

# Introduction

Serverless computing simplifies application development by removing the need to manage servers. Using AWS Lambda, API Gateway, and DynamoDB, you can create a scalable, cost-effective REST API for tasks like adding and retrieving user data. This setup lets developers focus on code, not infrastructure.

## Overview:

**AWS Lambda:** Executes code in response to events like HTTP requests without managing servers. In this project, it handles adding and retrieving user data from DynamoDB.

**API Gateway:** Acts as the HTTP interface for Lambda, triggering functions based on requests (POST/GET). No server management is required.

**DynamoDB:** A fast, fully managed NoSQL database that stores user data and allows easy querying based on user IDs.

## Key Features

**AWS Lambda:** Event-driven, scales automatically, and is cost-efficient with pay-per-use pricing.

**API Gateway:** Integrates with Lambda, offers security features, and simplifies REST API setup.

**DynamoDB:** High availability, fast queries, and fully managed—perfect for user data retrieval.

## Application:

AWS Lambda and API Gateway create a serverless REST API that scales based on traffic, with DynamoDB providing efficient data storage and retrieval. Together, they eliminate server management and reduce costs, ideal for handling various workloads.

## 3rd year Project Relation

For the Offsync project, which collects data offline and syncs it once online, this architecture is ideal. Lambda and API Gateway ensure seamless syncing, while DynamoDB handles data storage and retrieval efficiently. The pay-per-use model further optimizes costs for offline-heavy scenarios, and built-in security features protect data.

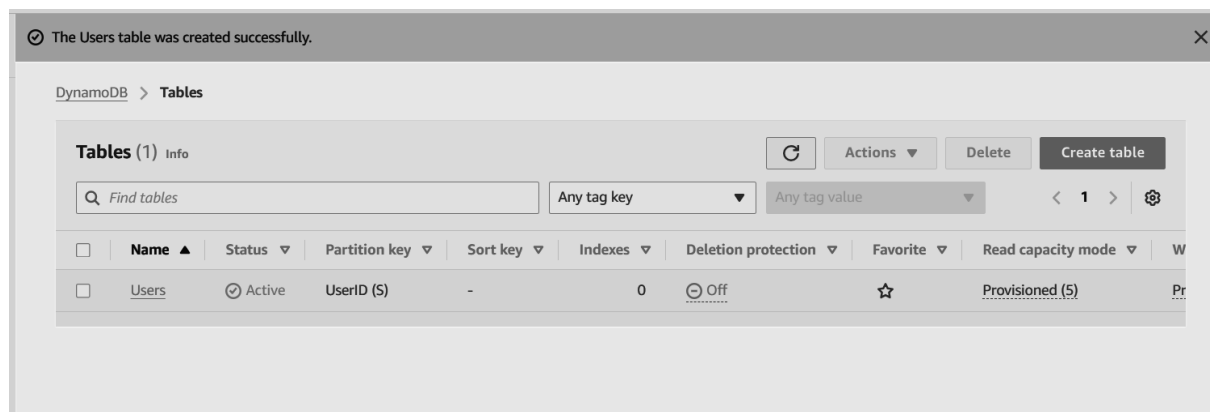
# Step by Step Execution:

## Dynamo DB:

Go to DynamoDB dashboard → Tables → Create table.

Name your table "Users" with the partition key as **UserID (string)**.

Click on **Create Table**.



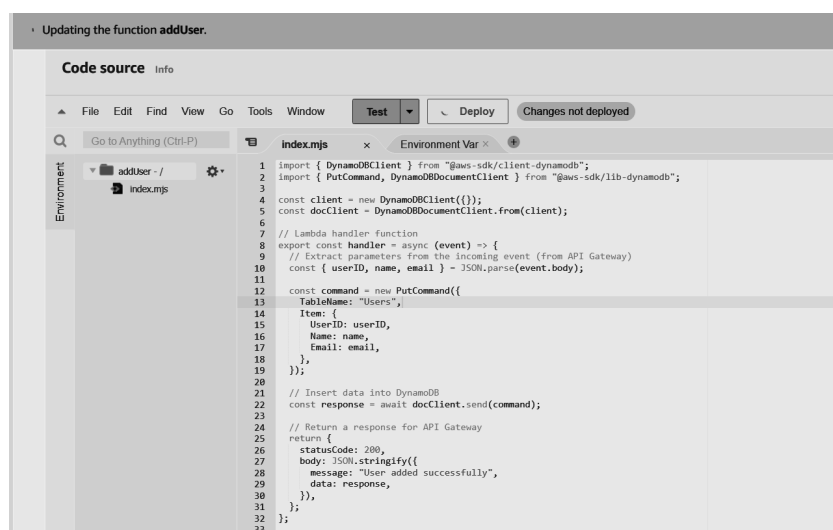
## Lambda Functions

Navigate to the Lambda dashboard → Create Function.

