



Build a Three-Tier Web App



tahirgroot@gmail.com

User Information

1

Get User Data

```
{  
  "email": "test@example.com",  
  "name": "Test User",  
  "userId": "1"  
}
```

Introducing Today's Project!

In this project, I will demonstrate how to setup a three tier web app from scratch! I'll start with the presentation tier, then set up the logic tier and finally set up the data tier before tying them all together.

Tools and concepts

Services I used were Amazon S3, Cloudfront, DynamoDB, Lambda, API Gateway, Key concepts I learnt include Lambda functions, CORS errors, updating the javascript file with the API invoke URL, and testing the invoke URL within the browser

Project reflection

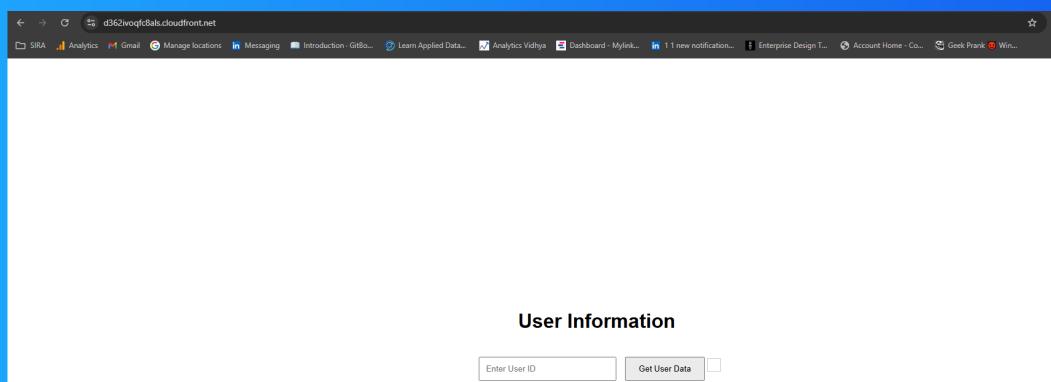
This project took me approximately 2 hours. The most challenging part was resolving the cors errors in both API Gateway and Lambda. It was most rewarding to see the final outcome - data getting returned in my web app.

I did this project to learn about the three tier architecture and setup my own webapp. This project absolutely met my goals and I could see a fully functioning web app by the end.

Presentation tier

For the presentation tier, I will set up how my website will be displayed and available to my end users. This is because the presentation tier is responsible for storing my website's files(Amazon S3) + website distribution (Amazon Cloudfront).

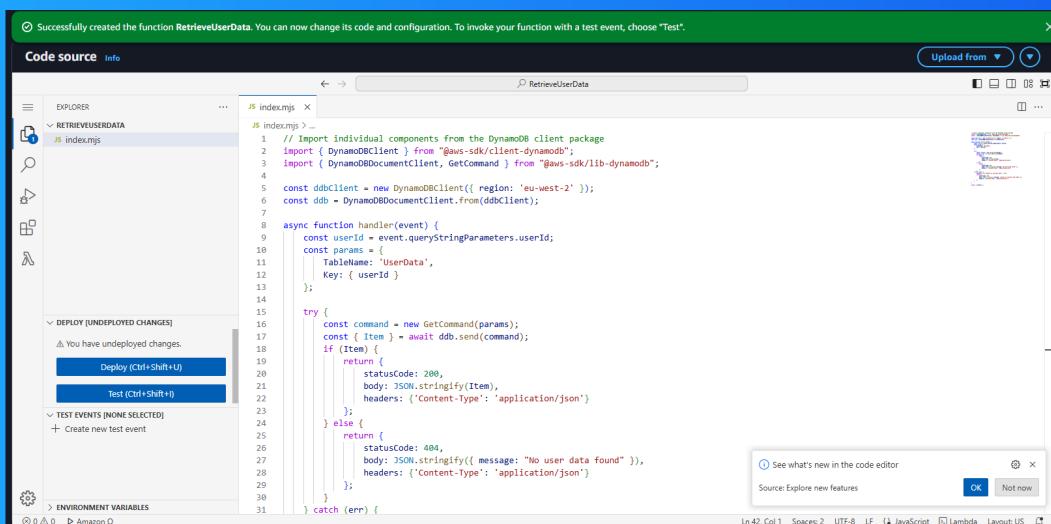
I accessed my delivered website by visiting the cloudfront distribution url. This distribution url is working because I also set up an origin access control that lets my s3 bucket restrict access to only my cloudfront distribution.



