Three Tier Web App on AWS

Three-tier architecture is a Web application architecture that organizes applications into three logical and physical computing tiers.

- **Presentation Layer →** Shows the UI of Application.
- **Logic Layer →** How the Data will be Processed.
- **Data Layer →** How the Data will be Stored.

In our web Application for we are going to use **S3, CloudFront** for Presentation Layer, **API Gateway and Lambda Function** for Logical Tier and **DynamoDB** for Data Tier.

Let's Start the Project →

Step 1: We have created three files.

- **index.html** – File contains the basic skeleton of our website.
- **style.css** –This file defines the style to HTML tags.
- **script.js** –This File adds dynamic behaviour to our website.

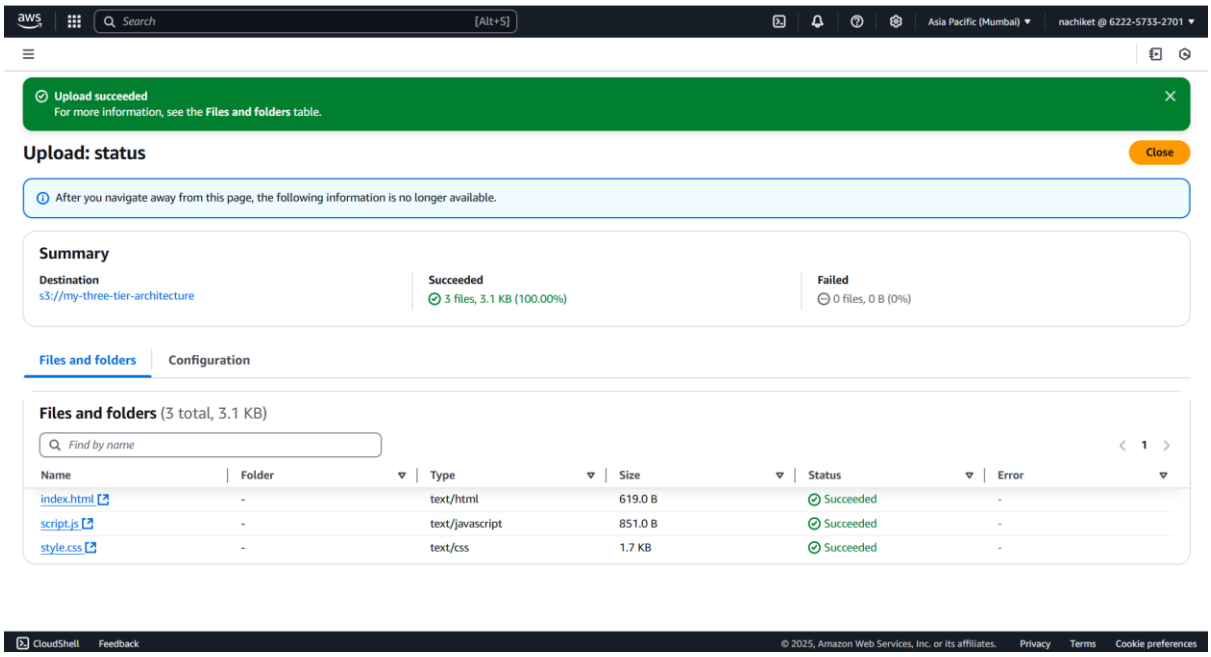
We are logged in as a **IAM user**.

Step 2: Let's Setup the Presentation Tier .

- Create an S3 Bucket and upload files in it.

What is S3 Bucket?

➔ Amazon S3 is an object storage service that stores data as objects within buckets. An object is a file and any metadata that describes the file. A bucket is a container for objects.



Step 3: Setup the CloudFront

What is CloudFront and how CloudFront Speed up delivery?

