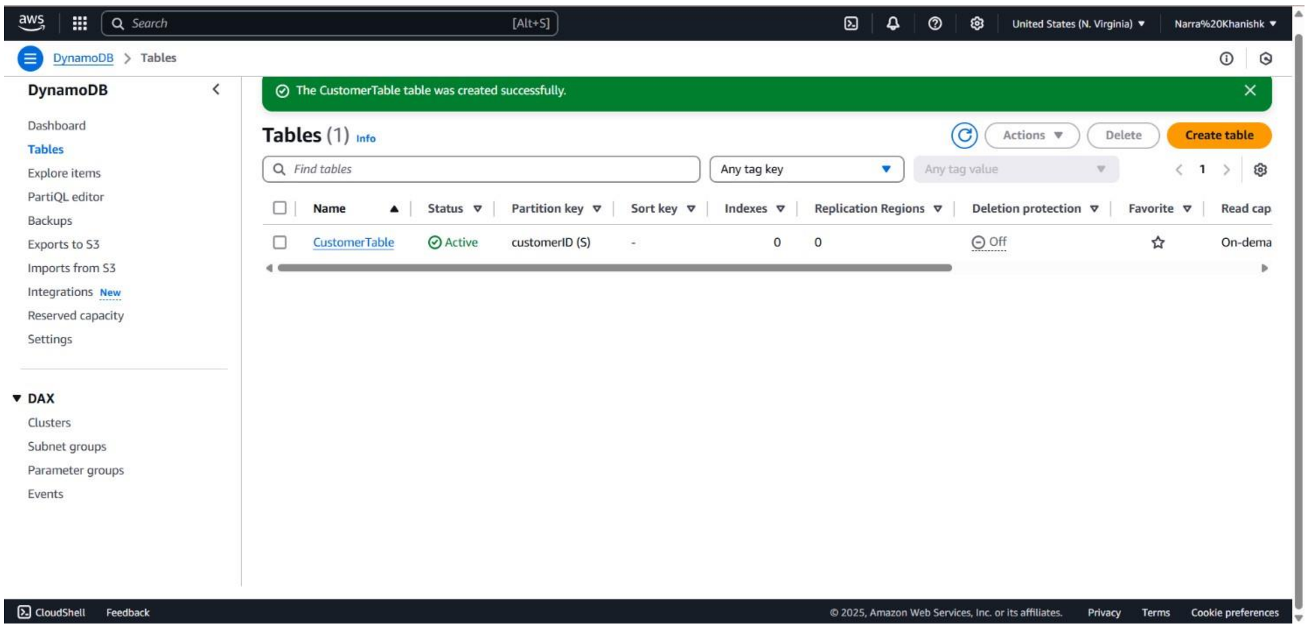
Detailed metrics
Inactive

Data tracing
Inactive

Fig. 4.1: Configuration of API Gateway resources and methods for enabling RESTful communication with backend services.

The API Gateway (CRMAPI) is configured with multiple resources, including /add-customer, /deletecustomer, /get-all-customers, /get-customer, /populate, and /update-customer. The stage name "dev" is active, and the Invoke URL is displayed, allowing the frontend to communicate with the backend. The API Gateway logs and tracing options are visible but currently inactive [5]. This setup enables seamless request handling for the Cloud-Based CRM [2].

