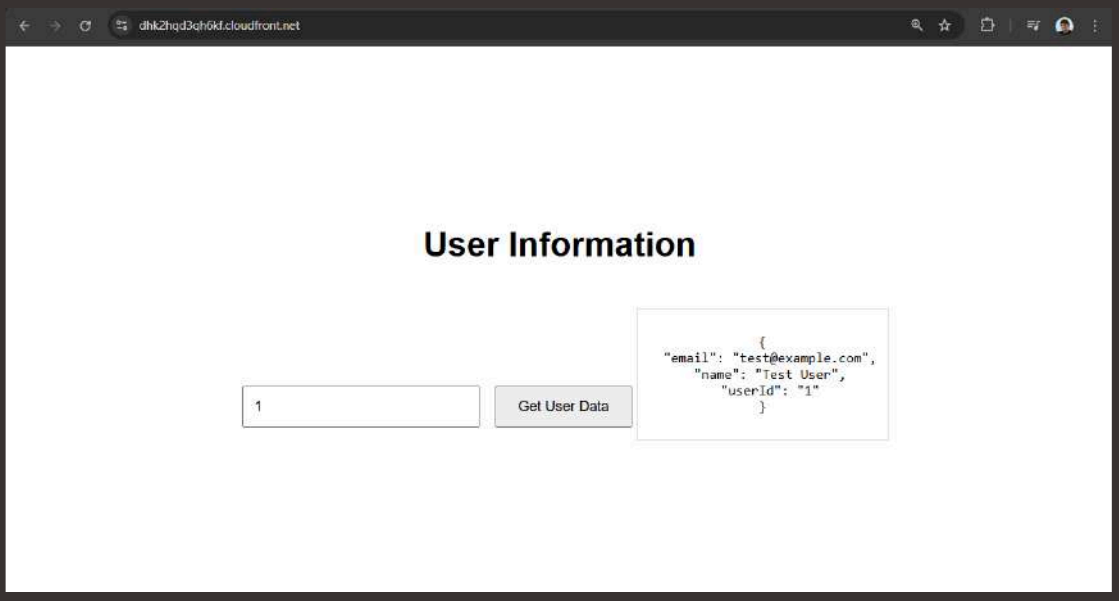
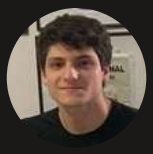
nextwork.org

Build a Three-Tier Web App

Nicolás Aversa





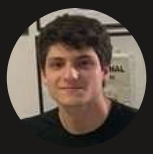
Nicolás Aversa
NextWork Student

Introducing Today's Project!

In this project, I will demonstrate how to build a complete serverless three-tier application architecture using AWS services. I'm doing this project to learn how to properly structure a cloud-native application by separating concerns into presentation, logic and data layers while maintaining seamless integration between all components.

Tools and concepts

Services I used were S3, CloudFront, Lambda, API Gateway, IAM, and DynamoDB. Key concepts I learnt include Lambda functions, and mainly troubleshooting console errors, while the architecture setup was quick since I'd done it before in the earlier projects.

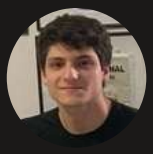


Nicolás Aversa
NextWork Student

Project reflection

This project took me approximately two and a half hours. The most challenging part was troubleshooting the console errors - particularly resolving the CORS policy blockage by updating my Lambda's response headers, fixing the API Gateway configuration, and ensuring proper cache invalidation in CloudFront after uploading the corrected script.js. It was most rewarding to see my CloudFront-distributed site successfully fetch and display real-time data from DynamoDB, confirming all three architecture tiers were communicating.

I took on this project to gain my first hands-on experience building a complete serverless application on AWS. As a beginner, I wanted to learn how to connect essential serverless services - S3 for storing website files, CloudFront for global distribution, Lambda for backend logic, API Gateway for RESTful endpoints, and DynamoDB for data storage - all without managing servers. This project absolutely met my goals. It successfully helped me understand how these services integrate in a real-world scenario and gave me my first real grasp of how serverless architectures actually work in practice.

