



# AWS Project : Serverless web app

## Documentation

 By : Abouchiba Mohamed Yassir

To see the files used in this project, go to this link:  
[https://github.com/M-Yassir/AWS-Projects/tree/main/Serverless\\_Web\\_App](https://github.com/M-Yassir/AWS-Projects/tree/main/Serverless_Web_App)

# Introduction

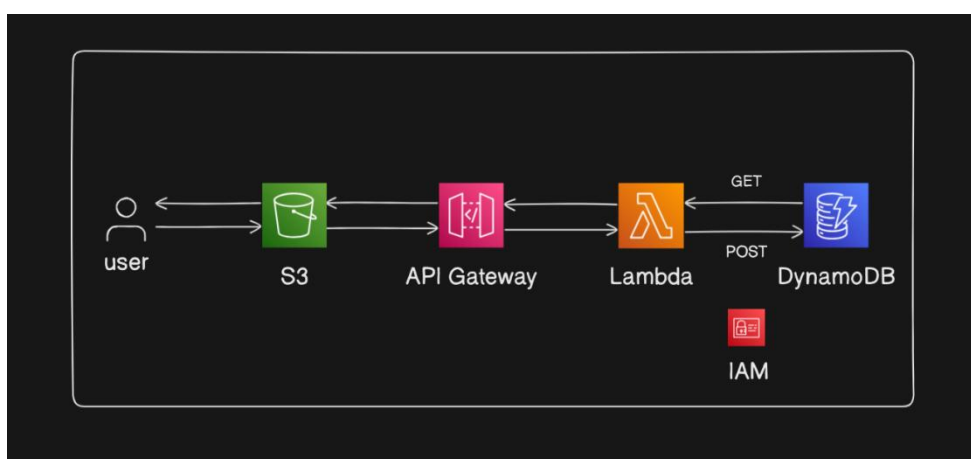
## Project Description

This project demonstrates the development of a serverless web application for user registration and management using Amazon Web Services (AWS). The application allows users to submit their details (username, password, email, and phone number) through a web form and retrieves all registered users for display. Built entirely within the AWS Free Tier, it showcases how serverless architecture can solve common challenges like scalability, cost-efficiency, and ease of deployment.

## Key Features

- User Registration: A web form to collect and store user data.
- User Retrieval: A button to fetch and display all registered users.
- Serverless Architecture: Uses AWS Lambda, API Gateway, and DynamoDB for backend functionality.
- Static Website Hosting: The frontend is hosted on Amazon S3.

## AWS architecture diagram



## Step-by-Step implementation

- NB:** - All the necessary code is on the repository given above.  
- The creation of the index.html file isn't covered here as it isn't our topic.

### Step1: Creating a DynamoDB table and Lambda functions:

First we go to DynamoDB service then we click on '*Create table*'

We fill in the necessary informations like it is shown below:

#### Create table

**Table details** info  
DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.

**Table name**  
This will be used to identify your table.  
  
Between 3 and 255 characters, containing only letters, numbers, underscores (\_), hyphens (-), and periods (.).

**Partition key**  
The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.  
   
1 to 255 characters and case sensitive.

**Sort key - optional**  
You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.  
   
1 to 255 characters and case sensitive.

**Table settings**

☒ **Default settings**  
The fastest way to create your table. You can modify most of these settings after your table has been created. To modify these settings now, choose 'Customize settings'.

☐ **Customize settings**  
Use these advanced features to make DynamoDB work better for your needs.

**Default table settings**  
These are the default settings for your new table. You can change some of these settings after creating the table.

Setting	Value	Editable after creation
Table class	DynamoDB Standard	Yes
Capacity mode	On-demand	Yes
Maximum read capacity units	-	Yes
Maximum write capacity units	-	Yes
Local secondary indexes	-	No
Global secondary indexes	-	Yes
Encryption key management	Owned by Amazon DynamoDB	Yes
Deletion protection	Off	Yes
Resource-based policy	Not active	Yes

**Tags**  
Tags are pairs of keys and optional values, that you can assign to AWS resources. You can use tags to control access to your resources or track your AWS spending.  
No tags are associated with the resource.  
  
You can add 50 more tags.

We click on '*Create table*'

We go to IAM to create a role to let the lambda function access DynamoDB:



Then, we go to the Lambda service and we create a function to get user's informations:

**Create function** [Info](#)

Choose one of the following options to create your function.

☒ **Author from scratch**  
Start with a simple Hello World example.

☐ **Use a blueprint**  
Build a Lambda application from sample code and configuration presets for common use cases.

☐ **Container image**  
Select a container image to deploy for your function.

---

**Basic information**

**Function name**  
Enter a name that describes the purpose of your function.  
  
Function name must be 1 to 64 characters, must be unique to the Region, and can't include spaces. Valid characters are a-z, A-Z, 0-9, hyphens (-), and underscores (\_).

