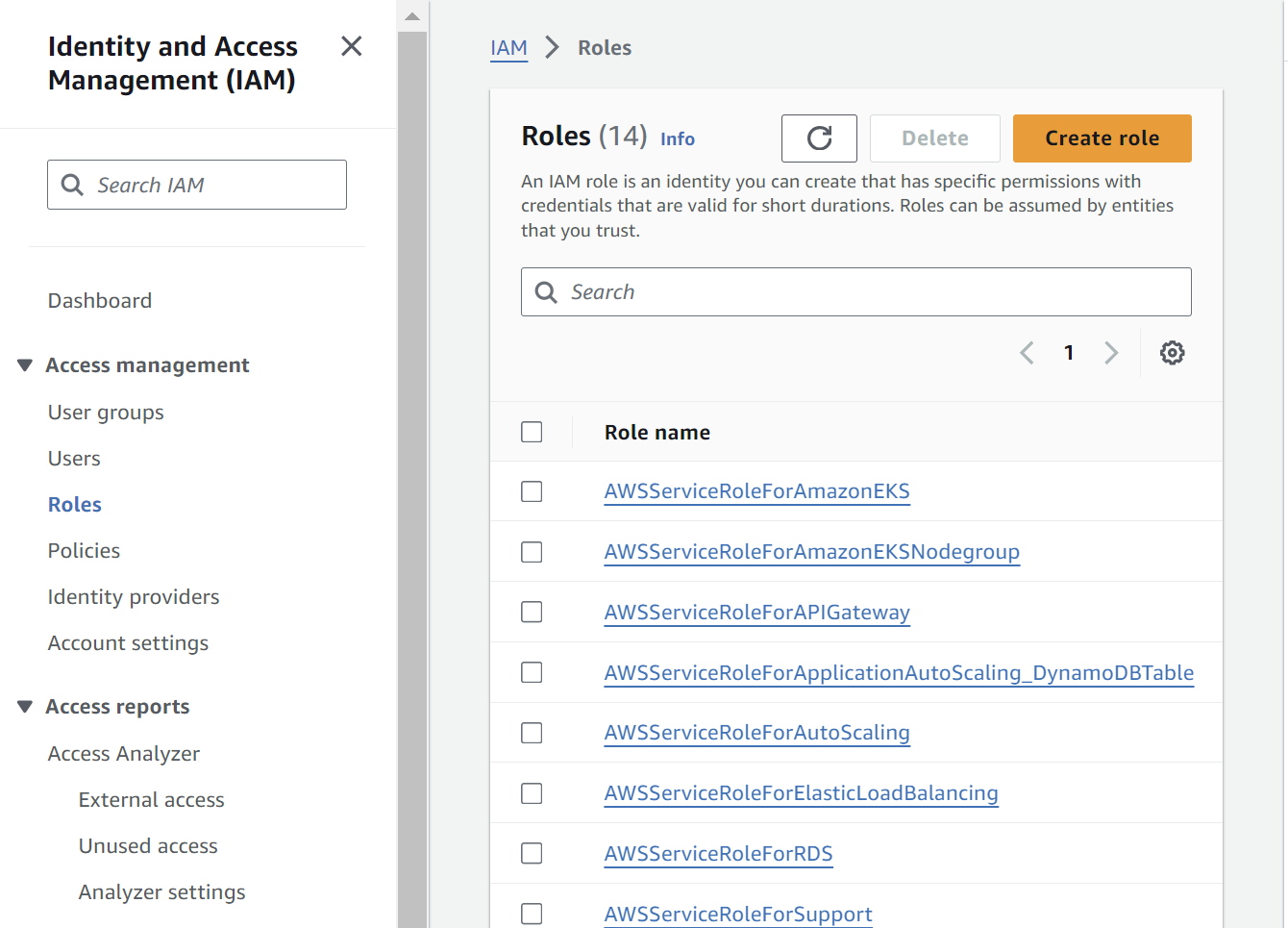
**Experiment-2**

**To create simple backend logic and communication with front end application.**

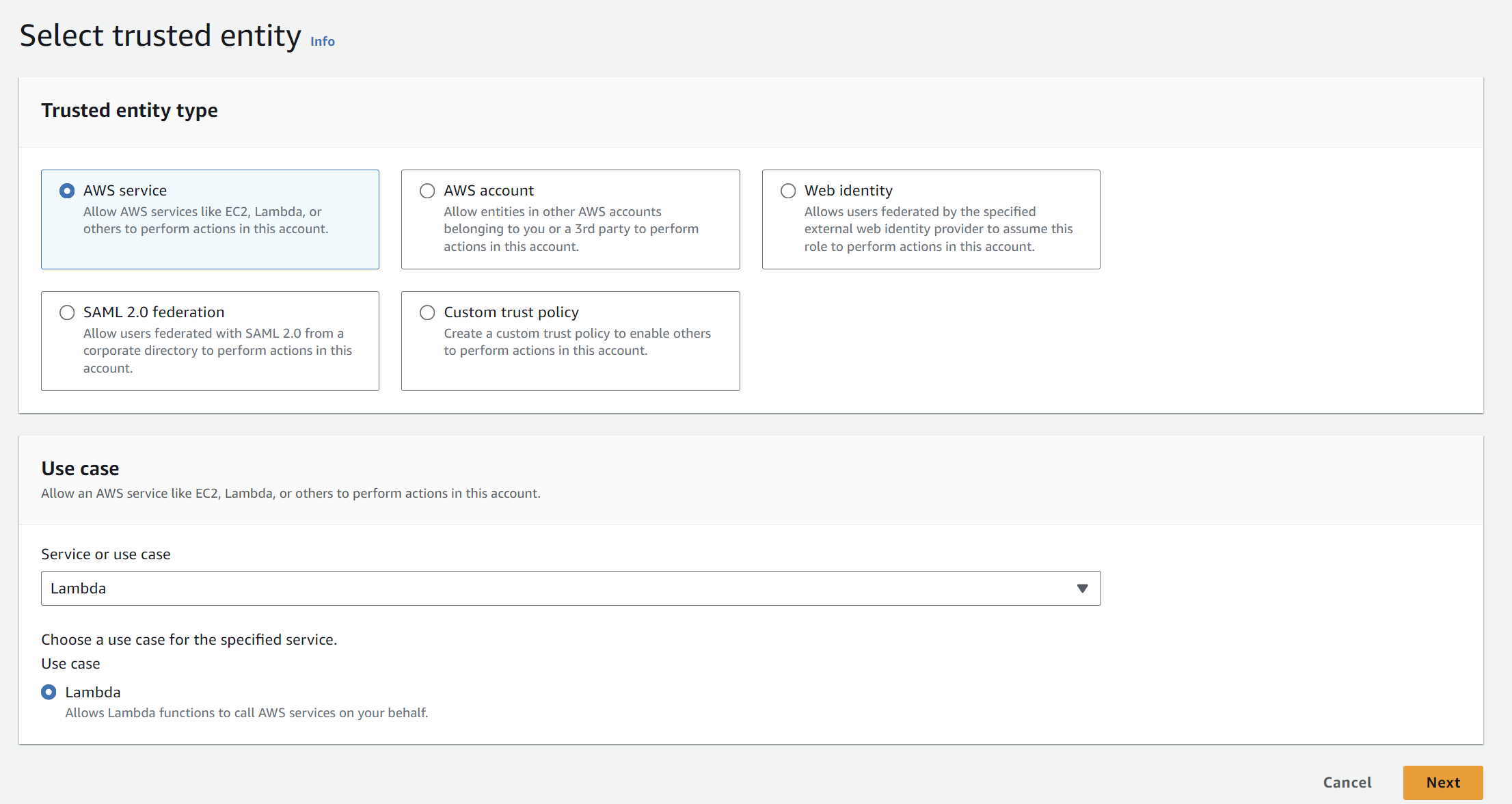
* **Technologies used:** React, Node
* **AWS services used:** API Gateway, Lambda Function, S3 Bucket, DynamoDB, IAM

**IAM Role Creation**

**Step-1:** Go to AWS console and search for IAM. Go to roles and click on “Create role”.



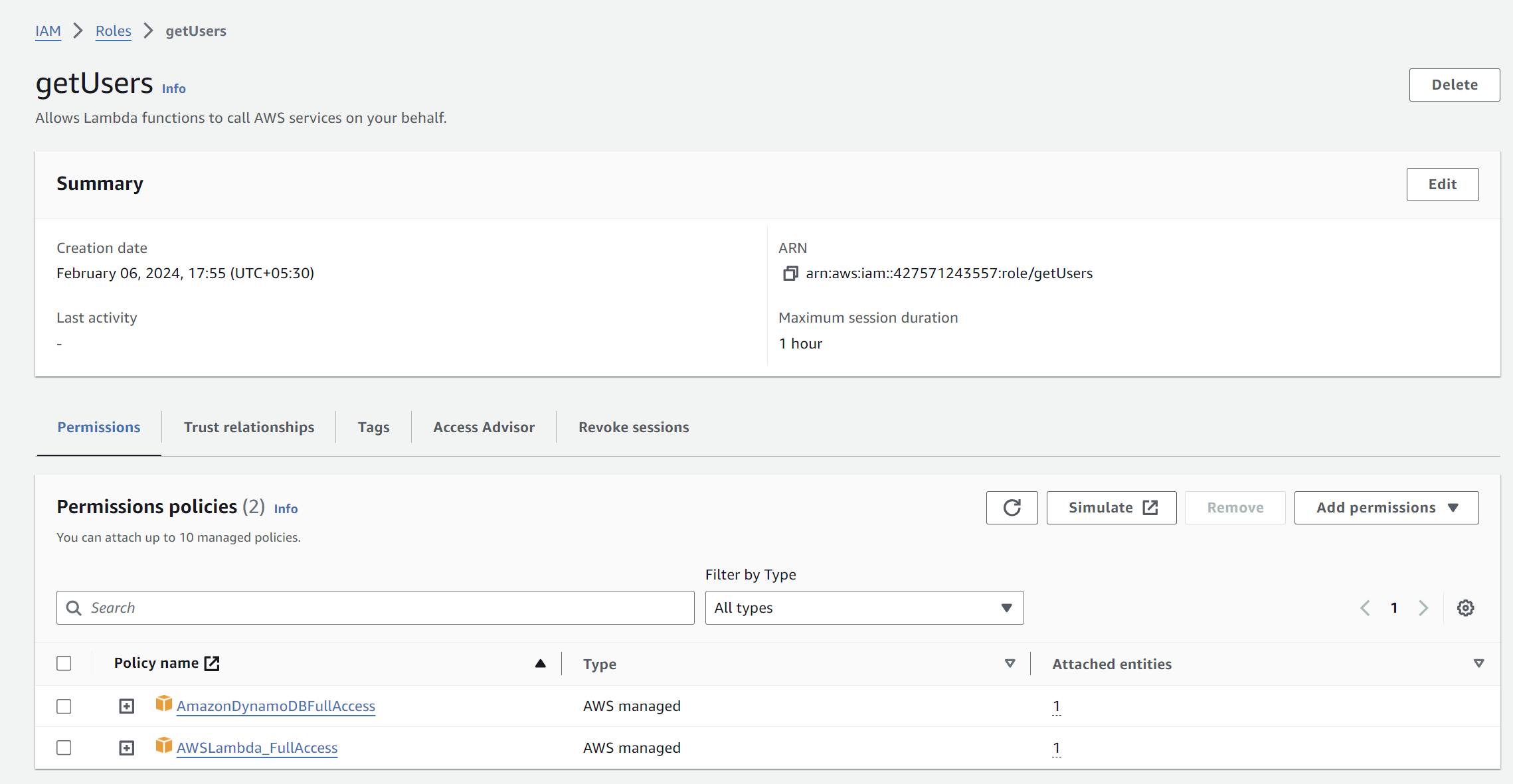
**Step-2:** Select the “AWS service” option and select “Lambda”