Name the function **AddUserFunction**, keep default settings, and create.

Go to **Configuration** → **Permissions** and note the execution role name.

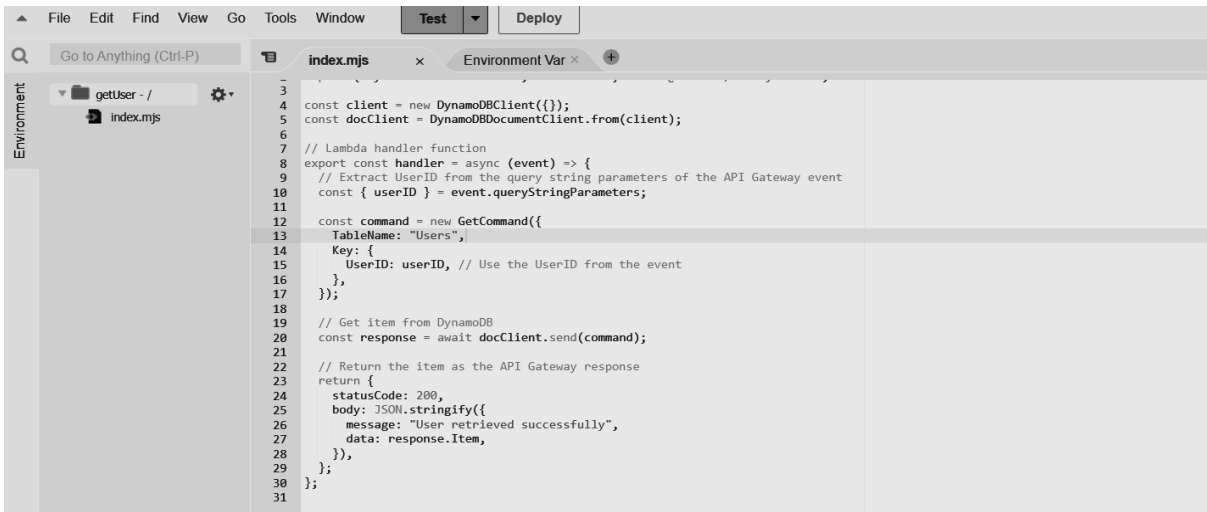
In the **Code** section, write logic to add a user to DynamoDB (refer to AWS documentation for DynamoDB JS SDK).



## GetUserFunction

Create another function called **getUser**.

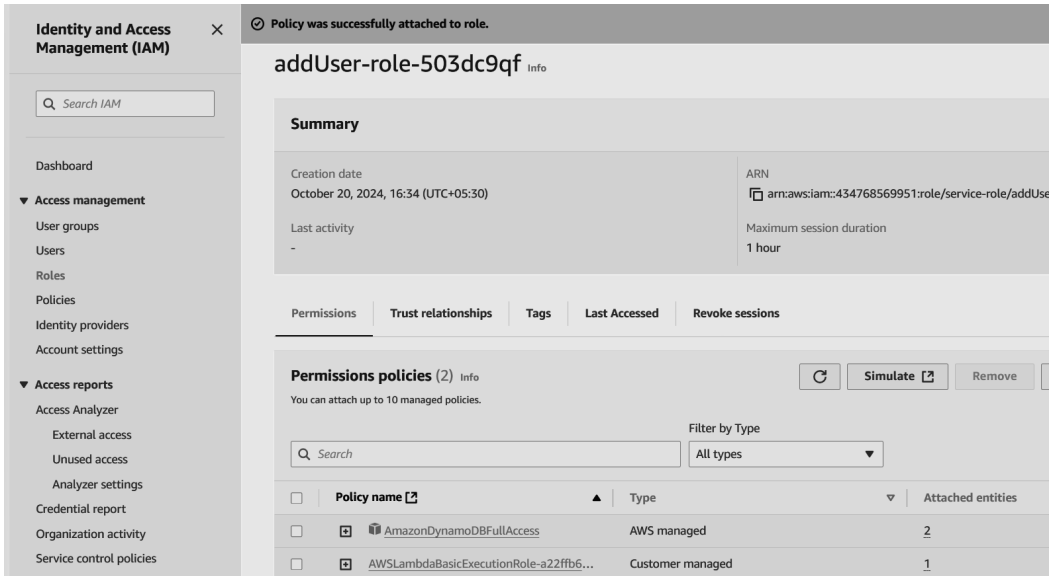
Use the existing execution role from the previous step.  
In the **Code** section, write logic to retrieve user data (use AWS documentation).



```
1 // ...
2
3
4 const client = new DynamoDBClient({});
5 const docClient = DynamoDBDocumentClient.from(client);
6
7 // Lambda handler function
8 export const handler = async (event) => {
9   // Extract UserID from the query string parameters of the API Gateway event
10   const { userID } = event.queryStringParameters;
11
12   const command = new GetCommand({
13     TableName: "Users",
14     Key: {
15       UserID: userID, // Use the UserID from the event
16     },
17   });
18
19   // Get item from DynamoDB
20   const response = await docClient.send(command);
21
22   // Return the item as the API Gateway response
23   return {
24     statusCode: 200,
25     body: JSON.stringify({
26       message: "User retrieved successfully",
27       data: response.Item,
28     }),
29   };
30 };
31
```

