S PRANAV

URL Shortener Service on AWS — Project Report

---

1. Introduction

- This project implements a URL shortener service using AWS serverless technologies.
- It provides a simple interface to convert long URLs into short, shareable links.
- The service stores URL mappings securely and redirects users via the short URLs.

---

2. Project Overview

Component	Technology	Purpose
Frontend	HTML, JavaScript	UI for inputting URLs and displaying short URLs
Backend	AWS Lambda (Node.js)	Handles URL shortening and redirect logic
API Gateway	AWS API Gateway	Exposes REST API endpoints for the Lambda
Database	AWS DynamoDB	Stores mappings between short codes and original URLs
Hosting	AWS Amplify / S3	Hosts the frontend static website

---

3. Features

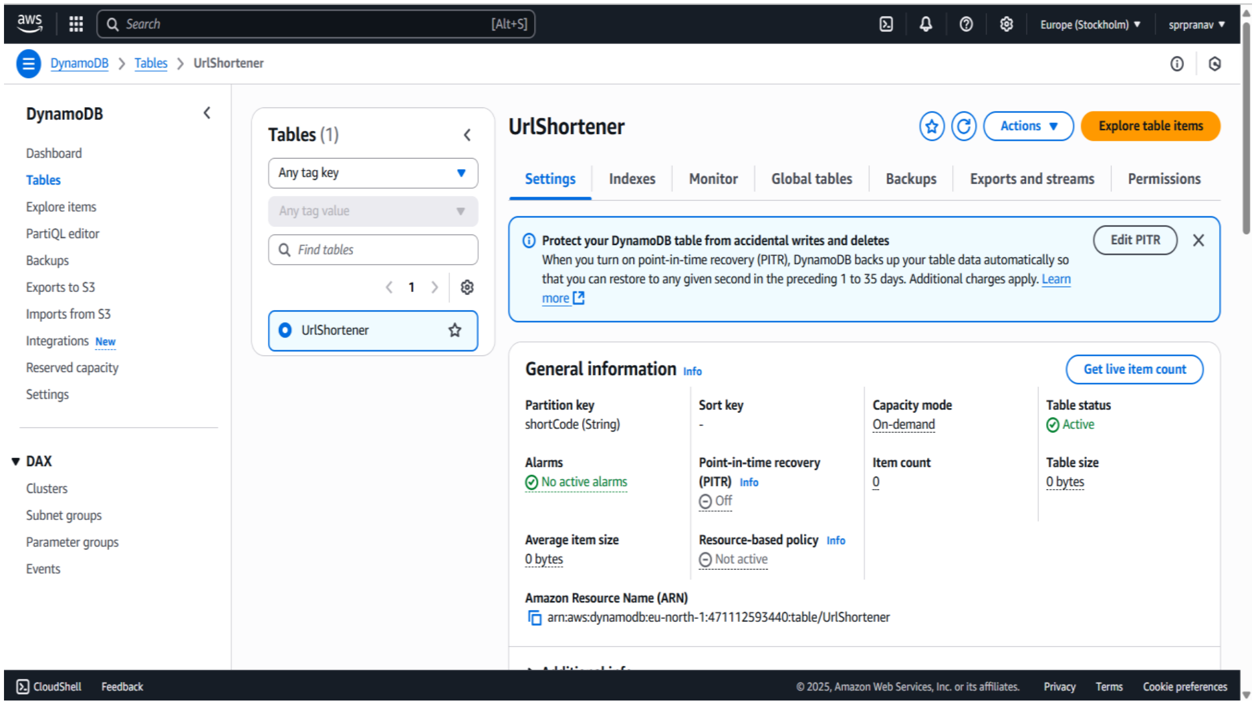
- Generates unique 6-character short codes for URLs.
- Stores mappings in DynamoDB for fast retrieval.
- Redirects short URLs to the original URLs using Lambda and API Gateway.
- Supports CORS for frontend API calls.
- Provides a clean, simple frontend interface for users.

---

4. Step-by-Step Procedure

Step 1: Create DynamoDB Table

- Table name: UrlShortener
- Primary key: shortCode (String)
- Used default settings eligible for AWS Free Tier.



