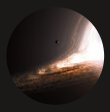
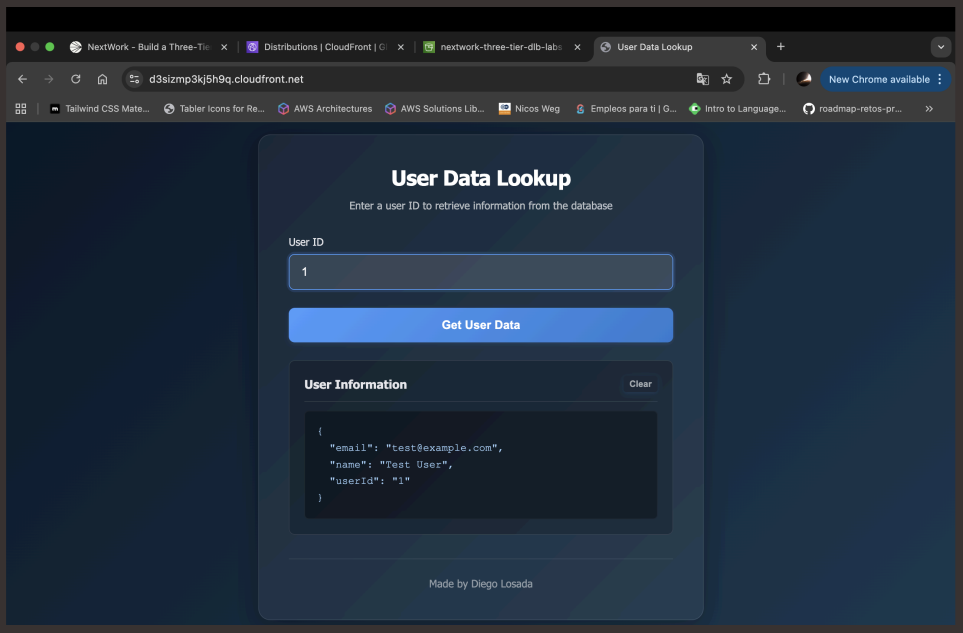
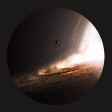
nextwork.org

Build a Three-Tier Web App



Diego Losada





Diego Losada
NextWork Student

Introducing Today's Project!

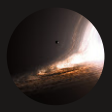
In this project, I will demonstrate how to build a full three-tier serverless application using AWS services. I'm doing this project to learn how to: - Create a storage bucket for website files using S3. - Distribute content globally and improve performance with CloudFront. - Implement serverless backend logic with Lambda functions. - Build APIs to handle user requests using API Gateway. - Store and retrieve user data securely with DynamoDB. - Connect all these services seamlessly to create a scalable three-tier architecture.

Tools and concepts

Services I used were AWS S3, CloudFront, Lambda, API Gateway, and DynamoDB. Key concepts I learnt include serverless architecture, three-tier architecture, API integration, DynamoDB tables, Lambda function permissions, and caching with CloudFront.

Project reflection

This project took me approximately 1.5 hours. The most challenging part was resolving the API connection issues between CloudFront and API Gateway and ensuring the Lambda function had the correct permissions. It was most rewarding to see the full three-tier application working end-to-end, with data properly fetched and displayed on my website.



Diego Losada
NextWork Student

nextwork.org

I chose to do this project today because I wanted to practice building a scalable serverless application and understand how the presentation, logic, and data tiers interact.