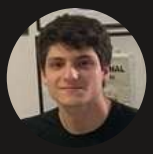


Nicolás Aversa
NextWork Student

Presentation tier

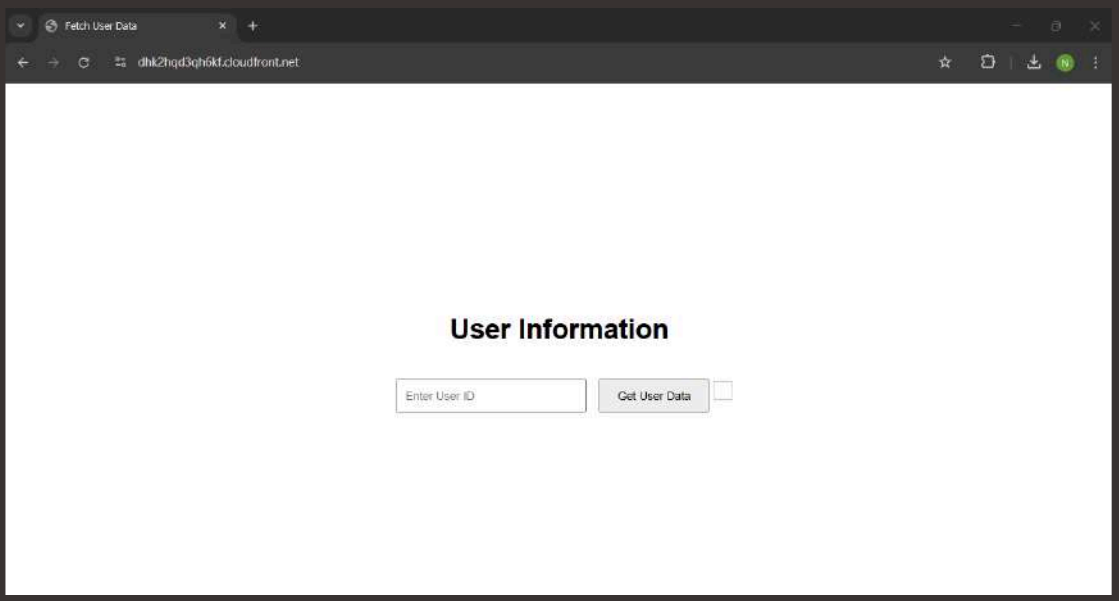
For the presentation tier, I will set up an S3 bucket to store website's files and configure CloudFront as a CDN because this provides a highly available and globally distributed foundation for delivering my web content. The S3 bucket will securely store all frontend assets like HTML, CSS and JavaScript files, while CloudFront will cache and serve these files from edge locations worldwide to ensure fast loading times for users regardless of their geographic location.

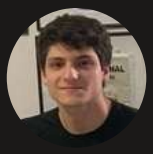
I accessed my delivered website by opening my CloudFront distribution domain name in a new tab. I created an origin access control (OAC), which lets you keep your S3 bucket and objects not publicly accessible, while still making sure they can be accessed through CloudFront. Then, I pasted the OAC policy into my S3 bucket's bucket policy, to explicitly grant the OAC permission to the bucket's contents.



