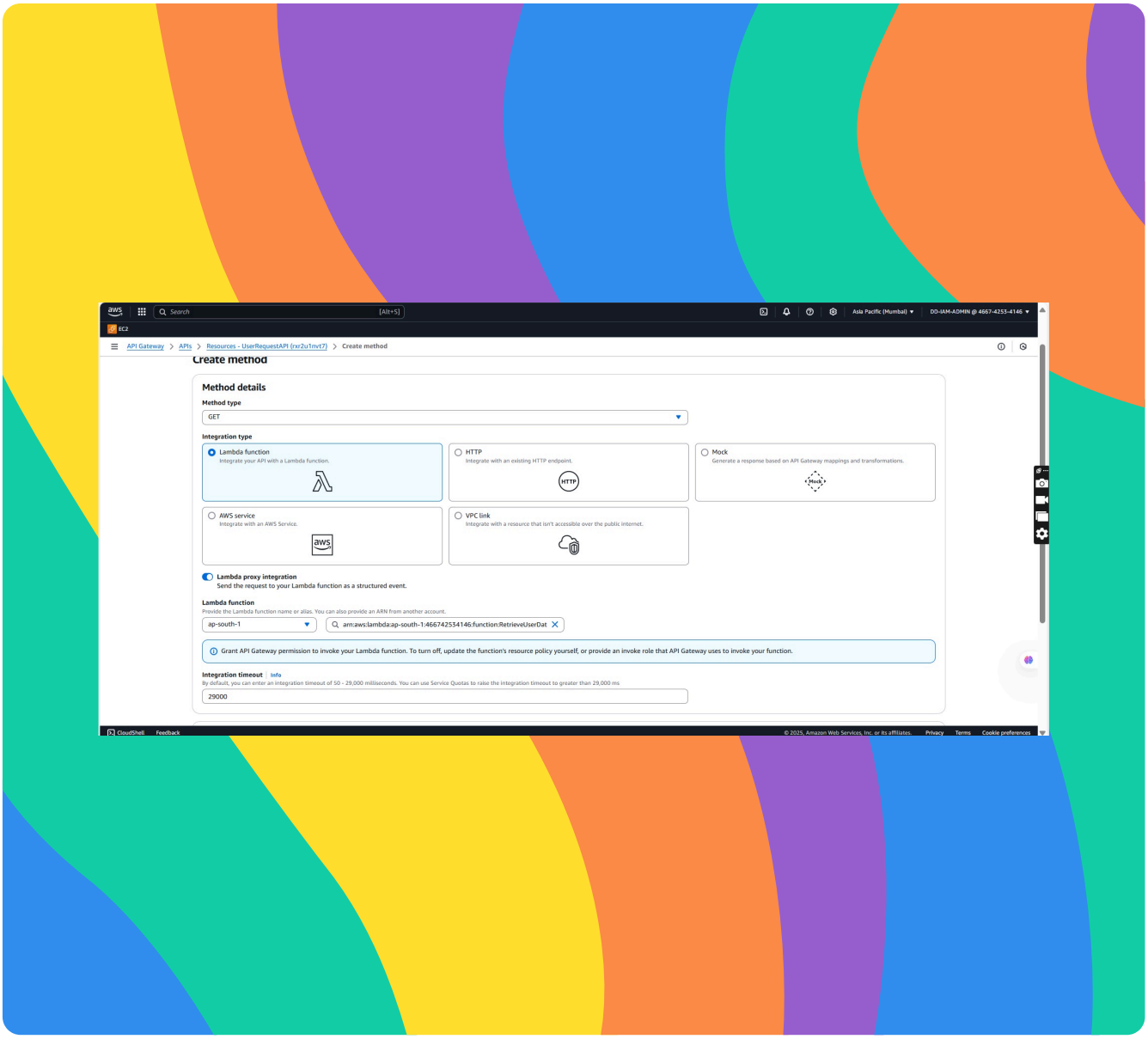
nextwork.org

APIs with Lambda + API Gateway

DI **Dineshraj Dhanapathy**



DI
Dineshraj Dhanapathy
NextWork Student

Introducing Today's Project!