UrlShortener

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

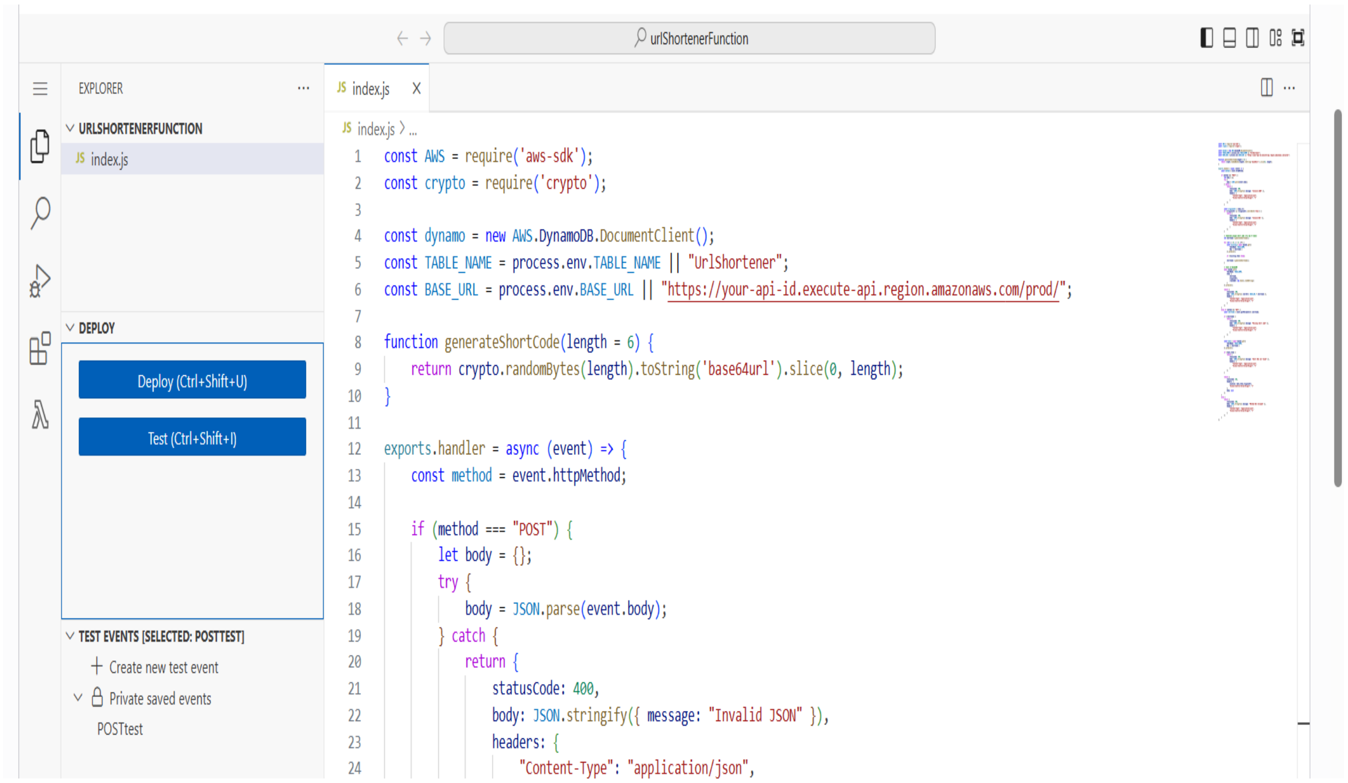
General information Info

Partition key: shortCode (String)	Sort key: -	Capacity mode: On-demand	Table status: Active
Alarms: No active alarms	Point-in-time recovery (PITR) Info: Off	Item count: 0	Table size: 0 bytes
Average item size: 0 bytes	Resource-based policy Info: Not active		

Amazon Resource Name (ARN)
arn:aws:dynamodb:eu-north-1:471112593440:table/UrlShortener