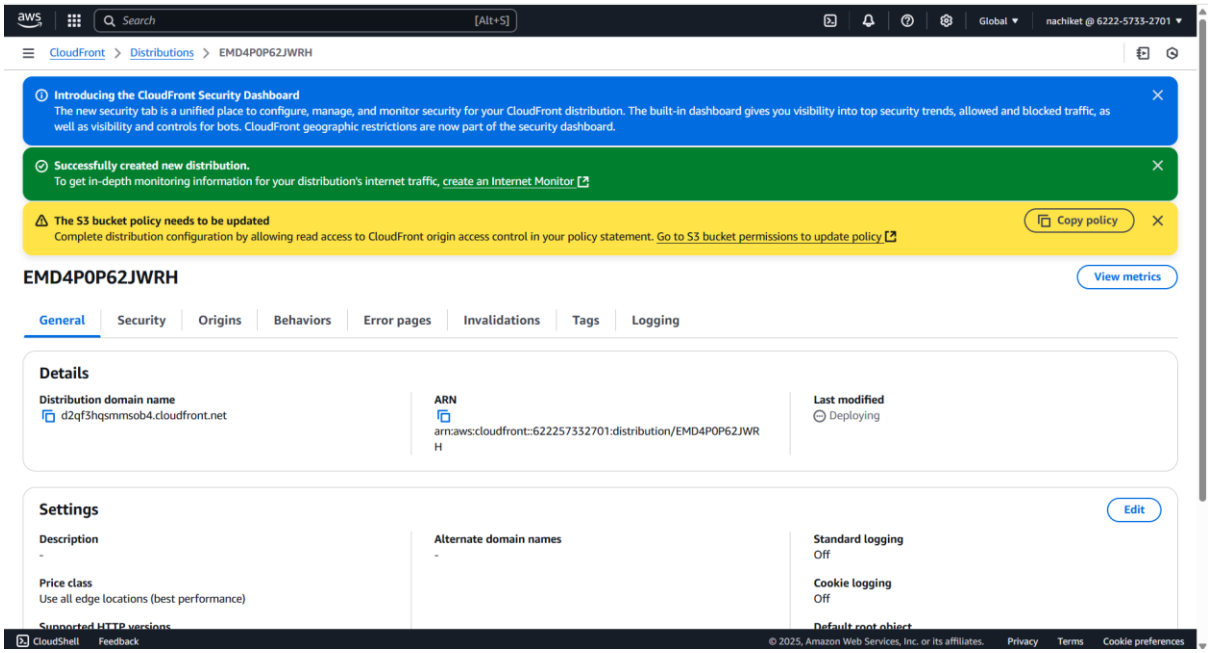
➔ CloudFront caches content in globally distributed edge locations to serve users from the nearest point, reducing latency and improving speed.

During Setup, we have one option called as Origin Access Control Settings. What it is?

➔ When serving content from an **Amazon S3 bucket** through **CloudFront**, it is important to ensure that the S3 bucket is not publicly accessible to prevent unauthorized access. **Origin Access Control (OAC)** enhances security by restricting direct access to the S3 bucket, allowing only CloudFront to retrieve files.

- We have set the **Default Root Object → index.html**

We have successfully created the distribution.



Copy the distribution name and hit it on web browser.

➔ Oops! Site cannot be reached.

Why this happened?

➔ When accessing a CloudFront distribution by copying its domain name into a web browser, an error such as **"Oops! Site cannot be reached"** may occur. This typically happens due to misconfigured permissions on the **Amazon S3 bucket** serving as the origin. If the **S3 bucket policy** does not explicitly allow CloudFront to access its objects, CloudFront will be unable to retrieve and serve the content. To resolve this, a proper **bucket policy** should be configured to grant access **only to CloudFront** via **Origin Access Control (OAC)** while blocking direct public access.

S3 Bucket Policy :