Step 2: DynamoDB Table with Items

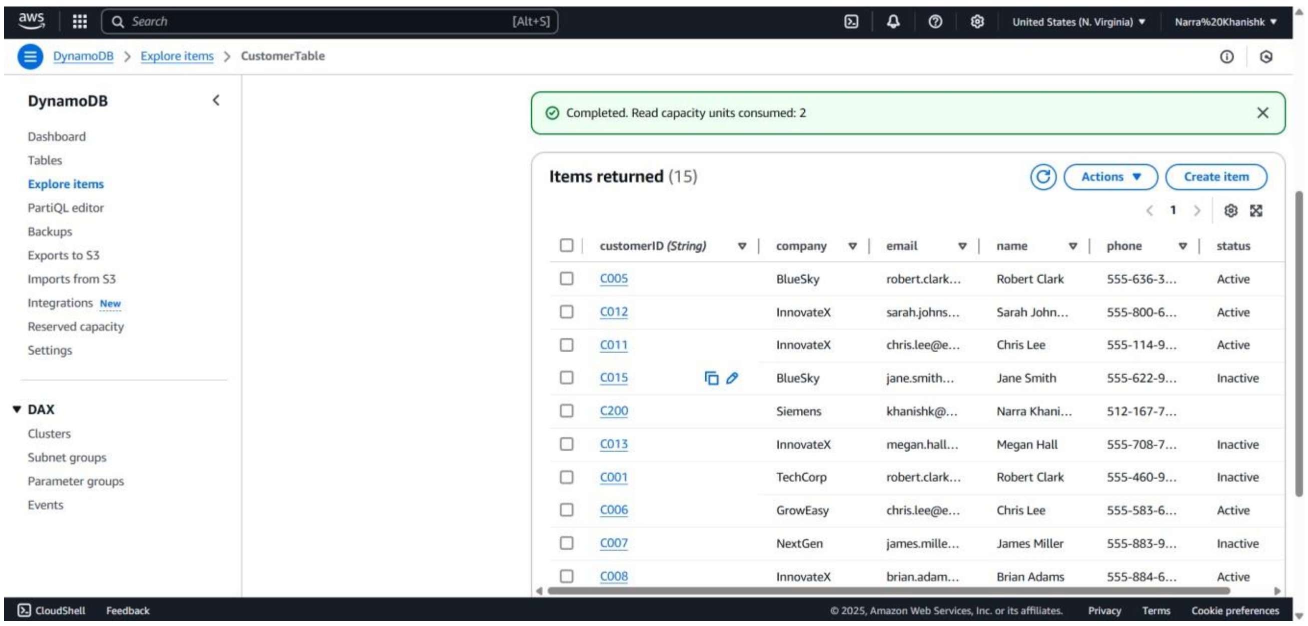


The CustomerTable table was created successfully.

Tables (1) Info

Name	Status	Partition key	Sort key	Indexes	Replication Regions	Deletion protection	Favorite	Read cap
CustomerTable	Active	customerID (S)	-	0	0	Off	☆	On-dema

Fig. 4.2: CustomerTable created in DynamoDB with 'customerID' as the partition key.



Completed. Read capacity units consumed: 2

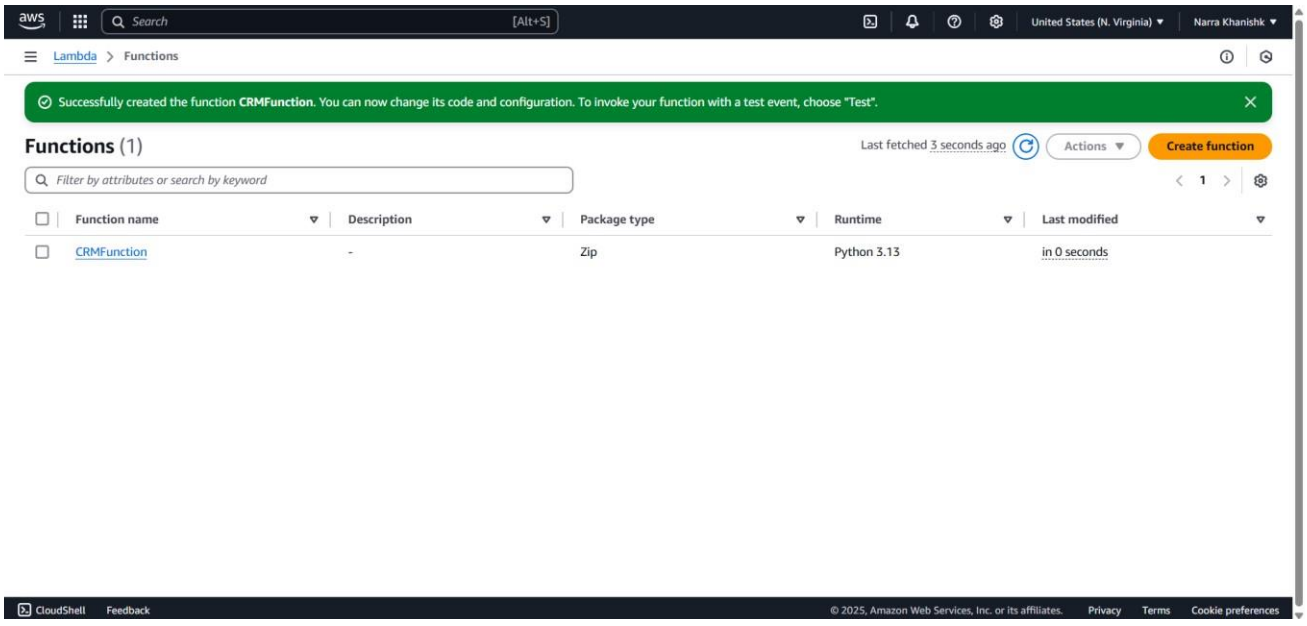
Items returned (15)