**Step-3**: Attach the following permissions to the role:

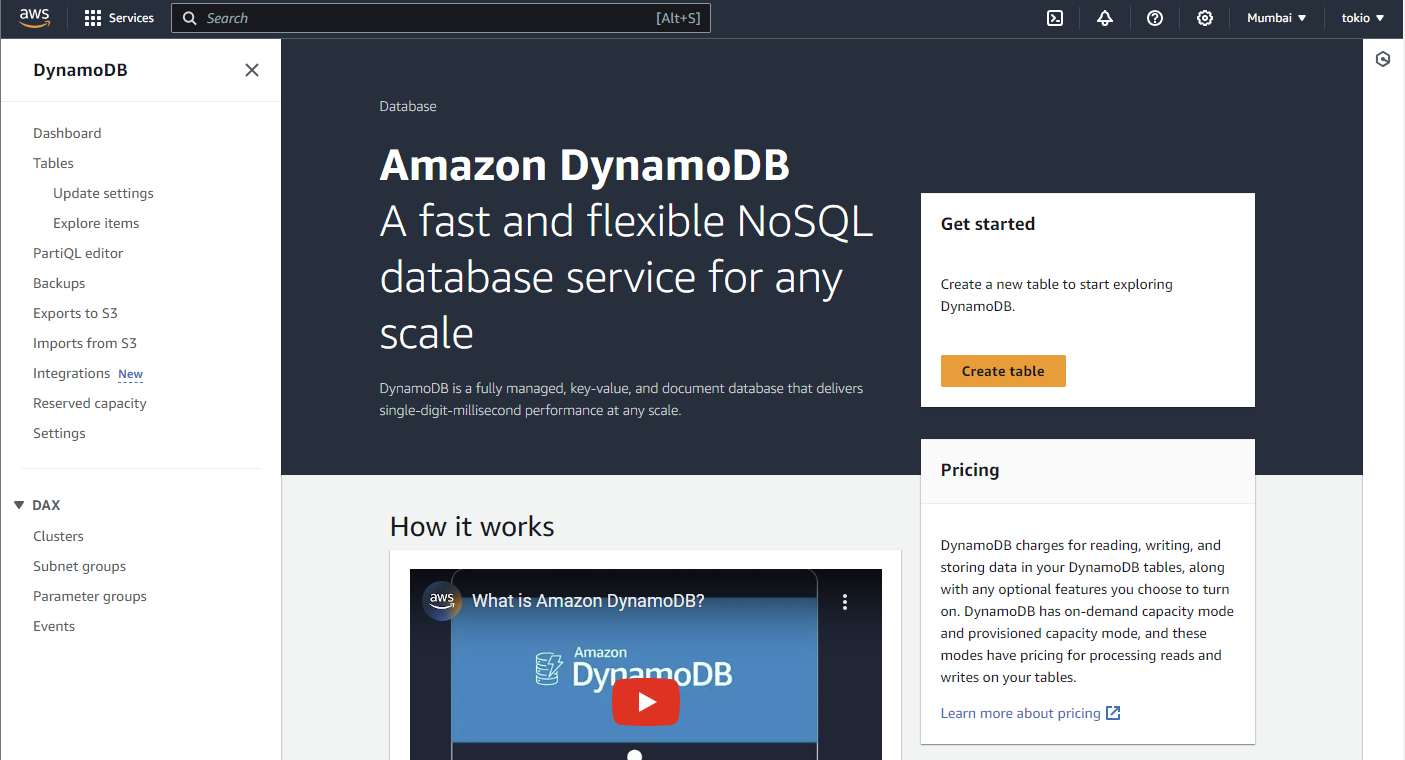
AmazonDynamoDBFullAccess

AWSLambda\_FullAccess

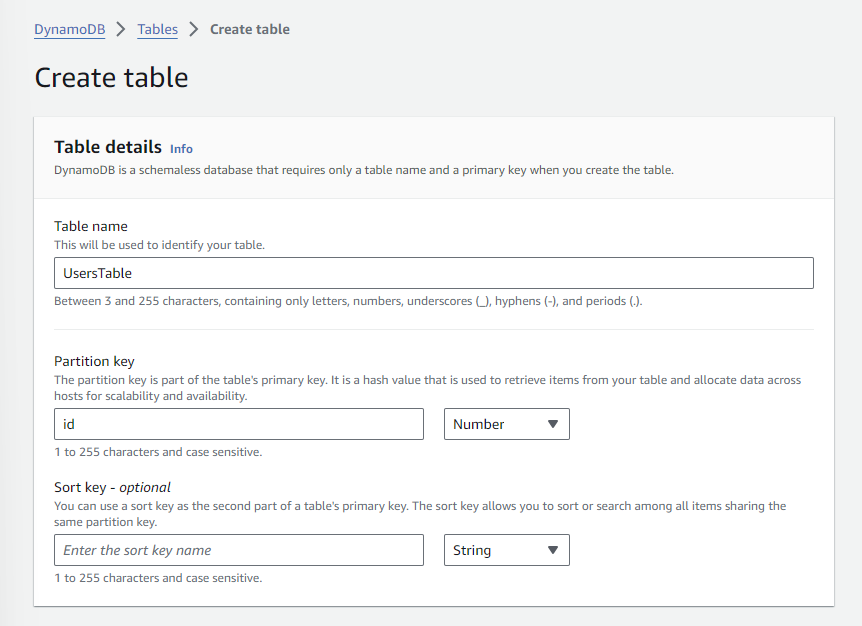


**DynamoDB Table**

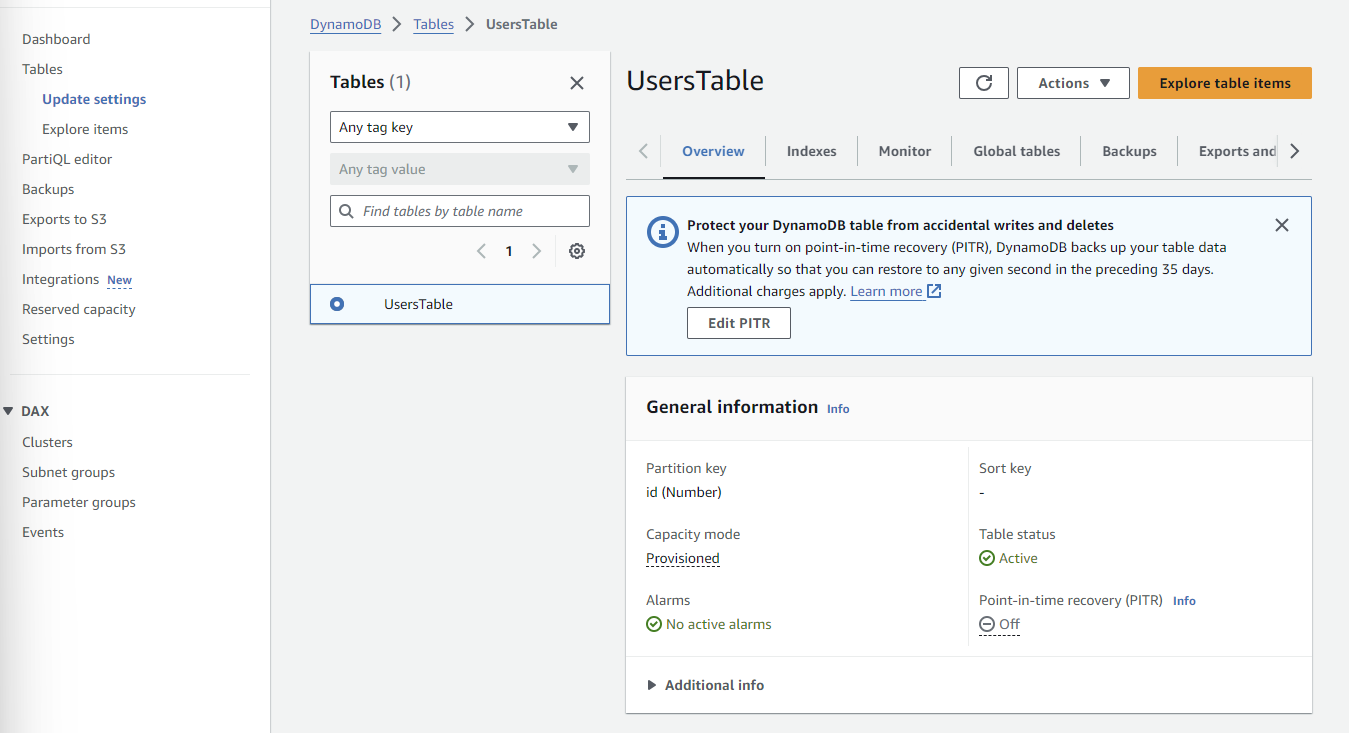
**Step-1:** Go to DynamoDB and click on “Create table”



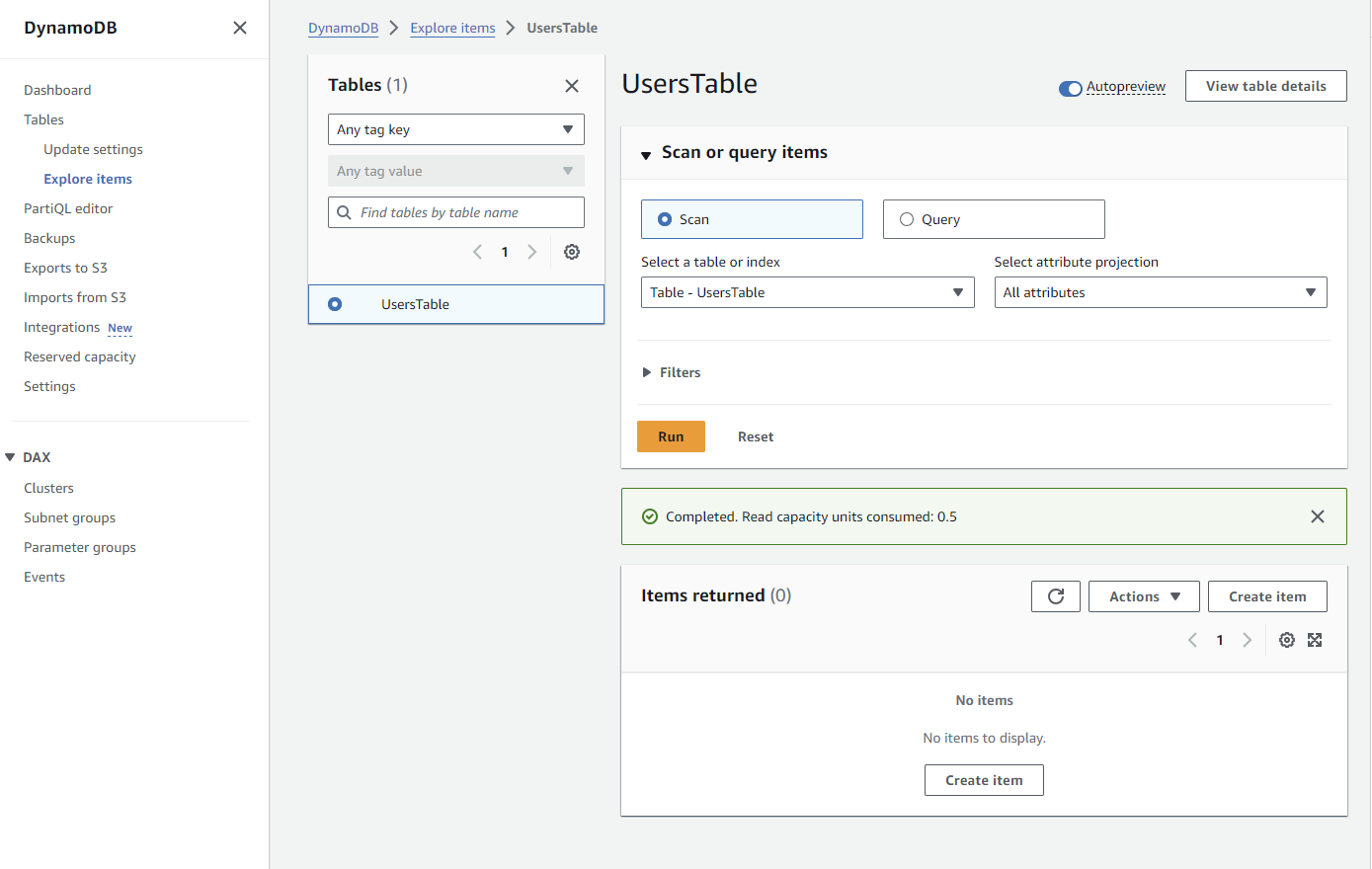
**Step-2:** Fill in the table name and the partition key as “id” and then proceed to create table.



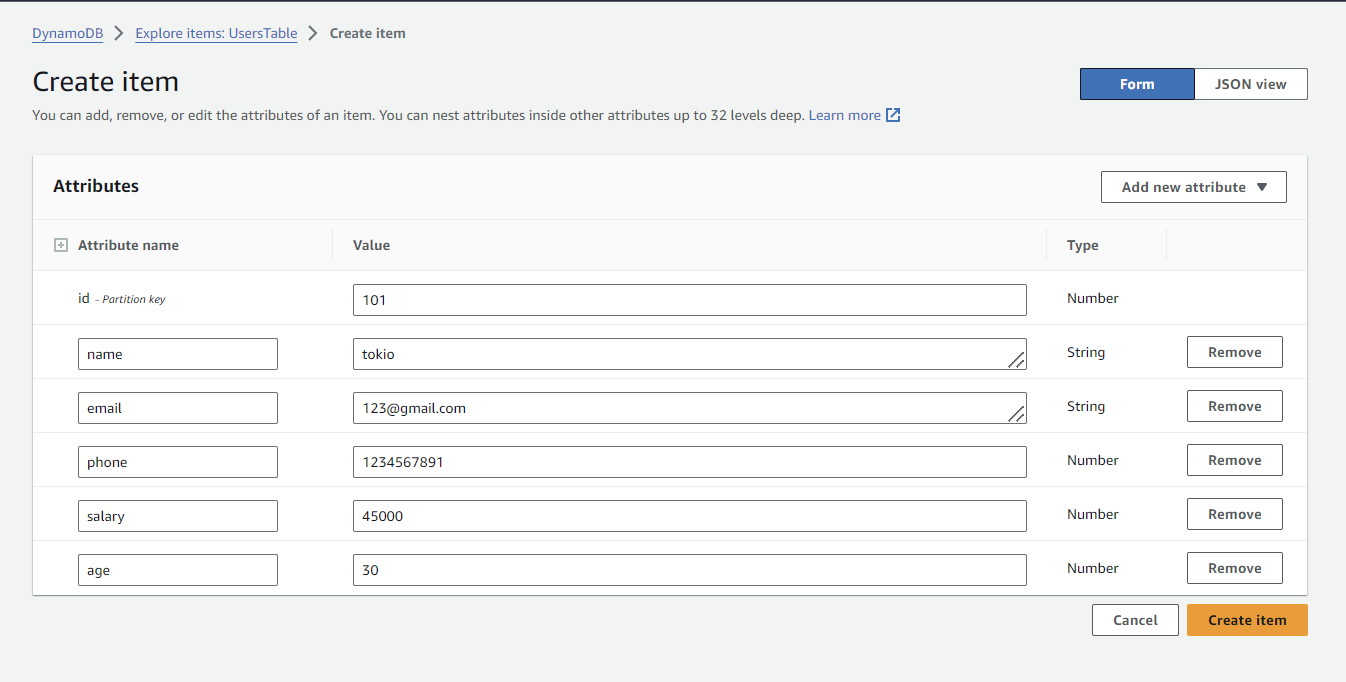
**Step-3:** Now go to “Explore table items”.



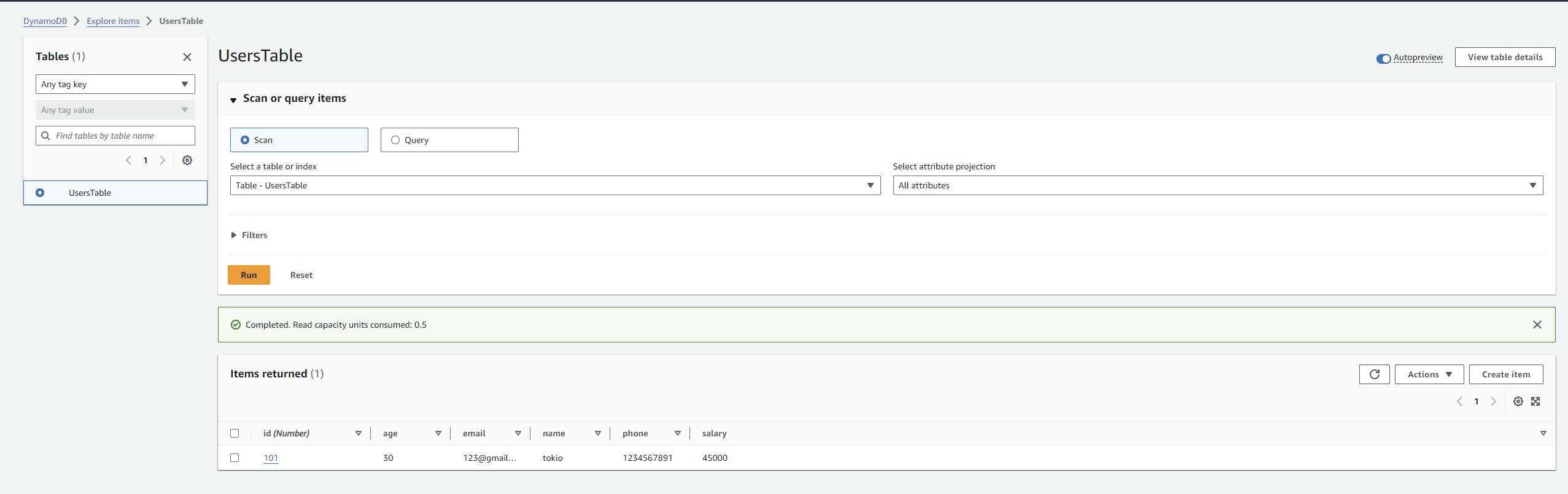
**Step-4:** Now click on “Create item”. For testing purpose, we are manually populating the data with just one entry.