Nicolás Aversa
NextWork Student

NextWork.org



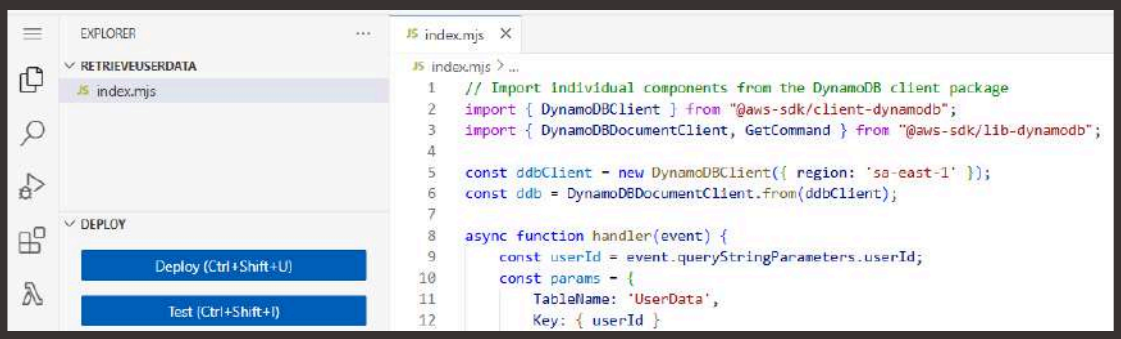


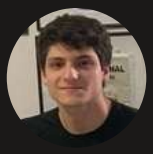
Nicolás Aversa
NextWork Student

Logic tier

For the logic tier, I will set up a Lambda function that retrieves user data from a DynamoDB table. As a way to expose that functionality to the outside world, I'll use API Gateway to handle requests and route them to the right place.

The Lambda function retrieves data by looking for specific user data based on a 'userId' and returns that data. If there's an error e.g. the userId doesn't exist in the database, it returns an error message.



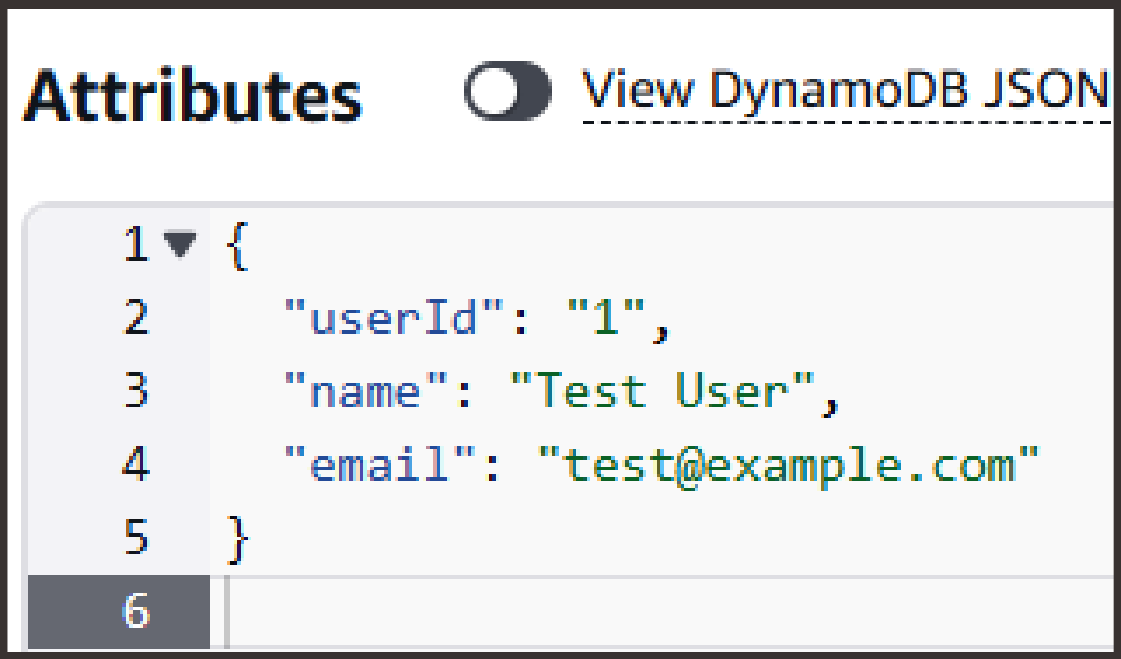


Nicolás Aversa
NextWork Student

Data tier

For the data tier, I will set up a DynamoDB table to store some user data, which will then be retrieved by the Lambda function.