customerID (String)	company	email	name	phone	status
C005	BlueSky	robert.clark...	Robert Clark	555-636-3...	Active
C012	InnovateX	sarah.johns...	Sarah John...	555-800-6...	Active
C011	InnovateX	chris.lee@e...	Chris Lee	555-114-9...	Active
C015	BlueSky	jane.smith...	Jane Smith	555-622-9...	Inactive
C200	Siemens	khanishk@...	Narra Khani...	512-167-7...	
C013	InnovateX	megan.hall...	Megan Hall	555-708-7...	Inactive
C001	TechCorp	robert.clark...	Robert Clark	555-460-9...	Inactive
C006	GrowEasy	chris.lee@e...	Chris Lee	555-583-6...	Active
C007	NextGen	james.mille...	James Miller	555-883-9...	Inactive
C008	InnovateX	brian.adam...	Brian Adams	555-884-6...	Active

Fig. 4.3: CustomerTable items displayed in DynamoDB.

This visual shows the CustomerTable in DynamoDB after populating it with 100 customer records using the /populate endpoint. Each record includes customerID, customerName, email, phone, status, and company [3].

Step 3: Lambda Function Creation


Successfully created the function **CRMFunction**. You can now change its code and configuration. To invoke your function with a test event, choose "Test".

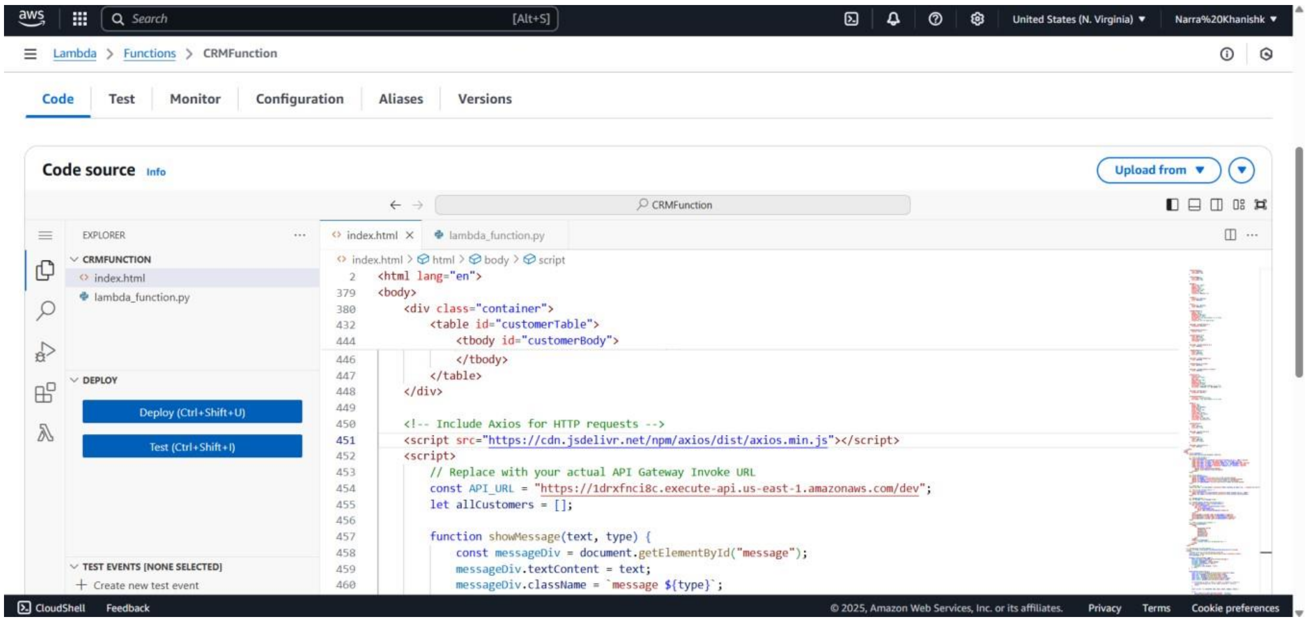
Functions (1)

Function name	Description	Package type	Runtime	Last modified
CRMFunction	-	Zip	Python 3.13	in 0 seconds

Fig. 4.4: Lambda function CRMFunction created with Python 3.13 runtime.