**Step-5:** Now add the values for the below mentioned fields

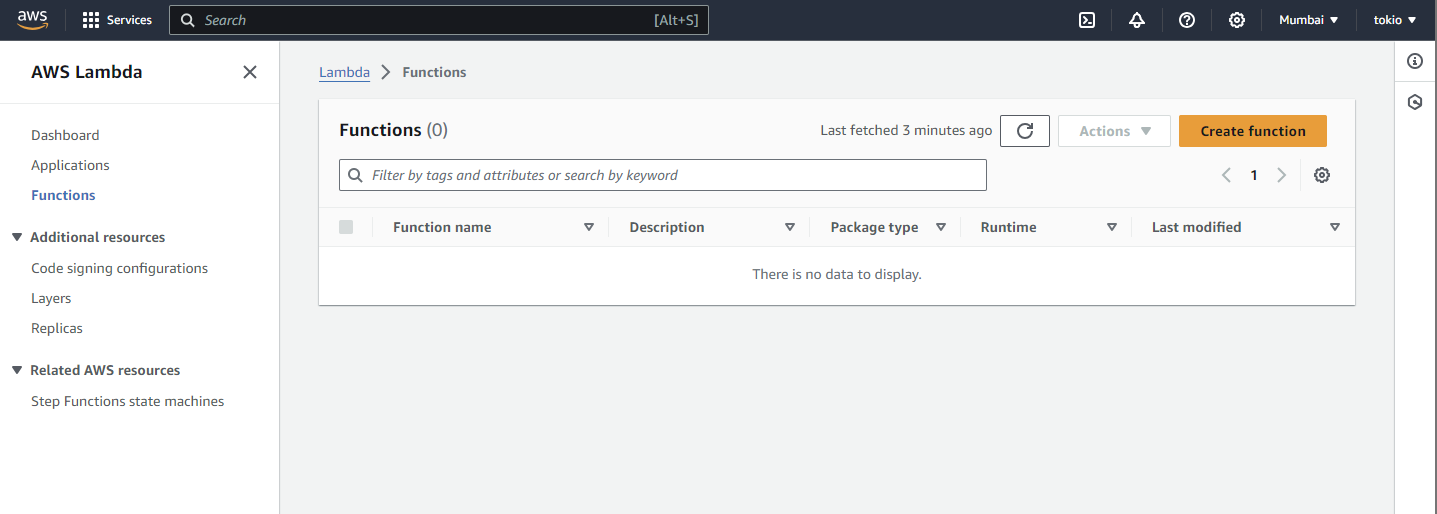


**Step-6:** Now you can see that the values are inserted successfully.

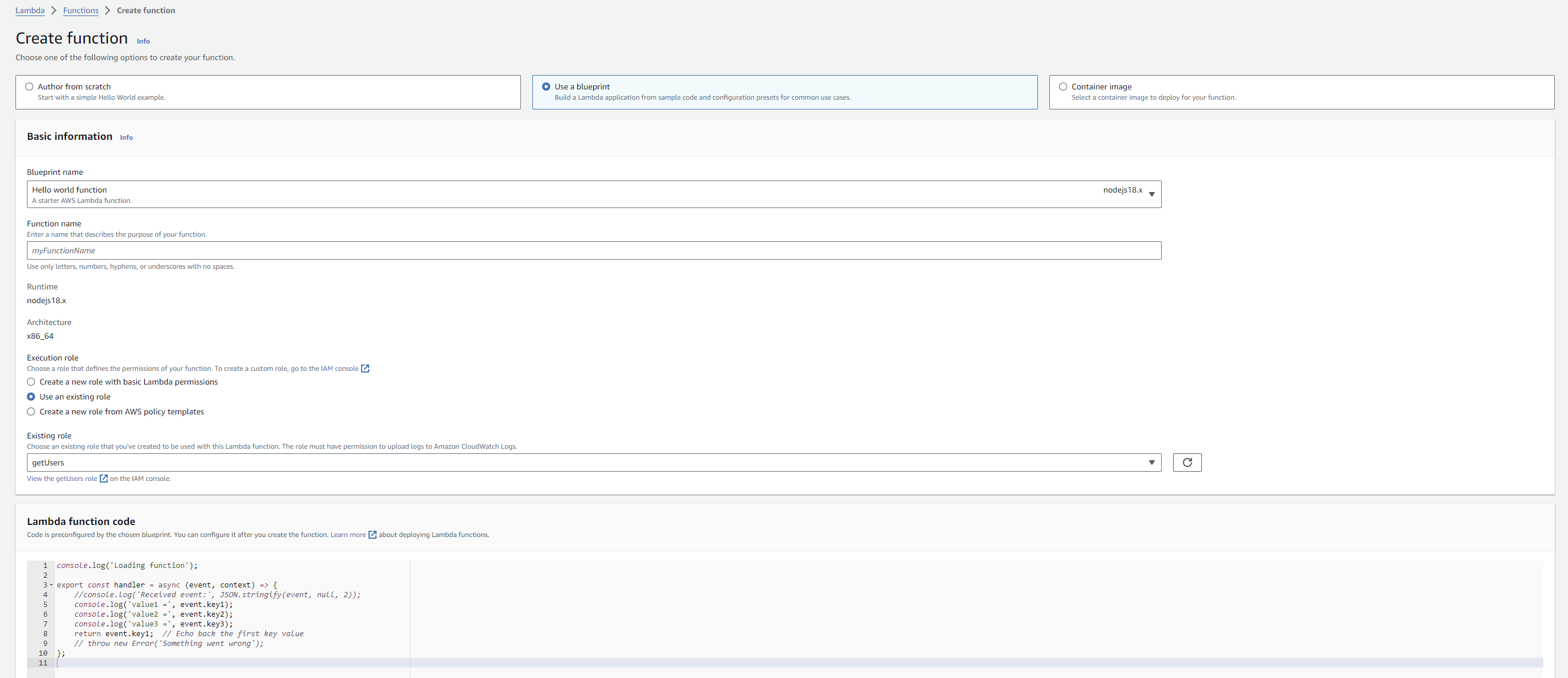


**Lambda Function**

**Step-1:** Go to Lambda and click on “Create function”



**Step-2:** Fill the function name, language as Node and attach the IAM role that you just created.



**Step-3:** Now copy the below code in the code editor in lambda function. Deploy the code and save it.

import { DynamoDBClient, ScanCommand } from '@aws-sdk/client-dynamodb';

// Create a DynamoDB client instance

const dynamoDBClient = new DynamoDBClient({});

const params = {

TableName: 'UsersTable'

};

const listItems = async () => {

try {

// Execute the ScanCommand to scan items in the DynamoDB table

const data = await dynamoDBClient.send(new ScanCommand(params));

return data;

} catch (error) {

return error;

}

};

export const handler = async (event, context) => {

try {

const data = await listItems();

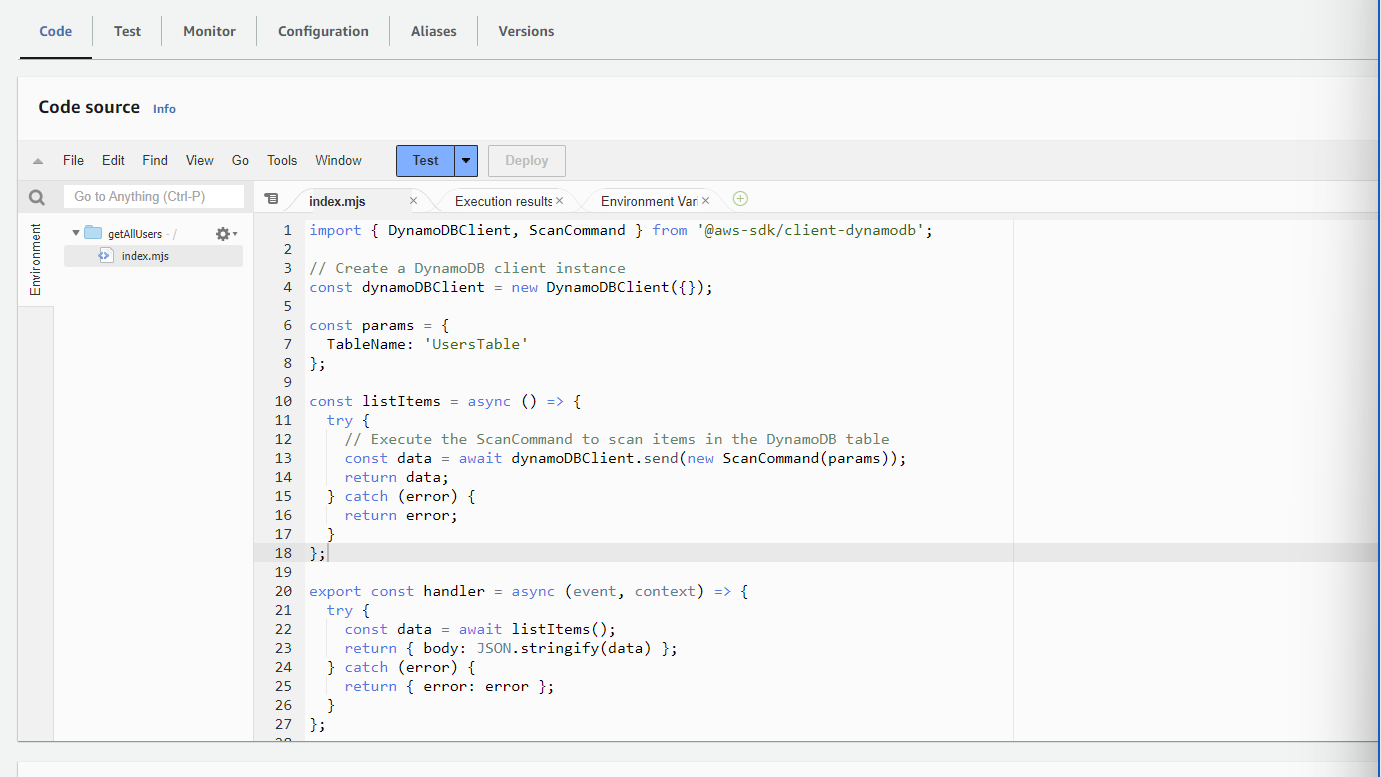
return { body: JSON.stringify(data) };

} catch (error) {

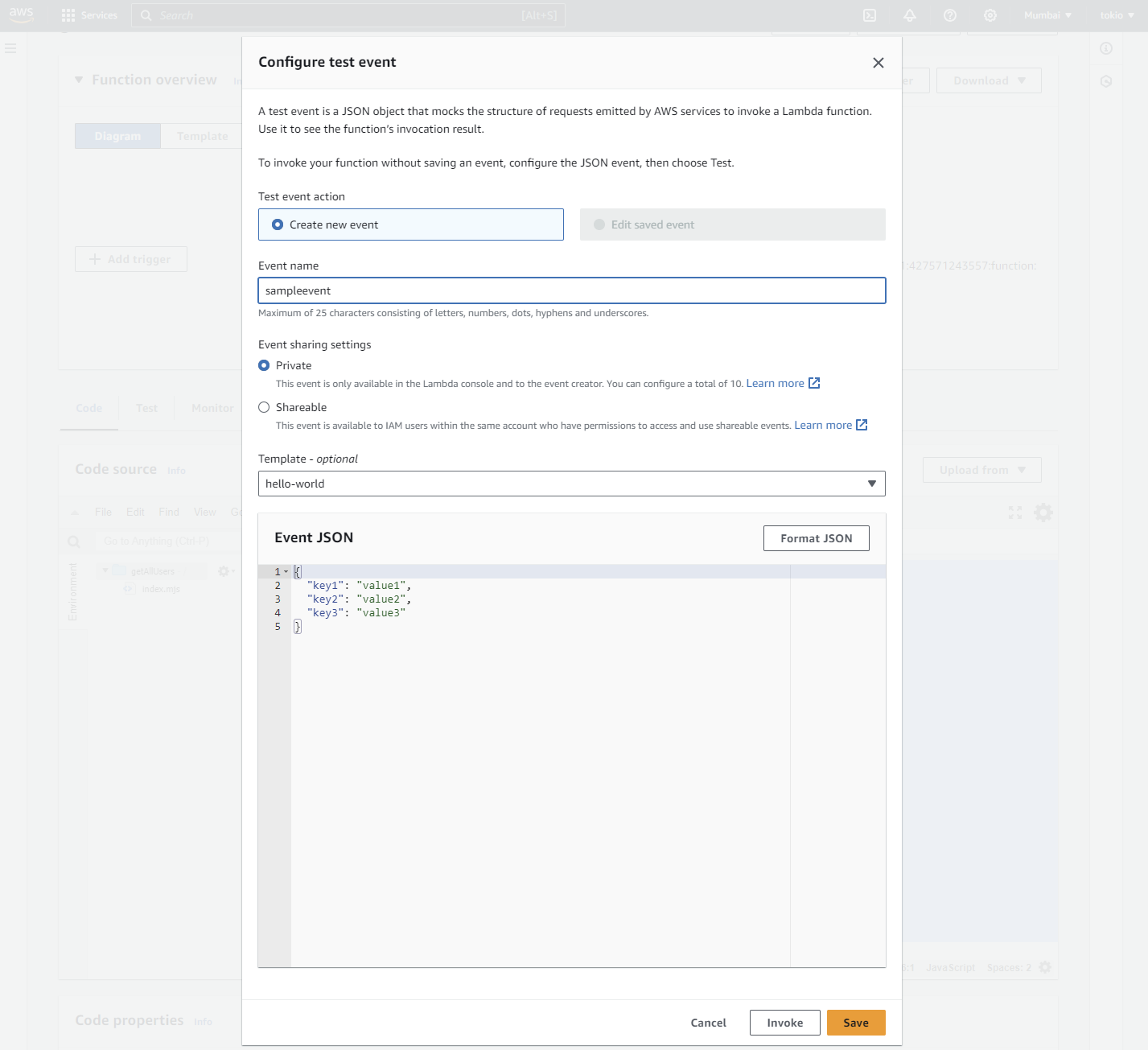
return { error: error };