## IAM role config

Go to **IAM Roles**, find the auto-created role from **AddUserFunction**.  
Attach the **AmazonDynamoDBFullAccess** policy to the role.



Identity and Access Management (IAM)

Policy was successfully attached to role.

addUser-role-503dc9qf Info

**Summary**

Creation date	October 20, 2024, 16:34 (UTC+05:30)	ARN	arn:aws:iam:434768569951:role/service-role/addUser-role-503dc9qf
Last activity	-	Maximum session duration	1 hour

Permissions | Trust relationships | Tags | Last Accessed | Revoke sessions

**Permissions policies (2)** Info

You can attach up to 10 managed policies.

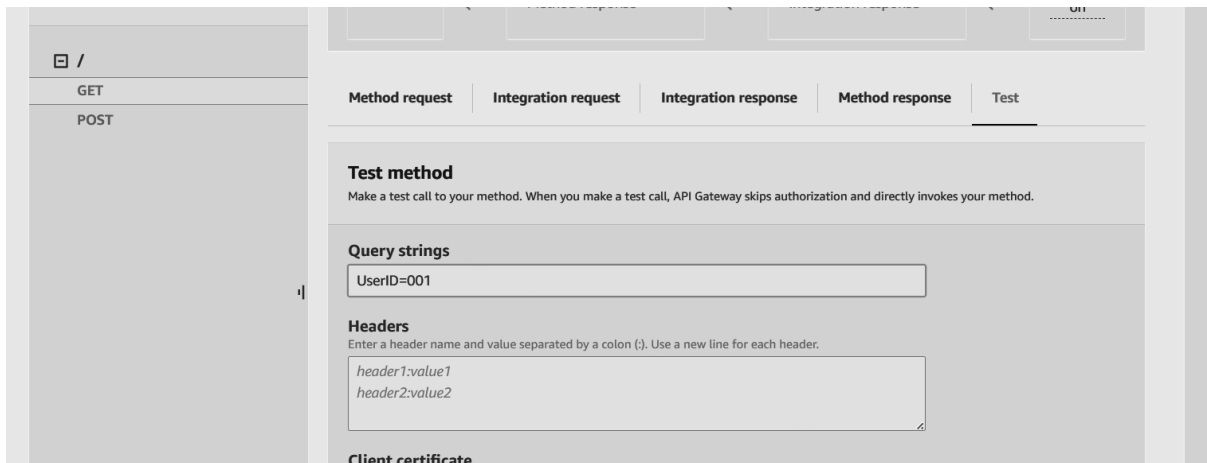
Filter by Type: All types

<input type="checkbox"/>	Policy name <small>Info</small>	Type	Attached entities
<input type="checkbox"/>	AmazonDynamoDBFullAccess	AWS managed	2
<input type="checkbox"/>	AWSLambdaBasicExecutionRole-a22ffb6...	Customer managed	1