This visual shows the creation of the CustomerFunction in the Lambda console. The function is configured with the LambdaCRMRole and set to use Python 3.13 as the runtime [1].

Step 4: Frontend & Backend Code


Lambda > Functions > CRMFunction

Code source

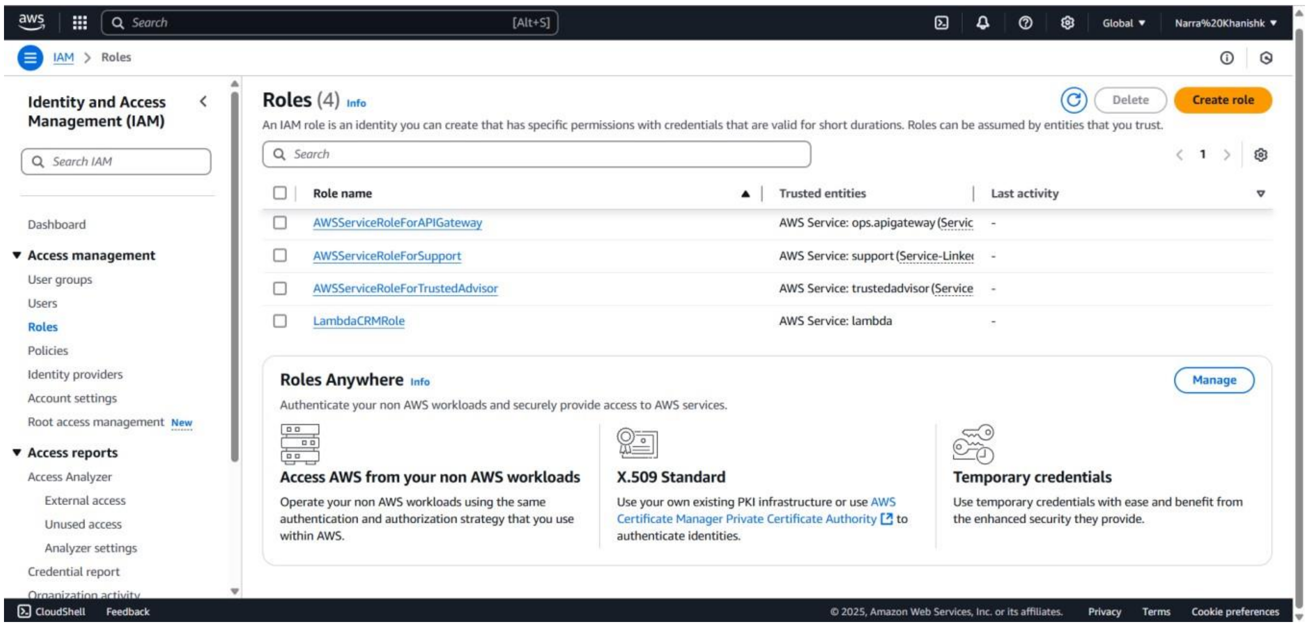
CRMFUNCTION: index.html, lambda_function.py

```
<html lang="en">
<body>
    <div class="container">
        <table id="customerTable">
            <tbody id="customerBody">
            </tbody>
        </table>
    </div>

    <!-- Include Axios for HTTP requests -->
    <script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>
    <script>
        // Replace with your actual API Gateway Invoke URL
        const API_URL = "https://1drxfnci8c.execute-api.us-east-1.amazonaws.com/dev";
        let allCustomers = [];

        function showMessage(text, type) {
            const messageDiv = document.getElementById("message");
            messageDiv.textContent = text;
            messageDiv.className = `message ${type}`;
```

Fig. 4.5: CRMFunction code editor with API integration and customer data logic.

This visual shows the AWS Lambda console with a CRM function, displaying the code editor with index.html and lambda_function.py files. The code includes an API Gateway URL, Axios for HTTP requests, and a table for customer data management.

Step 5: Attaching DynamoDB Permissions to the IAM Role



Roles (4) Info

An IAM role is an identity you can create that has specific permissions with credentials that are valid for short durations. Roles can be assumed by entities that you trust.