Bucket policy

Edit Delete

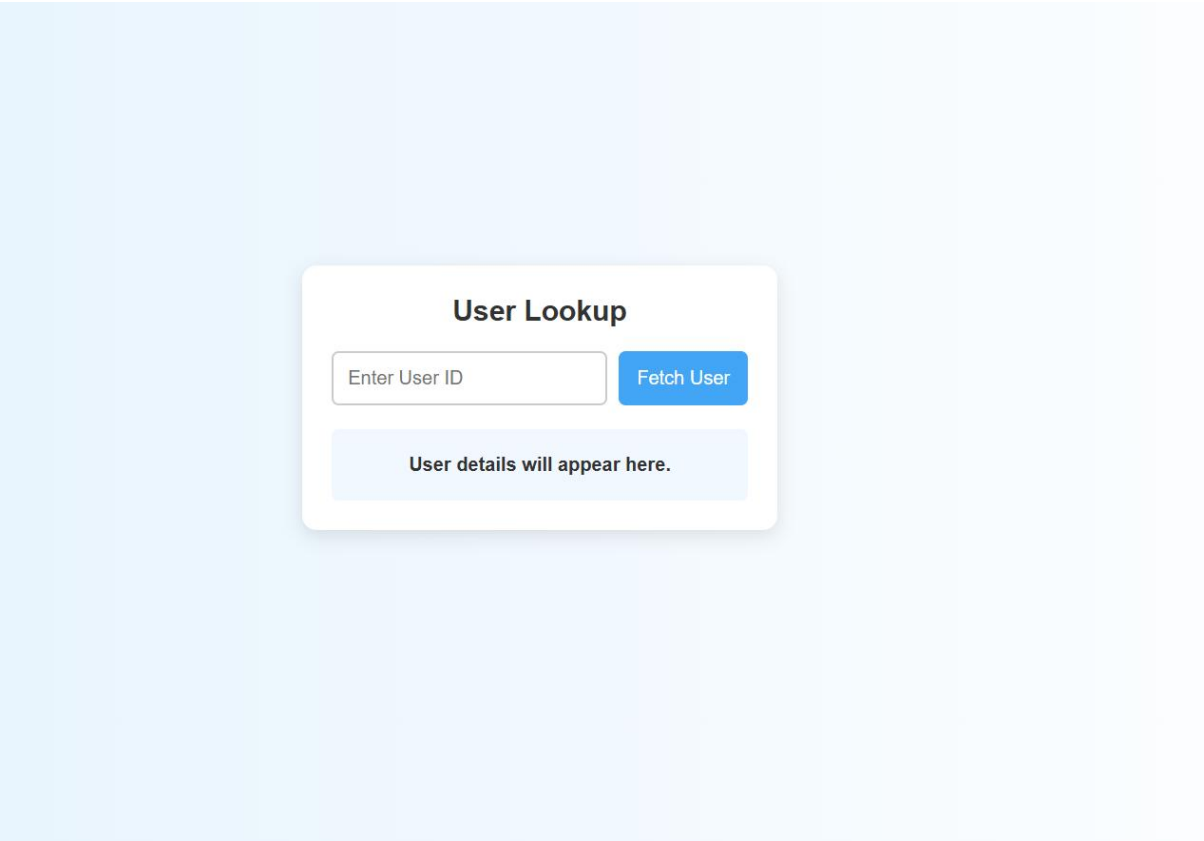
The bucket policy, written in JSON, provides access to the objects stored in the bucket. Bucket policies don't apply to objects owned by other accounts. Learn more

Public access is blocked because Block Public Access settings are turned on for this bucket
To determine which settings are turned on, check your Block Public Access settings for this bucket. Learn more about using Amazon S3 Block Public Access

Copy


```
{
  "Version": "2008-10-17",
  "Id": "PolicyForCloudFrontPrivateContent",
  "Statement": [
    {
      "Sid": "AllowCloudFrontServicePrincipal",
      "Effect": "Allow",
      "Principal": {
        "Service": "cloudfront.amazonaws.com"
      },
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::my-three-tier-architecture/*",
      "Condition": {
        "StringEquals": {
          "AWS:SourceArn": "arn:aws:cloudfront::622257332701:distribution/EMD4P0P62JWRH"
        }
      }
    }
  ]
}
```


- **Refresh the URL**



How CloudFront Different from S3 Static Website hosting ?

➔ **CloudFront** is a **Content Delivery Network (CDN)** that caches content at edge locations worldwide, reducing latency and improving speed for users across different regions.

➔ **S3 Static Website Hosting** serves files directly from a specific AWS **S3 bucket region**, meaning users far from that region may experience higher latency.