}

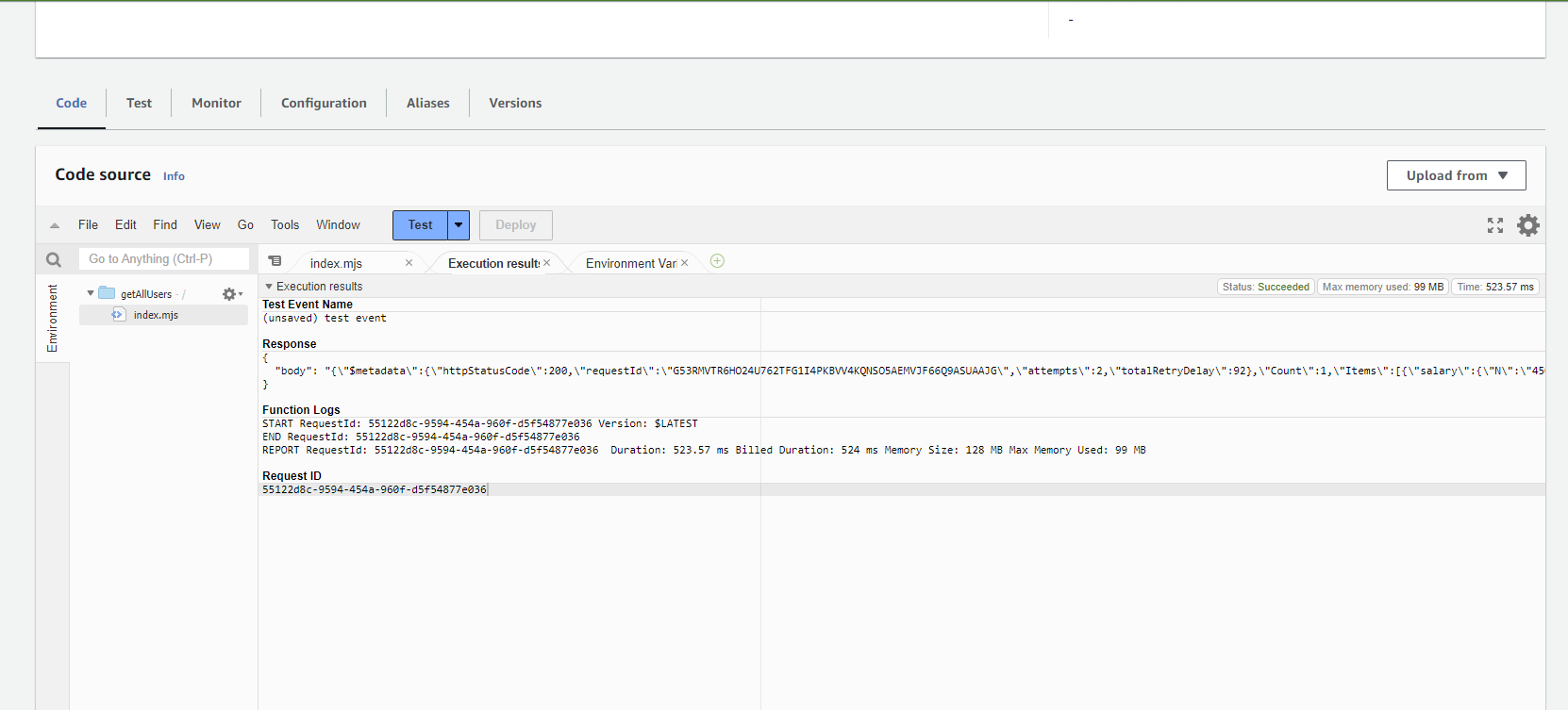
};



**Step-4:** Now test the function. Create a sampleevent and save it, then run the test

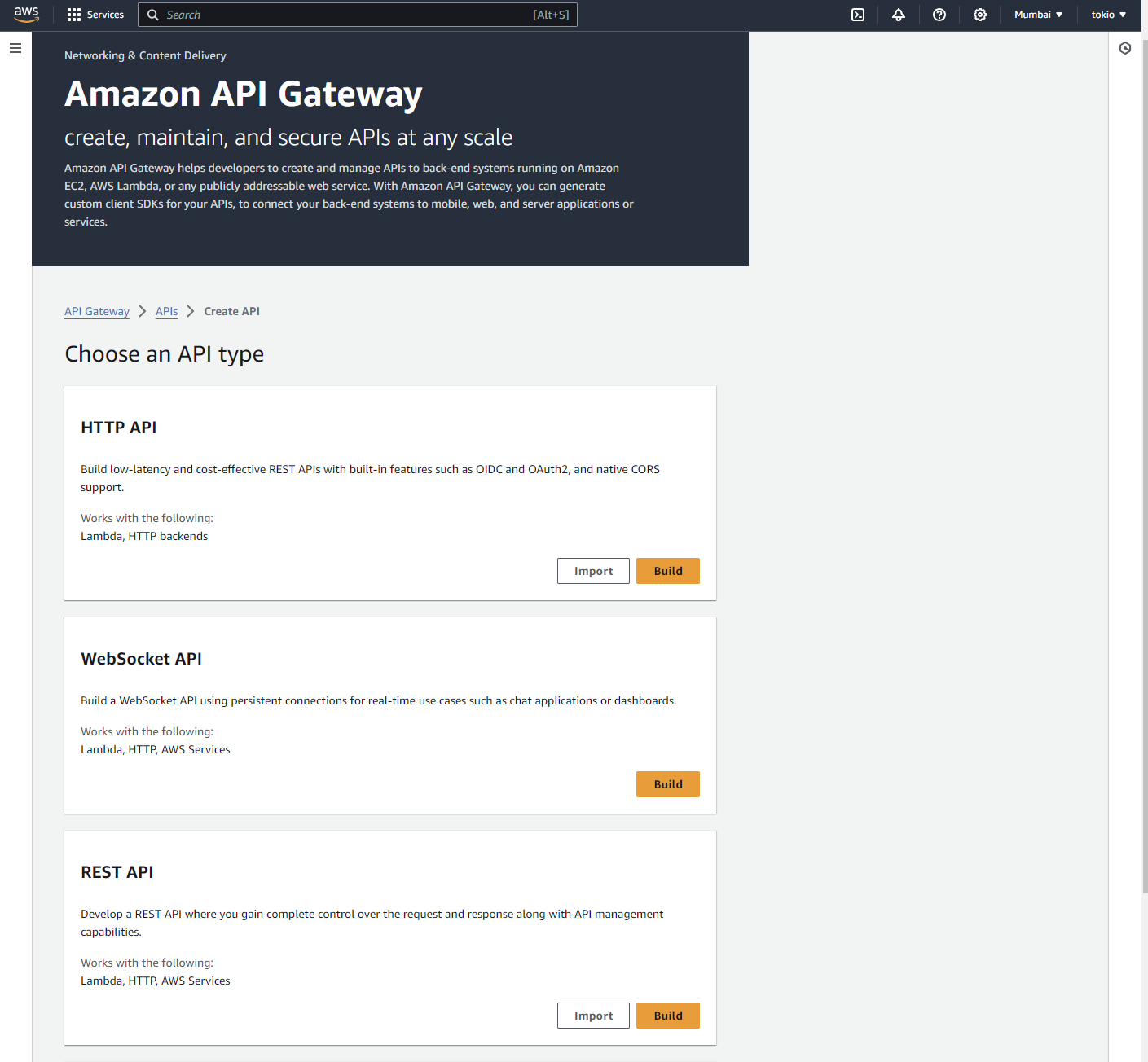


**Step-5:** Now you can see that the execution results show positive response i.e. our lambda function is successfully executing.

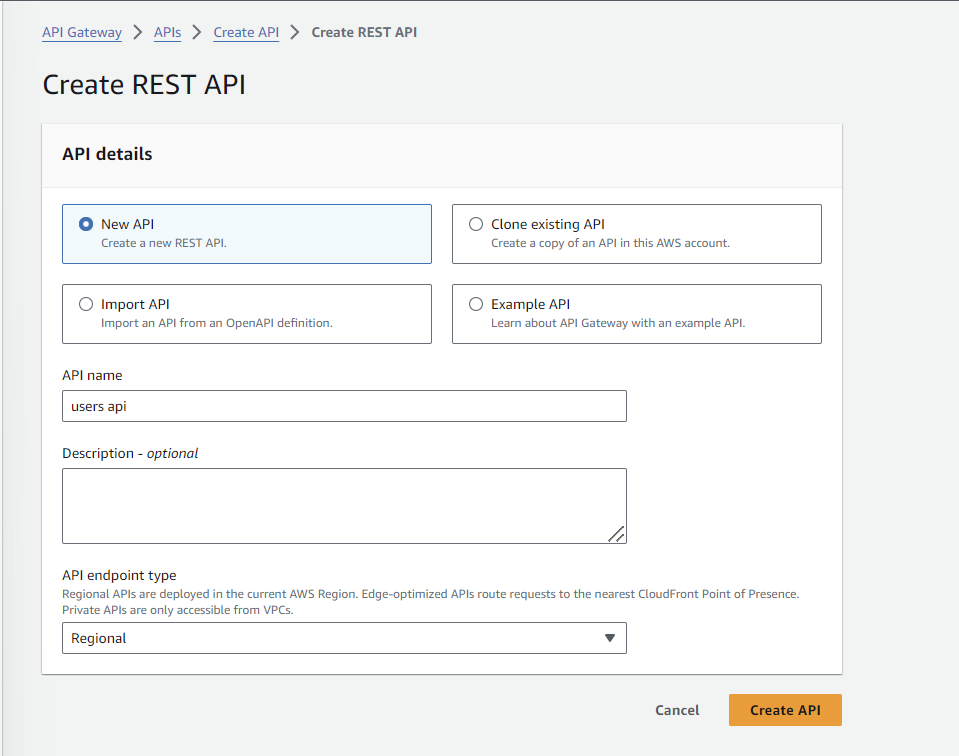


**API Gateway**

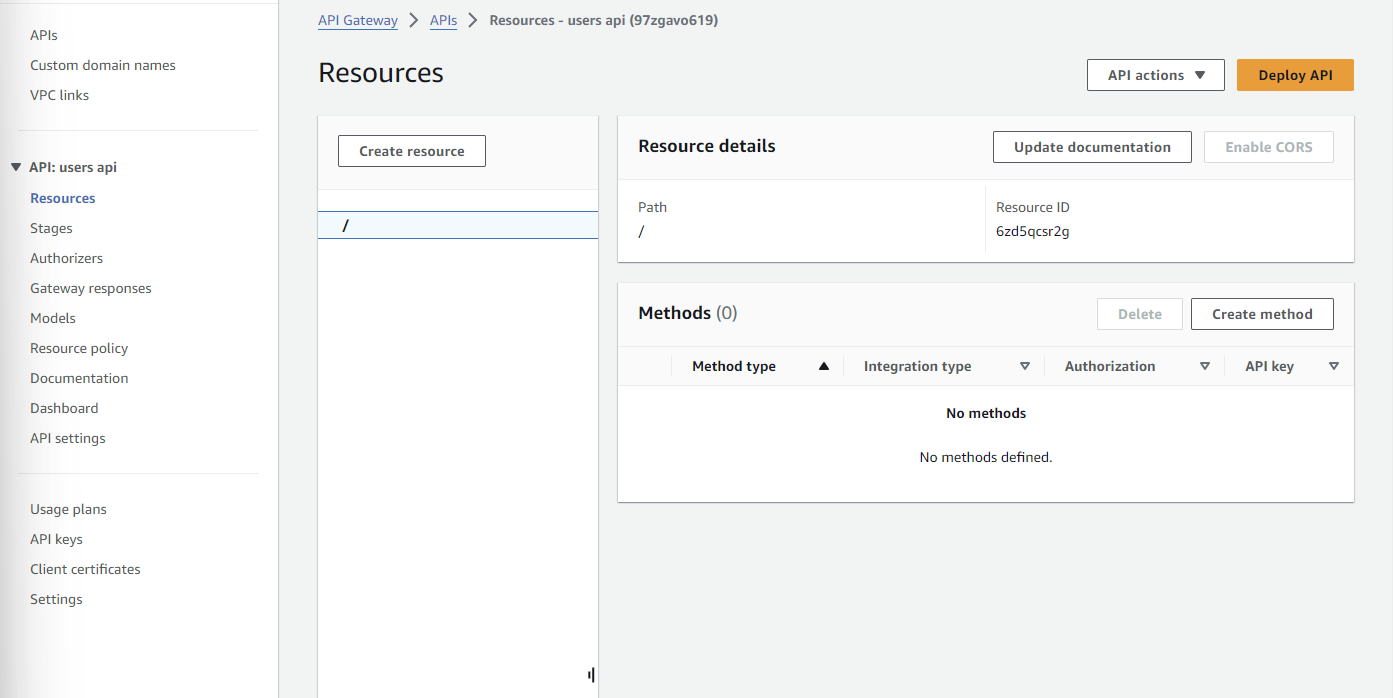
**Step-1:** Go to Amazon API Gateway and chose “Build” option from REST API.



