

Neocracy Travel Connect

Harjot Singh and Rishi Pawar

Class: CSE-C

Roll No: 12320802723 & 11420802723

Trainer: Saurabh Dwivedi



Assignments	AWS Services used
Assignment -1	EC2, EBS
Assignment-2	EC2, S3, SNS
Assignment-3	RDS, DynamoDB, CloudWatch
Assignment-4	EC2, ELB, Auto-Scaling, Cloud Watch
Assignment-5	Lambda, API Gateway, S3, DynamoDB

Project Report: Neocracy Travel Connect

Project Overview

Neocracy Travel Connect is a full-stack, serverless contact and booking web application designed for **Neocracy India Travel**, a travel service provider. The project allows users to explore destinations and directly submit travel queries through a responsive contact form. The entire backend is built using **AWS** serverless technologies ensuring scalability, zero server management, and cost-efficiency.

Key Features

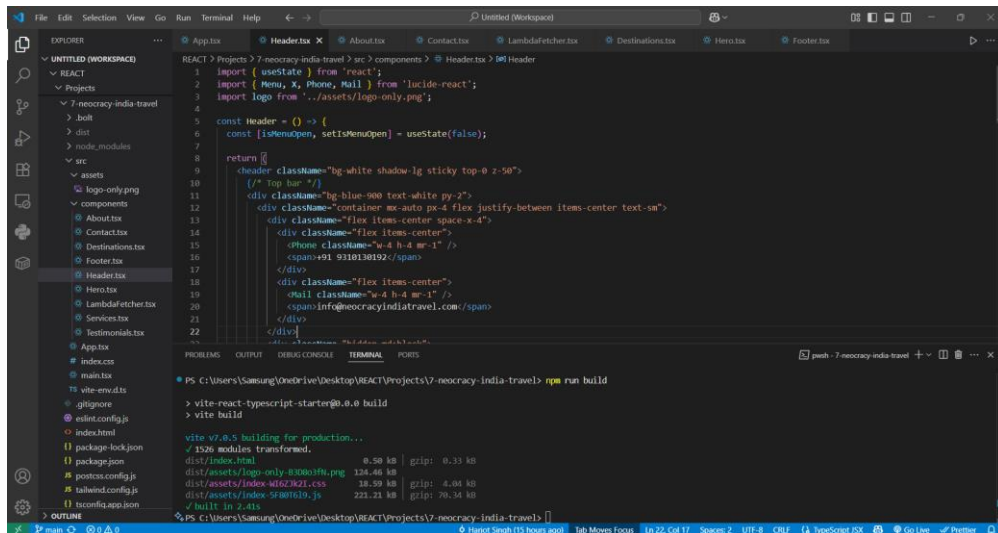
- Responsive front-end travel website built with **React + TypeScript + Tailwind CSS**
 - Static site hosted on **AWS S3** with a live custom endpoint
 - Functional contact form with the following capabilities:
 - Collects user inputs (name, email, phone, destination, message)
 - Sends form data to **AWS Lambda** via **API Gateway**
 - **Email notification** sent through **Amazon SNS**
 - Data saved into **Amazon DynamoDB**
 - Backend entirely serverless (no EC2 or manual servers involved)
 - Secured access via **IAM Roles**
-

Technologies Used

- **Frontend:** React, TypeScript, Tailwind CSS
 - **Hosting:** AWS S3 (Static Website Hosting)
 - **Backend:** AWS Lambda (Python 3)
 - **API Gateway:** For routing frontend requests to Lambda
 - **SNS:** For sending real-time email alerts
 - **DynamoDB:** For persisting form submissions
 - **IAM Roles:** For secure resource permissions
 - **Axios:** For HTTP requests from frontend to backend
-

Implementation Steps:

1. Created a React project using Vite with TypeScript support



```
1 import { useState } from 'react';
2 import { Menu, X, Phone, Mail } from 'lucide-react';
3 import logo from '../assets/logo-only.png';
4
5 const Header = () => {
6   const [isOpen, setIsMenuOpen] = useState(false);
7
8   return (
9     <header className="bg-white shadow-lg sticky top-0 z-50">
10      <div className="bg-blue-900 text-white py-2">
11        <div className="container mx-auto px-4 flex justify-between items-center text-sm">
12          <div className="flex items-center space-x-4">
13            <div className="flex items-center">
14              <Phone className="w-4 h-4 mr-1"/>
15              <span>+91 9310130192</span>
16            </div>
17            <div className="flex items-center">
18              <Mail className="w-4 h-4 mr-1"/>
19              <span>info@neocracyindiatravel.com</span>
20            </div>
21          </div>
22        </div>
23      </div>
24    </header>
25  );
26}
```

PS C:\Users\Samsung\OneDrive\Desktop\REACT\Projects\7-neocracy-india-travel> npm run build

> vite-react-typescript-starter@0.0.0 build

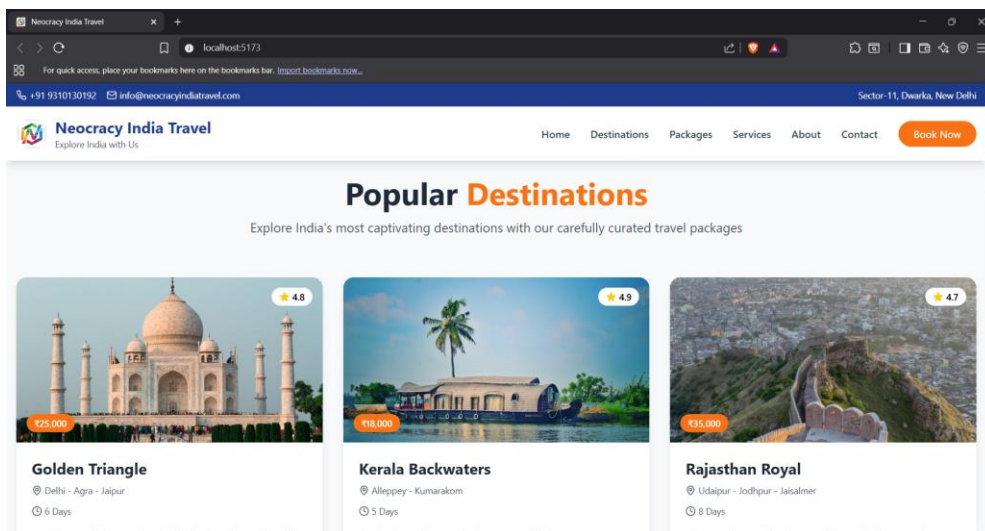
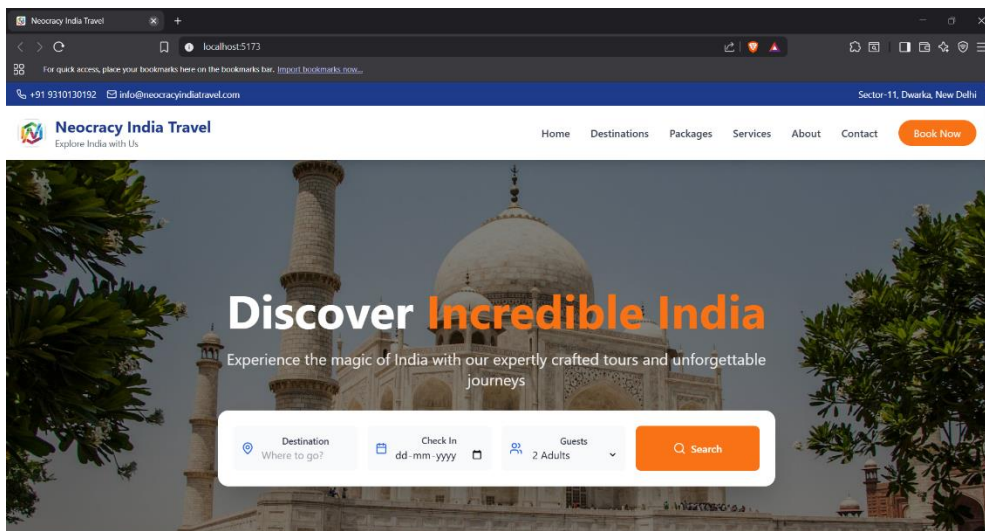
> vite build

