2. Creating a Serverless API

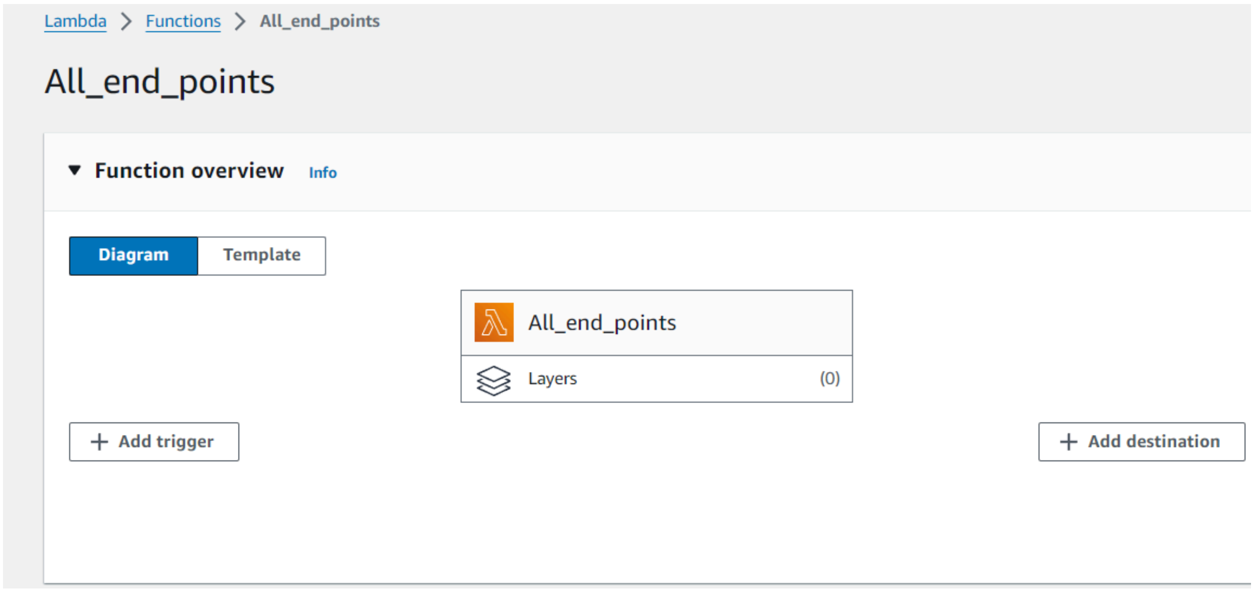
Objective: Develop a serverless API using AWS Lambda and API Gateway.

Approach:

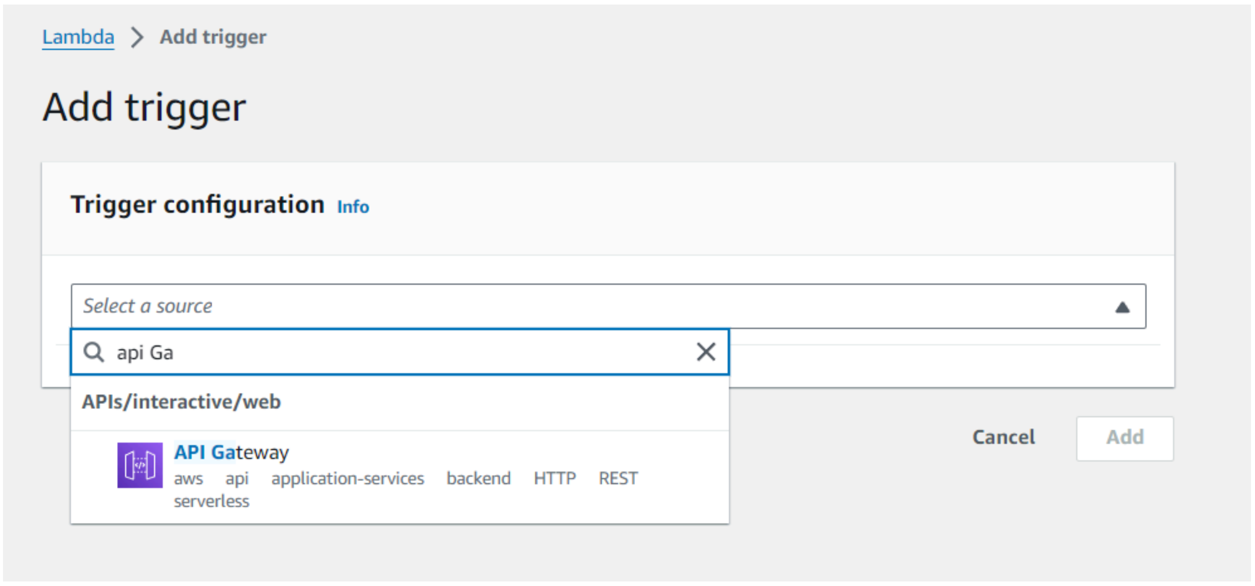
- Define API: Design a simple RESTful API (e.g., for a todo list application).
- Lambda Functions: Create Lambda functions for each API method (GET, POST, PUT, DELETE).
- API Gateway Setup: Use API Gateway to set up the API endpoints, connecting each endpoint to the corresponding Lambda function.
- Testing: Test the API using tools like Postman or AWS API Gateway test functionality.

Goal: Gain hands-on experience in building and deploying a serverless API, understanding the integration between Lambda and API Gateway.