In this project, I will demonstrate how to build and run the logic tier of a web application using AWS services. I'm doing this project to learn how backend code processes user actions, handles data, and connects to other parts of the app. This will help me understand how the logic tier supports application functionality and improves my skills in designing scalable, server-side solutions.

Tools and concepts

Services I used were AWS Lambda and API Gateway. Key concepts I learned include Lambda functions, which let you run backend code without managing servers, and API Gateway, which acts as the front door to trigger Lambda securely. These services helped me understand how to build and manage scalable, serverless applications on AWS.

DI
Dineshraj Dhanapathy
NextWork Student

Project reflection

This project took me approximately 1 hour to complete. The most challenging part was troubleshooting the access denied errors and correctly configuring permissions between API Gateway and Lambda. It was most rewarding to see my hosted website and API working seamlessly together and to understand how different AWS services integrate to build a full-stack, serverless application.

I did this project today to strengthen my understanding of how AWS Lambda and API Gateway work together in building serverless applications. This project met my goals by helping me practice setting up Lambda functions, connecting them to REST APIs via API Gateway, and understanding request handling. It also gave me hands-on experience with deploying, documenting, and testing backend functionality without managing any servers.

DI
Dineshraj Dhanapathy
NextWork Student

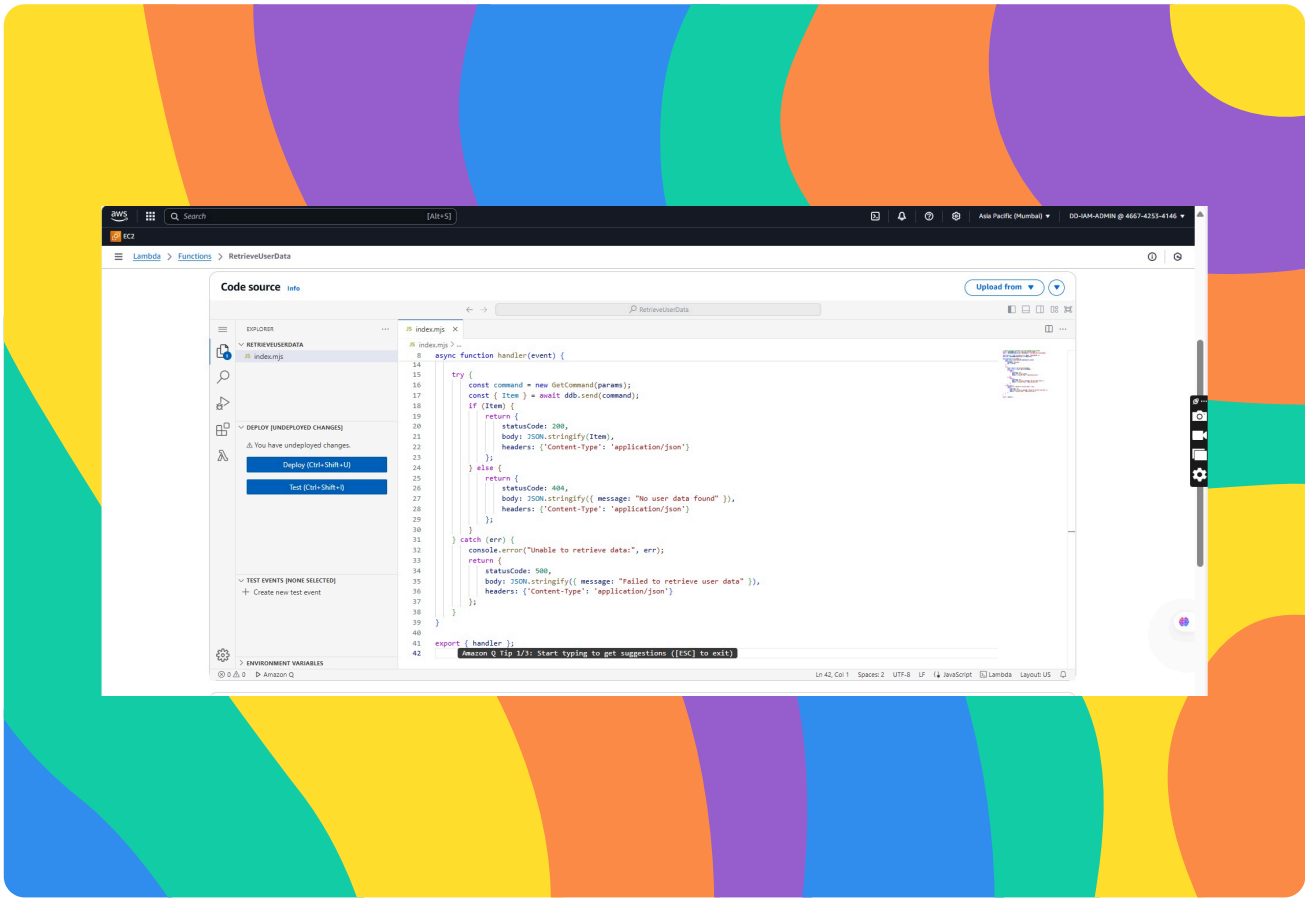
Lambda functions

AWS Lambda is a serverless compute service that lets you run code without managing servers. It automatically handles scaling, availability, and infrastructure, running your code only when triggered and charging only for the compute time used. I'm using Lambda in this project to process backend logic, like retrieving user data, in response to events. This makes my application more efficient, cost-effective, and easier to manage as it grows.

The code I added to my function will retrieve data from a DynamoDB table by searching for a specific user based on their 'userId'. It sends a request to the database, checks if the user exists, and returns the matching data. If the user isn't found or there's an issue with the query, it returns an error message instead. This sets up the logic to handle backend data retrieval, even though the actual DynamoDB table and userId setup will come later in the next project.

DI

Dineshraj Dhanapathy
NextWork Student



DI
Dineshraj Dhanapathy
NextWork Student

API Gateway

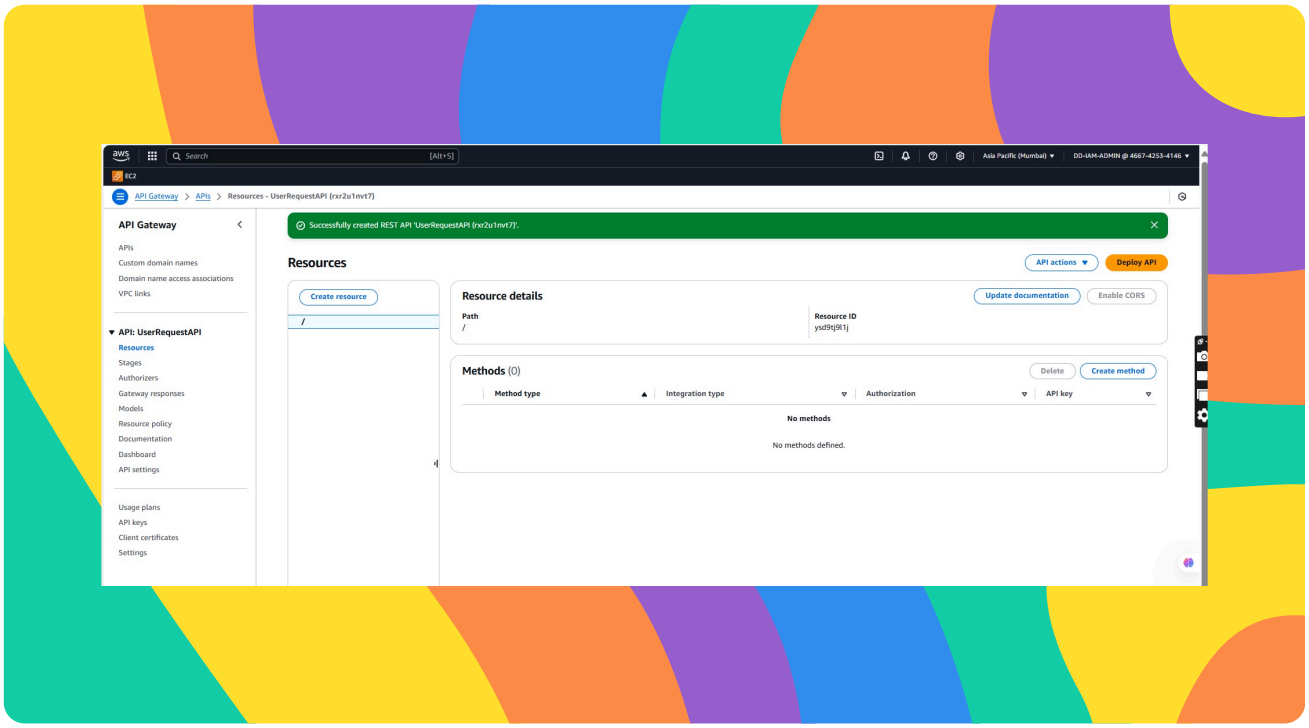
APIs are interfaces that allow different software systems to communicate with each other. They define how requests for information or actions should be made and how responses are delivered. There are different types of APIs, like REST, HTTP, and WebSocket. REST APIs use standard HTTP methods to interact with data and are widely used for their simplicity and compatibility. My API is a REST API, which I'm using to connect user requests to my Lambda function for backend processing.