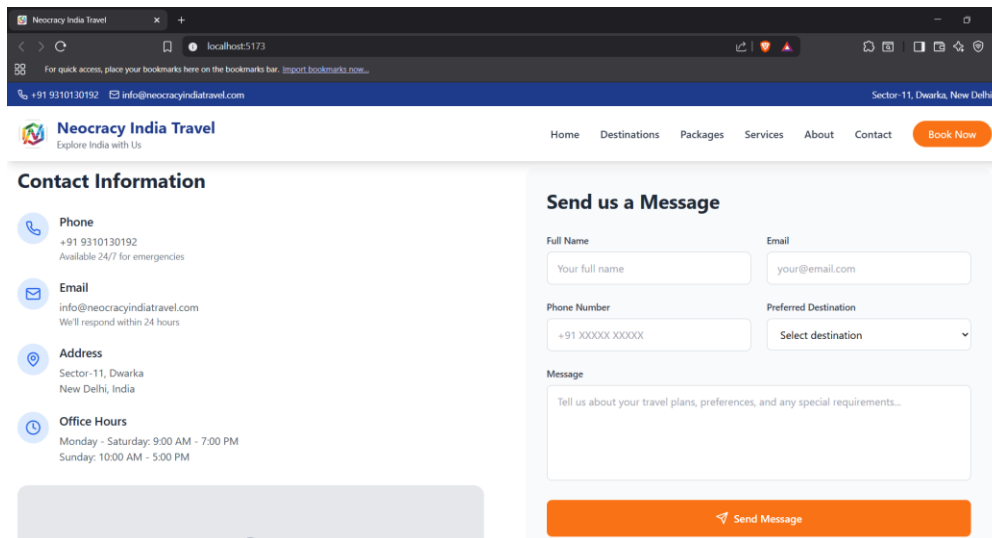
vite v7.8.5 building for production...

✓ 1526 modules transformed.

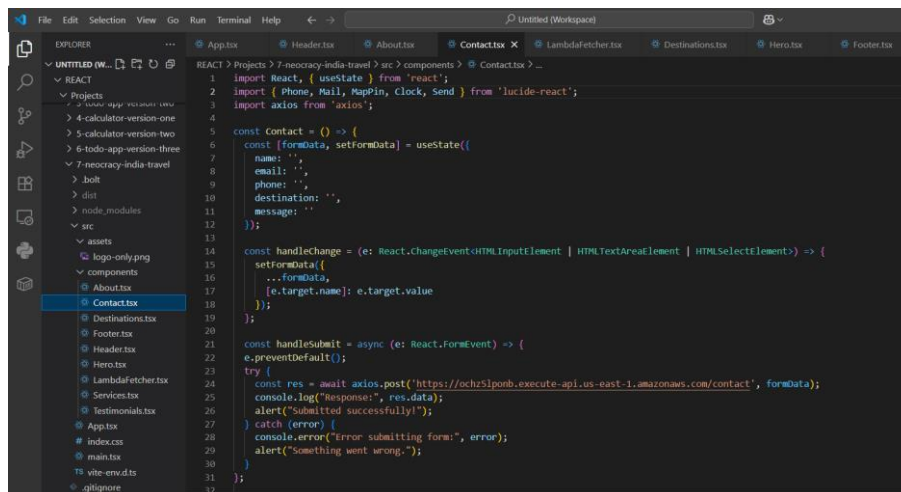
dist/assets/index.html	0.50 kB	gzip: 0.33 kB
dist/assets/logo-only-8386a3f6.png	134.46 kB	gzip: 4.04 kB
dist/assets/index-WfGZ3G2L.css	18.59 kB	gzip: 76.34 kB
dist/assets/index-SR0061D9.js	223.23 kB	gzip: 76.34 kB