Role name	Trusted entities	Last activity
AWSServiceRoleForAPIGateway	AWS Service: ops.apigateway (Servic	-
AWSServiceRoleForSupport	AWS Service: support (Service-Linke	-
AWSServiceRoleForTrustedAdvisor	AWS Service: trustedadvisor (Service	-
LambdaCRMRole	AWS Service: lambda	-

Roles Anywhere Info

Authenticate your non AWS workloads and securely provide access to AWS services.

Access AWS from your non AWS workloads
Operate your non AWS workloads using the same authentication and authorization strategy that you use within AWS.

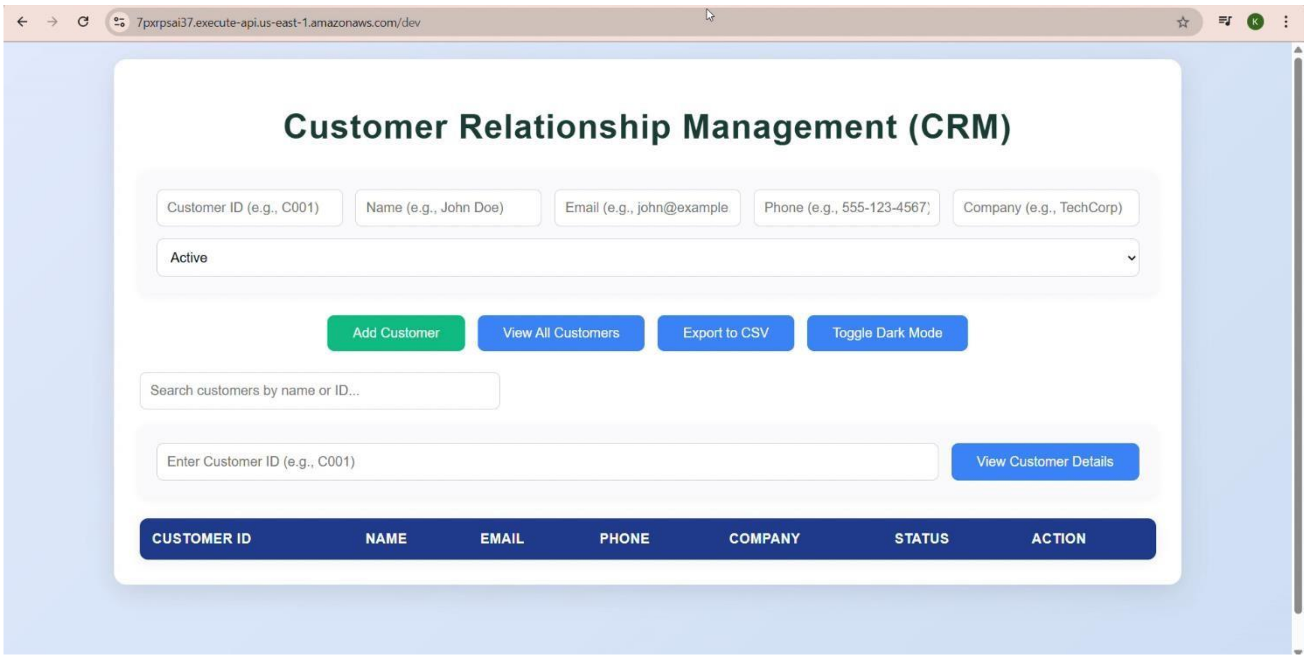
X.509 Standard
Use your own existing PKI infrastructure or use AWS Certificate Manager Private Certificate Authority to authenticate identities.

Temporary credentials
Use temporary credentials with ease and benefit from the enhanced security they provide.

Fig. 4.6: AmazonDynamoDBFullAccess policy attached to LambdaCRMRole for DynamoDB access.

This visual displays the process of attaching the AmazonDynamoDBFullAccess policy to the LambdaCRMRole. This policy grants the Lambda function permissions to perform CRUD operations on the CustomerTable in DynamoDB [1].