Amazon API Gateway is a fully managed service that allows developers to create, publish, monitor, and secure APIs at any scale. It acts as the entry point for applications to access backend services like AWS Lambda. I'm using API Gateway in this project to connect my frontend to the Lambda function, allowing user requests to be routed, processed, and returned securely. It also provides authentication, authorization, and traffic management features that help ensure my app is both secure and scalable.

DI

Dineshraj Dhanapathy
NextWork Student

When a user makes a request, API Gateway receives it and acts as the interface between the user and the backend. It then forwards the request to the appropriate Lambda function based on the API configuration. The Lambda function processes the request—such as retrieving or updating data—and sends a response back. API Gateway then takes this response and returns it to the user. This setup allows seamless, secure communication between users and serverless backend logic without managing any servers.



DI
Dineshraj Dhanapathy
NextWork Student

API Resources and Methods

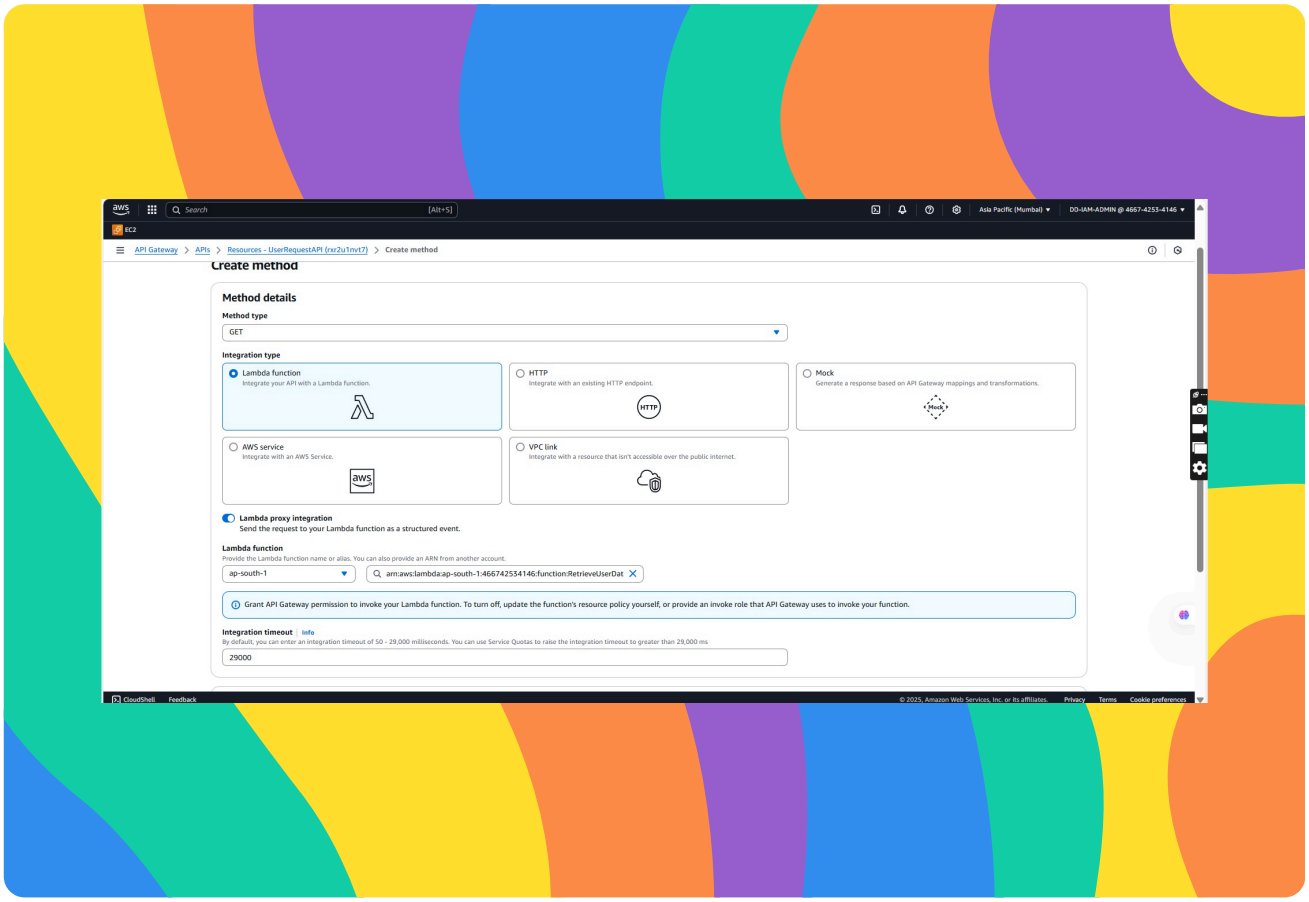
An API is made up of resources, which are specific paths or endpoints that represent different parts of the API's functionality. Each resource corresponds to a particular type of data or operation, like /users for user profiles or /messages for chat messages. These resources help structure the API logically and allow developers to handle different types of requests separately. In this project, I'm creating a resource that will connect to my Lambda function to handle user data requests.