✓ built in 2.41s

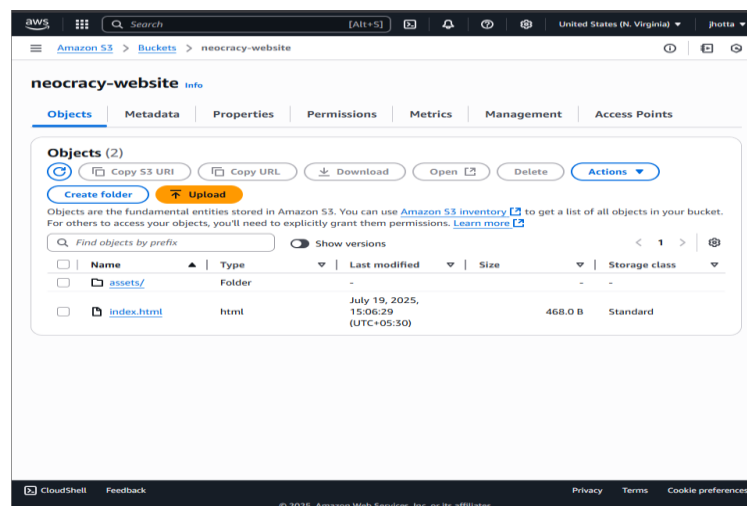




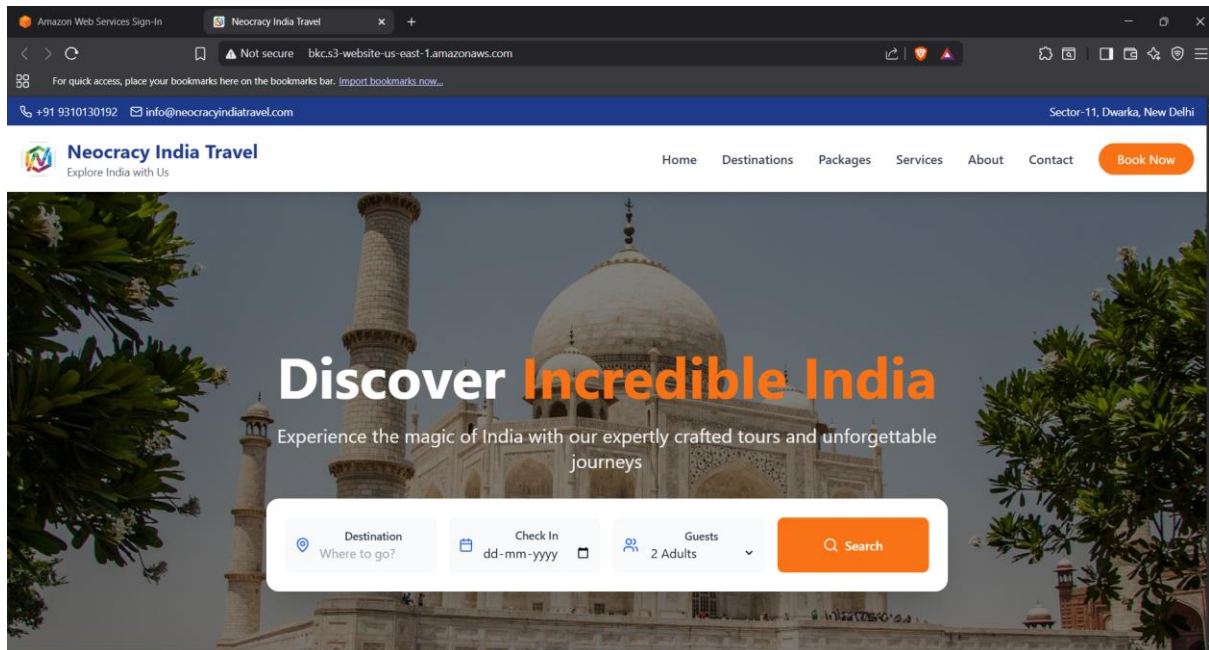
2. Used Axios to handle HTTP POST requests from React to API Gateway



3. Built production files using npm run build and uploaded the /dist folder to an AWS S3 bucket with public read permissions