## API gateway

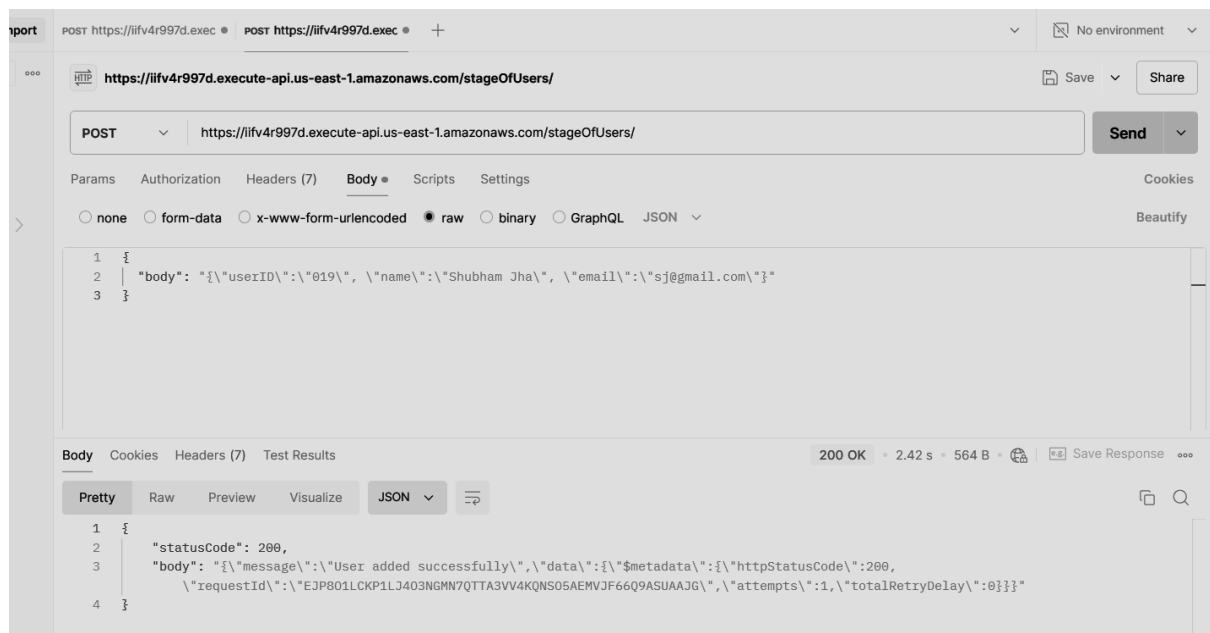
Navigate to **API Gateway** → **APIs** → **Create API** → **REST API**.  
Name the API and create.

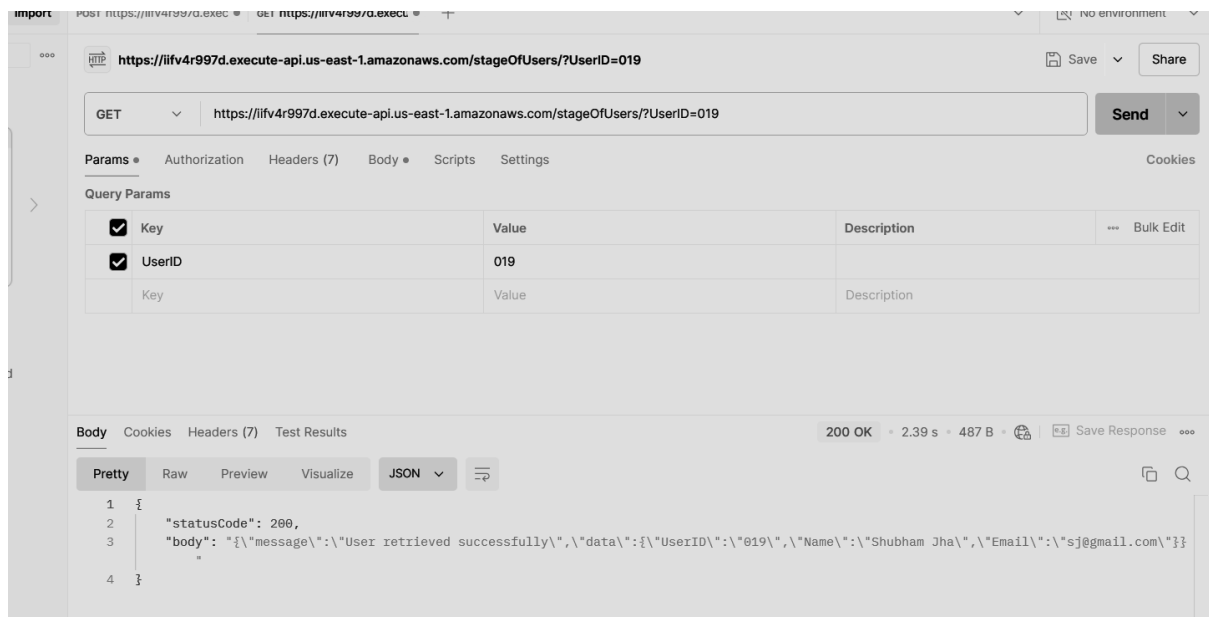
Create **POST** and **GET** methods for adding and getting users.  
Use the appropriate Lambda functions for each method.  
For **GET**, edit **Integration Request** → Add mapping template for JSON requests.



## Testing the API (Using postman web)

Deploy the API and copy the endpoint URL.  
Use Postman to test both the **POST** (addUser) and **GET** (getUser) methods.  
Ensure the API works and data is reflected in the DynamoDB table.





## Conclusion:

In conclusion, building a serverless REST API with AWS Lambda, API Gateway, and DynamoDB is a smart and efficient way to develop modern applications. AWS Lambda handles the heavy lifting by running your code without worrying about servers, while API Gateway smoothly directs HTTP requests to the right Lambda functions. DynamoDB offers fast, reliable storage for user data, perfectly fitting into this setup. Together, they simplify the development process while providing strong performance, easy scaling, and top-notch security—making this approach ideal for managing user data in web apps.