Logic tier

For the logic tier, I will set up a lambda function to handle requests (e.g. look up userid to retrun some user data) and also an API with API Gateway to receive requests from the user and hand it over to Lambda.

The Lambda function retrieves data by looking up a userid (that the user enters over the web app) in DynamoDB. The aws SDK is used in the function code so we can use templates and libraries that lets us find the correct DynamoDB table + request data.



The screenshot shows the AWS Lambda code editor interface. At the top, a green banner indicates "Successfully created the function RetrieveUserData. You can now change its code and configuration. To invoke your function with a test event, choose "Test". Below this, the "Code source" tab is selected, showing the file "index.mjs". The code itself is a JavaScript function that uses the AWS SDK's DynamoDB client to query a table named "UserData" for a specific user ID. It handles both successful and failed queries, returning the user data or an error message respectively. On the left sidebar, there are sections for "EXPLORER", "TEST EVENTS (NONE SELECTED)", and "ENVIRONMENT VARIABLES". A tooltip at the bottom right encourages users to "See what's new in the code editor" and "Explore new features".

```
JS index.mjs
1 // Import individual components from the DynamoDB client package
2 import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
3 import { DynamoDBDocumentClient, GetCommand } from "@aws-sdk/lib-dynamodb";
4
5 const ddclient = new DynamoDBClient({ region: 'eu-west-2' });
6 const ddb = DynamoDBDocumentClient.from(ddclient);
7
8 async function handler(event) {
9     const userId = event.queryStringParameters.userId;
10    const params = {
11        TableName: "UserData",
12        Key: { userId }
13    };
14
15    try {
16        const command = new GetCommand(params);
17        const [item] = await ddb.send(command);
18        if (item) {
19            return {
20                statusCode: 200,
21                body: JSON.stringify(item),
22                headers: {"Content-Type": "application/json"}
23            };
24        } else {
25            return {
26                statusCode: 404,
27                body: JSON.stringify({ message: "No user data found" }),
28                headers: {"Content-Type": "application/json"}
29            };
30        }
31    } catch (err) {
32    }
}
```

Data tier

For the data tier, I will set up a dynamoDB database that stores user data. At the moment there is no user data for us to return to our web app's users. The data in my database will get returned once i setup Dynamo DB and connect it with Lambda.

The partition key for my DynamoDB table is userid. which means that when my table looks up data, it will look it up based on userid. Then, it can return all data(values) related to the item with that ID.



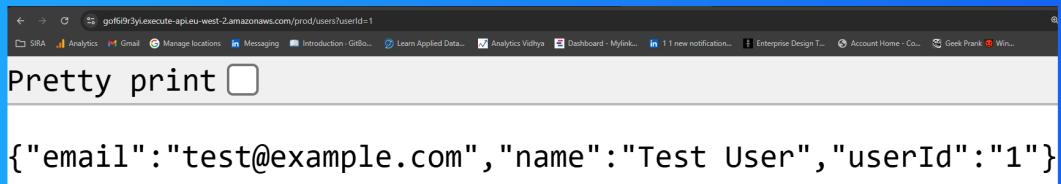
A screenshot of a DynamoDB item editor. The top bar has a dark background with the word "Attributes" in white and a "View DynamoDB JSON" button. Below the bar is a code editor window with a dark theme. The code shows a single object with five numbered lines:

```
1 ▼ {}
2   "userId": "1",
3   "name": "Test User",
4   "email": "test@example.com"
5 }
```

Logic and Data tier

Once all three layers of my three-tier architecture are set up, the next step is to connect the presentation and logic tier. This is because there is no way for my API to catch requests that users make through my distributed site!

To test my API, I visited the invoke URL of the prod stage API. This lets us test whether I can use the API and retrieve user data. The results were some user data in JSON when we looked up userId=1. This proved a logic + data tier connection.