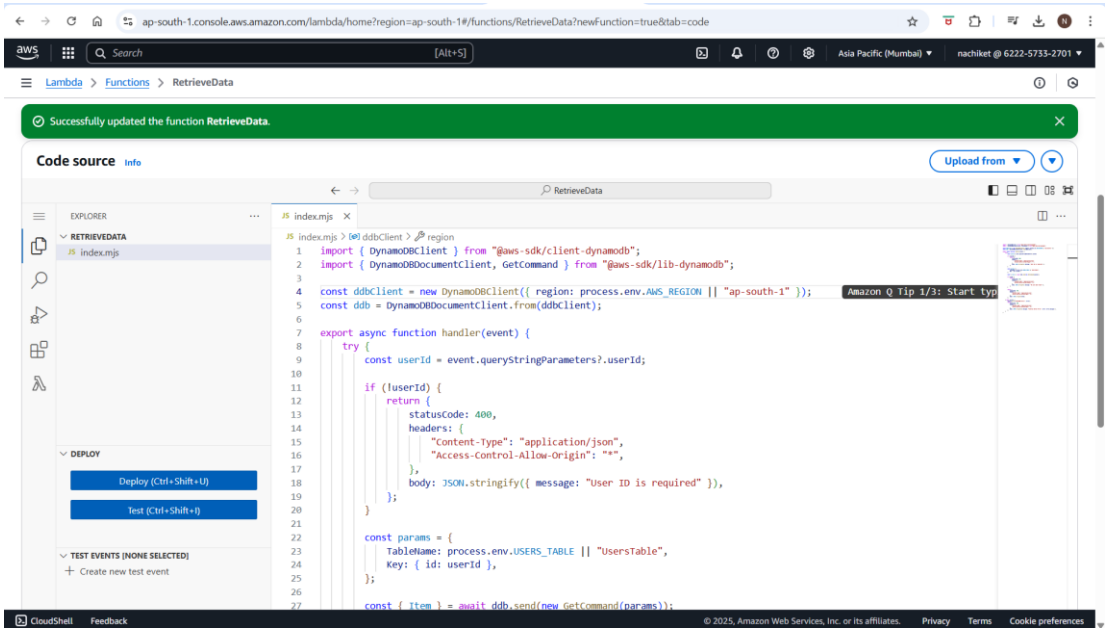
Hurray! We have completed with Presentation Tier Part.

Step 4 – Let's setup the Logic Tier.

What is Lambda Function?

➔ AWS **Lambda** is a **serverless compute service** that lets you run code without provisioning or managing servers. It automatically scales based on the number of incoming requests and executes code only when triggered, making it a **cost-efficient** and **highly scalable** solution for various applications.

- Create Lambda Function by giving Function name,
- select the Runtime,
- write the Lambda Function Code.



When to use Lambda and when to use EC2?

- ➔ If you **want a fully managed, auto-scaling, event-driven service**, go with **AWS Lambda**.
- ➔ If you **need full control, long-running applications, or custom infrastructure**, use **EC2**.