Step 2: Develop Lambda Function

- Language: Node.js 16.x runtime
- Lambda function handles:
  - **POST** requests to generate and store short URLs.
  - **GET** requests to redirect short URLs.
- Environment variables configured:
  - TABLE_NAME = UrlShortener
  - BASE_URL = https://{api-id}.execute-api.{region}.amazonaws.com/prod/



```
const AWS = require('aws-sdk');
const crypto = require('crypto');

const dynamo = new AWS.DynamoDB.DocumentClient();
const TABLE_NAME = process.env.TABLE_NAME || "UrlShortener";
const BASE_URL = process.env.BASE_URL || "https://your-api-id.execute-api.region.amazonaws.com/prod/";

function generateShortCode(length = 6) {
    return crypto.randomBytes(length).toString('base64url').slice(0, length);
}

exports.handler = async (event) => {
    const method = event.httpMethod;

    if (method === "POST") {
        let body = {};
        try {
            body = JSON.parse(event.body);
        } catch {
            return {
                statusCode: 400,
                body: JSON.stringify({ message: "Invalid JSON" }),
                headers: {
                    "Content-Type": "application/json",
```

index.js

```
const AWS = require('aws-sdk');

const crypto = require('crypto');


const dynamo = new AWS.DynamoDB.DocumentClient();

const TABLE_NAME = process.env.TABLE_NAME || "UrlShortener";

const BASE_URL = process.env.BASE_URL || "https://your-api-id.execute-
api.region.amazonaws.com/prod/";


function generateShortCode(length = 6) {

    return crypto.randomBytes(length).toString('base64url').slice(0, length);

}


exports.handler = async (event) => {

    const method = event.httpMethod;
```

```
if (method === "POST") {
    let body = {};
    try {
        body = JSON.parse(event.body);
    } catch {
        return {
            statusCode: 400,
            body: JSON.stringify({ message: "Invalid JSON" }),
            headers: {
                "Content-Type": "application/json",
                "Access-Control-Allow-Origin": "*"
            }
        };
    }

    const originalUrl = body.url;
    if (!originalUrl || !originalUrl.startsWith('http')) {
        return {
            statusCode: 400,
            body: JSON.stringify({ message: "Invalid URL" }),
            headers: {
                "Content-Type": "application/json",
                "Access-Control-Allow-Origin": "*"
            }
        };
    }

    // Generate unique short code (try max 5 times)
    let shortCode = generateShortCode();
```