A screenshot of a web browser window. The address bar shows a URL starting with "gof6s9r3y.execute-api.eu-west-2.amazonaws.com/prod/users?userId=1". The page content is a JSON object:

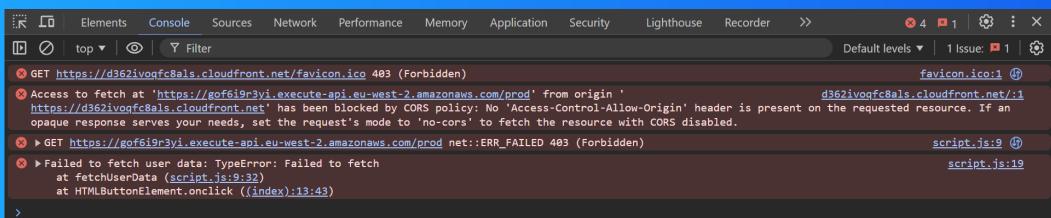
```
{"email": "test@example.com", "name": "Test User", "userId": "1"}
```

Console Errors

The error in my distributed site was because there was an error within script.js (one of the website files i had uploaded into s3). The script.js file is referencing a prod stage API URL placeholder and not my API's actual URL.

To resolve the error, I updated script.js by replacing some placeholder text with the API's prod stage invoke URL. I then reuploaded script.js into it into S3 because the S3 bucket is still storing the last uploaded version(i.e. with the error).

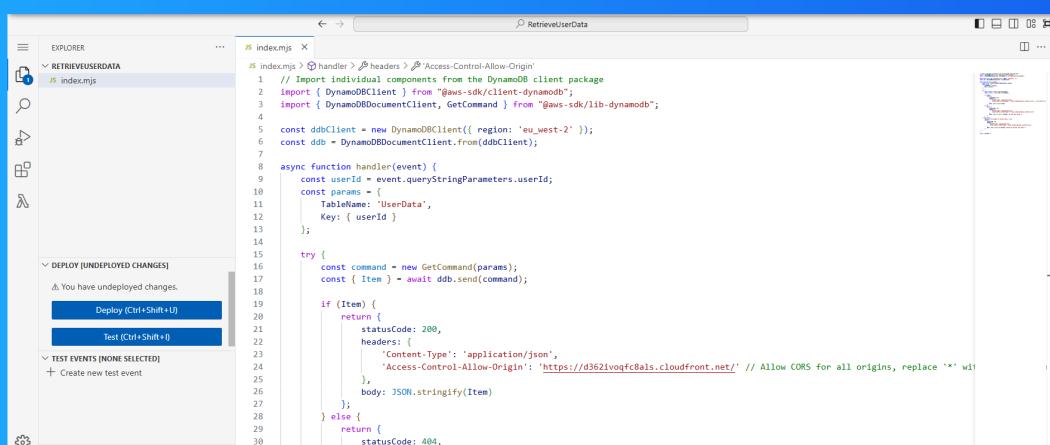
I ran into a second error after updating script.js. This was an error on the browser side with CORS because API Gateway, by default, is only configured to allow requests from users directly running its invoke URL in the browser(now with cloudfront).



Resolving CORS Errors

To resolve the CORS error, I first went into my API(API GATEWAY) and enabled CORS on the /users resource. I then made sure GET requests are enabled, and referenced my CloudFront domain as the domain getting access.

I also updated my Lambda function because it needs to be able to return CORS headers to show that it has the permission to invoke the API's invoke URL and return a response. This means adding 'Access-Control-Allow-Origin' as a header in the response.