Each resource consists of methods, which are the actions that can be performed on that resource, such as GET, POST, PUT, or DELETE. These methods correspond to standard HTTP operations—GET retrieves data, POST sends new data, PUT updates existing data, and DELETE removes data. In this project, I'm using the GET method to retrieve user information through the /users resource connected to my Lambda function.

I created a GET method for the /users resource. This method allows clients to send HTTP GET requests to retrieve user data. When a request is made, it triggers my connected Lambda function, which processes the request and returns the appropriate user information. This setup enables a seamless flow of data from the backend to the user through the API.

DI

Dineshraj Dhanapathy
NextWork Student

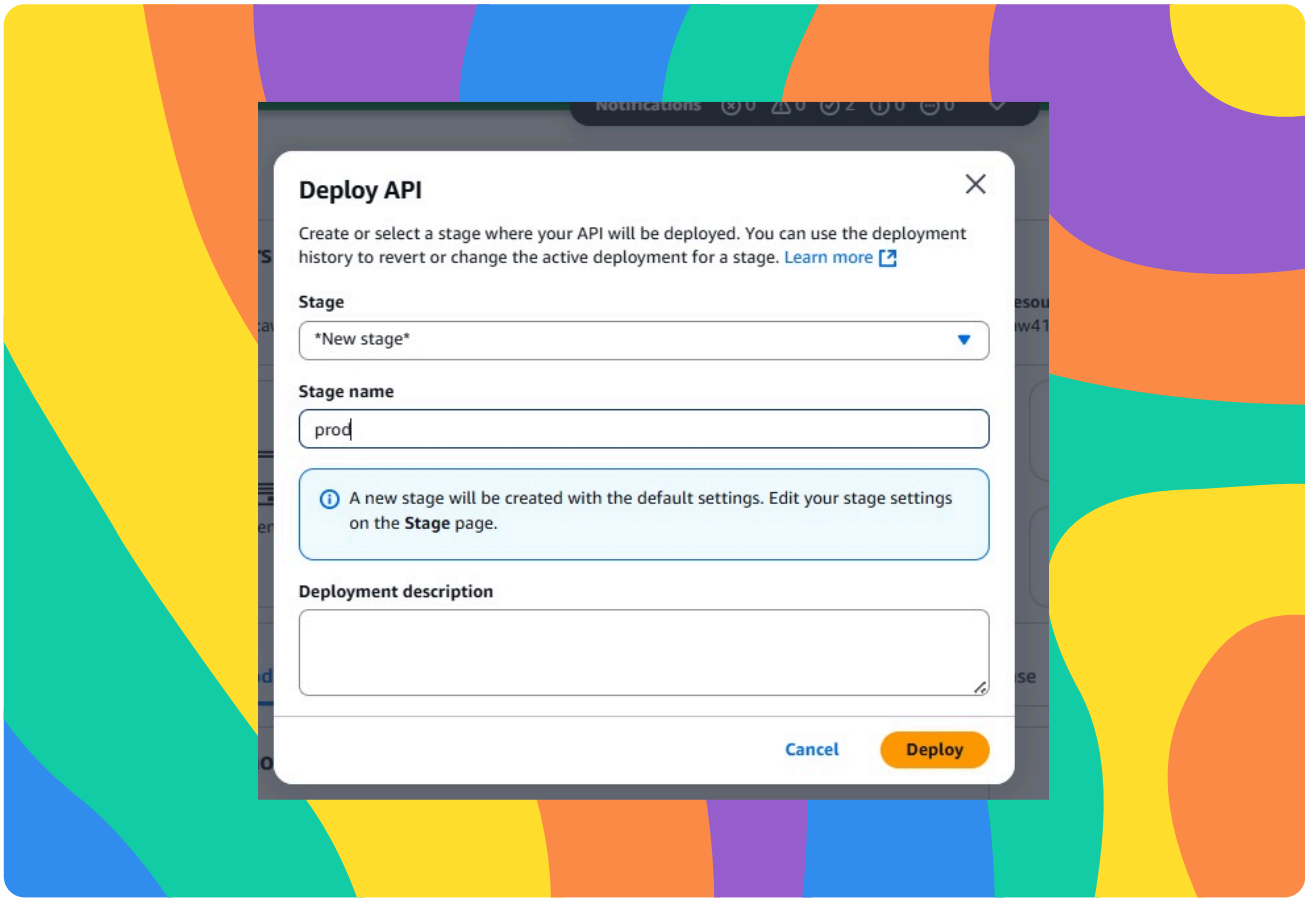


DI
Dineshraj Dhanapathy
NextWork Student

API Deployment

When you deploy an API, you deploy it to a specific stage. A stage is like a versioned environment (such as dev, test, or prod) that allows you to manage and test different phases of your API. Each stage has its own unique endpoint URL. I deployed to a stage called prod so I can test my API in a development environment before making it available to users in production.

To visit my API, I copied the invoke URL from the prod stage in the API Gateway console and pasted it into my browser, adding /users to the end of the URL to target the resource I created. The API displayed an error because I hadn't yet set up the necessary permissions for API Gateway to invoke my Lambda function or the Lambda function wasn’t returning the expected response.


Deploy API
Create or select a stage where your API will be deployed. You can use the deployment history to revert or change the active deployment for a stage. Learn more