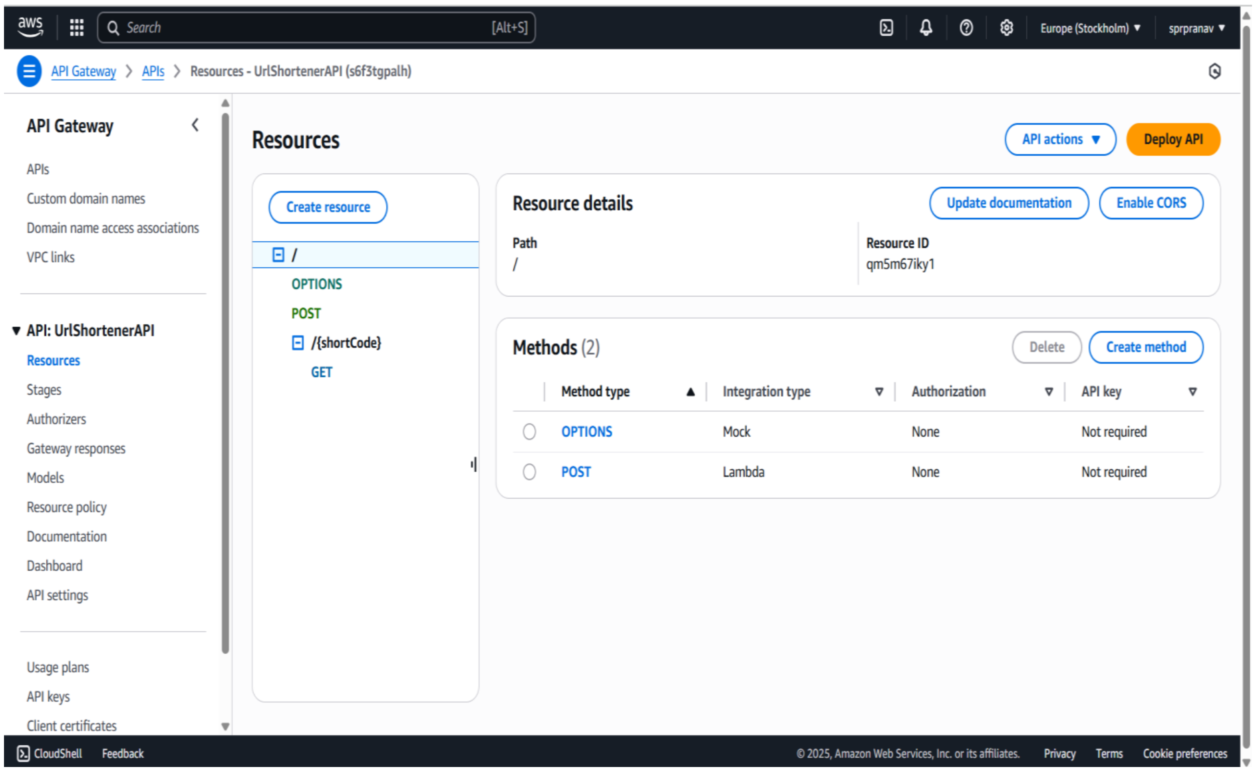
```
for (let i = 0; i < 5; i++) {
    const existing = await dynamo.get({
        TableName: TABLE_NAME,
        Key: { shortCode }
    }).promise();

    if (!existing.Item) break;

    shortCode = generateShortCode();
}

// Save in DynamoDB
await dynamo.put({
    TableName: TABLE_NAME,
    Item: {
        shortCode,
        originalUrl,
        createdAt: new Date().toISOString()
    }
}).promise();

return {
    statusCode: 200,
    body: JSON.stringify({ shortUrl: BASE_URL + shortCode }),
    headers: {
        "Content-Type": "application/json",
        "Access-Control-Allow-Origin": "*"
    }
};
```

```
    }
    else if (method === "GET") {
        const shortCode = event.pathParameters?.shortCode;

        if (!shortCode) {
            return {
                statusCode: 400,
                body: JSON.stringify({ message: "Missing short code" }),
                headers: {
                    "Content-Type": "application/json",
                    "Access-Control-Allow-Origin": "*"
                }
            };
        }

        const data = await dynamo.get({
            TableName: TABLE_NAME,
            Key: { shortCode }
        }).promise();

        if (!data.Item) {
            return {
                statusCode: 404,
                body: JSON.stringify({ message: "Short URL not found" }),
                headers: {
                    "Content-Type": "application/json",
                    "Access-Control-Allow-Origin": "*"
                }
            };
        }
```

```
            return {
                statusCode: 301,
                headers: {
                    Location: data.Item.originalUrl,
                    "Access-Control-Allow-Origin": "*"
                },
                body: null
            };
        }
        else {
            return {
                statusCode: 405,
                body: JSON.stringify({ message: "Method Not Allowed" }),
                headers: {
                    "Content-Type": "application/json",
                    "Access-Control-Allow-Origin": "*"
                }
            };
        }
    };
```

Step 3: Configure API Gateway

- Created REST API with:
  - POST / method linked to Lambda for URL shortening.
  - GET /{shortCode} method linked to Lambda for redirecting.
- Enabled Lambda Proxy integration.
- Configured CORS on both methods to allow browser calls.



Path: /
Resource ID: qm5m67iky1

Methods (2)

Method type	Integration type	Authorization	API key
OPTIONS	Mock	None	Not required
POST	Lambda	None	Not required