**Runtime** [Info](#)  
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.

**Architecture** [Info](#)  
Choose the instruction set architecture you want for your function code.  
☒ x86\_64  
☐ arm64

**Permissions** [Info](#)  
By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

▼ **Change default execution role**

**Execution role**  
Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).

☐ Create a new role with basic Lambda permissions

☒ Use an existing role

☐ Create a new role from AWS policy templates

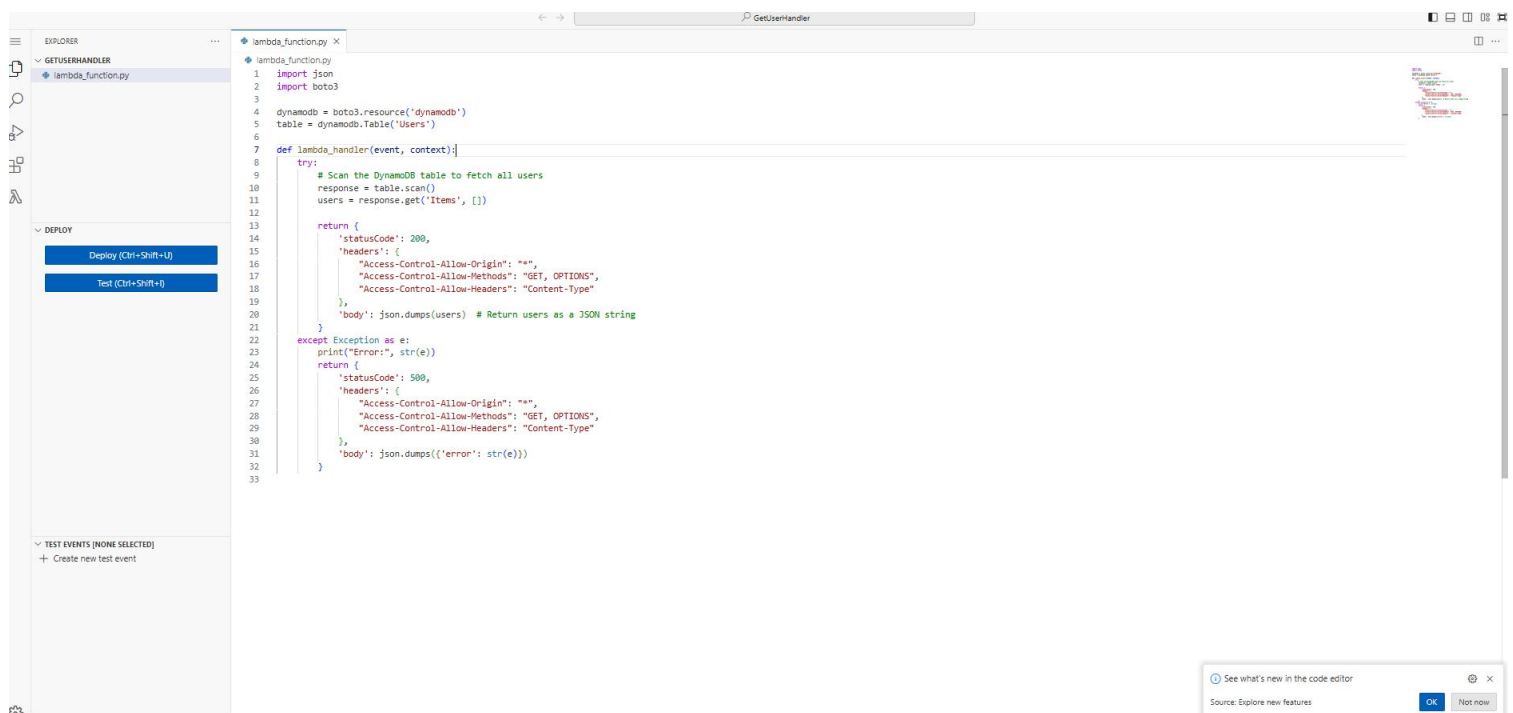
**Existing role**  
Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.

[View the lambdaDynamoDB-role role](#) on the IAM console.

► **Additional Configurations**  
Use additional configurations to set up code signing, function URL, tags, and Amazon VPC access for your function.

[Cancel](#) [Create function](#)

We click on 'create function', after that, we go to 'code source' and we type the following python code (the code is on the repository):



```
1 import json
2 import boto3
3
4 dynamodb = boto3.resource('dynamodb')
5 table = dynamodb.Table('Users')
6
7 def lambda_handler(event, context):
8     try:
9         # Scan the DynamoDB table to fetch all users
10        response = table.scan()
11        users = response.get('Items', [])
12
13        return {
14            'statusCode': 200,
15            'headers': {
16                "Access-Control-Allow-Origin": "*",
17                "Access-Control-Allow-Methods": "GET, OPTIONS",
18                "Access-Control-Allow-Headers": "Content-Type"
19            },
20            'body': json.dumps(users) # Return users as a JSON string
21        }
22    except Exception as e:
23        print("Error:", str(e))
24        return {
25            'statusCode': 500,
26            'headers': {
27                "Access-Control-Allow-Origin": "*",
28                "Access-Control-Allow-Methods": "GET, OPTIONS",
29                "Access-Control-Allow-Headers": "Content-Type"
30            },
31            'body': json.dumps({'error': str(e)})
32        }
33
```

We create another function to register users, the same steps are here:

Choose one of the following options to create your function.

☒ **Author from scratch**  
Start with a simple Hello World example.

☐ **Use a blueprint**  
Build a Lambda application from sample code and configuration presets for common use cases.

☐ **Container image**  
Select a container image to deploy for your function.

---

### Basic information

**Function name**  
Enter a name that describes the purpose of your function.

RegisterUserHandler

Function name must be 1 to 64 characters, must be unique to the Region, and can't include spaces. Valid characters are a-z, A-Z, 0-9, hyphens (-), and underscores (\_).