Stage
New stage
Stage name
prod
A new stage will be created with the default settings. Edit your stage settings on the Stage page.
Deployment description
Cancel
Deploy

DI
Dineshraj Dhanapathy
NextWork Student

API Documentation

For my project's extension, I am writing API documentation because it clearly explains how my API works, making it easier for other developers (and future me) to understand and use it correctly. Documentation outlines endpoints, methods, parameters, and responses, helping prevent confusion and bugs. You can do this in JSON using tools like Swagger or directly from the API Gateway console, ensuring your API is well-structured, professional, and ready for real-world use.

Once I prepared my documentation, I can publish it to a specific stage in API Gateway. You have to publish your API to a specific stage because each stage represents a version of your API (like development or production), and the documentation needs to match that version. This ensures developers are seeing the correct details for the right version, avoiding confusion when working across multiple environments in the API lifecycle.

DI

Dineshraj Dhanapathy
NextWork Student

My published and downloaded documentation showed me detailed metadata about my API, such as its version, title, and structure. It included the /users resource and the GET method attached to it. The automatically generated sections provided a general overview, while the manual parts allowed me to add more specific and user-friendly explanations. This combination gave a complete picture of how my API works and made it easier for developers to understand how to interact with it effectively.

NextWork.org

Everyone should be in a job they love.

Check out nextwork.org for more projects