The partition key for my DynamoDB table is 'userId', which means DynamoDB will organize and distribute all data based on unique user identifiers. This lets my Lambda function instantly retrieve any user's data through direct partition access, ensuring fast, scalable performance as the user base grows.

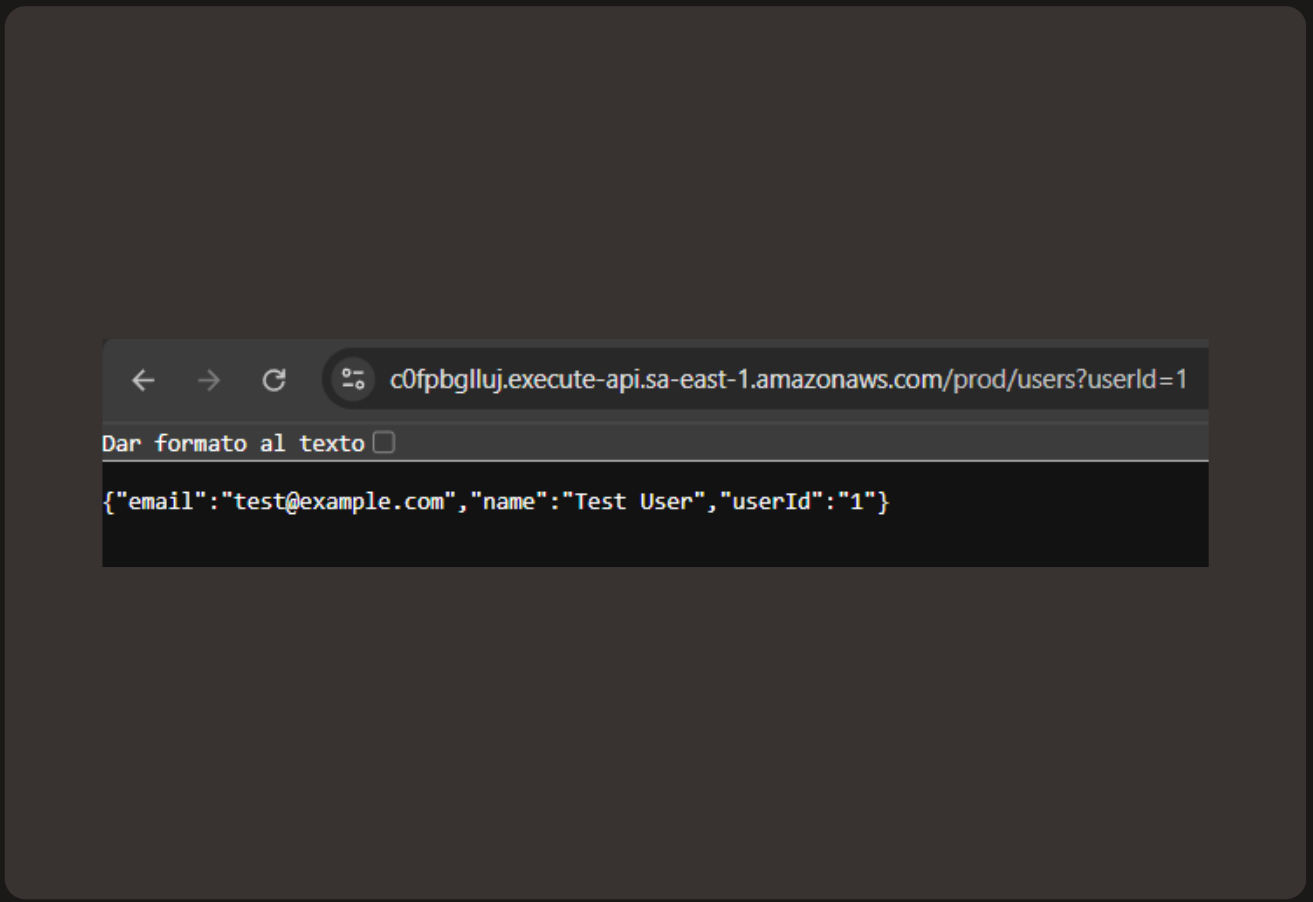


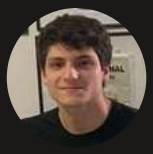
Nicolás Aversa
NextWork Student

Logic and Data tier

Once all three layers of my three-tier architecture are set up, the next step is to update my script.js file with JavaScript code because I need my web app to be able to make a request to my API Gateway endpoint and display the returned data.

To test my API, I appended /users?userId=1 to the end of my prod stage API's Invoke URL. The results were successful; my table's data got returned by the API.





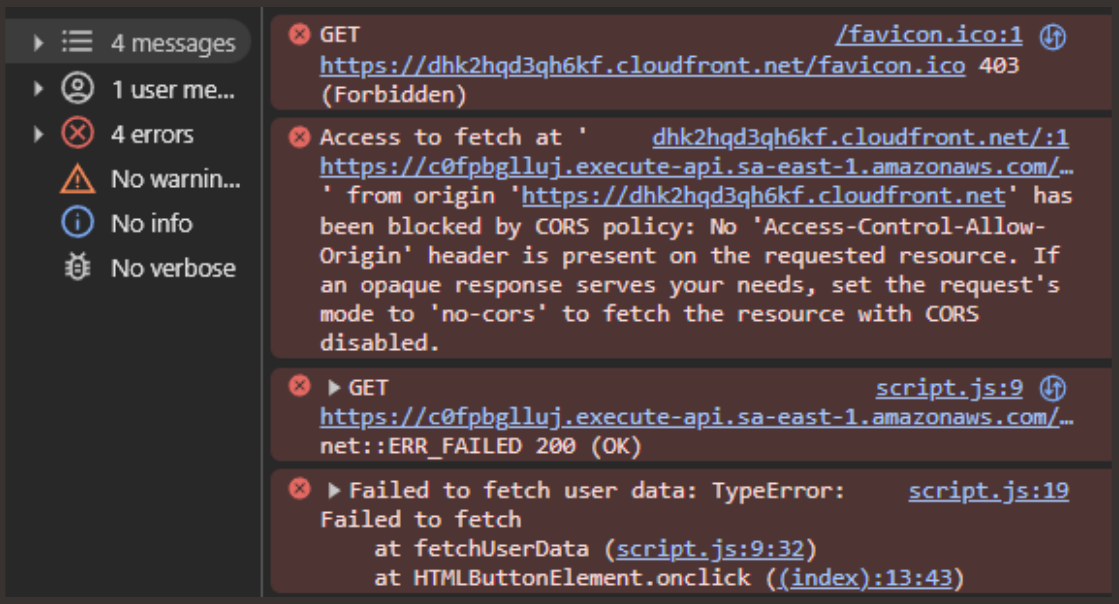
Nicolás Aversa
NextWork Student

Console Errors

The error in my distributed site was because my API was not actually getting fetched. I was not referencing it.

To resolve the error, I updated script.js by replacing the '[YOUR-PROD-API-URL]' for my prod stage API's Invoke URL. I then reuploaded it into S3 because this file now has the latest version. However, since CloudFront was still serving the cached version, I had to create an invalidation to force the CDN to fetch script.js from S3, ensuring all users would receive the corrected version of my file.

I ran into a third error after updating script.js. This was an error with CORS because my API Gateway is not configured to allow requests from my CloudFront URL. CORS (Cross-Origin Resource Sharing) is like a security bouncer for your browser. It decides whether the frontend (like my CloudFront-hosted site) is allowed to talk to a backend server (like API Gateway).