**Runtime** Info  
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.

Python 3.13

**Architecture** Info  
Choose the instruction set architecture you want for your function code.

☒ x86\_64  
☐ arm64

**Permissions** Info  
By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

▼ **Change default execution role**

**Execution role**  
Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).

☐ Create a new role with basic Lambda permissions  
☒ Use an existing role  
☐ Create a new role from AWS policy templates

**Existing role**  
Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.

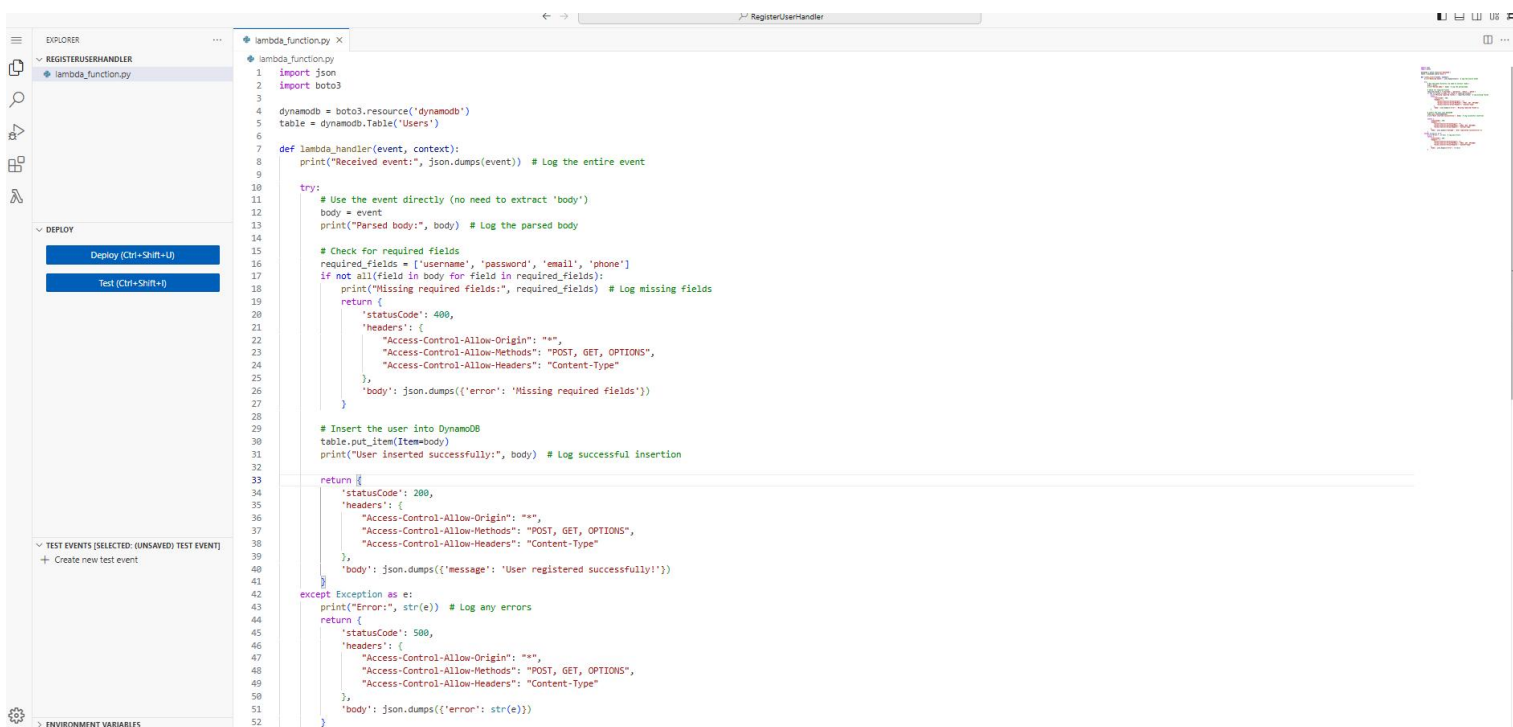
lambdaDynamoDB-role

[View the lambdaDynamoDB-role on the IAM console.](#)

► **Additional Configurations**  
Use additional configurations to set up code signing, function URL, tags, and Amazon VPC access for your function.

Cancel **Create function**

We click on 'create function', after that, we go to 'code source' and we type the following python code (the code is on the repository):



```
1 import json
2 import boto3
3
4 dynamodb = boto3.resource('dynamodb')
5 table = dynamodb.Table('Users')
6
7 def lambda_handler(event, context):
8     print("Received event: ", json.dumps(event)) # Log the entire event
9
10     try:
11         # Use the event directly (no need to extract 'body')
12         body = event
13         print("Parsed body: ", body) # Log the parsed body
14
15         # Check for required fields
16         required_fields = ['username', 'password', 'email', 'phone']
17         if not all(field in body for field in required_fields):
18             print("Missing required fields: ", required_fields) # Log missing fields
19             return {
20                 'statusCode': 400,
21                 'headers': {
22                     "Access-Control-Allow-Origin": "*",
23                     "Access-Control-Allow-Methods": "POST, GET, OPTIONS",
24                     "Access-Control-Allow-Headers": "Content-Type"
25                 },
26                 'body': json.dumps({'error': 'Missing required fields'})
27             }
28
29         # Insert the user into DynamoDB
30         table.put_item(Item=body)
31         print("User inserted successfully: ", body) # Log successful insertion
32
33         return {
34             'statusCode': 200,
35             'headers': {
36                 "Access-Control-Allow-Origin": "*",
37                 "Access-Control-Allow-Methods": "POST, GET, OPTIONS",
38                 "Access-Control-Allow-Headers": "Content-Type"
39             },
40             'body': json.dumps({'message': 'User registered successfully!'})
41         }
42     except Exception as e:
43         print("Error: ", str(e)) # Log any errors
44         return {
45             'statusCode': 500,
46             'headers': {
47                 "Access-Control-Allow-Origin": "*",
48                 "Access-Control-Allow-Methods": "POST, GET, OPTIONS",
49                 "Access-Control-Allow-Headers": "Content-Type"
50             },
51             'body': json.dumps({'error': str(e)})
52         }
```



## Step2: Creating an API to trigger our lambda function using API gateway:

First we go to API gateway service, and we create a REST API

Choose an API type [info](#)

**HTTP API**

Build low-latency and cost-effective REST APIs with built-in features such as OIDC and OAuth2, and native CORS support.