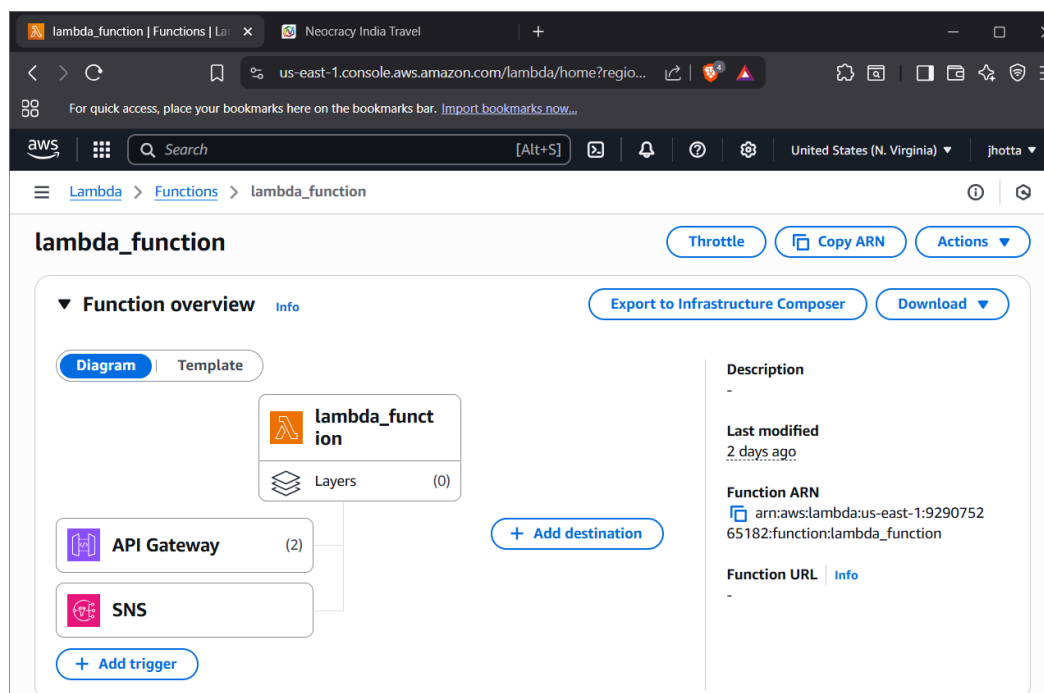


4. Enabled static website hosting on the S3 bucket and tested the live URL



5. AWS Lambda Function

- Created a Python 3 Lambda function
- Code parses incoming form data, formats it, and triggers two actions:
 - Publishes an SNS notification with form details
 - Stores data in DynamoDB with a timestamp and unique ID



Code source Info

lambda_function

```
1 import json
2 import boto3
3 import uuid
4 from datetime import datetime
5
6 sns = boto3.client('sns')
7 dynamodb = boto3.resource('dynamodb')
8 table = dynamodb.Table('ContactSubmissions')
9 SNS_TOPIC_ARN = 'arn:aws:sns:us-east-1:929075265182:mytopic'
10
11 def lambda_handler(event, context):
12     try:
13         body = json.loads(event.get('body', '{}'))
14
15         name = body.get('name', 'N/A')
16         email = body.get('email', 'N/A')
17         phone = body.get('phone', 'N/A')
18         destination = body.get('destination', 'N/A')
19         message = body.get('message', 'N/A')
20
21         # Save to DynamoDB
22         table.put_item(Item={
23             'id': str(uuid.uuid4()),
24             'name': name,
25             'email': email,
26             'phone': phone,
27             'destination': destination,
28             'message': message,
29             'timestamp': datetime.utcnow().isoformat()
30         })
31     except Exception as e:
```

Code source Info

lambda_function

```
11 def lambda_handler(event, context):
12     try:
13         # Compose message
14         sns_message = f"""
15         New Contact Form Submission:
16
17         Name: {name}
18         Email: {email}
19         Phone: {phone}
20         Destination: {destination}
21         Message: {message}
22         """
23
24         # Publish to SNS
25         response = sns.publish(
26             TopicArn=SNS_TOPIC_ARN,
27             Message=sns_message,
28             Subject="New Travel Query from Neocracy India Travel"
29         )
30
31         return {
32             "statusCode": 200,
33             "headers": {
34                 "Access-Control-Allow-Origin": "*",
35                 "Access-Control-Allow-Headers": "Content-Type",
36                 "Access-Control-Allow-Methods": "POST, OPTIONS"
37             },
38             "body": json.dumps({ "message": "Submitted successfully!" })
39         }
40     except Exception as e:
```

Code Test Monitor Configuration Aliases Versions

General configuration Triggers Permissions Destinations Function URL Environment variables Tags VPC RDS databases Monitoring and operations tools

Execution role

Role name
lambda_function-role-hq9vmdpr

Resource summary

To view the resources and actions that your function has permission to access, choose a service.