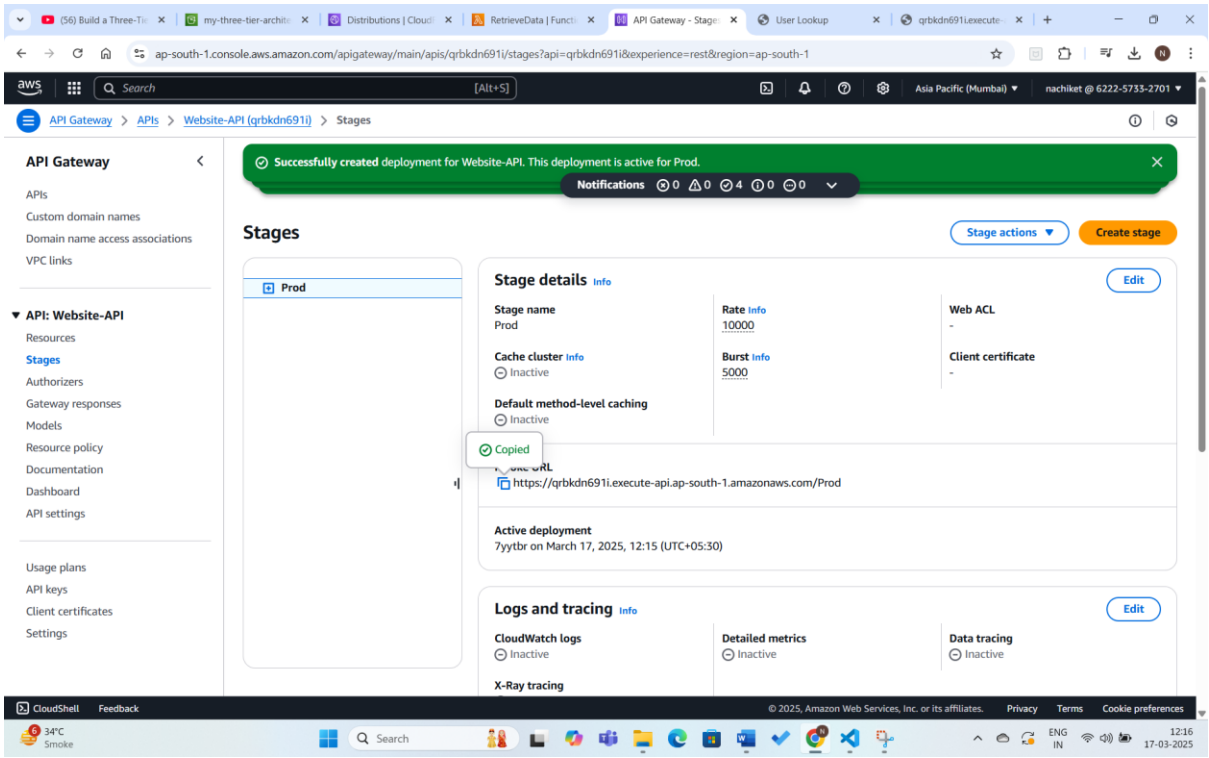
Step 5: Create API Gateway

- For Our project, we have selected REST API

What is API Gateway?

- ➔ In a distributed website architecture, when a user makes a request, the request is sent to Amazon API Gateway. The API Gateway acts as an entry point, processing the request and then triggering the appropriate AWS Lambda function to execute the backend logic.

- We have set the method type as GET. As, we are going to just retrieve the data.
- We have then again re-deployed API by name - Prod.