**Works with the following:**  
Lambda, HTTP backends

[Import](#) [Build](#)

**WebSocket API**

Build a WebSocket API using persistent connections for real-time use cases such as chat applications or dashboards.

**Works with the following:**  
Lambda, HTTP, AWS Services

[Build](#)

**REST API**

Develop a REST API where you gain complete control over the request and response along with API management capabilities.

**Works with the following:**  
Lambda, HTTP, AWS Services

[Import](#) [Build](#)

### Create REST API <sup>1</sup> [info](#)

**API details**

☒ **New API**  
Create a new REST API.

☐ **Clone existing API**  
Create a copy of an API in this AWS account.

☐ **Import API**  
Import an API from an OpenAPI definition.

☐ **Example API**  
Learn about API Gateway with an example API.

**API name**

**Description - optional**

**API endpoint type**  
Regional APIs are deployed in the current AWS Region. Edge-optimized APIs route requests to the nearest CloudFront Point of Presence. Private APIs are only accessible from VPCs.

[Cancel](#)

[Create API](#)

We create a GET and POST methods

### Create method

**Method details**

**Method type**

**Integration type**

☒ **Lambda function**  
Integrate your API with a Lambda function.

☐ **HTTP**  
Integrate with an existing HTTP endpoint.

☐ **Mock**  
Generate a response based on API Gateway mappings and transformations.

☐ **AWS service**  
Integrate with an AWS Service.

☐ **VPC link**  
Integrate with a resource that isn't accessible over the public internet.

☒ **Lambda proxy integration**  
Send the request to your Lambda function as a structured event.

**Lambda function**  
Provide the Lambda function name or alias. You can also provide an ARN from another account.

☒ **Grant API Gateway permission to invoke your Lambda function.**  
To turn off, update the function's resource policy yourself, or provide an invoke role that API Gateway uses to invoke your function.

**Integration timeout** [info](#)  
By default, you can enter an integration timeout of 30 - 29,000 milliseconds. You can use Service Quotas to raise the integration timeout to greater than 29,000 ms.

**Method request settings**

**URL query string parameters**

**HTTP request headers**

**Request body**

[Cancel](#) [Create method](#)

## Create method

**Method details**

Method type

POST

Integration type

☒ **Lambda function**  
Integrate your API with a Lambda function.

☐ **HTTP**  
Integrate with an existing HTTP endpoint.

☐ **Mock**  
Generate a response based on API Gateway mappings and transformations.

☐ **AWS service**  
Integrate with an AWS Service.

☐ **VPC link**  
Integrate with a resource that isn't accessible over the public internet.

☒ **Lambda proxy integration**  
Send the request to your Lambda function as a structured event.

Lambda function

Provide the Lambda function name or alias. You can also provide an ARN from another account.

us-east-1

arn:aws:lambda:us-east-1:577638379285:function:RegisterUserHand

Integration timeout

Info

By default, you can enter an integration timeout of 50 - 29,000 milliseconds. You can use Service Quotas to raise the integration timeout to greater than 29,000 ms.

29000

Method request settings

URL query string parameters

HTTP request headers

Request body

Cancel

Create method

Then we deploy our API:

Resources

Create resource

GET

POST

Deploy API

Create or select a stage where your API will be deployed. You can use the deployment history to revert or change the active deployment for a stage. [Learn more](#)

Stage

\*New stage\*

Stage name

prod

☒ A new stage will be created with the default settings. Edit your stage settings on the **Stage** page.

Deployment description

Cancel

Deploy

## Enable CORS

### CORS settings

To allow requests from scripts running in the browser, configure cross-origin resource sharing (CORS) for your API. When you save your configuration, API Gateway replaces any existing CORS settings with your new configuration.

#### Gateway responses

API Gateway will configure CORS for the selected gateway responses.

☐ Default 4XX

☐ Default 5XX

#### Access-Control-Allow-Methods

☒ GET

☒ OPTIONS

☒ POST

#### Access-Control-Allow-Headers

API Gateway will configure CORS for the selected gateway responses.

Content-Type,X-Amz-Date,Authorization,X-Api-Key,X-Amz-Security-Token

#### Access-Control-Allow-Origin

Enter an origin that can access the resource. Use a wildcard "\*" to allow any origin to access the resource.

\*

#### Additional settings

Cancel

Save

!! Also enable CORS:



After this we should get an invoke URL

Stages

Stage actions Create stage

prod

Stage details Info Edit

Stage name: prod

Cache cluster info: Inactive

Default method-level caching: Inactive

Rate info: 10000

Burst info: 5000

Web ACL: -

Client certificate: -

Invoke URL: <https://qb1hfje9f1.execute-api.us-east-1.amazonaws.com/prod>

Active deployment: xunhm on February 02, 2025, 18:04 (UTC+01:00)

Logs and tracing Info Edit

CloudWatch logs: Inactive

X-Ray tracing: Inactive

Custom access logging: Inactive

Detailed metrics: Inactive

Data tracing: Inactive

We'll paste in the variable 'apiEndpoint' in the **index.html** file before using S3.

## Step3: Creating an S3 static web hosting:

We create an S3 bucket and enable static web hosting like it is shown below:

### Create bucket Info

Buckets are containers for data stored in S3.