AWS End User Messaging SMS and Voice V2
12 actions, 1 resource

By action By resource

Resource	Actions
	Allow: sms-voice:DescribeVerifiedDestinationNumbers Allow: sms-voice:SendDestinationNumberVerificationCode Allow: sms-voice>DeleteVerifiedDestinationNumber Allow: sms-voice:SetTextMessageSpendLimitOverride Allow: sms-voice>DeleteOptedOutNumber

6. IAM Role Permissions

The screenshot shows the AWS IAM console for the role **lambda_function-role-hq9vmdpr**. The **Summary** tab is active, displaying the role's creation date (July 23, 2025, 10:23 UTC+05:30), last activity (Yesterday), ARN (`arn:aws:iam::929075265182:role/service-role/lambda_function-role-hq9vmdpr`), and maximum session duration (1 hour). Below the summary, the **Permissions** tab is selected, showing 4 attached policies. A table lists these policies:

Policy name	Type	Attached entities
AmazonS3FullAccess	AWS managed	2
AmazonSNSFullAccess	AWS managed	2
AWSLambdaBasicExecutionRole	Customer managed	1
myDBPolicy	Customer inline	0

7. API Gateway Configuration

The first screenshot shows the **Routes** configuration for the API **my-first-api (ochz5lponb)**. The **Routes for my-first-api** section shows a tree structure with **/contact** (POST) and **/hello** (GET). The **Route details** for **POST /contact (ID: bpftn60)** are shown, including the ARN (`arn:aws:apigateway:us-east-1::apis/ochz5lponb/routes/bpftn60`), authorization status (No authorizer attached), and integration details (Integration: `brmj2jcf`).

The second screenshot shows the **CORS** configuration for the same API. The **Configure CORS** section allows setting various headers and methods. The current configuration is as follows:

Setting	Value
Access-Control-Allow-Origin	*
Access-Control-Allow-Headers	content-type
Access-Control-Allow-Methods	GET, POST, OPTIONS
Access-Control-Max-Age	3600 Seconds
Access-Control-Expose-Headers	No Expose Headers are allowed
Access-Control-Allow-Credentials	NO

8. SNS Setup

aws

Search

[Alt+S]

United States (N. Virginia)

jhotta

Amazon SNS

Topics

mytopic

mytopic

Edit

Delete

Publish message

Details

Name

mytopic

ARN

arn:aws:sns:us-east-1:929075265182:mytopic

Type

Standard

Display name

-

Topic owner

929075265182

<

Subscriptions

Access policy

Data protection policy

Delivery policy (HTTP/S)

Delivery status logging

Encryption

>

Subscriptions (2)

Edit

Delete

Request confirmation

Confirm subscription

Create subscription

Search

<

1

>

⚙

ID	Endpoint	Status	Protocol
<input type="radio"/> a2cd85d4-1953-43...	arn:aws:lambda:us-...	Confirmed	LAMBDA
<input type="radio"/> a4f2b28c-907b-4b...	official.igharjot@g...	Confirmed	EMAIL

9. DynamoDB Setup

aws

Search

[Alt+S]

United States (N. Virginia)

jhotta

DynamoDB

Tables

ContactSubmissions

Tables (1)

Any tag key

Any tag value

Find tables

<

1

>

⚙

ContactSubmissions

☆

ContactSubmissions

☆

🔄

Actions

Explore table items

<

Settings

Indexes

Monitor

Global tables

Backups

Exports and stre

>

ⓘ Protect your DynamoDB table from accidental writes and deletes

When you turn on point-in-time recovery (PITR), DynamoDB backs up your table data automatically so that you can restore to any given second in the preceding 1 to 35 days. Additional charges apply. [Learn more](#)