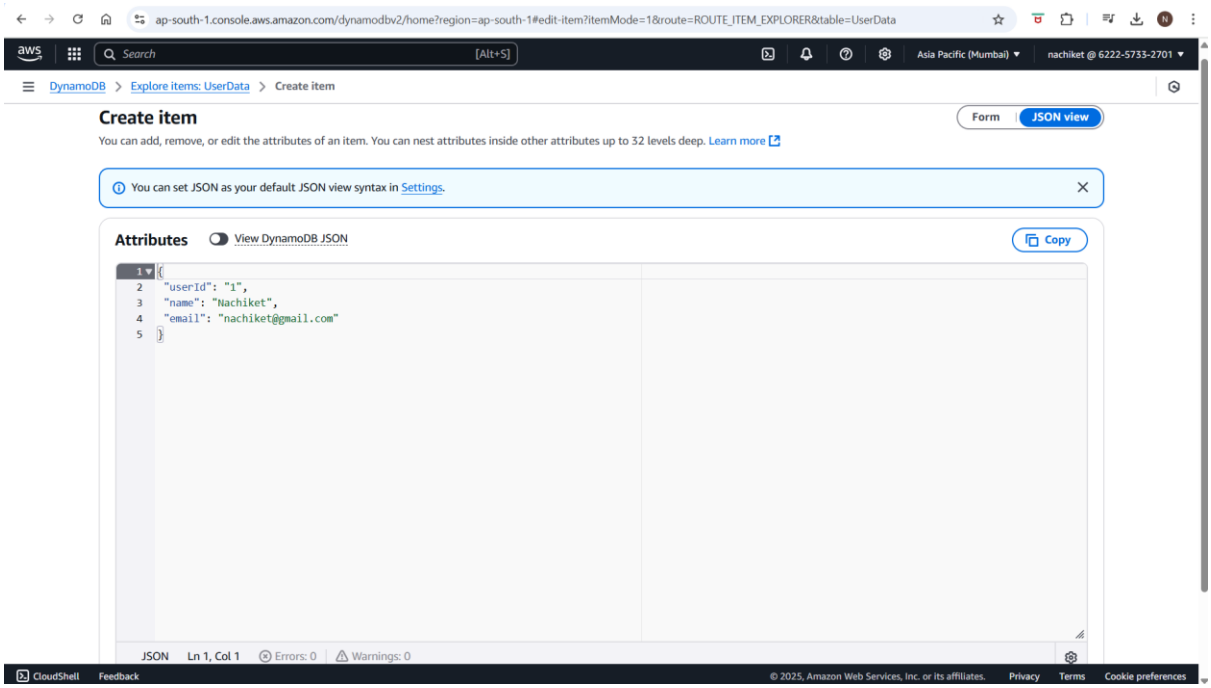


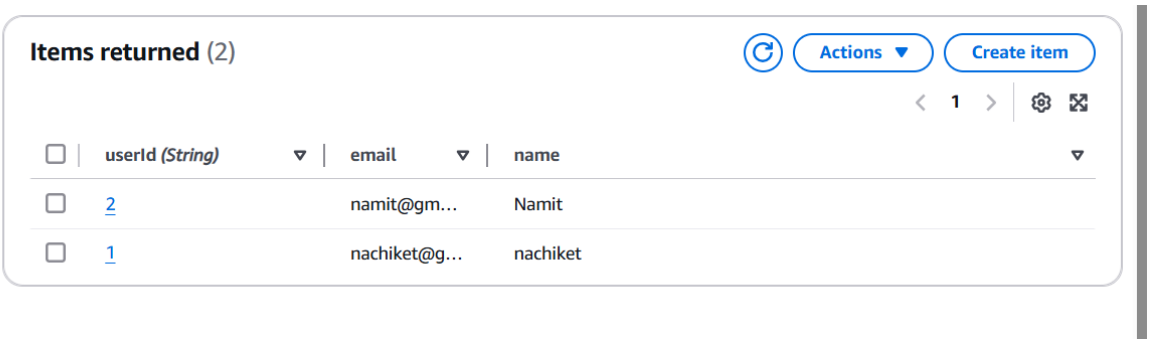
Copy the Invoke URL and paste in browser and we got an error. Why?

➔ Because we haven't setup the DynamoDB yet.

Step 6: Let's Setup the Data Tier

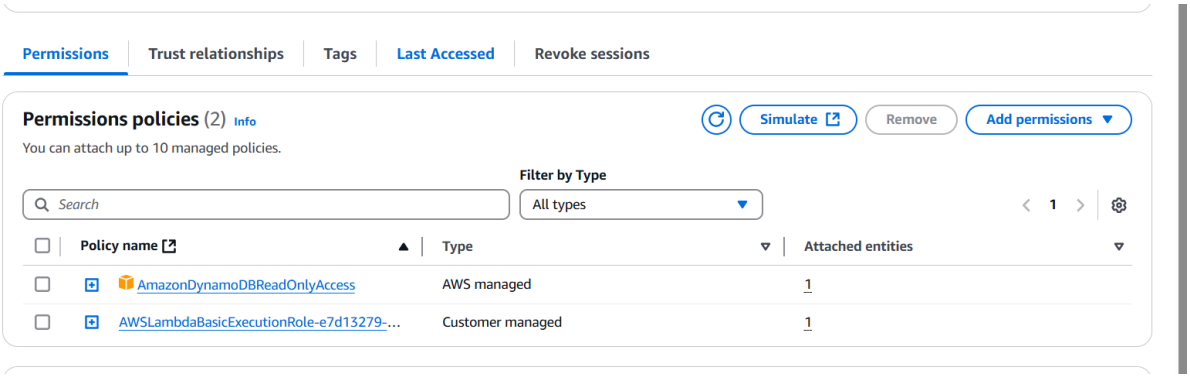
- Go to DynamoDB and create table
- Give the Table Name
- Create the item. i.e. add the data in the DynamoDB in JSON Format.



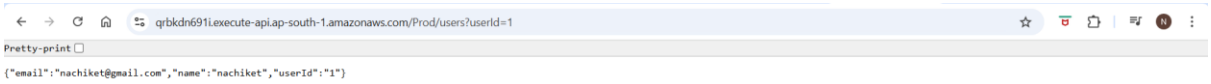


Step 7: Go again in Lambda Function.

- Go to Configuration Settings
- Add Permission Policies for DynamoDB



- Now Refresh the Invoke URL.



- **This proves, Connection between Logic and Data Tier is Successful.**

But why still not able to fetch UI on Frontend side?

➔ We haven't connected our API to Website yet.