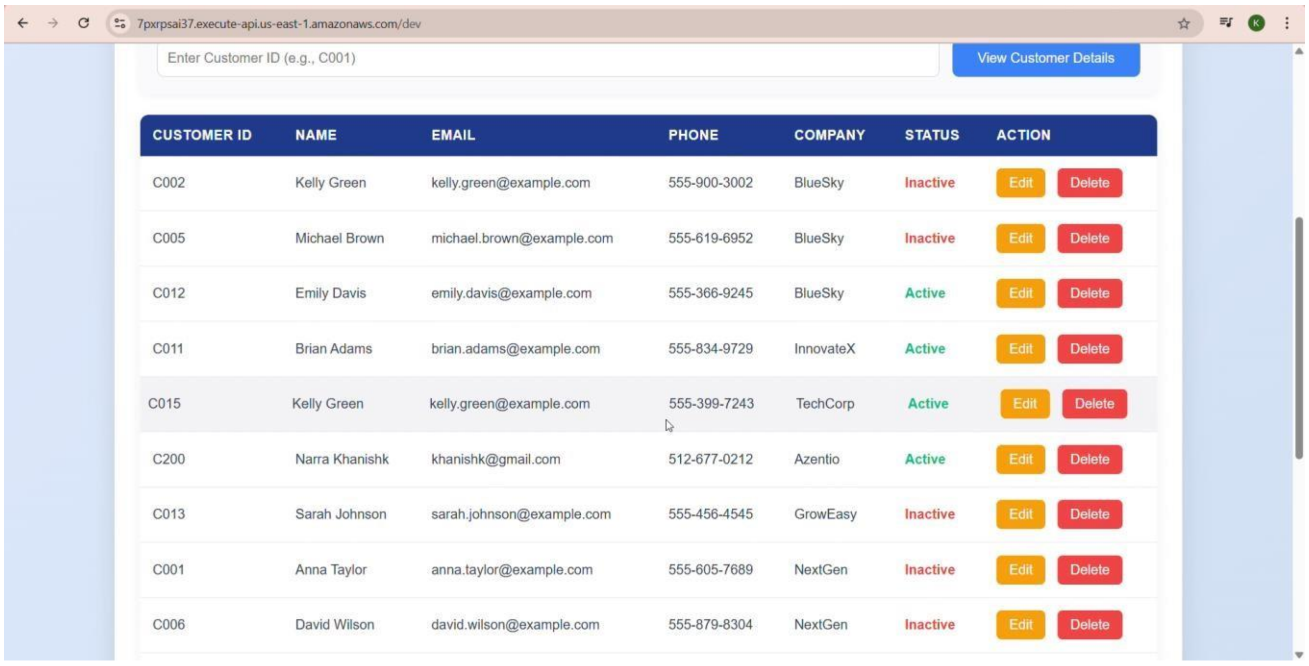
Step 6: Final output


7pxrpsai37.execute-api.us-east-1.amazonaws.com/dev
Customer Relationship Management (CRM)
Customer ID (e.g., C001)
Name (e.g., John Doe)
Email (e.g., john@example
Phone (e.g., 555-123-4567)
Company (e.g., TechCorp)
Active
Add Customer
View All Customers
Export to CSV
Toggle Dark Mode
Search customers by name or ID...
Enter Customer ID (e.g., C001)
View Customer Details

CUSTOMER ID	NAME	EMAIL	PHONE	COMPANY	STATUS	ACTION

Fig. 4.7: Cloud-Based CRM dashboard with customer input, search, export, and data table.