First define and create a lambda function:



Add Trigger point as Api Gateway



Lambda > Add trigger

Add trigger

Trigger configuration Info

Select a source

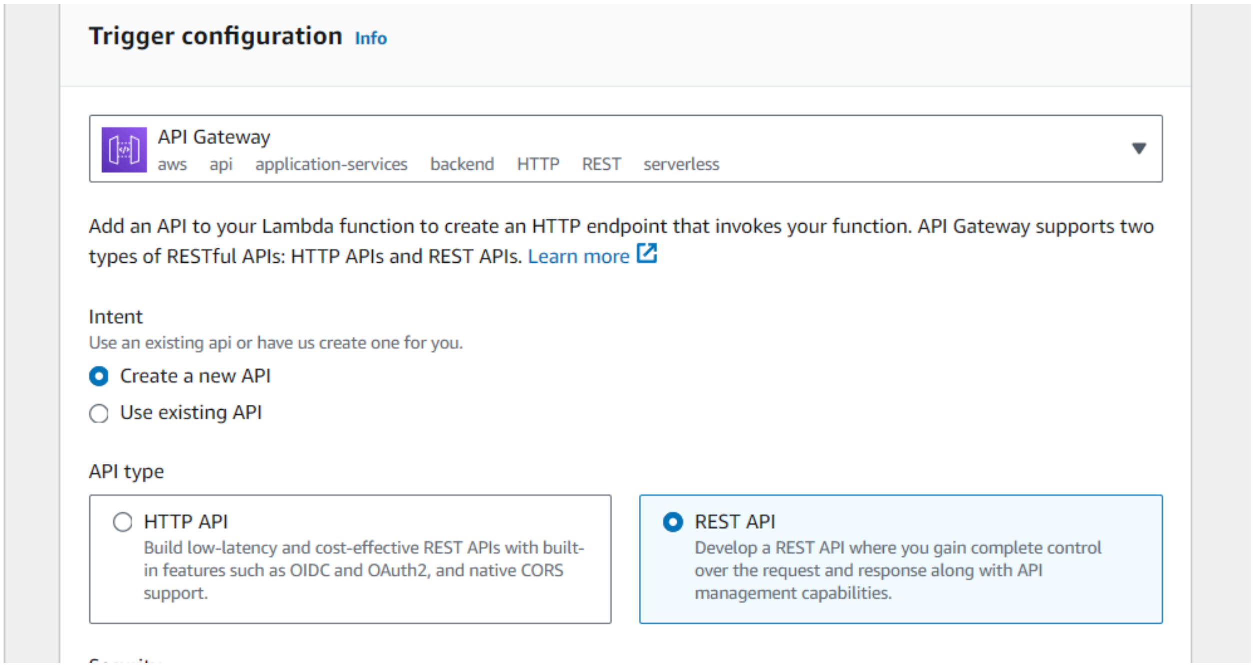
api Ga

APIs/interactive/web

API Gateway
aws api application-services backend HTTP REST serverless

Cancel Add

Create Rest Api



Trigger configuration Info

API Gateway
aws api application-services backend HTTP REST serverless

Add an API to your Lambda function to create an HTTP endpoint that invokes your function. API Gateway supports two types of RESTful APIs: HTTP APIs and REST APIs. Learn more

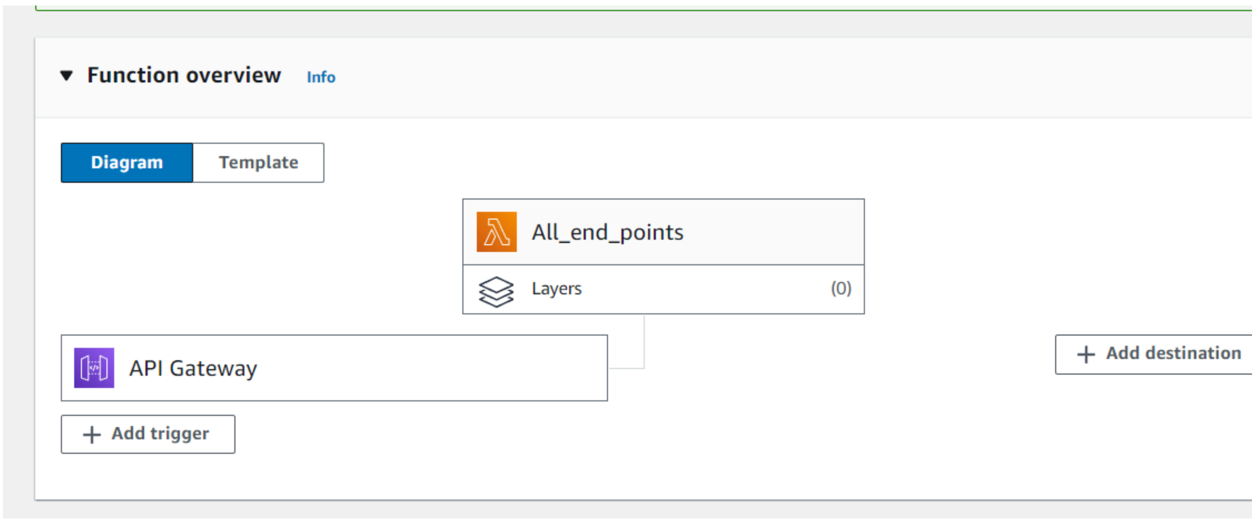
Intent
Use an existing api or have us create one for you.