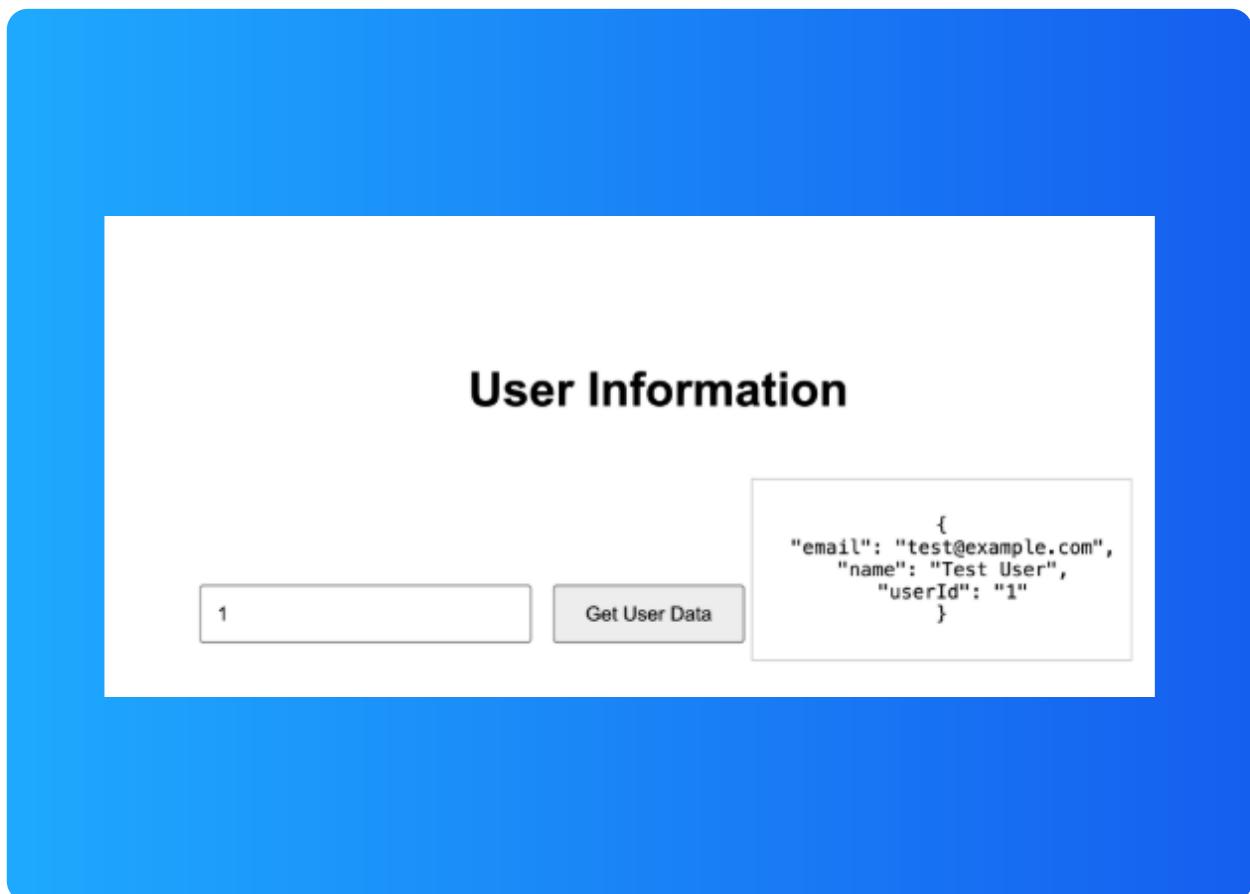


The screenshot shows the AWS Lambda function editor for a function named "RetrievalUserData". The code in the "index.js" file is as follows:

```
index.js
 1 // Import individual components from the DynamoDB client package
 2 import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
 3 import { DynamoDBDocumentClient, GetCommand } from "@aws-sdk/lib-dynamodb";
 4
 5 const ddbClient = new DynamoDBClient({ region: 'eu-west-2' });
 6 const ddb = DynamoDBDocumentClient.from(ddbClient);
 7
 8 async function handler(event) {
 9   const userId = event.queryStringParameters.userId;
10   const params = {
11     TableName: 'UserData',
12     Key: { userId }
13   };
14
15   try {
16     const command = new GetCommand(params);
17     const { Item } = await ddb.send(command);
18
19     if (Item) {
20       return {
21         statusCode: 200,
22         headers: {
23           'Content-Type': 'application/json',
24           'Access-Control-Allow-Origin': 'https://d362ivogfc8als.cloudfront.net/' // Allow CORS for all origins, replace '*' with your origin
25         },
26         body: JSON.stringify(Item)
27       };
28     } else {
29       return {
30         statusCode: 404,
31         body: JSON.stringify({ message: 'User not found' })
32       };
33     }
34   } catch (error) {
35     console.error(error);
36     return {
37       statusCode: 500,
38       body: JSON.stringify({ message: 'Internal server error' })
39     };
40   }
41 }
```

Fixed Solution

I verified the fixed connection between API Gateway and CloudFront by looking up user data in the distributed site again. In my final test user data could be returned - so a user request in the presentation tier gets data from the data tier.





NextWork.org

Everyone should be in a job they love.

Check out nextwork.org for
more projects