Step 4: Test API

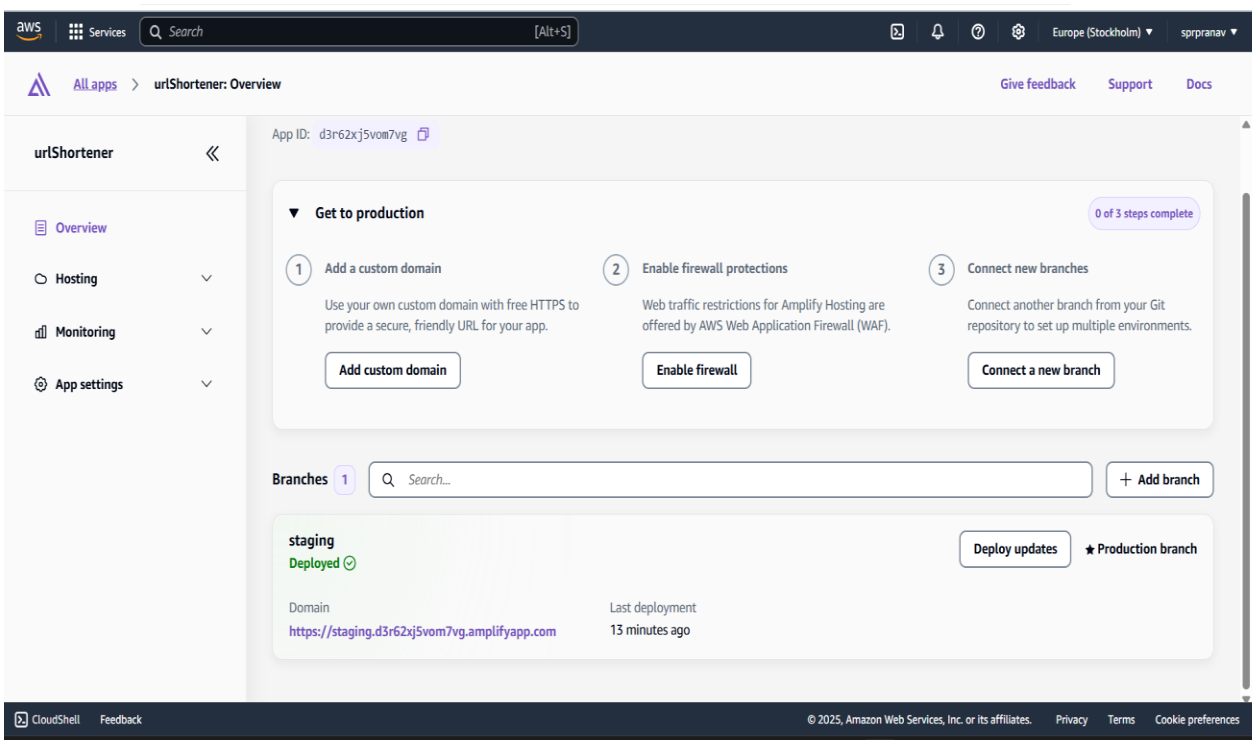
- Used Postman and browser to test POST (shortening) and GET (redirect) functionality.
- Verified correct redirection and error handling.

Step 5: Build Frontend

- Created simple HTML + JavaScript page (index.html).
- Connected frontend to API Gateway endpoint for URL shortening.
- Tested frontend locally.

Step 6: Deploy Frontend on AWS Amplify

- Created ZIP file of frontend files.
- Uploaded ZIP to AWS Amplify manual deploy.
- Accessed frontend via public Amplify URL.



App ID: d3r62xj5vom7vg

Get to production — 0 of 3 steps complete

1. Add a custom domain — Use your own custom domain with free HTTPS to provide a secure, friendly URL for your app. [Add custom domain]
2. Enable firewall protections — Web traffic restrictions for Amplify Hosting are offered by AWS Web Application Firewall (WAF). [Enable firewall]
3. Connect new branches — Connect another branch from your Git repository to set up multiple environments. [Connect a new branch]

Branches 1 — Search... — Add branch

staging — Deployed — Deploy updates — ★ Production branch

Domain: https://staging.d3r62xj5vom7vg.amplifyapp.com

Last deployment: 13 minutes ago

5. Challenges & Solutions

Challenge	Solution
CORS errors when frontend called API Gateway	Enabled CORS on API Gateway methods
AWS Lambda runtime & package errors	Updated Lambda runtime to Node.js 16.x, avoided unsupported imports
AWS S3 bucket public access restrictions	Switched to AWS Amplify hosting for frontend
Generating unique short codes without collision	Added retry logic to ensure unique short codes

---

6. Cost Considerations

- Project designed to run within AWS Free Tier limits.
- Services used: Lambda, API Gateway, DynamoDB, Amplify.
- Free tier covers most usage for development and small traffic.
- Monitoring AWS billing recommended to avoid unexpected costs.

---

7. Future Enhancements

- Add custom domains with HTTPS support.
- Implement user authentication and link management dashboard.
- Add analytics to track usage metrics.
- Improve frontend UI with React or other frameworks.
- Add expiration or usage limits for shortened URLs.

---

8. Conclusion

This project demonstrates how to build a scalable, serverless URL shortener using AWS services. It highlights serverless architecture benefits: cost-efficiency, scalability, and ease of deployment.