- [✓ Create a new API]
- Use existing API

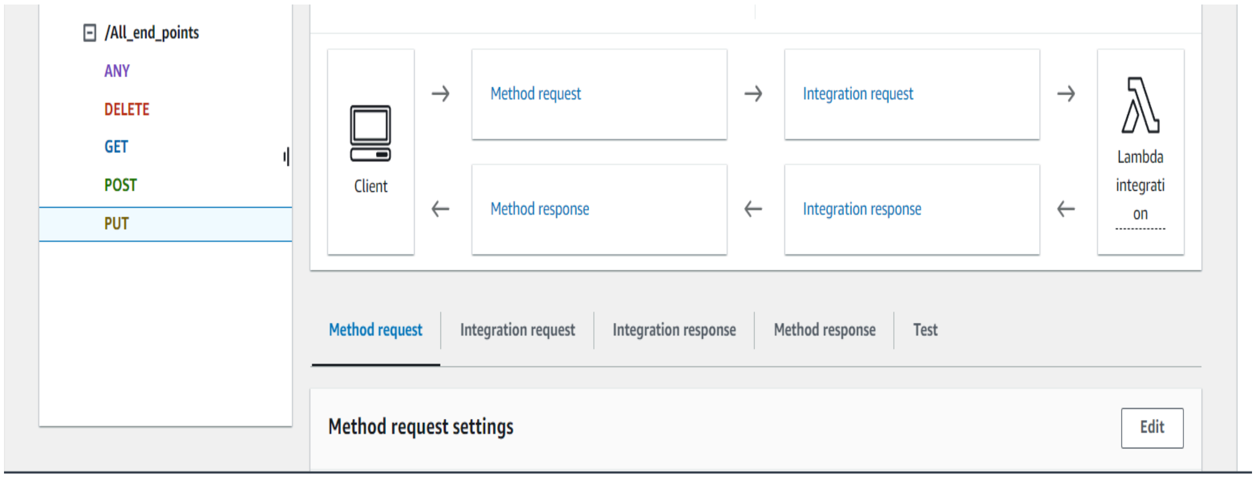
API type

- HTTP API
  Build low-latency and cost-effective REST APIs with built-in features such as OIDC and OAuth2, and native CORS support.
- [✓ REST API]
  Develop a REST API where you gain complete control over the request and response along with API management capabilities.

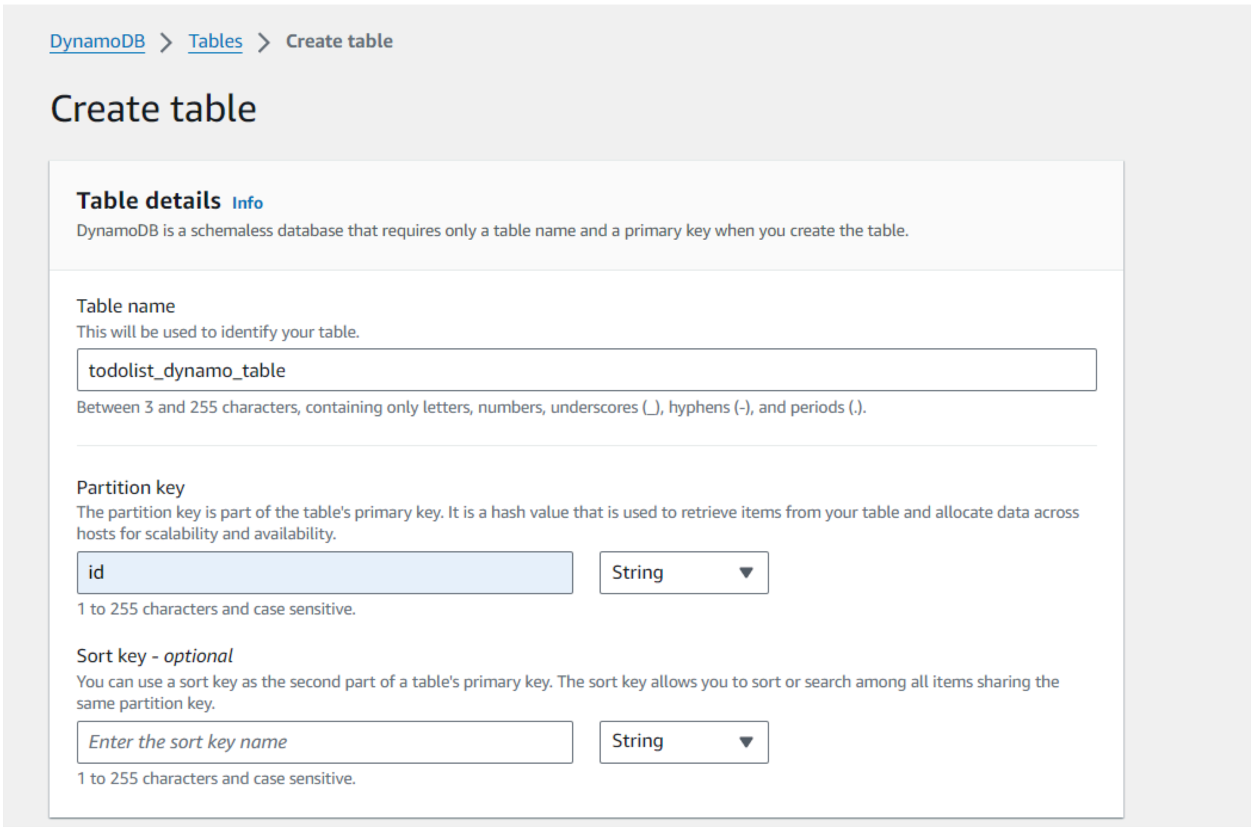
Api gateway Trigger Added



Create methods for the api:



Create dynamo db table:



DynamoDB > Tables > Create table

Create table

Table details Info

DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.

Table name

This will be used to identify your table.

todolist_dynamo_table

Between 3 and 255 characters, containing only letters, numbers, underscores (_), hyphens (-), and periods (.).

Partition key

The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.

id | String

1 to 255 characters and case sensitive.

Sort key - *optional*

You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.