#### General configuration

##### AWS Region

US East (N. Virginia) us-east-1

##### Bucket type Info

###### General purpose

Recommended for most use cases and access patterns. General purpose buckets are the original S3 bucket type. They allow a mix of storage classes that redundantly store objects across multiple Availability Zones.

###### Directory

Recommended for low-latency use cases. These buckets use only the S3 Express One Zone storage class, which provides faster processing of data within a single Availability Zone.

##### Bucket name Info

mstaticwebbucket777

Bucket name must be unique within the global namespace and follow the bucket naming rules. [See rules for bucket naming](#)

##### Copy settings from existing bucket - optional

Only the bucket settings in the following configuration are copied.

Choose bucket

Format: s3://bucket/prefix

#### Object Ownership Info

Control ownership of objects written to this bucket from other AWS accounts and the use of access control lists (ACLs). Object ownership determines who can specify access to objects.

##### ACLs disabled (recommended)

All objects in this bucket are owned by this account. Access to this bucket and its objects is specified using only policies.

##### ACLs enabled

Objects in this bucket can be owned by other AWS accounts. Access to this bucket and its objects can be specified using ACLs.

##### Object Ownership

Bucket owner enforced

#### Block Public Access settings for this bucket

Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies, or all. In order to ensure that public access to this bucket and its objects is blocked, turn on Block all public access. These settings apply only to this bucket and its access points. AWS recommends that you turn on Block all public access, but before applying any of these settings, ensure that your applications will work correctly without public access. If you require some level of public access to this bucket or objects within, you can customize the individual settings below to suit your specific storage use cases. [Learn more](#)

##### Block all public access

Turning this setting on is the same as turning on all four settings below. Each of the following settings are independent of one another.

- ☐ Block public access to buckets and objects granted through new access control lists (ACLs)  
S3 will block public access permissions applied to newly added buckets or objects, and prevent the creation of new public access ACLs for existing buckets and objects. This setting doesn't change any existing permissions that allow public access to S3 resources using ACLs.
- ☐ Block public access to buckets and objects granted through any access control lists (ACLs)  
S3 will ignore all ACLs that grant public access to buckets and objects.
- ☐ Block public access to buckets and objects granted through new public bucket or access point policies  
S3 will block new bucket and access point policies that grant public access to buckets and objects. This setting doesn't change any existing policies that allow public access to S3 resources.
- ☐ Block public and cross-account access to buckets and objects through any public bucket or access point policies  
S3 will ignore public and cross-account access for buckets or access points with policies that grant public access to buckets and objects.

##### Turning off block all public access might result in this bucket and the objects within becoming public

AWS recommends that you turn on block all public access, unless public access is required for specific and verified use cases such as static website hosting.

- ☐ I acknowledge that the current settings might result in this bucket and the objects within becoming public.

### Bucket Versioning

Versioning is a means of keeping multiple variants of an object in the same bucket. You can use versioning to preserve, retrieve, and restore every version of every object stored in your Amazon S3 bucket. With versioning, you can easily recover from both unintended user actions and application failures. [Learn more](#)

#### Bucket Versioning

- ☒ Disable  
☐ Enable

### Tags - optional (0)

You can use bucket tags to track storage costs and organize buckets. [Learn more](#)

No tags associated with this bucket.

[Add tag](#)

### Default encryption [Info](#)

Server-side encryption is automatically applied to new objects stored in this bucket.

#### Encryption type [Info](#)

- ☒ Server-side encryption with Amazon S3 managed keys (SSE-S3)  
☐ Server-side encryption with AWS Key Management Service keys (SSE-KMS)  
☐ Dual-layer server-side encryption with AWS Key Management Service keys (DSSE-KMS)  
Secure your objects with two separate layers of encryption. For details on pricing, see DSSE-KMS pricing on the [Storage](#) tab of the [Amazon S3 pricing page](#).

#### Bucket Key

Using an S3 Bucket Key for SSE-KMS reduces encryption costs by lowering calls to AWS KMS. S3 Bucket Keys aren't supported for DSSE-KMS. [Learn more](#)

- ☐ Disable  
☒ Enable

### ► Advanced settings

After creating the bucket, you can upload files and folders to the bucket, and configure additional bucket settings.

[Cancel](#)

[Create bucket](#)

We enable static web hosting:

### Edit static website hosting [Info](#)

#### Static website hosting

Use this bucket to host a website or redirect requests. [Learn more](#)

##### Static website hosting

- ☐ Disable  
☒ Enable

##### Hosting type

- ☒ Host a static website  
Use the bucket endpoint as the web address. [Learn more](#)  
☐ Redirect requests for an object  
Redirect requests to another bucket or domain. [Learn more](#)

For your customers to access content at the website endpoint, you must make all your content publicly readable. To do so, you can edit the S3 Block Public Access settings for the bucket. For more information, see [Using Amazon S3 Block Public Access](#)

##### Index document

Specify the home or default page of the website.

index.html

##### Error document - optional

This is returned when an error occurs.

error.html

##### Redirection rules - optional

Redirection rules, written in JSON, automatically redirect webpage requests for specific content. [Learn more](#)

1

JSON Ln 1, Col 1 Errors: 0 Warnings: 0

[Cancel](#)

[Save changes](#)

and we give it the following policy:

## Edit bucket policy [Info](#)

**Bucket policy**[Policy examples](#)[Policy generator](#)

The bucket policy, written in JSON, provides access to the objects stored in the bucket. Bucket policies don't apply to objects owned by other accounts. [Learn more](#)

**Bucket ARN**  
`arn:aws:s3::mstaticwebbucket777`

**Policy**

```
1 {
2   "Version": "2012-10-17",
3   "Statement": [
4     {
5       "Sid": "PublicReadGetObject",
6       "Effect": "Allow",
7       "Principal": "*",
8       "Action": "s3:GetObject",
9       "Resource": "arn:aws:s3::mstaticwebbucket777/*"
10    }
11  ]
12 }
```

**Edit statement**

Select a statement

Select an existing statement in the policy or add a new statement.