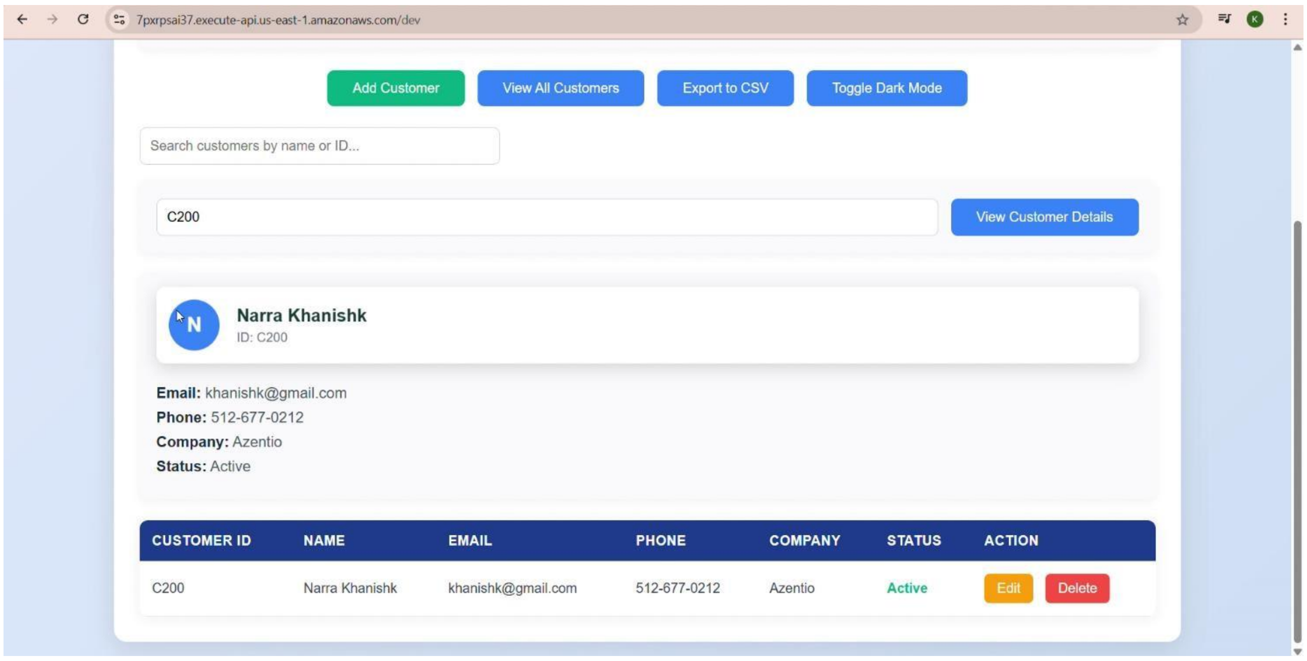
This visual shows the frontend interface of the Cloud-Based CRM system, displaying a customer management dashboard. It includes input fields for adding customers, buttons for viewing and exporting customer data, a search bar, and a table for listing customer details.


7pxrpsai37.execute-api.us-east-1.amazonaws.com/dev
Enter Customer ID (e.g., C001)
View Customer Details

CUSTOMER ID	NAME	EMAIL	PHONE	COMPANY	STATUS	ACTION
C002	Kelly Green	kelly.green@example.com	555-900-3002	BlueSky	Inactive	Edit Delete
C005	Michael Brown	michael.brown@example.com	555-619-6952	BlueSky	Inactive	Edit Delete
C012	Emily Davis	emily.davis@example.com	555-366-9245	BlueSky	Active	Edit Delete
C011	Brian Adams	brian.adams@example.com	555-834-9729	InnovateX	Active	Edit Delete
C015	Kelly Green	kelly.green@example.com	555-399-7243	TechCorp	Active	Edit Delete
C200	Narra Khanishk	khanishk@gmail.com	512-677-0212	Azentio	Active	Edit Delete
C013	Sarah Johnson	sarah.johnson@example.com	555-456-4545	GrowEasy	Inactive	Edit Delete
C001	Anna Taylor	anna.taylor@example.com	555-605-7689	NextGen	Inactive	Edit Delete
C006	David Wilson	david.wilson@example.com	555-879-8304	NextGen	Inactive	Edit Delete

Fig. 4.8: Customer dashboard displaying records with edit and delete options.

This visual shows a customer management dashboard listing multiple customers with details such as ID, name, email, phone, company, and status. The interface provides action buttons for editing or deleting customer records.