Edit PITR

×

General information

Info

Get live item count

Partition key

id (String)

Alarms

✔ No active alarms

Average item size

176.5 bytes

Sort key

-

Point-in-time recovery (PITR)

Info

⊖ Off

Resource-based policy

Info

⊖ Not active

Capacity mode

On-demand

Item count

4

Table status

✔ Active

Table size

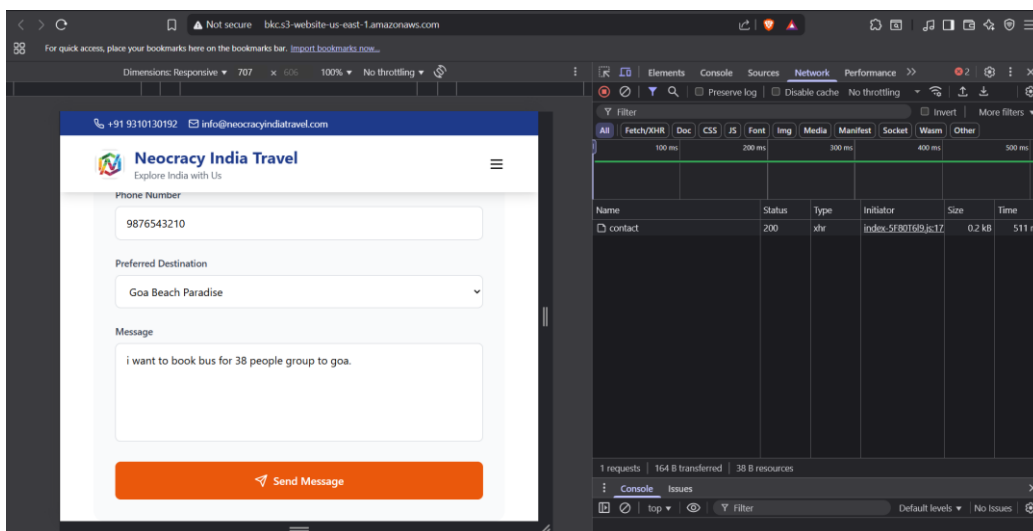
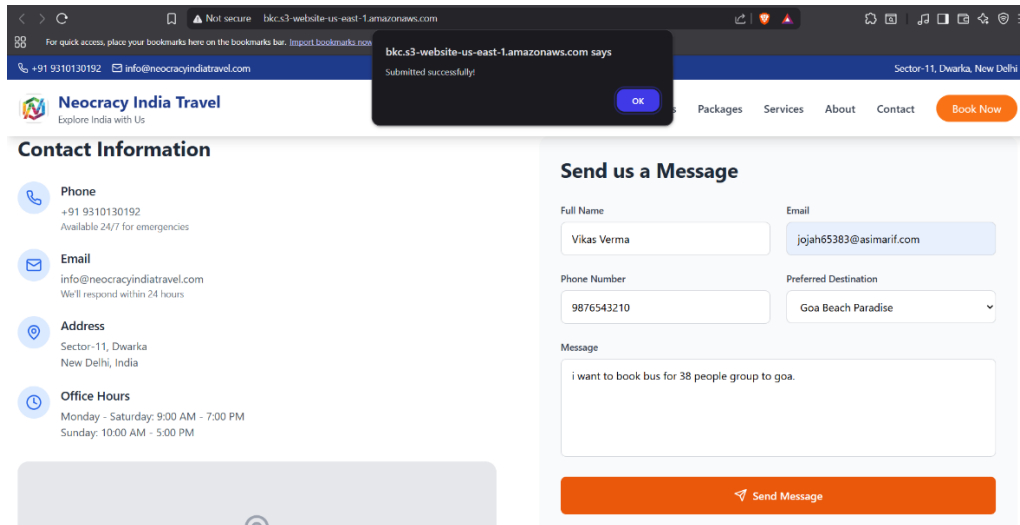
706 bytes

Amazon Resource Name (ARN)