[+ Add new statement](#)

After uploading the **index.html** file, we go now to get the link in the properties section:

## mstaticwebbucket777 [Info](#)

[Objects](#) [Metadata](#) [Properties](#) [Permissions](#) [Metrics](#) [Management](#) [Access Points](#)

**Objects (1)** [Copy S3 URI](#) [Copy URL](#) [Download](#) [Open](#) [Delete](#) [Actions](#) [Create folder](#) [Upload](#)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

Find objects by prefix

<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	<a href="#">index.html</a>	html	February 3, 2025, 12:21:32 (UTC+01:00)	7.2 KB	Standard

The link is located on this section, we click on it

[Amazon S3](#) > [Buckets](#) > mstaticwebbucket777

Use an accelerated endpoint for faster data transfers. [Learn more](#)

**Transfer acceleration**  
Disabled

**Object Lock** [Edit](#)  
Store objects using a write-once-read-many (WORM) model to help you prevent objects from being deleted or overwritten for a fixed amount of time or indefinitely. Object Lock works only in versioned buckets. [Learn more](#)

**Object Lock**  
Disabled

**Requester pays** [Edit](#)  
When enabled, the requester pays for requests and data transfer costs, and anonymous access to this bucket is disabled. [Learn more](#)

**Requester pays**  
Disabled

**Static website hosting** [Edit](#)  
Use this bucket to host a website or redirect requests. [Learn more](#)

**We recommend using AWS Amplify Hosting for static website hosting**  
Deploy a fast, secure, and reliable website quickly with AWS Amplify Hosting. Learn more about [Amplify Hosting](#) or [View your existing Amplify apps](#)

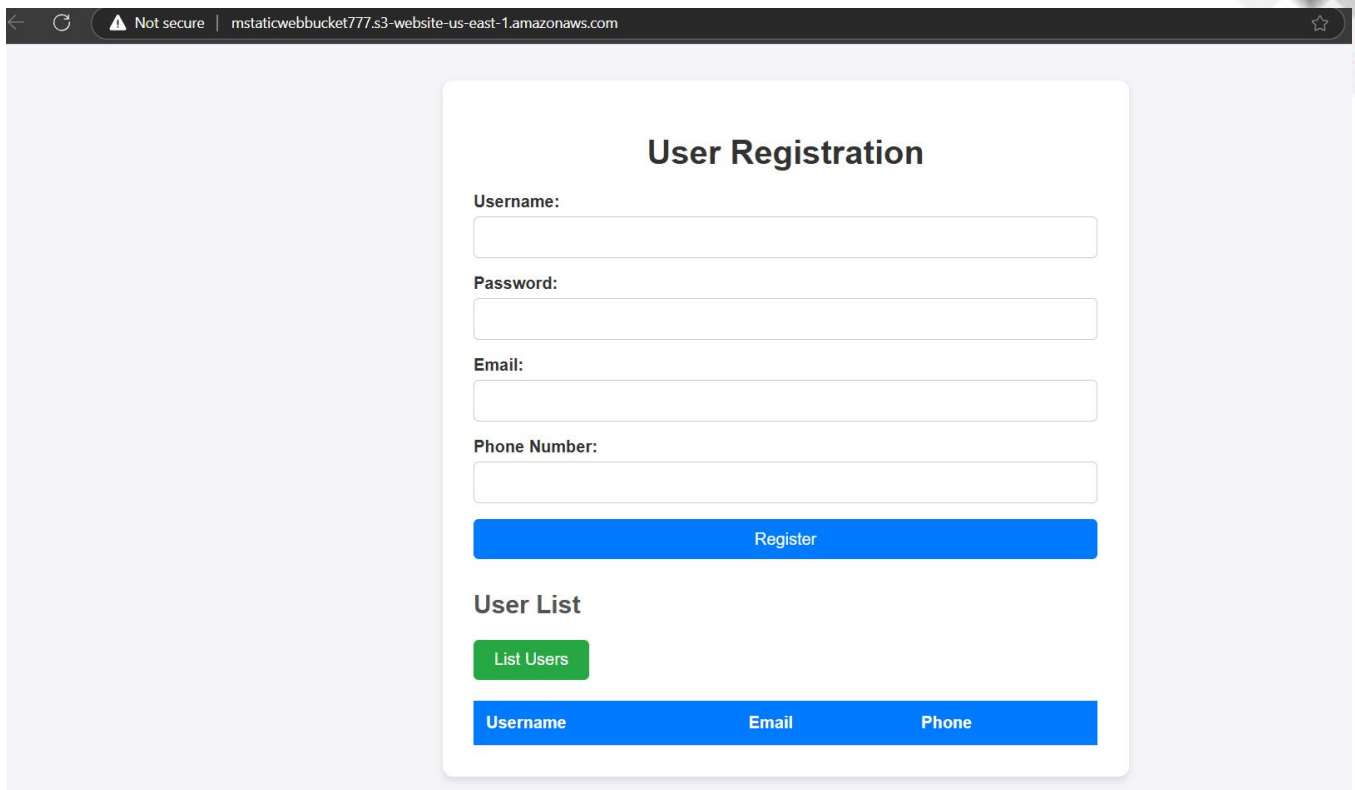
[Create Amplify app](#)