Enter the sort key name | String

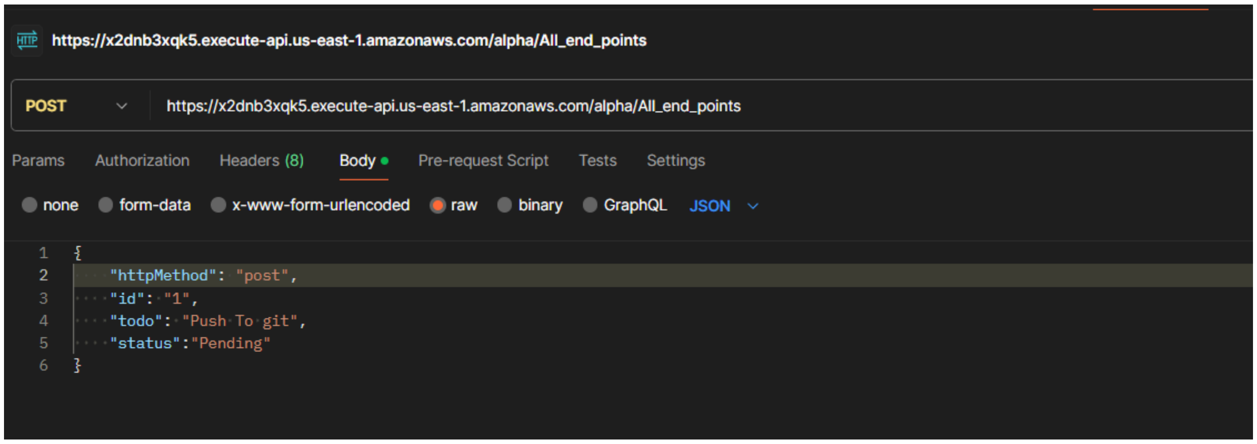
1 to 255 characters and case sensitive.

Lambda Function Code:

```
import json
import boto3
from botocore.exceptions import ClientError
dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('todolist_dynamo_table')
def lambda_handler(event, context):
    http_method = event.get("httpMethod").upper()
    if http_method == "POST":
        return post_request(event)
    elif http_method == "GET":
        return get_request(event)
    elif http_method == "DELETE":
        return delete_request(event)
    elif http_method == "PUT":
        return put_request(event)
    else:
        return {
            "statusCode": 405,
            "body": json.dumps({"error": "Not A valid Method"})
        }
```

```
def build_response(code, message, data=None):
    response_data = {
        "Code": code,
        "Message": message,
        "Data": data
    }
    return {
        "statusCode": code,
        "body": json.dumps(response_data)
    }

def post_request(event):
    try:
        id = event.get('id')
        todo = event.get('todo')
        status = event.get('status')
        item = {
            'id': id,
            'todo':todo,
            'status':status
        }

        table.put_item(Item=item)

        return build_response(200, "Success", {"message": "Insert Successful"})
    except ClientError as e:
        return build_response(500, "Internal Server Error", str(e))

def get_request(event):
    try:
        result = table.scan()
        items = result.get("Items",[])
        return build_response(200, "Success", items)
    except ClientError as e:
        return build_response(500, "Internal Server Error", str(e))

def delete_request(event):
    try:
        id = event.get('id')
        table.delete_item(Key={'id': id})
        return build_response(200, "Success", {"message": "Delete successful."})
    except ClientError as e:
        return build_response(500, "Internal Server Error", str(e))

def put_request(event):
    try:
        id = event.get('id')
        update_key = event.get('update_key')
        update_value = event.get('update_value')

        response = table.update_item(
            Key={'id': id},
            UpdateExpression=f'SET #updateKey = :value',
            ExpressionAttributeNames={'#updateKey': update_key},
            ExpressionAttributeValues={':value': update_value},
            ReturnValues='UPDATED_NEW'
        )
        return build_response(200, "Success", {"message": "Data updated successfully."})
    except ClientError as e:
        return build_response(500, "Internal Server Error", str(e))
```

Executing the post request from postman:



https://x2dnb3xqk5.execute-api.us-east-1.amazonaws.com/alpha/All_end_points

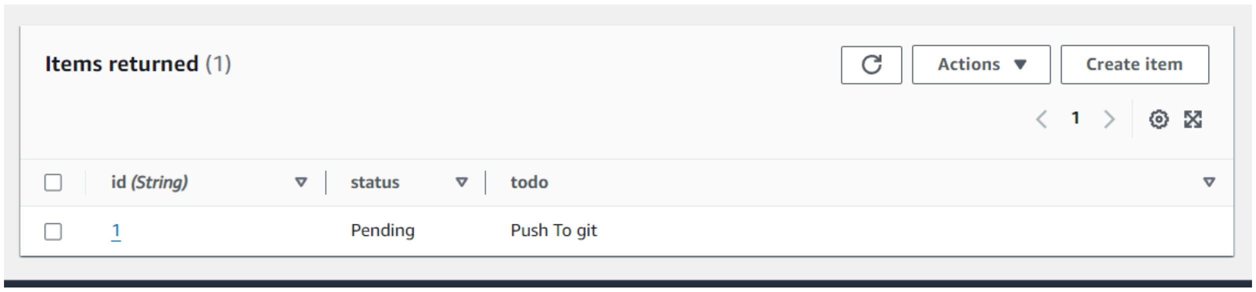
POST https://x2dnb3xqk5.execute-api.us-east-1.amazonaws.com/alpha/All_end_points

Params Authorization Headers (8) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded [✓ raw] binary GraphQL JSON

```
{
    "httpMethod": "post",
    "id": "1",
    "todo": "Push To git",
    "status":"Pending"
}
```