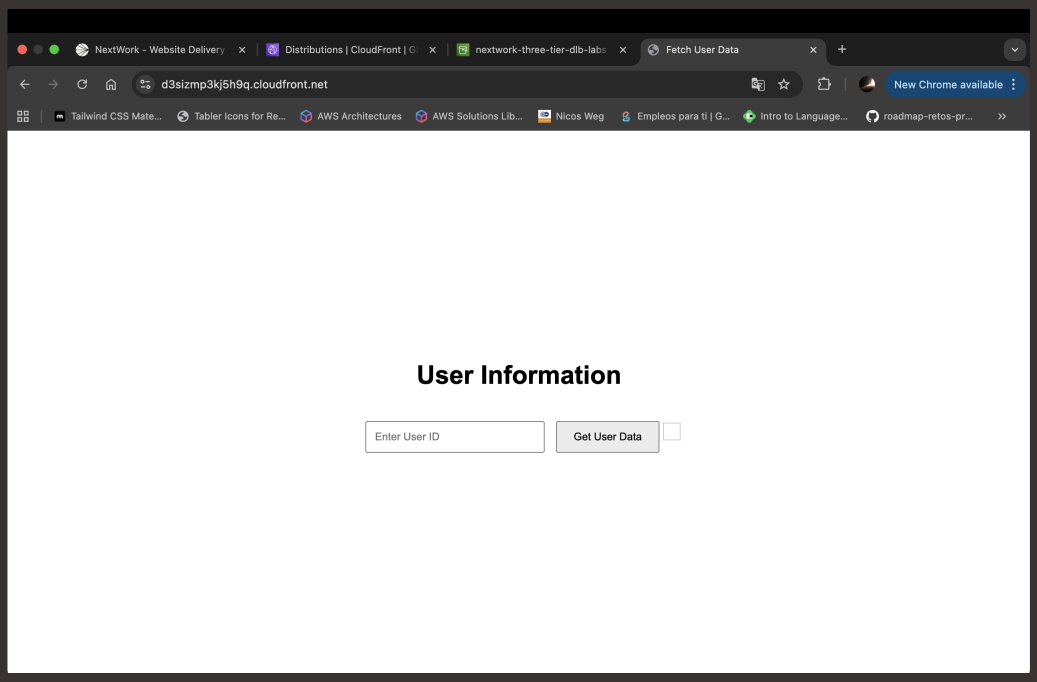
Diego Losada
NextWork Student

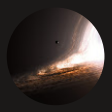
nextwork.org

Presentation tier

For the presentation tier, I will set up an S3 bucket to store my website's files and upload a simple index.html. I will also set up CloudFront to deliver the content globally. Because this ensures my website is accessible to users anywhere with fast load times, and it provides the foundation for the front-end layer of my three-tier architecture.

I accessed my delivered website by visiting the CloudFront distribution URL. This URL points to the content stored in my S3 bucket and ensures that the website loads quickly from anywhere in the world.





Diego Losada
NextWork Student

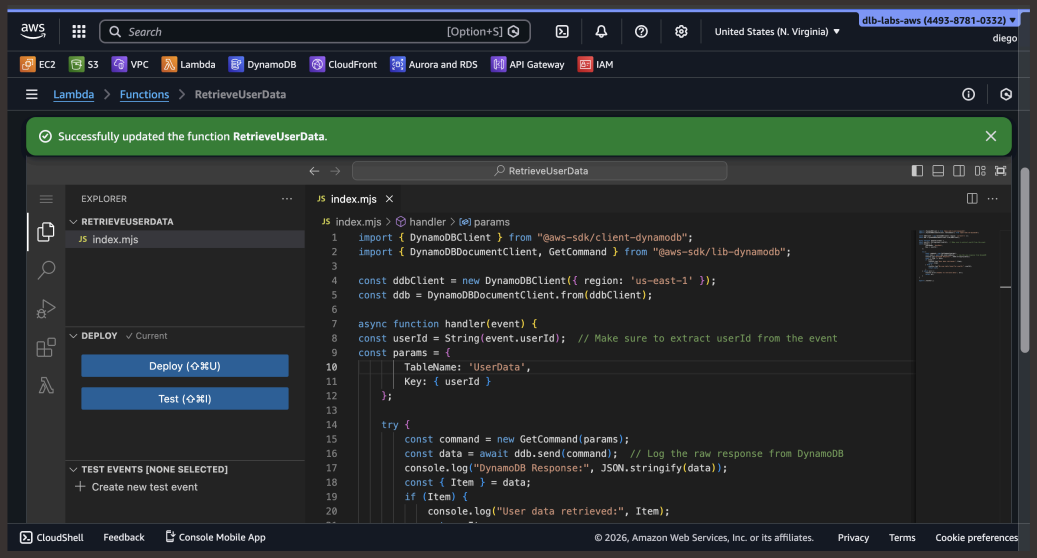
Logic tier

For the logic tier, I will set up a Lambda function and an API Gateway REST API because this allows my application to fetch and serve data from the DynamoDB database securely and serverlessly. The Lambda function contains the code that retrieves user data, and the API Gateway exposes this functionality through a GET endpoint that can be accessed by the front-end or other clients.

The Lambda function retrieves data by using the AWS SDK to query the DynamoDB table based on a provided userId. It sends a GetCommand to DynamoDB, receives the corresponding item if it exists, and returns that data to the API Gateway, which then delivers it to the front-end or any client making the request.