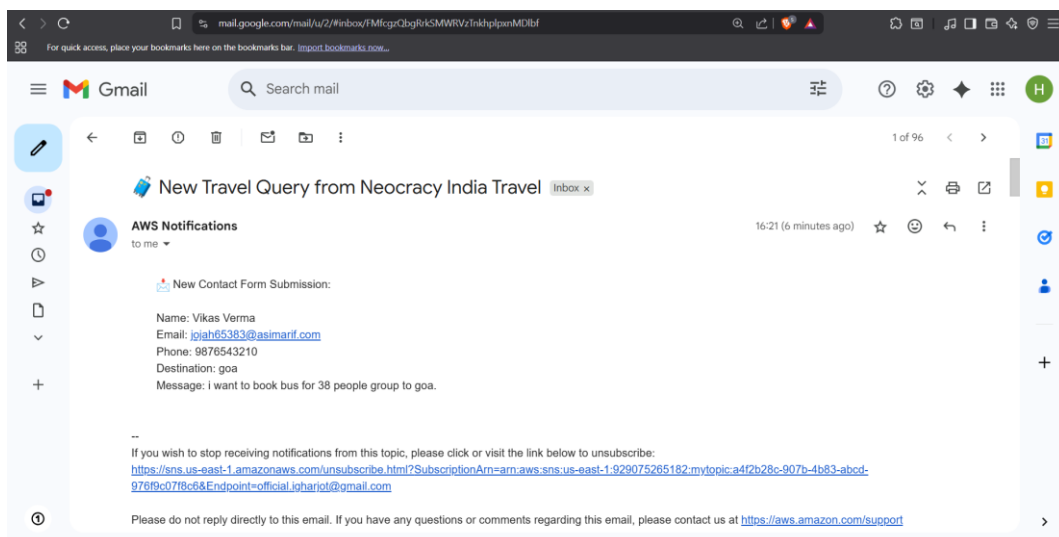
arn:aws:dynamodb:us-east-1:929075265182:table/ContactSubmissions

10. Testing & Debugging

- Used browser DevTools to monitor API requests



- Checked SNS delivery via email inbox



- Confirmed DynamoDB records through AWS Console

Run Reset

Completed - Items returned: 36 - Items scanned: 36 - Efficiency: 100% - RCUs consumed: 2

Table: ContactSubmissions - Items returned (6)

Scan started on July 26, 2025, 16:29:03

<input type="checkbox"/>	id (String)	destination	email	message	name	phone	timestamp
<input type="checkbox"/>	40a5e625-cff9-4d19-...	Golden Triangle	harjot@exa...	This is a tes...	Harjot Singh	9310130192	2025-07-25T08:22:20.825495
<input type="checkbox"/>	88718122-6683-4bef-...	goa	jojah65383...	i want to b...	Vikas Verma	9876543210	2025-07-26T10:51:56.010682
<input type="checkbox"/>	7698ca4c-36b4-4df1-...	himalayan	example@g...	HY	Harshit Kha...	9810941417	2025-07-25T05:12:33.890473
<input type="checkbox"/>	3ffd96aa-3e4a-4b19-...	rajasthan	example@g...	Hi, this is a ...	Example	9812346678	2025-07-25T08:24:34.695897
<input type="checkbox"/>	2e4d25d2-6c27-4c0b-...	varanasi	igharjot@g...	I wanted to ...	Harjot Singh	9310130192	2025-07-24T13:01:48.727036
<input type="checkbox"/>	59c85cd2-99cf-45e2-...	goa	jojah65383...	i want to b...	Vikas Verma	9876543210	2025-07-26T10:51:03.979424

Conclusion

Neocracy Travel Connect demonstrates the power of serverless architecture using AWS. It eliminates the need for traditional servers while still delivering scalable, secure, and fast performance. This project solidified practical knowledge in deploying React apps on AWS, building Lambda functions, and orchestrating AWS services using IAM, API Gateway, SNS, and DynamoDB.

Future Enhancements

- Integrate with Google Sheets or export data as CSV
- Build an admin dashboard to manage queries

Project Submitted by: Harjot Singh and Rishi Pawar
B.Tech CSE, BPIT, GGSIPU
GitHub: github.com/igharjot

Project Submitted to: Saurabh Dwivedi