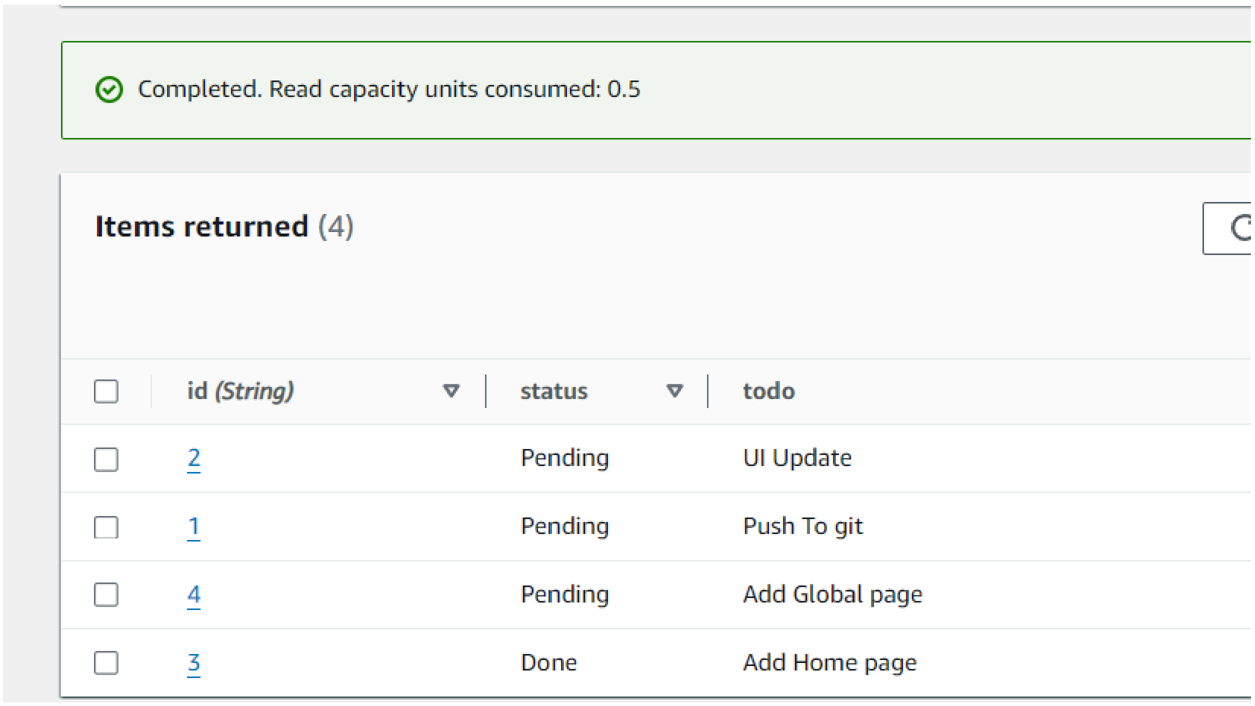
Viewing value in DynamoDb table as:



Items returned (1)

Actions ▼ Create item

	id (String)	status	todo
	1	Pending	Push To git

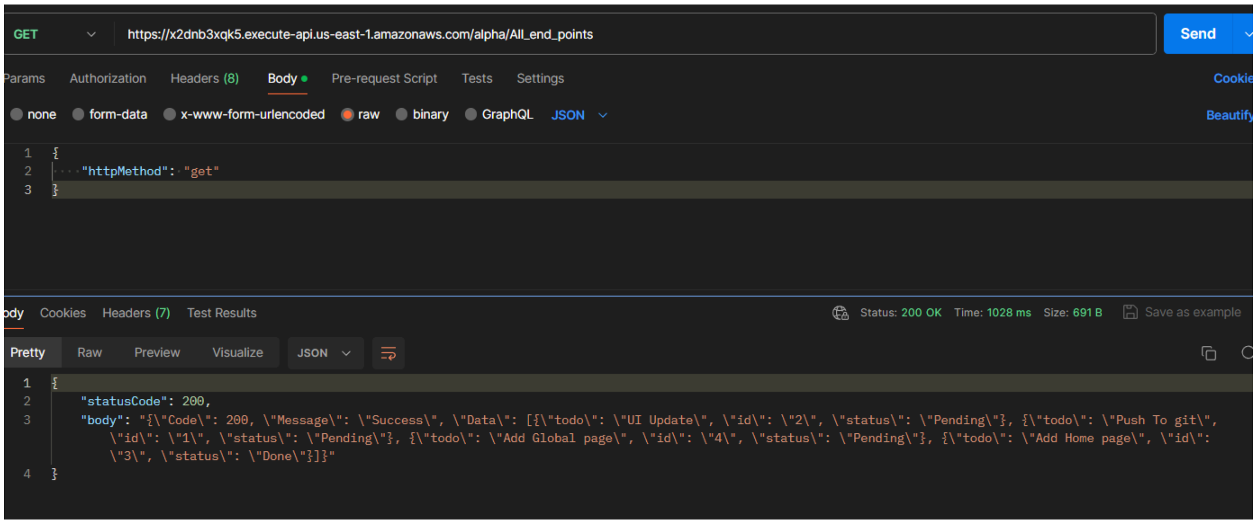


Completed. Read capacity units consumed: 0.5

Items returned (4)

	id *(String)*	status	todo
☐	2	Pending	UI Update
☐	1	Pending	Push To git
☐	4	Pending	Add Global page
☐	3	Done	Add Home page

Hitting Get request from postman:



GET https://x2dnb3xqk5.execute-api.us-east-1.amazonaws.com/alpha/All_end_points

Send
Params Authorization Headers (8) Body Pre-request Script Tests Settings
none form-data x-www-form-urlencoded raw binary GraphQL JSON