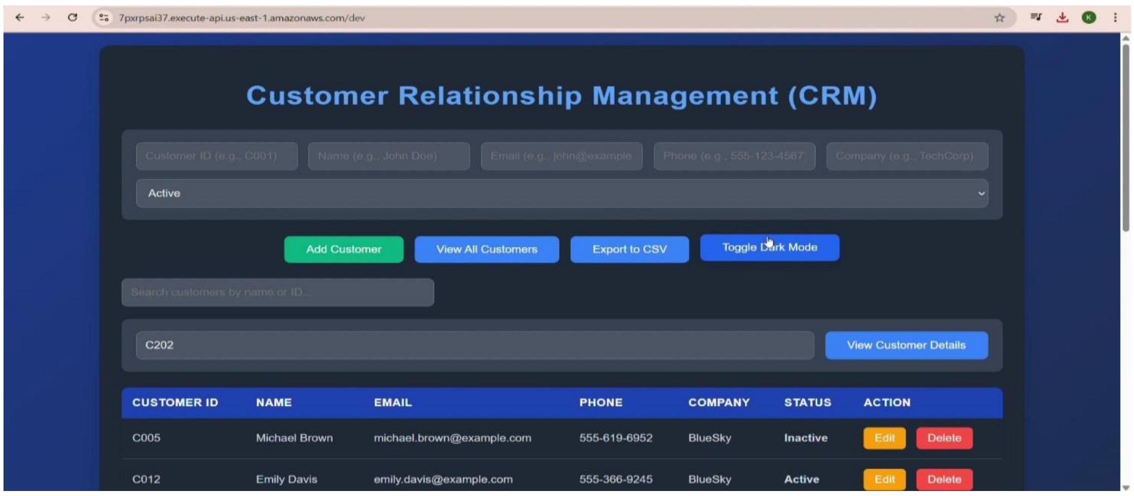
Narra Khanishk
ID: C200

Email: khanishk@gmail.com
Phone: 512-677-0212
Company: Azentio
Status: Active

CUSTOMER ID	NAME	EMAIL	PHONE	COMPANY	STATUS	ACTION
C200	Narra Khanishk	khanishk@gmail.com	512-677-0212	Azentio	Active	Edit Delete

Fig. 4.9: Customer dashboard showing search results based on query.

This visual shows a customer management dashboard displaying customer details based on a search query.



Customer Relationship Management (CRM)

Active

C202

CUSTOMER ID	NAME	EMAIL	PHONE	COMPANY	STATUS	ACTION
C005	Michael Brown	michael.brown@example.com	555-619-6952	BlueSky	Inactive	Edit Delete
C012	Emily Davis	emily.davis@example.com	555-366-9245	BlueSky	Active	Edit Delete

Fig. 4.10: Customer interface with form, data table, status, actions, and theme toggle.

5. Learning Outcomes

1. Understanding Serverless Architecture:

   Learned how to build a serverless application using AWS Lambda, API Gateway, and DynamoDB. Understood the benefits of serverless computing, such as automatic scaling and cost efficiency [1].

2. Working with AWS Services:

   Gained hands-on experience with AWS Lambda, API Gateway, DynamoDB, IAM, and CloudWatch[2]. Learned to configure IAM roles and permissions for secure access to AWS resources.

3. IAM Role Configuration:

   Understood the importance of IAM roles in securing AWS resources. Learned to create and attach policies to roles, ensuring least-privilege access for Lambda functions [4].

4. Debugging and Problem-Solving:

   Developed skills in debugging serverless applications using CloudWatch logs. Resolved issues like 500 Internal Server Errors, CORS problems, and Decimal serialization errors [5].

5. Frontend-Backend Integration:

   Integrated a frontend with a backend API using Axios. Implemented CORS to allow cross-origin requests [7].

6. Best Practices in Development:

   Learned the importance of error handling, logging, and data serialization in serverless applications. Gained experience in packaging and deploying Lambda functions with dependencies [1].

6. Conclusion

The CRM System project showcases the effectiveness of serverless architecture in building scalable and cost-efficient applications. By integrating AWS services like Lambda, API Gateway, DynamoDB, IAM, CloudWatch, and Sumerian, the system enables seamless CRUD operations on customer data through a web-based interface. Key challenges such as CORS, data validation, and serialization were addressed, resulting in a stable solution. AWS Sumerian added interactive elements, enhancing user experience. The project highlights the flexibility and scalability of serverless technologies, with future improvements planned for authentication, advanced search, and frontend deployment via S3 and CloudFront.

7. References

[1] AWS Lambda Developer Guide:
https://docs.aws.amazon.com/lambda/latest/dg/welcome.html

[2] Amazon API Gateway Developer Guide:
https://docs.aws.amazon.com/apigateway/latest/developerguide/welcome.html

[3] Amazon DynamoDB Developer Guide:
https://docs.aws.amazon.com/dynamodb/latest/developerguide/Introduction.html

[4] AWS IAM Documentation:
https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html

[5] Amazon CloudWatch Documentation:
https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/WhatIsCloudWatch.html

[6] "Building a Serverless Application with AWS Lambda and DynamoDB" – AWS Blog: https://aws.amazon.com/blogs/compute/building-a-serverless-application-with-aws-lambda-anddynamodb/

[7] "How to Enable CORS on API Gateway" – AWS Knowledge Centre:
https://aws.amazon.com/premiumsupport/knowledge-center/api-gateway-cors-errors/