**S3 static website hosting**  
Enabled

**Hosting type**  
Bucket hosting

**Bucket website endpoint**  
When you configure your bucket as a static website, the website is available at the AWS Region-specific website endpoint of the bucket. [Learn more](#)  
<http://mstaticwebbucket777.s3-website-us-east-1.amazonaws.com>

Now, we should get this:



Not secure | mstaticwebbucket777.s3-website-us-east-1.amazonaws.com

### User Registration

Username:

Password:

Email:

Phone Number:

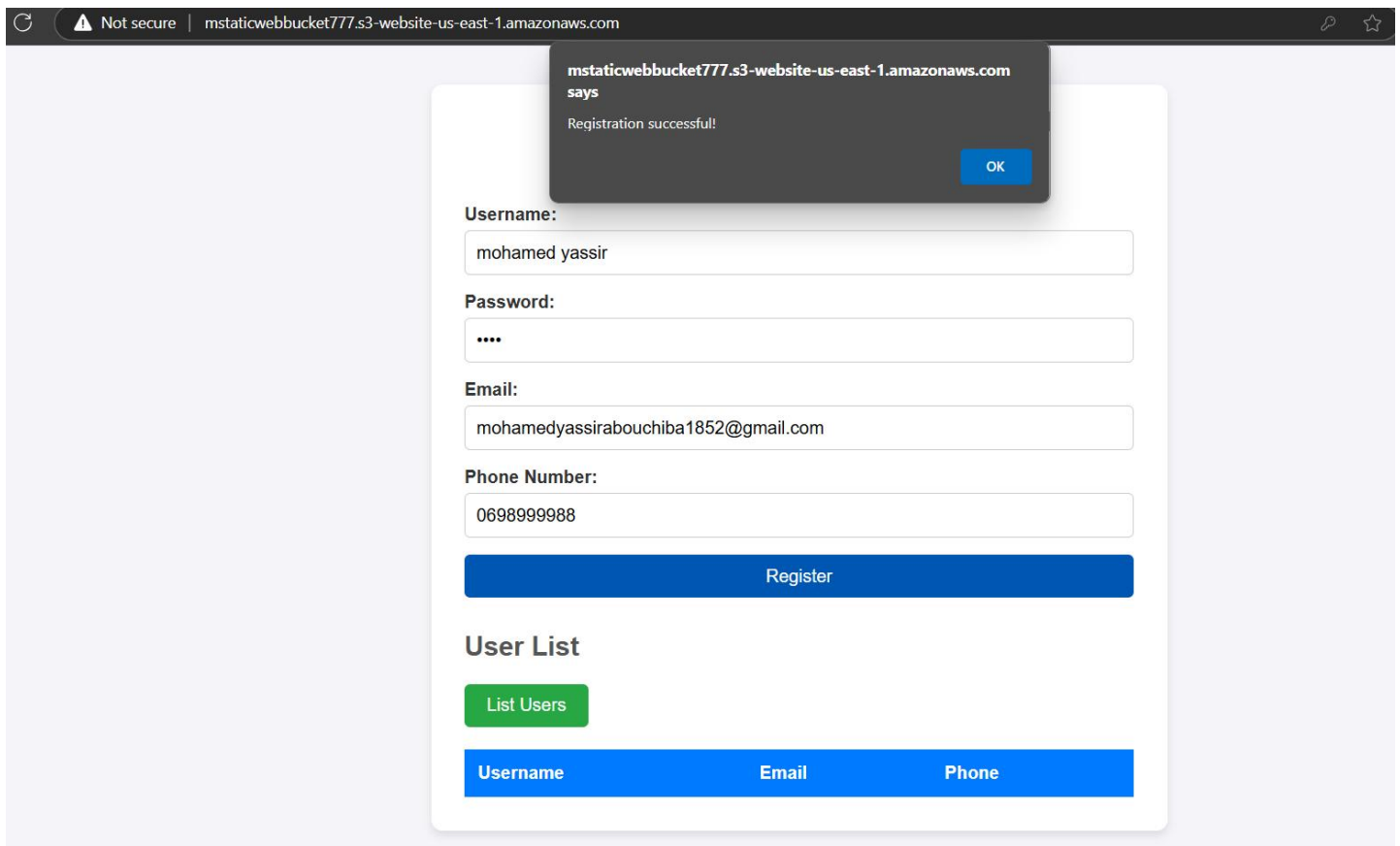
Register

### User List

List Users

Username	Email	Phone
----------	-------	-------

If we try to register a user, it shows the following:



Not secure | mstaticwebbucket777.s3-website-us-east-1.amazonaws.com

mstaticwebbucket777.s3-website-us-east-1.amazonaws.com says  
Registration successful!  
OK

Username:  
mohamed yassir

Password:  
....