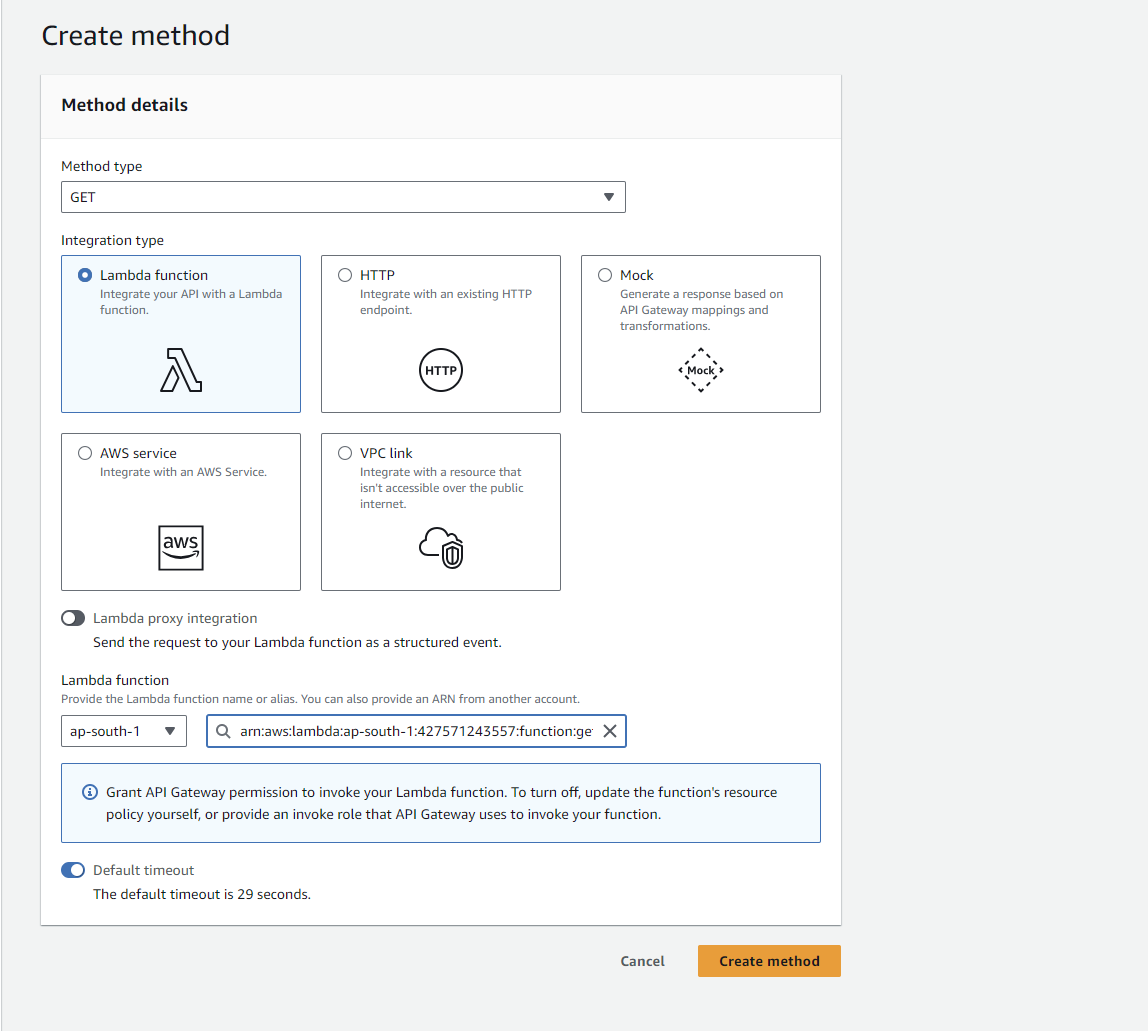
**Step-2:** Click on “New API” and name it



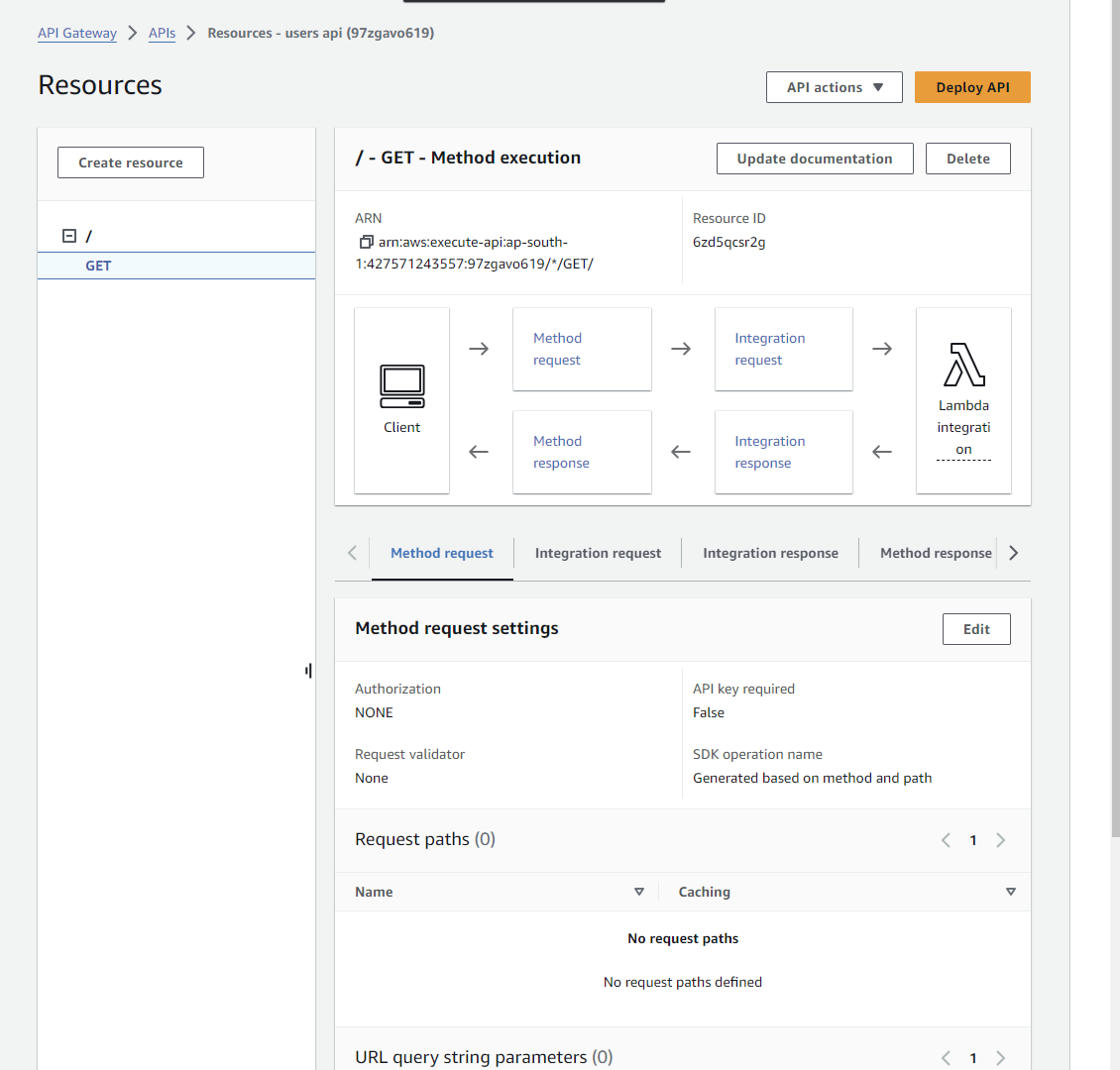
**Step-3:** Click on “Create method”.

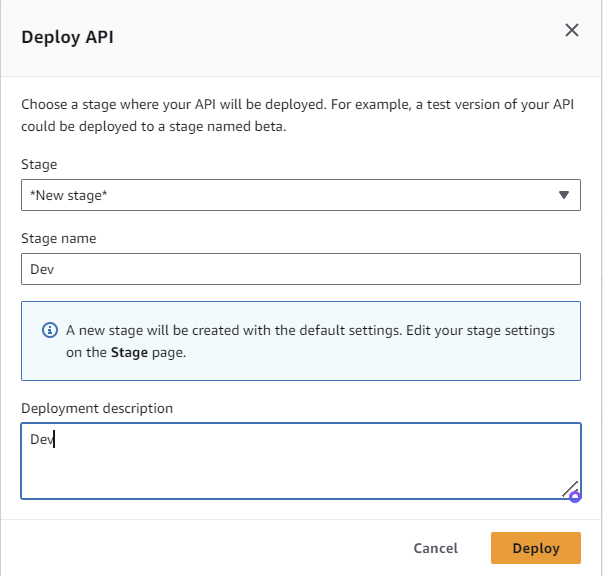


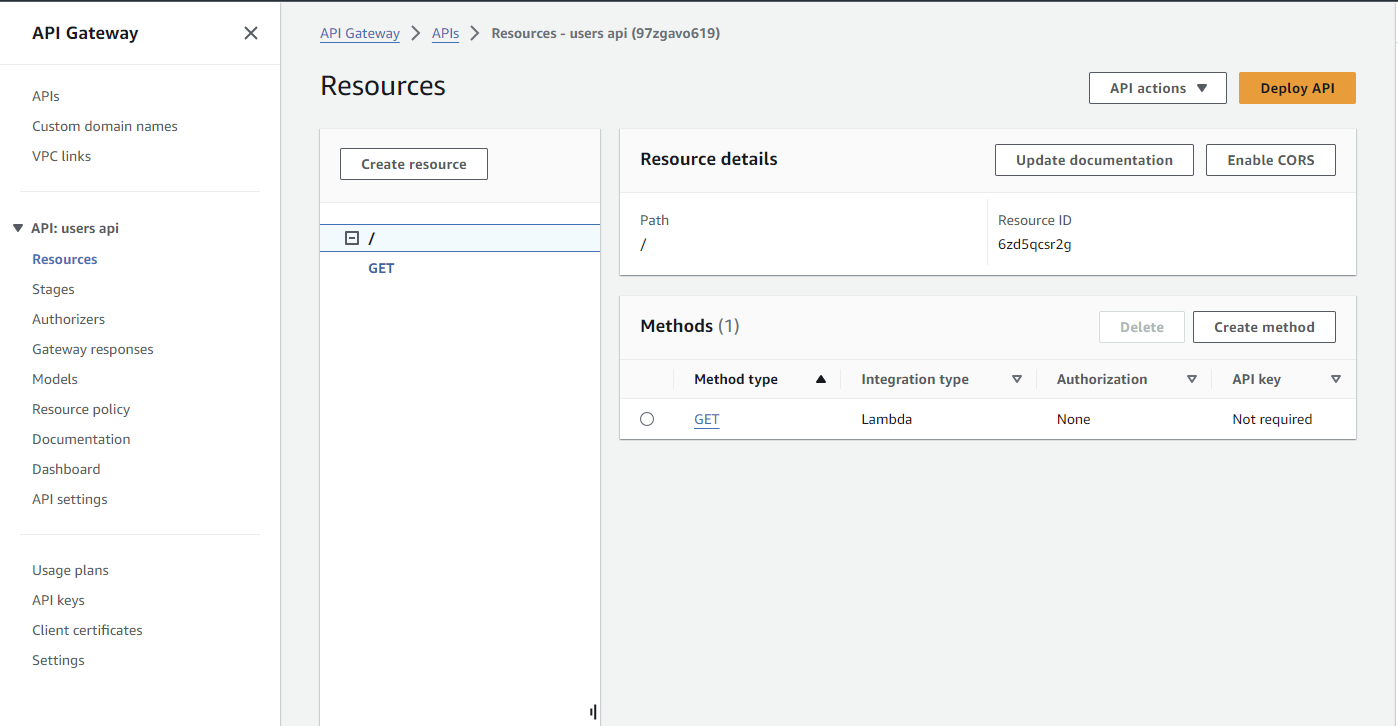
**Step-4:** Select the method type as “GET”, the region(make sure it is as same as the s3 bucket region), and select the lambda function.



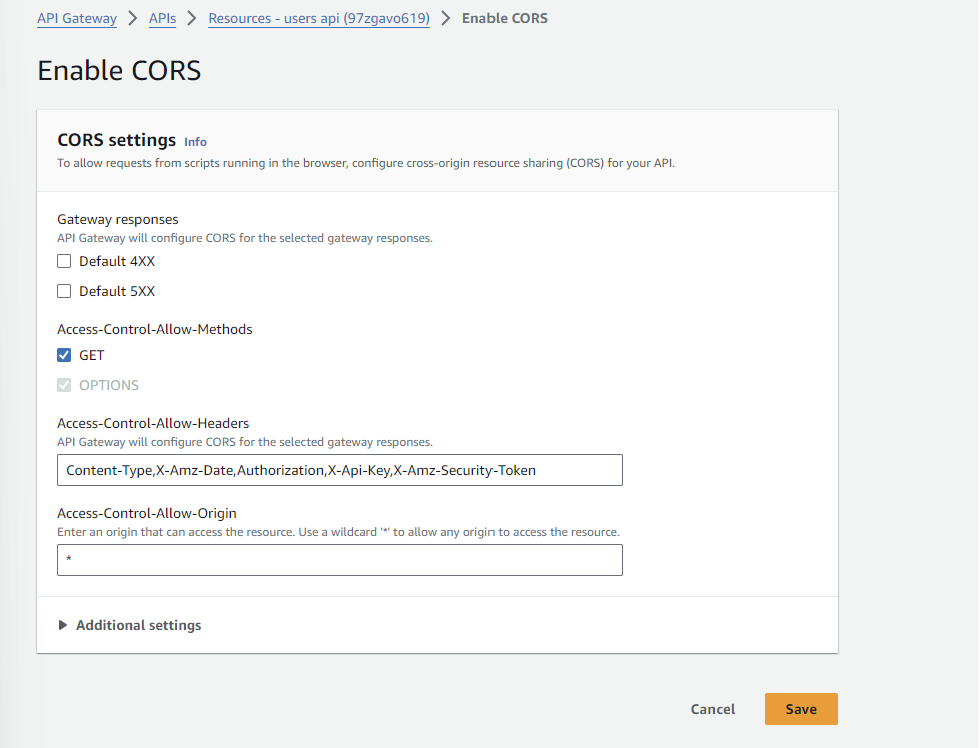
**Step-5:** Now click on “Deploy API”