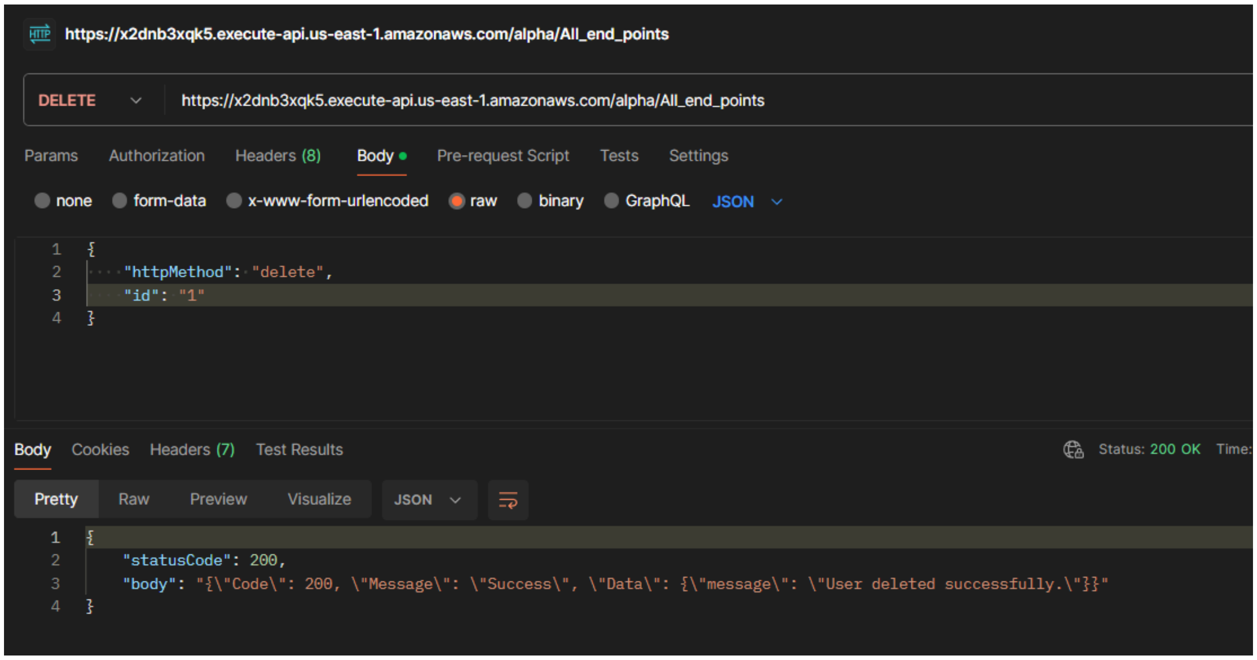
```
{
    "httpMethod": "get"
}
```

Body Cookies Headers (7) Test Results
Save as example
Pretty Raw Preview Visualize JSON

Status: 200 OK Time: 1028 ms Size: 691 B

```
{
    "statusCode": 200,
    "body": "{\"Code\": 200, \"Message\": \"Success\", \"Data\": [{\"todo\": \"UI Update\", \"id\": \"2\", \"status\": \"Pending\"}, {\"todo\": \"Push To git\",
        \"id\": \"1\", \"status\": \"Pending\"}, {\"todo\": \"Add Global page\", \"id\": \"4\", \"status\": \"Pending\"}, {\"todo\": \"Add Home page\", \"id\":
        \"3\", \"status\": \"Done\"}]}"
}
```

Hitting delete request:



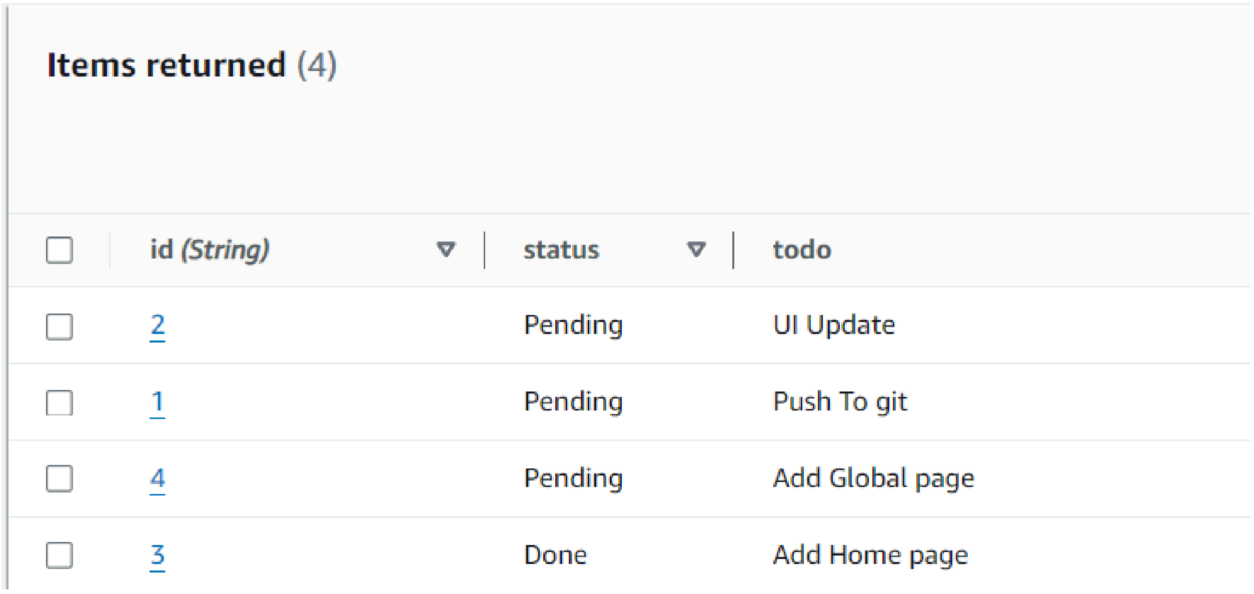
DELETE https://x2dnb3xqk5.execute-api.us-east-1.amazonaws.com/alpha/All_end_points

```
{
    "httpMethod": "delete",
    "id": "1"
}
```