Diego Losada
NextWork Student

nextwork.org

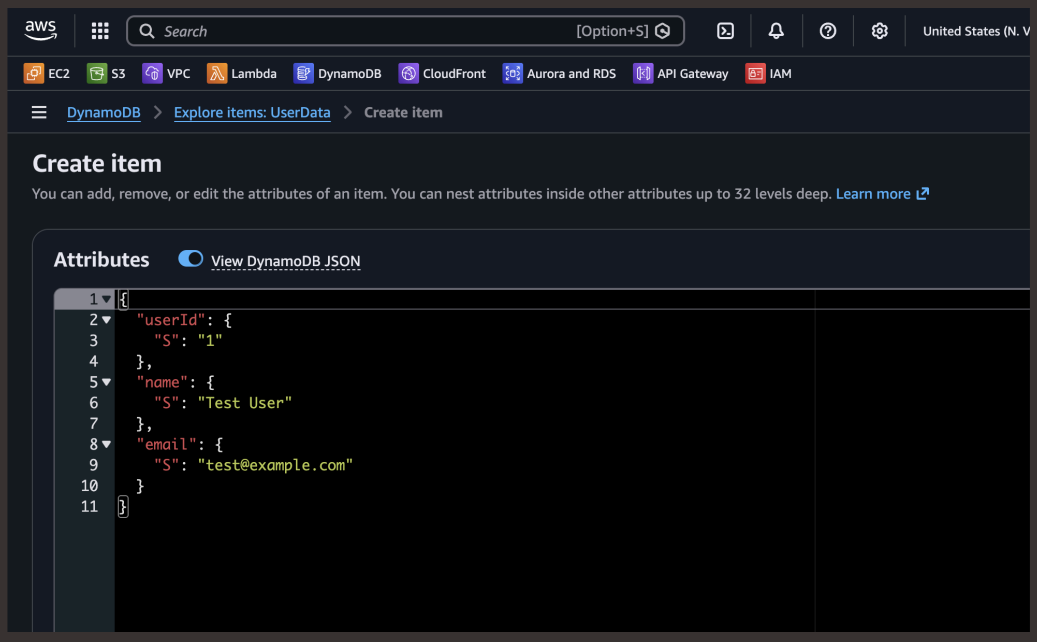


Diego Losada
NextWork Student

Data tier

For the data tier, I will set up a DynamoDB table because it provides a fully managed, scalable, and fast database to store and retrieve user data for my application. By adding user records to the table, my Lambda function can query and return the correct information when users make requests through the API.

The partition key for my DynamoDB table is userId, which means DynamoDB uses this value to uniquely identify and store each user's data and to quickly locate the correct item when my Lambda function queries the table.

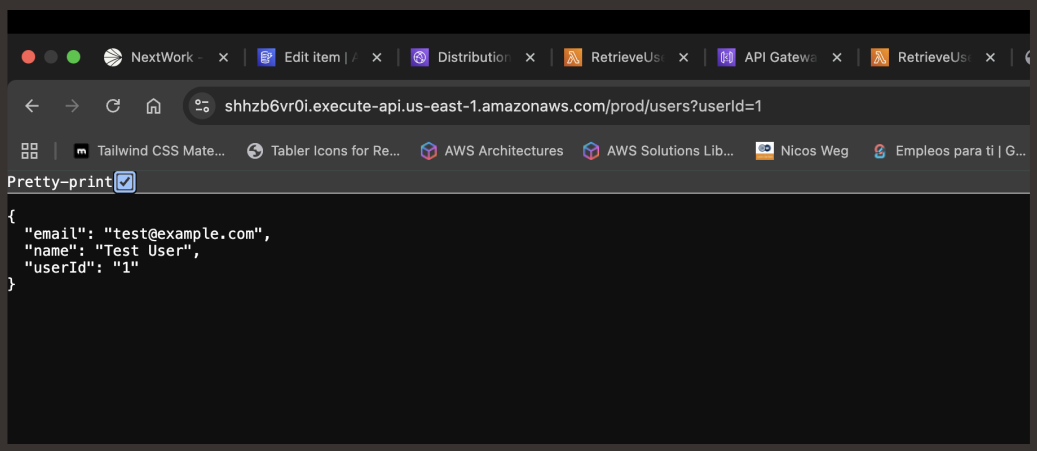


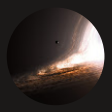
Diego Losada
NextWork Student

Logic and Data tier

Once all three layers of my three-tier architecture are set up, the next step is to connect them by updating my script.js to make API requests to my backend, because this allows the presentation layer to communicate with the logic and data layers and display real user data on the website.