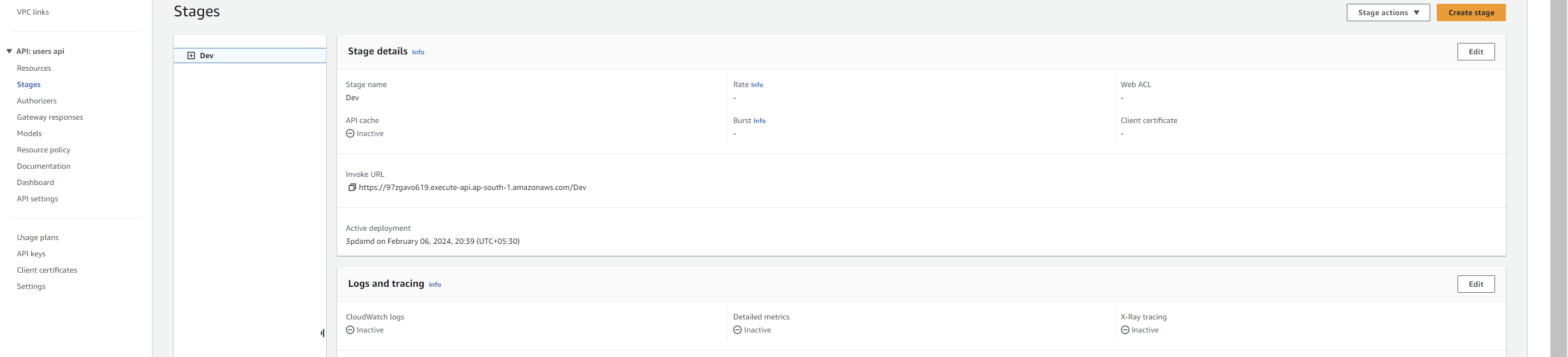




**Step-6:** Now click on “Enable CORS” and select the GET option and save. Then deploy it again.



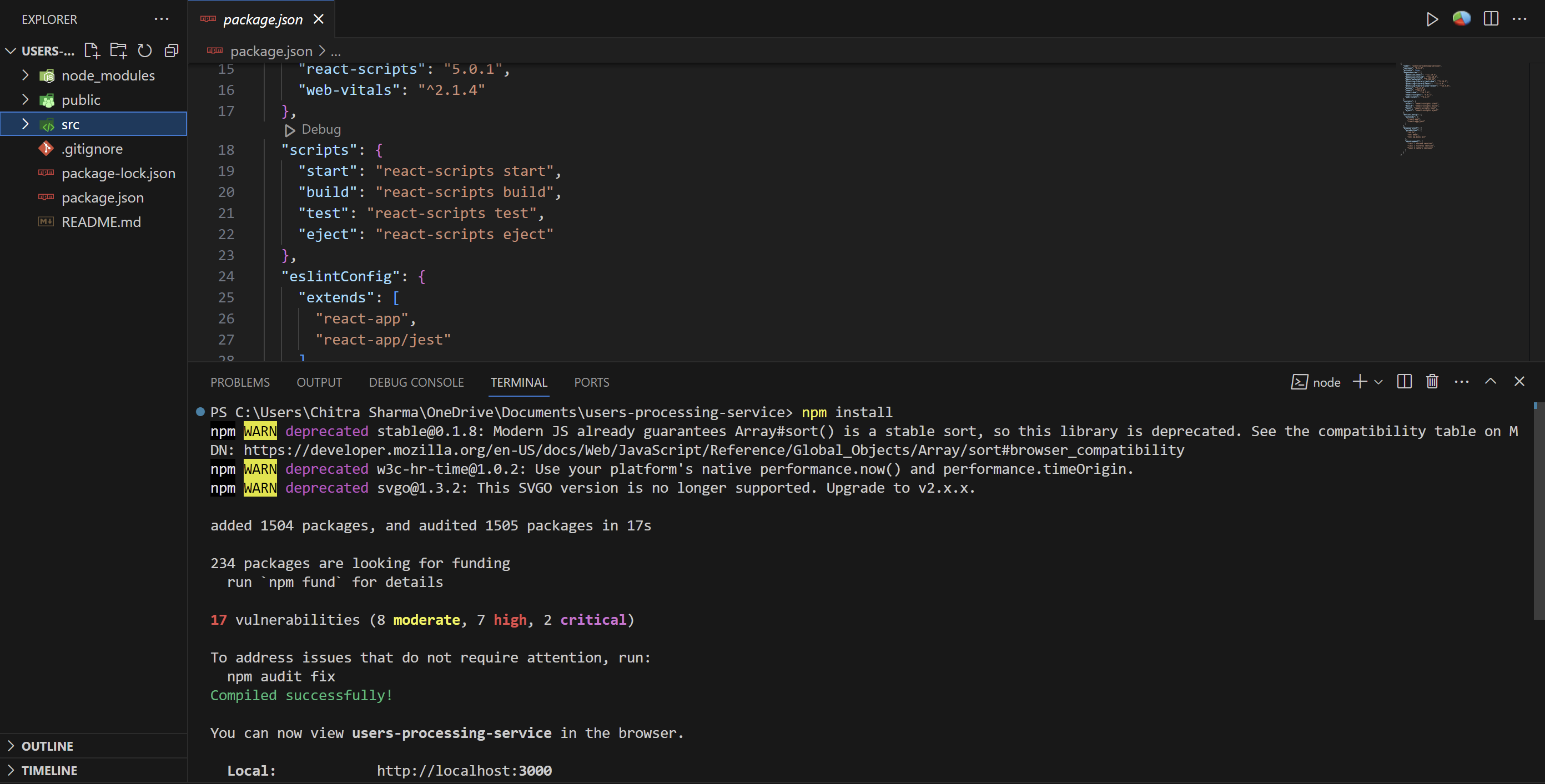
**Step-7:** Copy the “Invoke URL” from here. It will be used later.



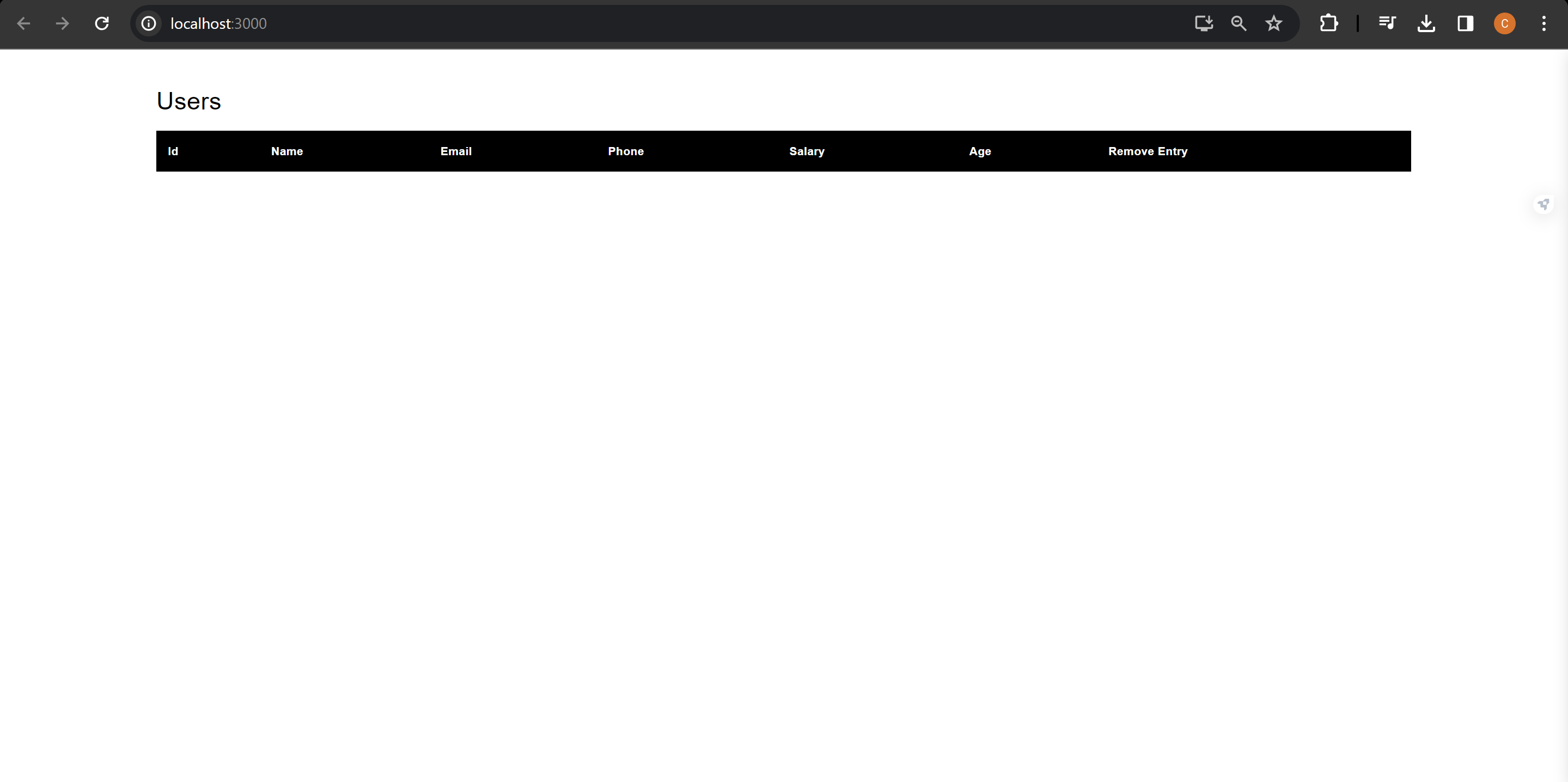
**Simple Storage Service(S3)**

**Step-1:** Unzip the code files and open terminal and in code directory run **npm install**. Then **node index.js** to check if the application is working or not.