Email:  
mohamedyassirabouchiba1852@gmail.com

Phone Number:  
0698999988

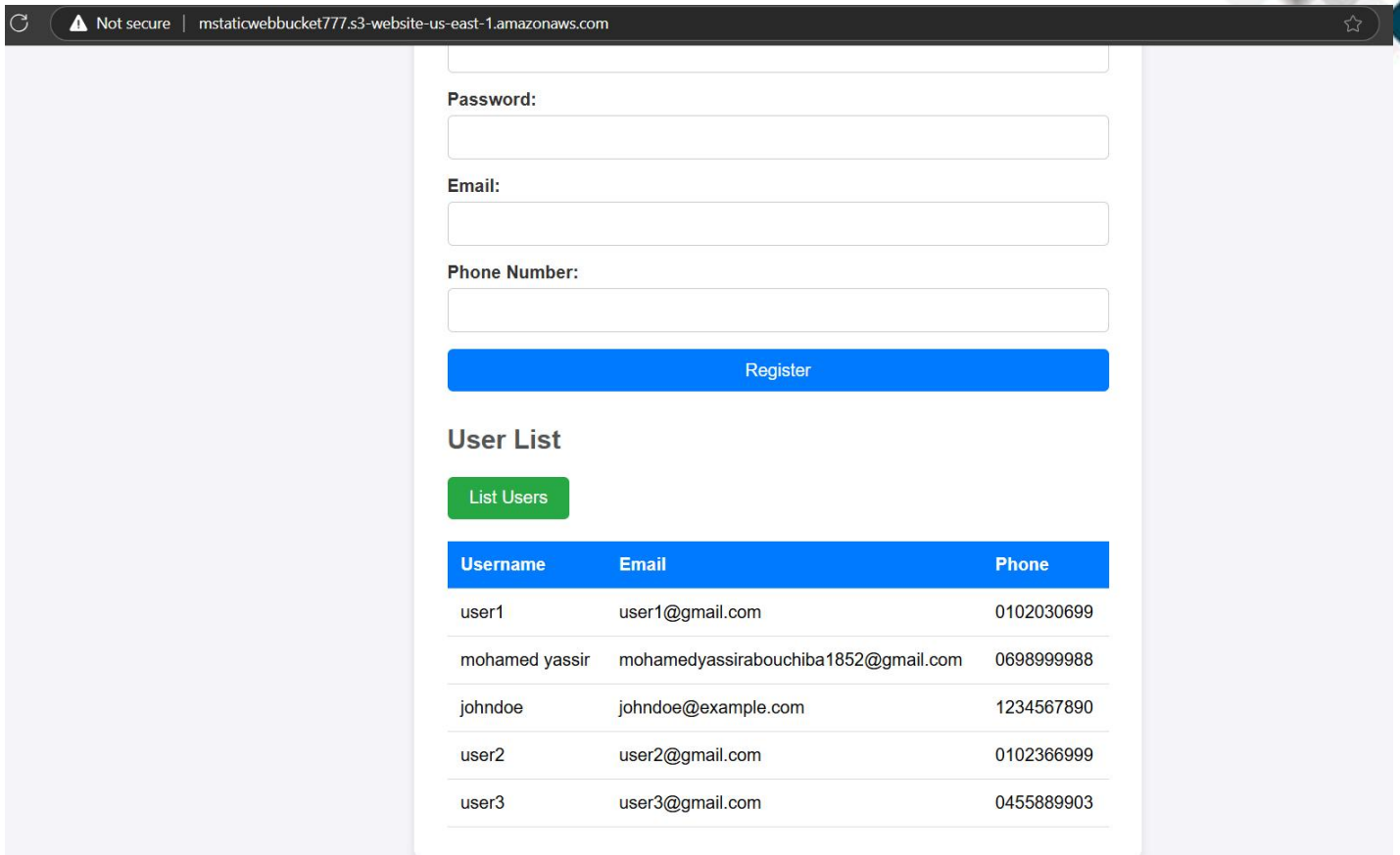
Register

### User List

List Users

Username	Email	Phone
----------	-------	-------

If we try now to list Users, it shows the following:



The screenshot shows a web browser window with the address bar displaying "mstaticwebbucket777.s3-website-us-east-1.amazonaws.com". The page contains a registration form with fields for Password, Email, and Phone Number, followed by a blue "Register" button. Below the form is a section titled "User List" with a green "List Users" button. A table displays the list of users with columns for Username, Email, and Phone.

Username	Email	Phone
user1	user1@gmail.com	0102030699
mohamed yassir	mohamedyassirabouchiba1852@gmail.com	0698999988
johndoe	johndoe@example.com	1234567890
user2	user2@gmail.com	0102366999
user3	user3@gmail.com	0455889903

This means that the operation of reading and writing to the DynamoDB was successful.

## Conclusion

This project has been an insightful journey into building a fully serverless web application using AWS services. By leveraging AWS Lambda, API Gateway, DynamoDB, and S3, I successfully implemented a scalable and cost-efficient user registration and management system within the AWS Free Tier.

### Challenges Faced & Lessons Learned:

#### 1) CORS Configuration in API Gateway & Lambda

One of the major challenges was handling CORS (Cross-Origin Resource Sharing) errors when making API requests from the frontend to API Gateway. I learned how to properly configure CORS headers in API Gateway and Lambda responses to allow secure communication between the frontend and backend.

#### 2) Setting Up API Gateway & Lambda Integration

Initially, API Gateway didn't route requests correctly to the appropriate Lambda functions.

I learned how to properly configure method integrations in API Gateway, ensuring that GET and POST requests were correctly processed by their respective Lambda functions.

#### 3) S3 Bucket Permissions & Web Hosting

Hosting the frontend on Amazon S3 required configuring the right public access permissions and an S3 Bucket Policy to allow users to access the web application.

I learned how to correctly set up S3 static website hosting, adjust bucket policies, and ensure that the "Block Public Access" settings didn't interfere with serving the frontend.

### Final Takeaways:

Through this project, I gained hands-on experience in serverless web development and learned how AWS services work together to create a fully functional application. Some key skills I developed include:

- ✓ Designing and deploying REST APIs with API Gateway & Lambda
- ✓ Configuring CORS policies for secure cross-origin requests
- ✓ Managing DynamoDB for efficient data storage and retrieval
- ✓ Setting up an S3-hosted static website with proper security policies

This project has deepened my understanding of serverless architecture and its advantages in terms of scalability, cost-efficiency, and ease of deployment. Moving forward, I am excited to apply these skills to build even more advanced cloud-based solutions.