Replace your CloudFront URL in script.js

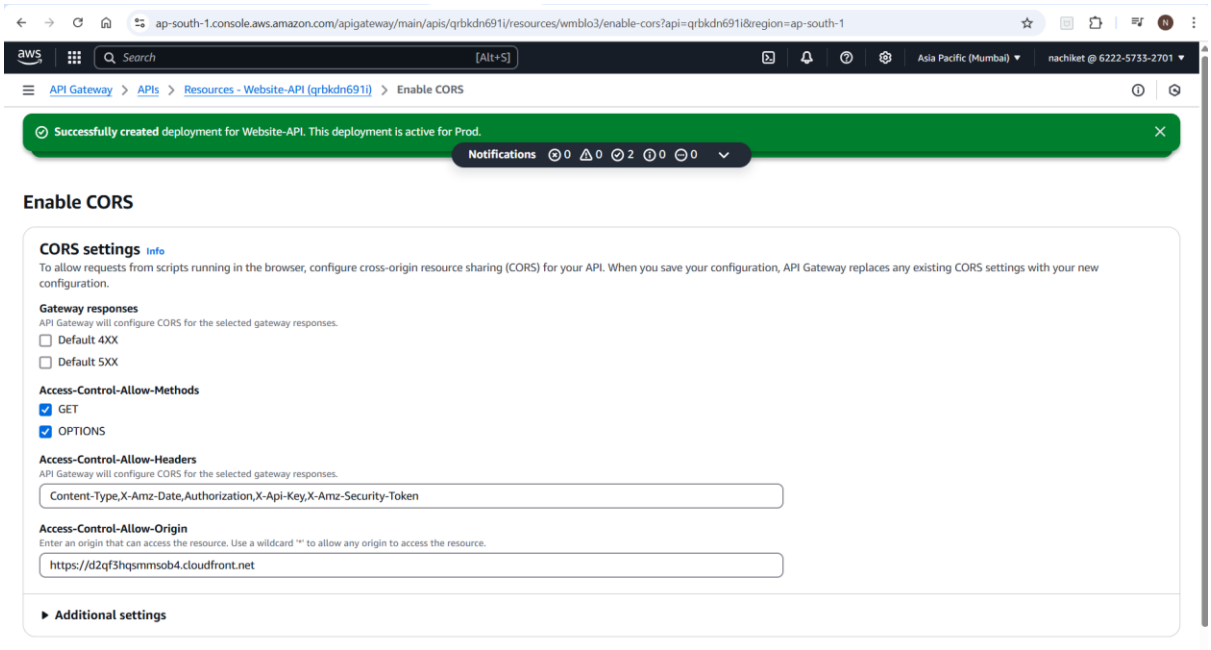
- URL – (CloudFront URL)/prod/users (prod from API Gateway and Users is the table name from DynamoDB)

Step 8: Go to API Gateway Options

- In API Gateway option enable CORS and Tick the GET and OPTIONS from Access control allow methods.

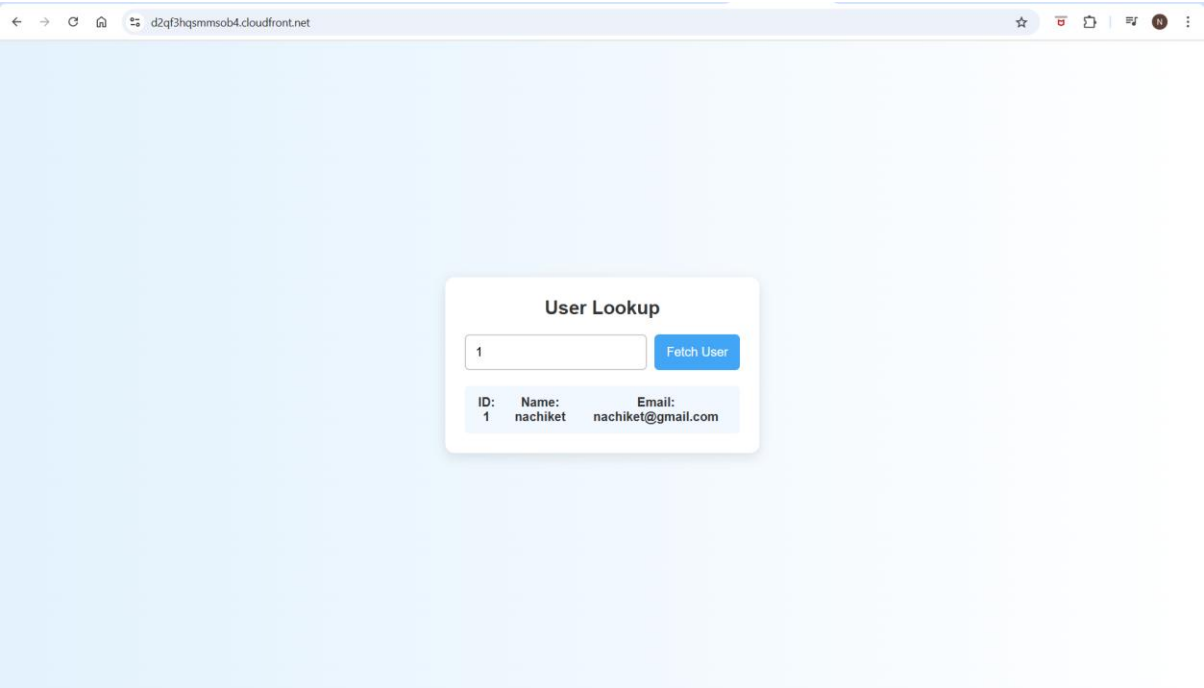
What is CORS?