**Note:** Make sure node is installed on your system.

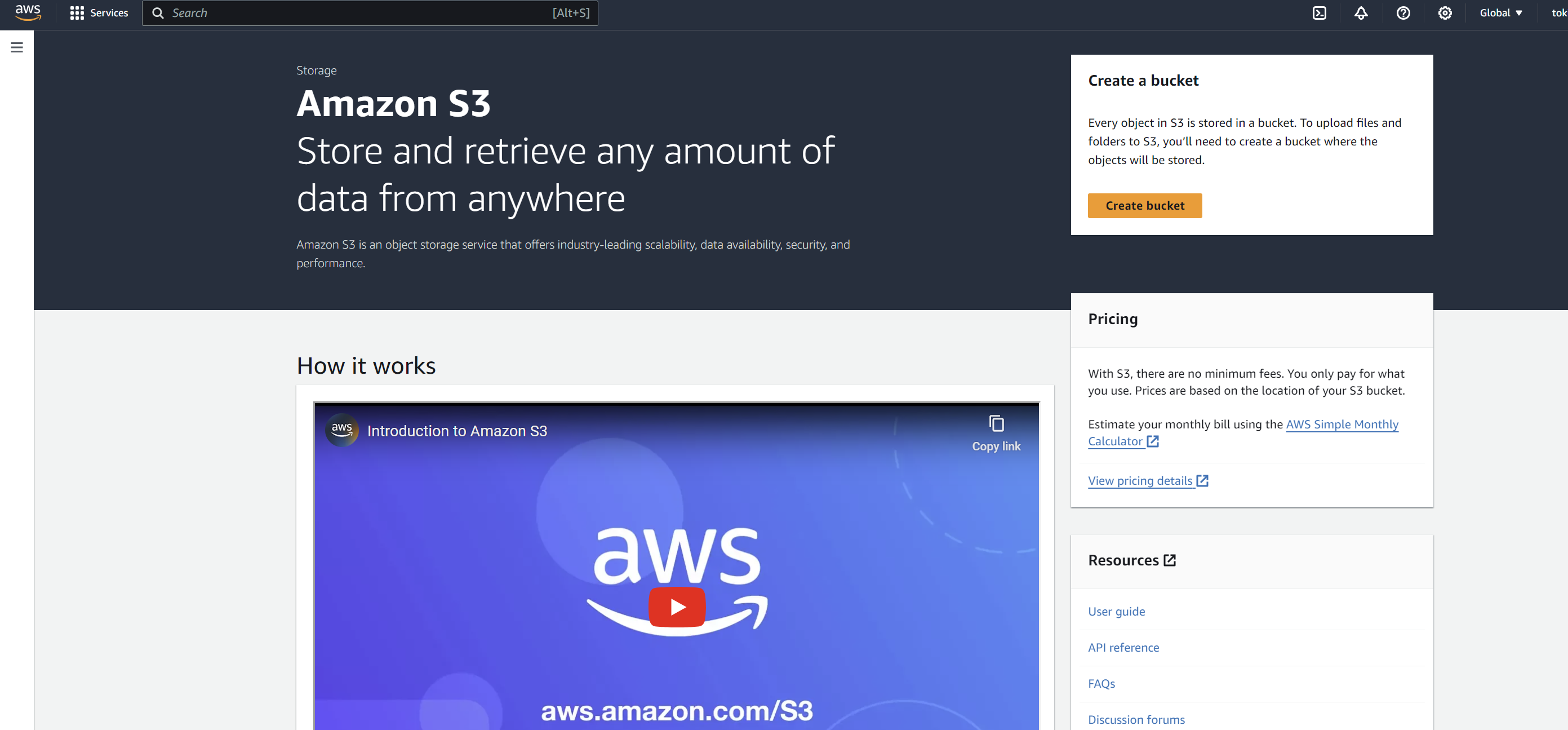


**Step-2:** After running node index.js, the application is accessible on localhost:3000

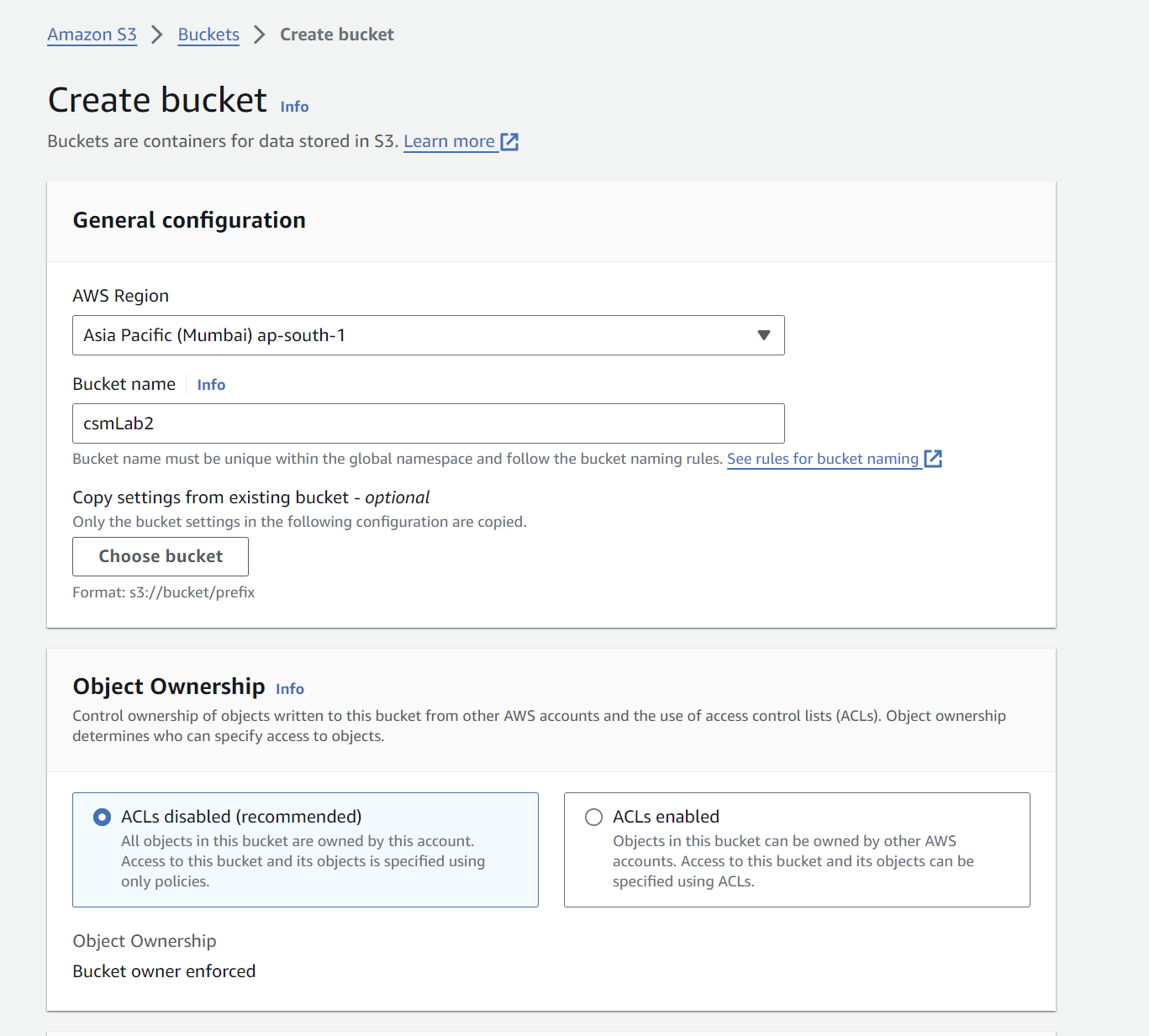


**Step-3:** Now run “**npm run build**” to create a build of your application that can be uploaded to S3 bucket.

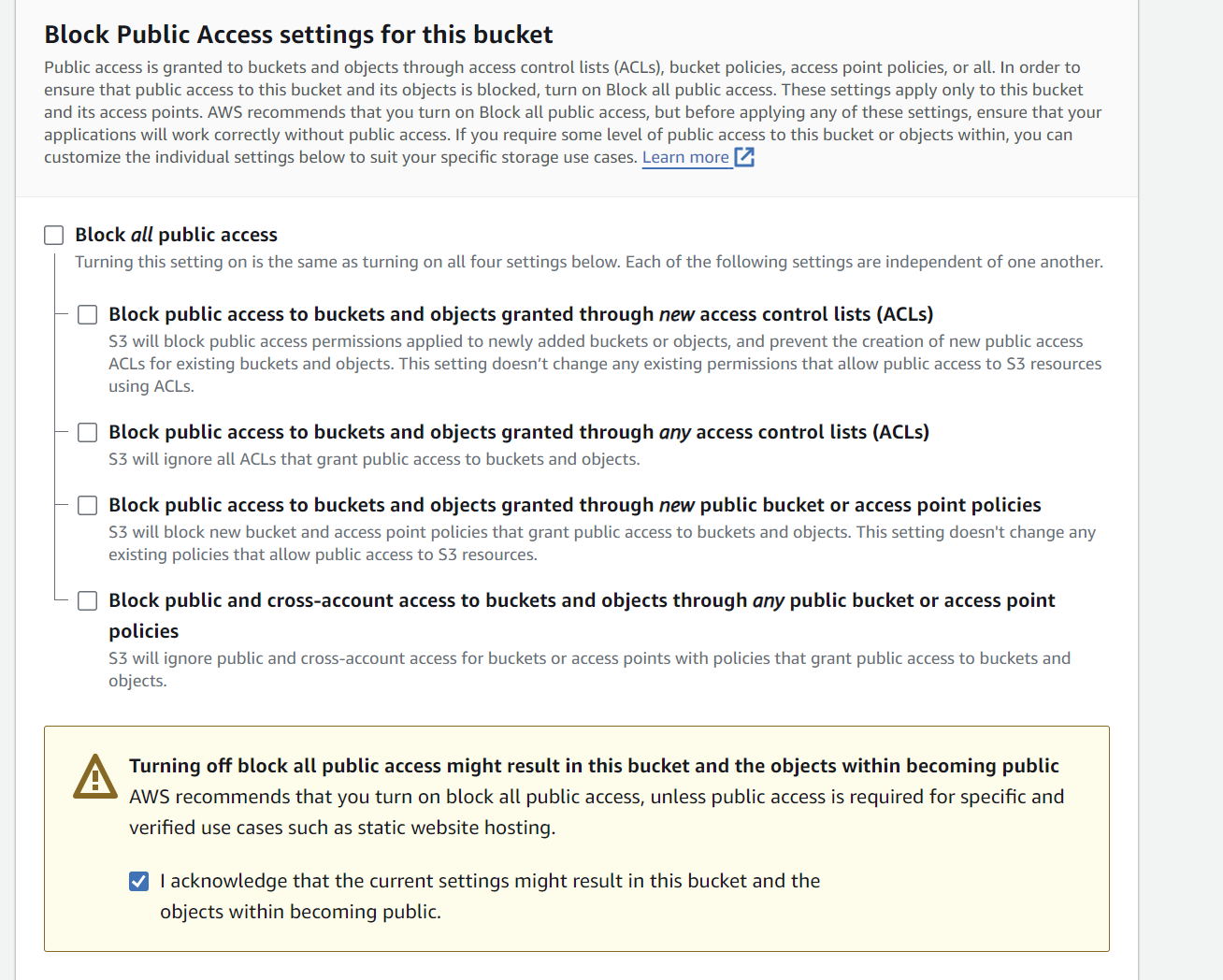
**Step-4:** Go to S3 and click on “Create bucket”.



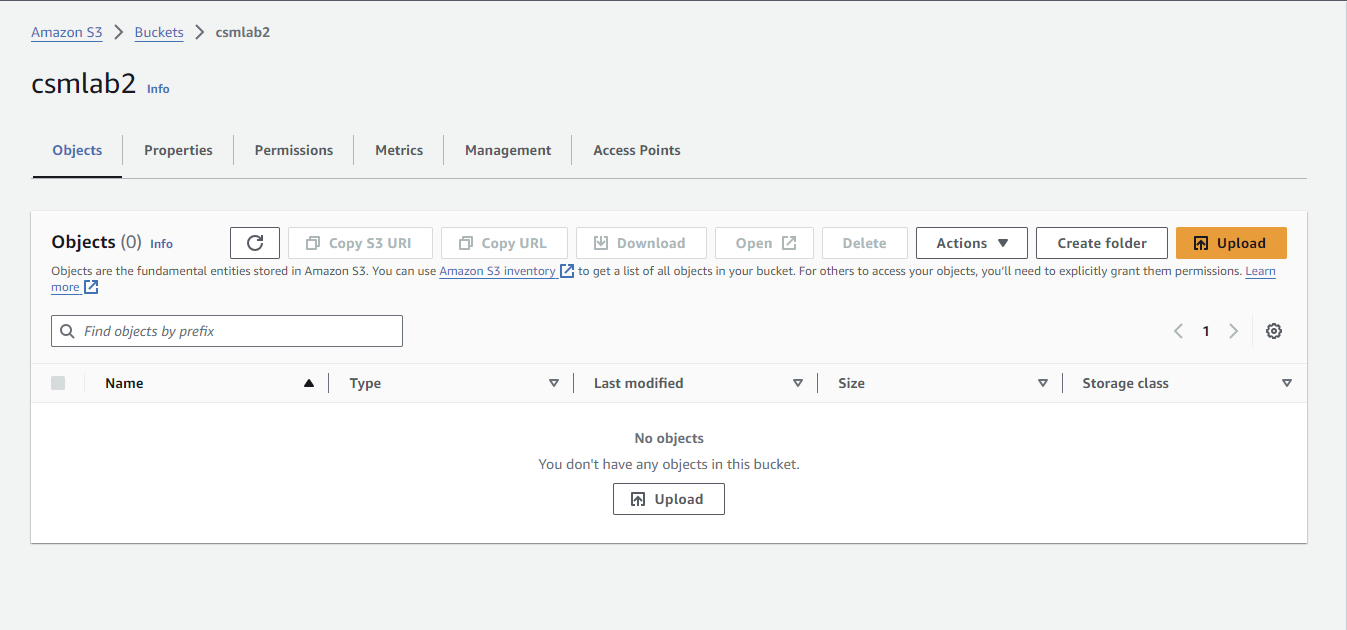
**Step-5:** Choose the unique bucket name and bucket region.



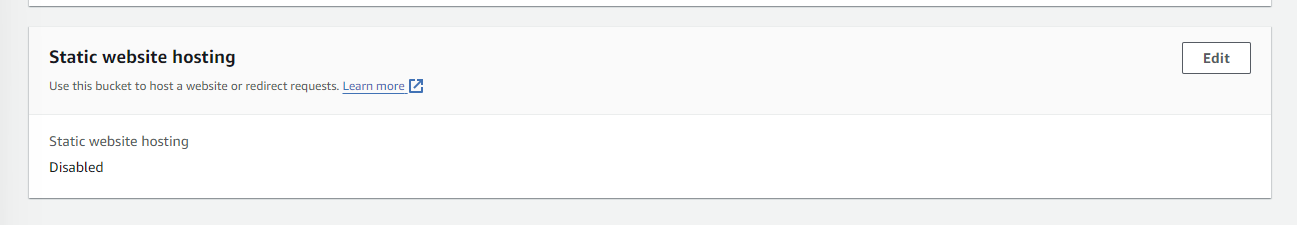
**Step-6:** Make sure you uncheck the box for “Block all public access”



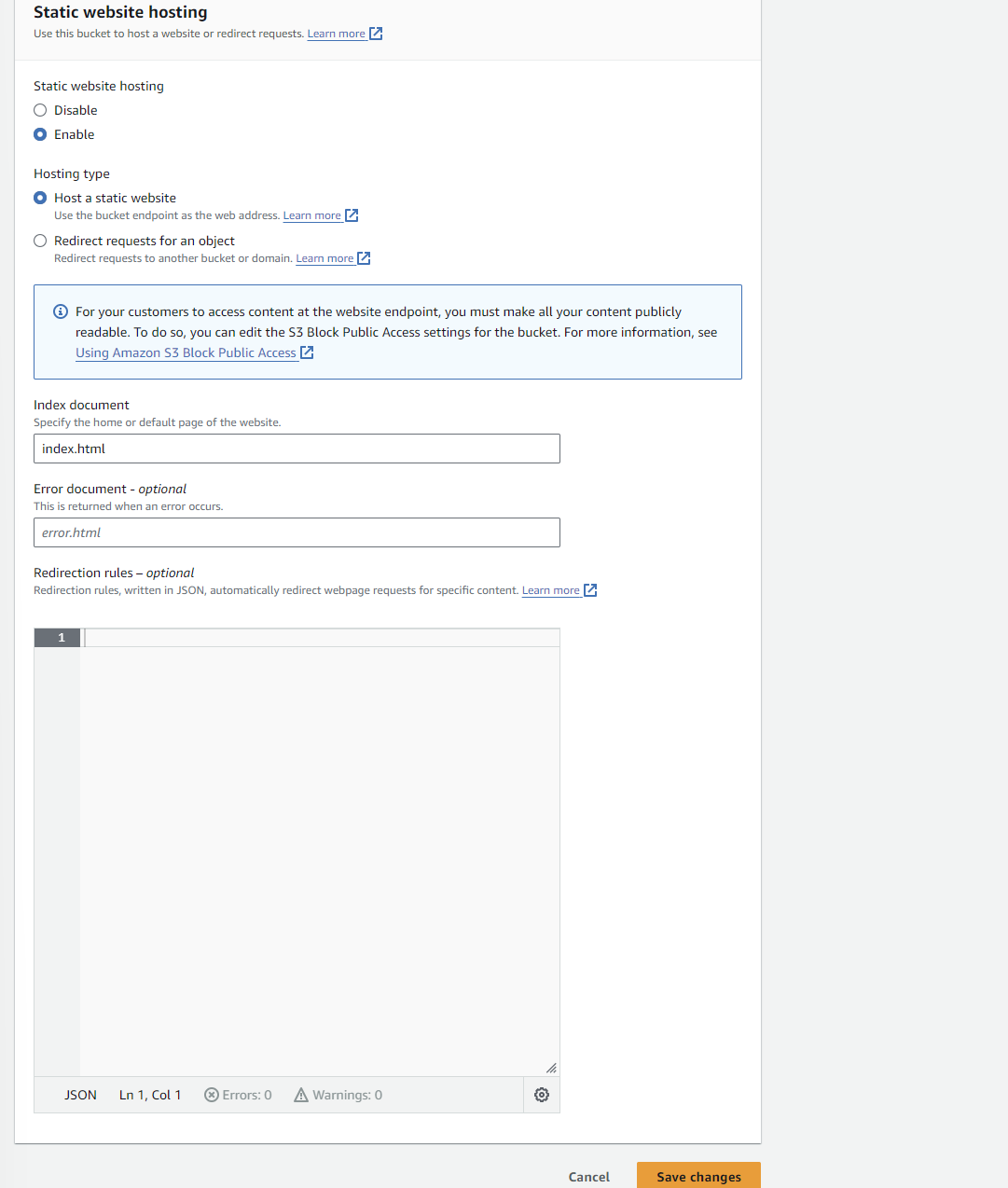
**Step-7:** Now the bucket is created. Go to the “Properties” tab .