Status: 200 OK

```
{
    "statusCode": 200,
    "body": "{\"Code\": 200, \"Message\": \"Success\", \"Data\": {\"message\": \"User deleted successfully.\"}}"
}
```

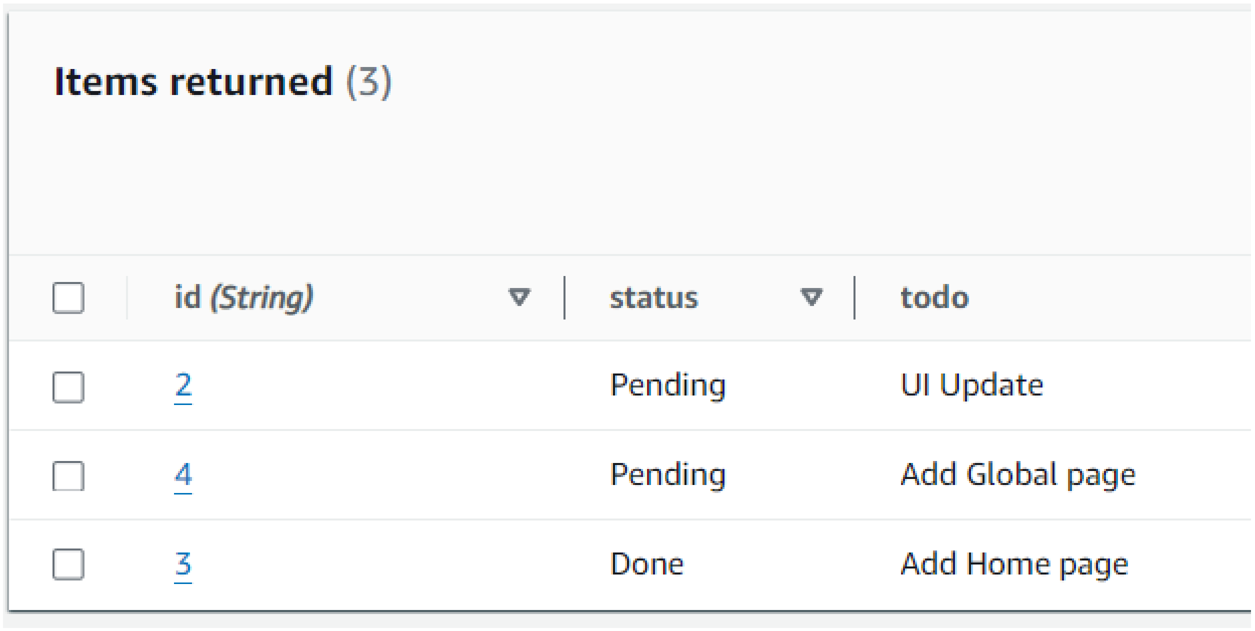
Data before delete request:



Items returned (4)

	id (String)	status	todo
☐	2	Pending	UI Update
☐	1	Pending	Push To git
☐	4	Pending	Add Global page
☐	3	Done	Add Home page

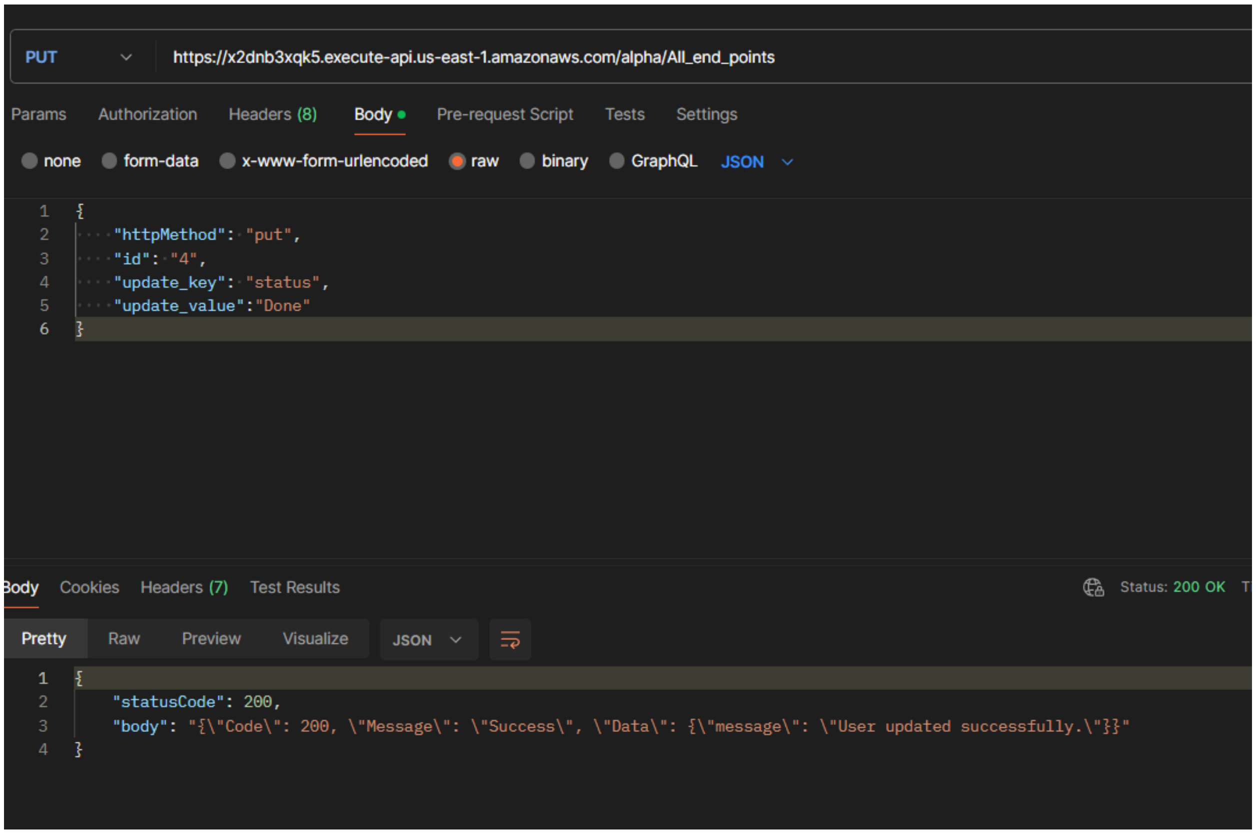
Data after delete request:



Items returned (3)

	id (String)	status	todo
☐	2	Pending	UI Update
☐	4	Pending	Add Global page
☐	3	Done	Add Home page

Put request:



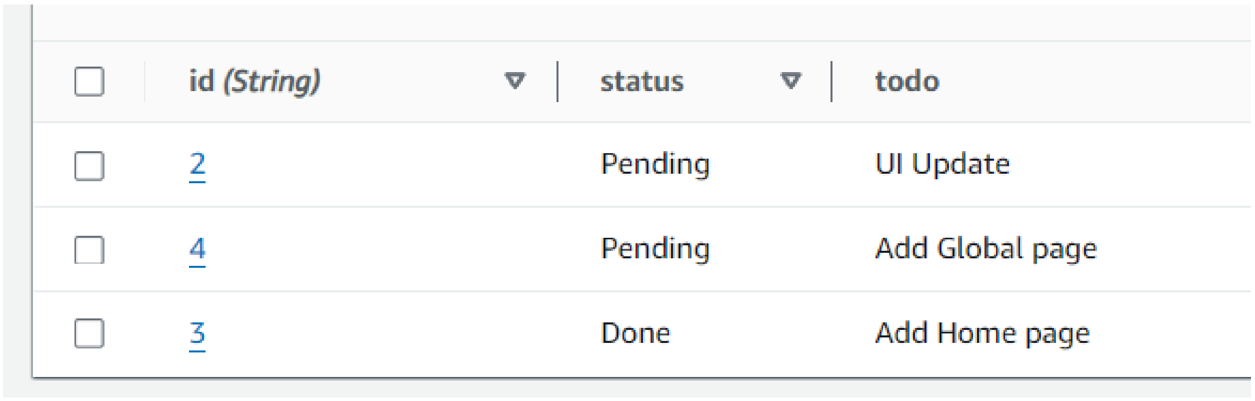
PUT https://x2dnb3xqk5.execute-api.us-east-1.amazonaws.com/alpha/All_end_points

```
{
    "httpMethod": "put",
    "id": "4",
    "update_key": "status",
    "update_value":"Done"
}
```

Status: 200 OK

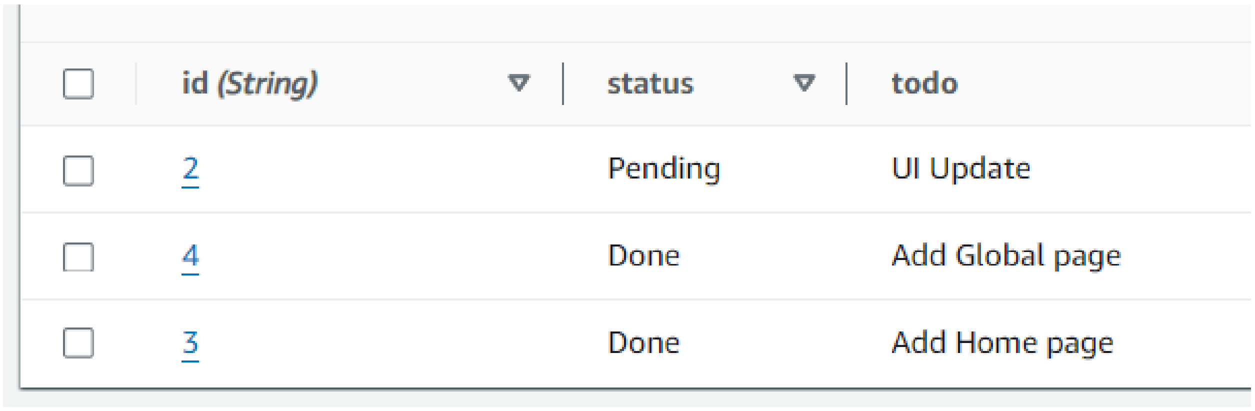
```
{
    "statusCode": 200,
    "body": "{\"Code\": 200, \"Message\": \"Success\", \"Data\": {\"message\": \"User updated successfully.\"}}"
}
```

Data before update(put)



id (String)	status	todo
2	Pending	UI Update
4	Pending	Add Global page
3	Done	Add Home page

Data after update(put)



id (String)	status	todo
2	Pending	UI Update
4	Done	Add Global page
3	Done	Add Home page

This Concludes the serverless lab 2 of Creating a Serverless API.