4 messages
1 user me...
4 errors
No warnin...
No info
No verbose
GET /favicon.ico:1
https://dhk2hqd3qh6kf.cloudfront.net/favicon.ico 403 (Forbidden)
Access to fetch at ' dhk2hqd3qh6kf.cloudfront.net/:1
https://c0fpbglluj.execute-api.sa-east-1.amazonaws.com/…
' from origin 'https://dhk2hqd3qh6kf.cloudfront.net' has been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource. If an opaque response serves your needs, set the request's mode to 'no-cors' to fetch the resource with CORS disabled.
GET script.js:9
https://c0fpbglluj.execute-api.sa-east-1.amazonaws.com/…
net::ERR_FAILED 200 (OK)
Failed to fetch user data: TypeError: script.js:19
Failed to fetch
at fetchUserData (script.js:9:32)
at HTMLButtonElement.onclick ((index):13:43)

Nicolás Aversa
NextWork Student

Resolving CORS Errors

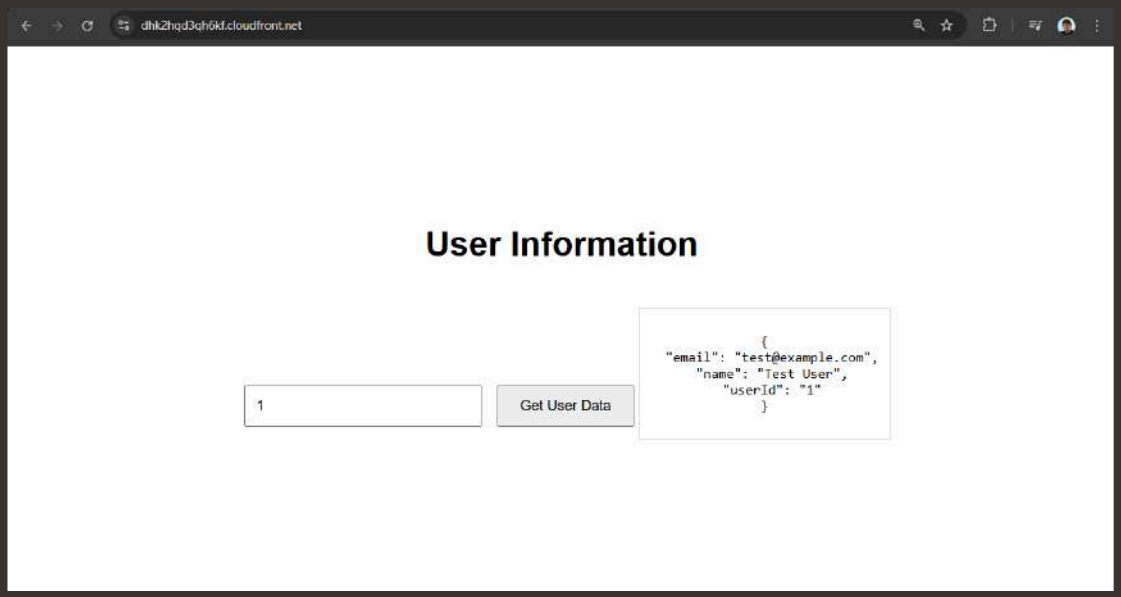
To resolve the CORS error, I first headed to Amazon API Gateway and enabled CORS on my /users resource. In the settings, I set my CloudFront distribution domain name as the Access-Control-Allow-Origin value.

I also updated my Lambda function because since proxy integration is enabled, API Gateway simply forwards the request to my Lambda function and expects the Lambda to return the full HTTP response, including the CORS headers. The changes I made were including the Access-Control-Allow-Origin header in the response, referencing my CloudFront domain name.

Nicolás Aversa
NextWork Student

Fixed Solution

I verified the fixed connection between API Gateway and CloudFront by doing one more refresh of my CloudFront domain name. After that, I typed 1 in the User ID input, clicked on 'Get User Data' and the data was successfully retrieved.



NextWork.org

Everyone should be in a job they love.

Check out nextwork.org for more projects