To test my API, I ran the edited API Gateway URL in my web browser. The results were a successful response showing the user data returned from my DynamoDB table, confirming that the API and Lambda function were working correctly.





Diego Losada
NextWork Student

Console Errors

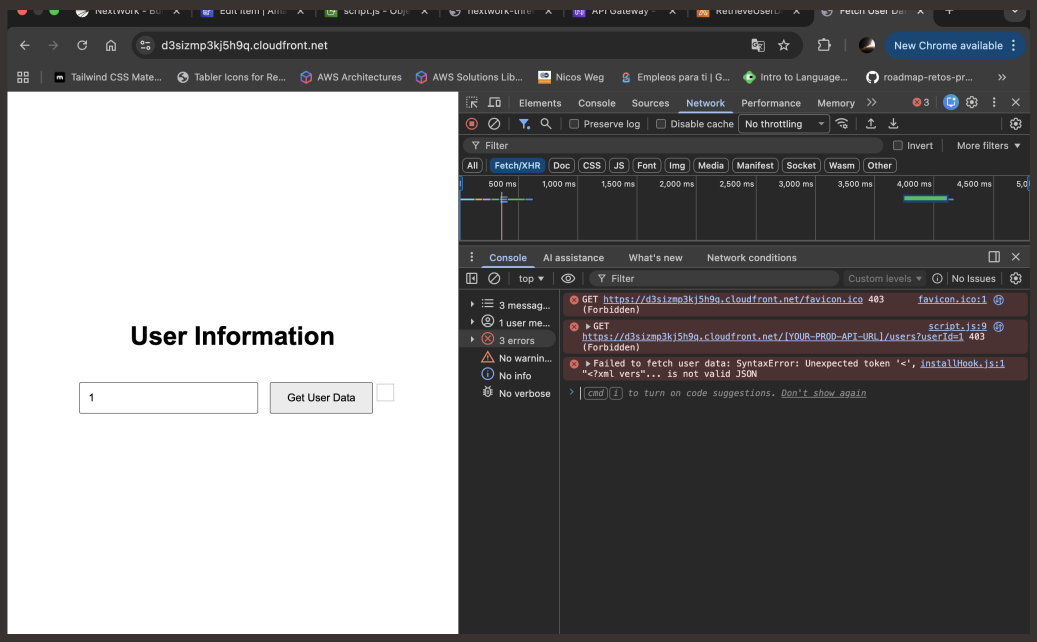
The error in my distributed site was because my script.js was still pointing to a placeholder API URL instead of the real API Gateway production URL, so the browser tried to call https://[YOUR-PROD-API-URL]/users?userId=1, which doesn’t exist and caused the request to fail.

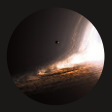
To resolve the error, I updated script.js by replacing the placeholder API URL with my real API Gateway production endpoint. I then reuploaded it into S3 because CloudFront serves the files from S3, and updating the bucket ensures the website uses the new JavaScript code when users load the page.

I ran into a second error after updating script.js. This was an error with the CloudFront distribution not being fully updated, and I hadn’t cleared my browser cache before testing again. As a result, the browser was still using an old version of script.js, which caused a 403 error when trying to fetch data and returned an HTML/XML page instead of valid JSON.