**Step-8:** Scroll down to the bottom and edit the static website hosting settings.



**Step-9**: Enable the static website hosting option and in the index document, write “index.html”.



**Step-10:** Now choose the “Permissions” tab and edit the bucket policy.

{

"Version": "2012-10-17",

"Statement": [

{

"Sid": "Statement1",

"Principal": "\*",

"Effect": "Allow",

"Action": [

"s3:GetObject",

"s3:PutObject",

"s3:PutObjectAcl",

"s3:DeleteObject",

"s3:ListBucket"

],

"Resource": [

"arn:aws:s3:::csmlab2/\*",

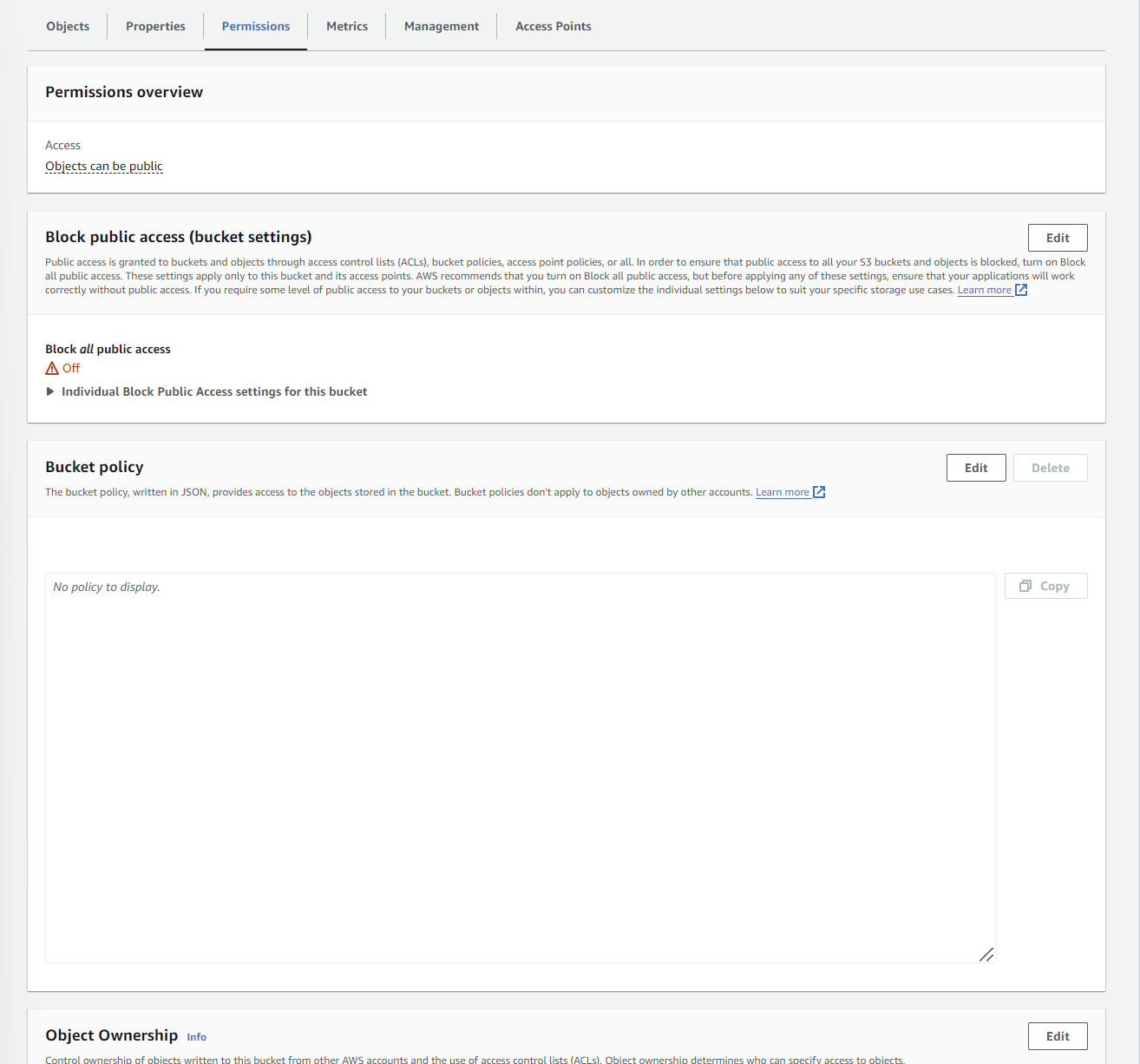
"arn:aws:s3:::csmlab2"

]

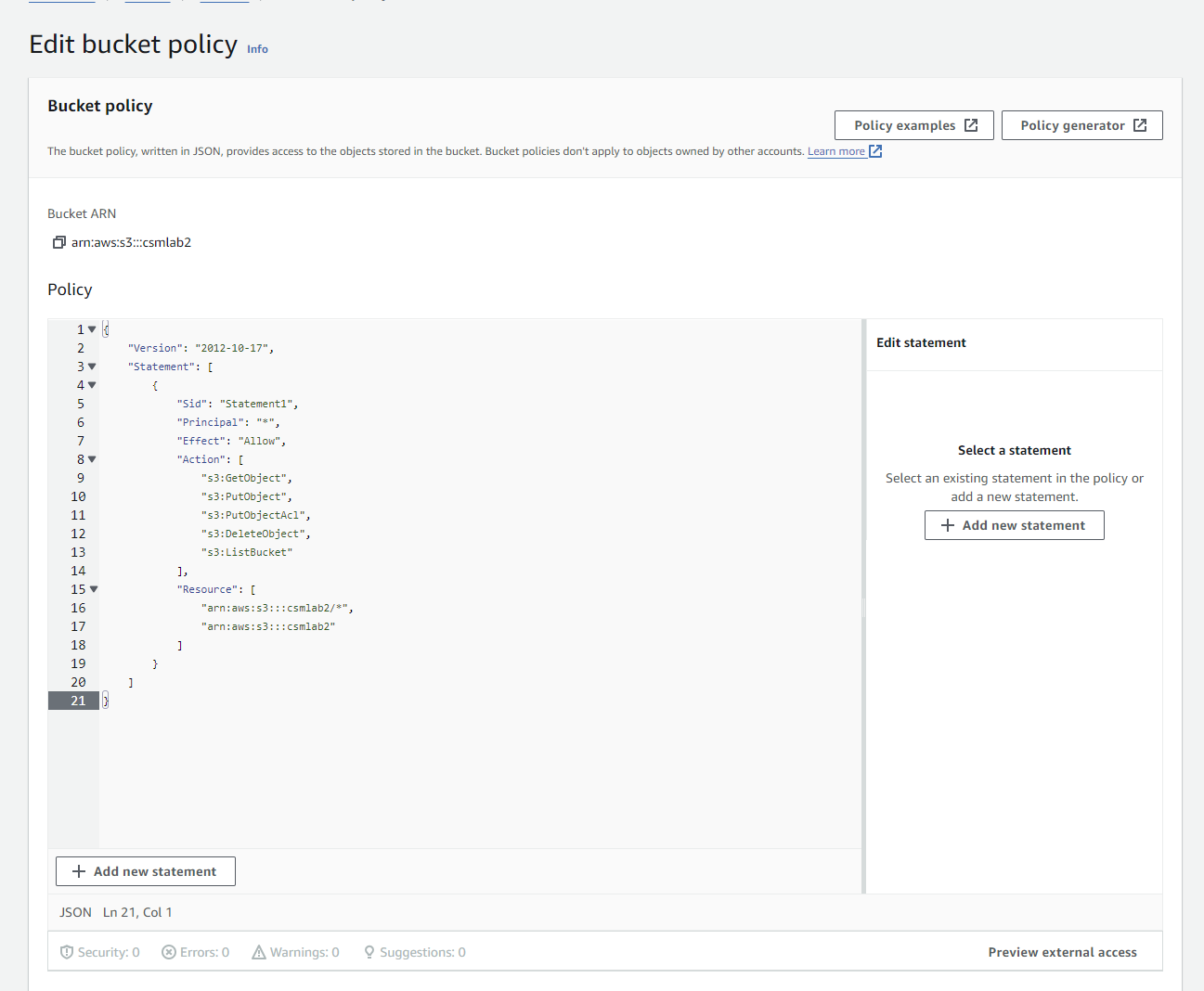
}

]

}

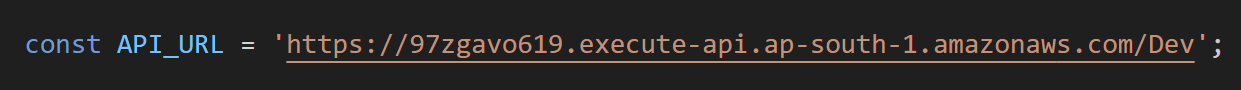


**Step-11:** Add the above JSON and save the changes**.**

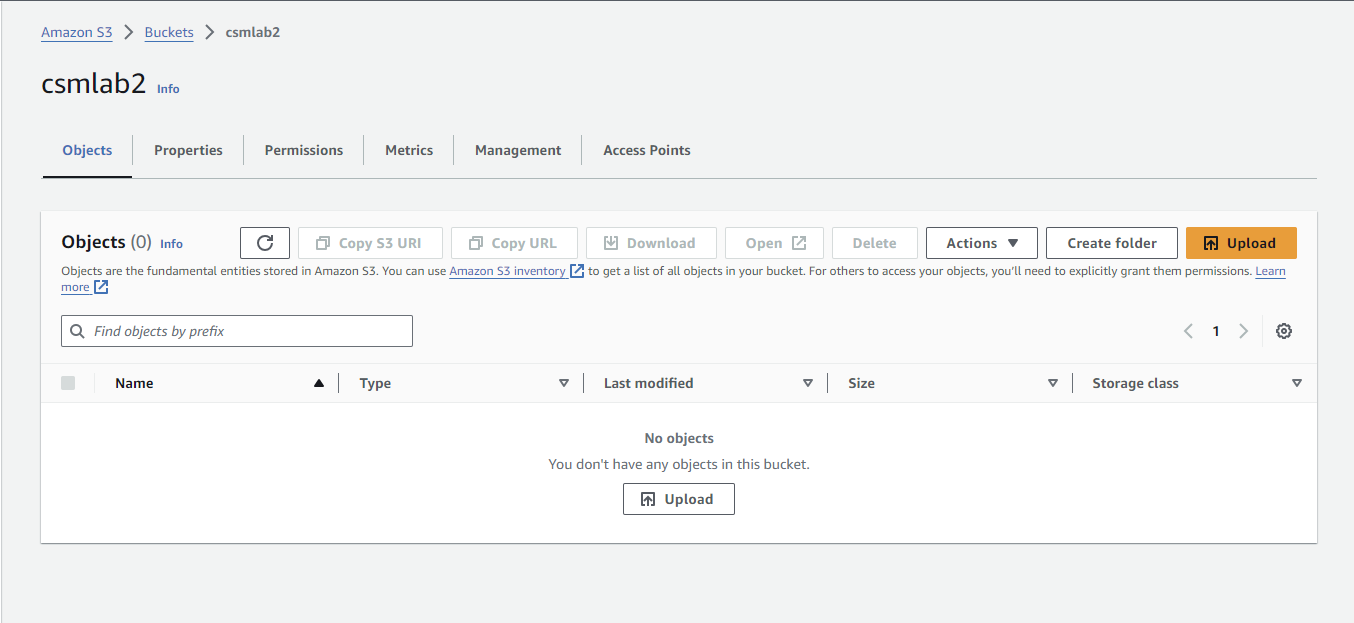


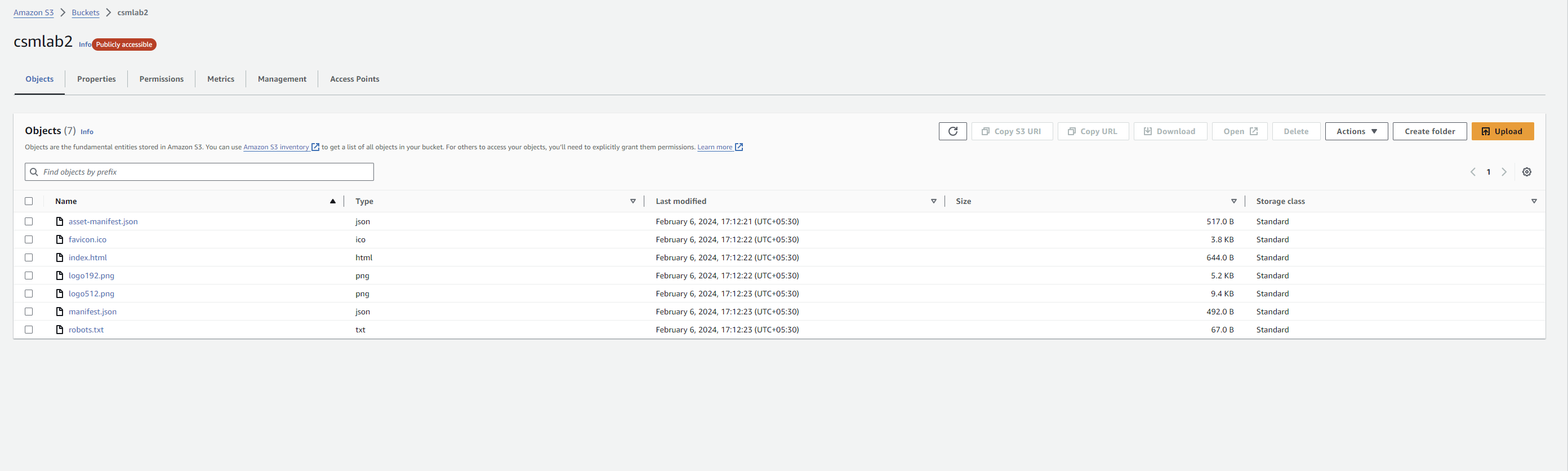
**NOTE:** Now before uploading the files and folder to the s3 bucket, change the API URL with your invoke URL that you have copied earlier from the API gateway in the src ->components -> Users.jsx file.

Here:

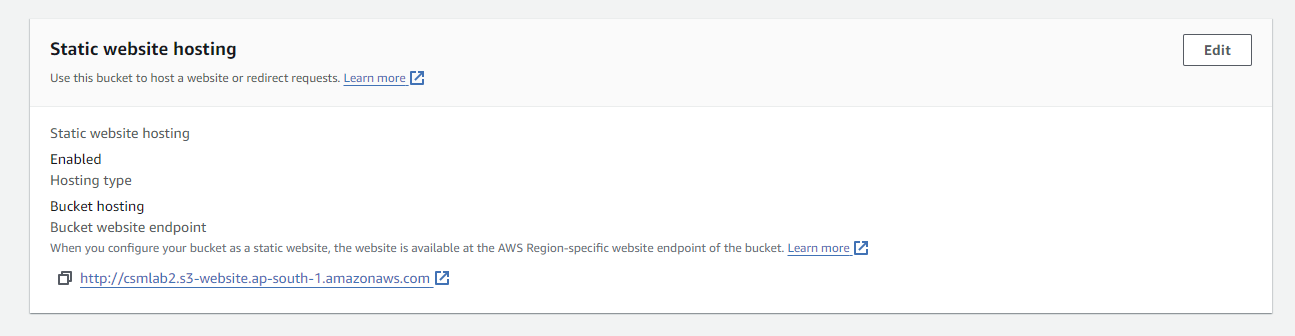


**Step-12:** Now upload the files and folder inside the build folder that you created locally.





**Step-13:** Now access you website using the bucket website endpoint given in the properties tab at the bottom.



**Step-14:** Now we can see that it is publicly accessible and it is also fetching data from the DynamoDB table that we created. We populated the table with one entry and it is getting displayed here.