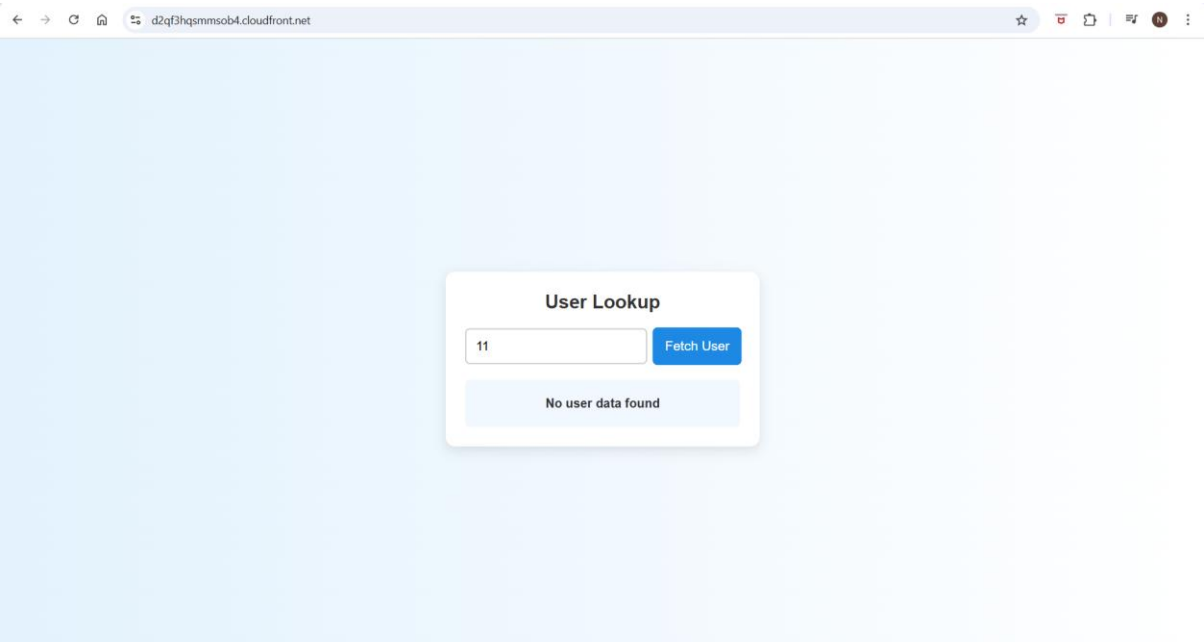
➔ Modern web applications often need to fetch data from APIs hosted on different domains (e.g., a frontend hosted on **example.com** making a request to an API at **api.example.com**). Without CORS, browsers would block these requests as a security measure.



- In Access Control Allow Origin Option – Add your CloudFront URL.

- Hurray! Now we can retrieve the data from CloudFront Distribution.


d2qf3hqsmmsob4.cloudfront.net
User Lookup
1
Fetch User
ID: Name: Email:
1 nachiket nachiket@gmail.com


d2qf3hqsmmsob4.cloudfront.net
User Lookup
11
Fetch User
No user data found