Diego Losada
NextWork Student

nextwork.org





Diego Losada
NextWork Student

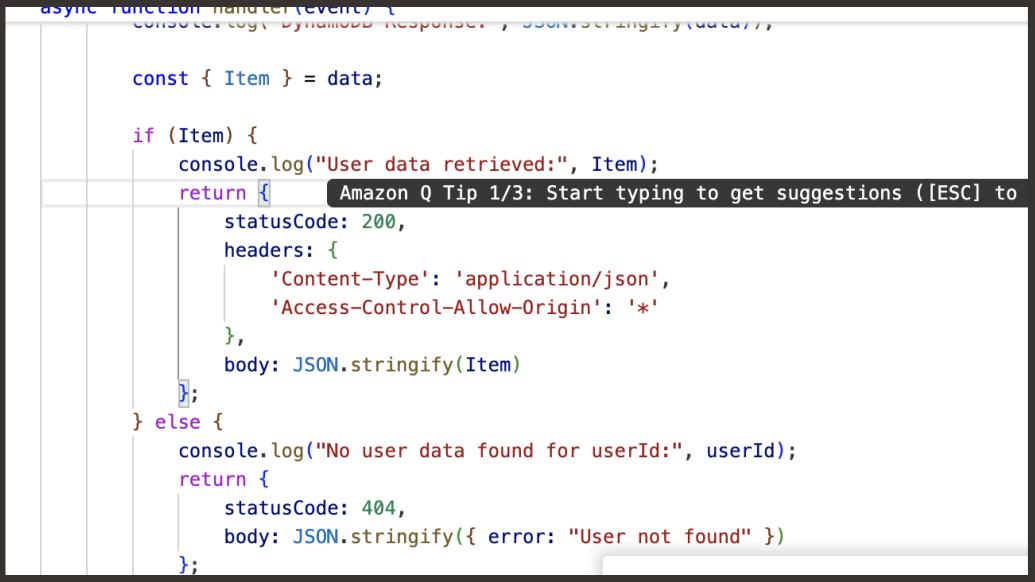
Resolving CORS Errors

Nothing, i didn't get the CORS error

I also updated my Lambda function because the API needed to return the proper CORS headers for the browser to accept responses from a different origin. The changes I made were: Added the Access-Control-Allow-Origin header with the value of * (or my CloudFront domain) in the Lambda function’s response. Added Access-Control-Allow-Methods and Access-Control-Allow-Headers to allow the GET request from the frontend. Ensured that these headers were included in both successful responses and error responses, so the browser wouldn’t block the requests.

Diego Losada
NextWork Student

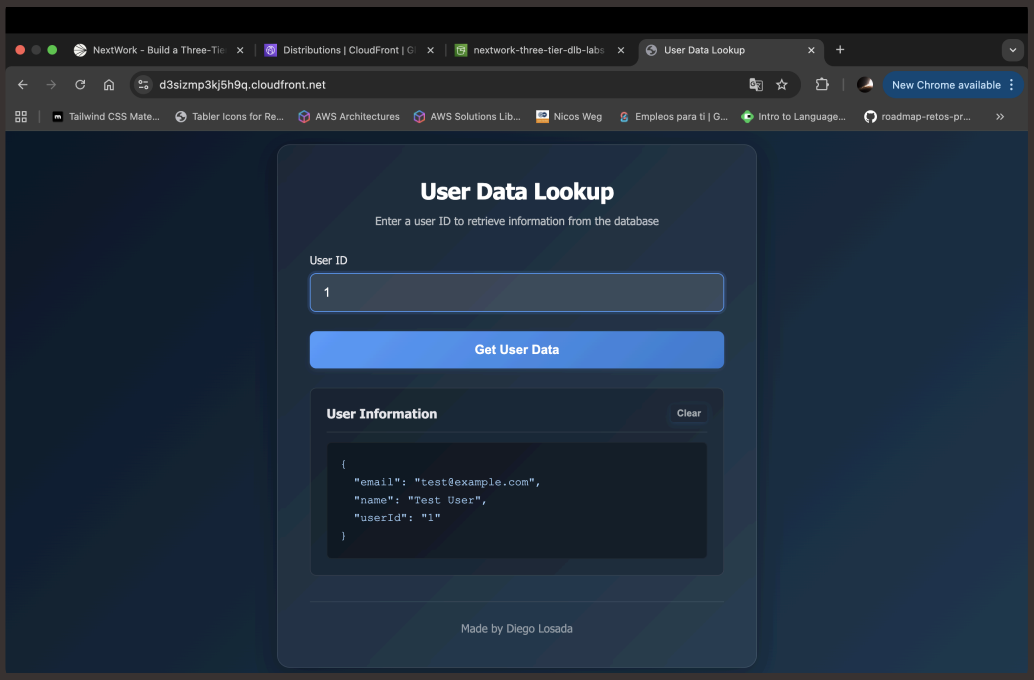
nextwork.org



Diego Losada
NextWork Student

Fixed Solution

I verified the fixed connection between API Gateway and CloudFront by clearing my browser cache, reloading the CloudFront URL, and confirming that the API requests now returned the expected JSON data without any 403 or CORS errors. I also checked the network tab in the developer tools to ensure that each request to the API Gateway endpoint went through CloudFront and received the correct response.



nextwork.org

The place to learn & showcase your skills

Check out nextwork.org for more projects