Build Amazon DynamoDB Integration with AWS SAM and OpenAPI

You can integrate an API method directly with an Amazon DynamoDB without the need for a compute layer such as Lambda in the middle. This is achieved through the **AWS service integration** type.

Use AWS SAM and OpenAPI to create an API Gateway REST API with a direct Amazon DynamoDB integration

1. Using AWS Cloud9 console, return to the root folder `module-3/dynamodb`
2. This code belongs in your SAM template file `template.yaml`

   Review the code and then **copy/paste** it into the `template.yaml` file.

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Description: API Gateway with DynamoDB integration using SAM

Globals:

    # Enable Logs
    Api:
        MethodSettings:
            - ResourcePath: "/*"
              HttpMethod: "*"
              DataTraceEnabled: True
              LoggingLevel: INFO
              MetricsEnabled: True

Resources:
  DynamoDBTable:
    Type: 'AWS::DynamoDB::Table'
    Properties:
      TableName: MyTable
      AttributeDefinitions:
        - AttributeName: id
          AttributeType: S
      KeySchema:
        - AttributeName: id
          KeyType: HASH
      ProvisionedThroughput:
        ReadCapacityUnits: 5
        WriteCapacityUnits: 5

# APIGW Rest API for DynamoDB Integration Example
  RestApiForDynamoDB:
    Type: AWS::Serverless::Api
    Properties:
      StageName: dev
      DefinitionBody: # an OpenApi definition
        'fn::Transform':
          Name: 'AWS::Include'
          Parameters:
            Location: 'openapi.yaml'
        OpenApiVersion: 3.0.3
        EndpointConfiguration:
          Type: REGIONAL

  # IAM Role to allow APIGW to call DynamoDB
  RestApiRole:
    Type: 'AWS::IAM::Role'
    Properties:
      AssumeRolePolicyDocument:
        Version: 2012-10-17
        Statement:
          - Effect: Allow
            Principal:
              Service:
                - apigateway.amazonaws.com
            Action:
              - 'sts:AssumeRole'
      Policies:
      - PolicyName: AllowDynamoDBExec
        PolicyDocument:
          Version: 2012-10-17
          Statement:
            - Effect: Allow
              Action:
                - "dynamodb:Query"
                - "dynamodb:PutItem"
                - "dynamodb:UpdateItem"
              Resource:
                - !GetAtt DynamoDBTable.Arn

Outputs:
  MyApiUrl:
    Description: URL of the API
    Value: !Sub "https://${RestApiForDynamoDB}.execute-api.${AWS::Region}.amazonaws.com/dev"
```

The `DynamoDBTable` resource defines a **DynamoDB** table that will be accessed by the API. The `RestApiRole` resource defines a *Role* that will be used by the routes of the API.

The *Policies* attribute **(lines 58-69)** specifies the actions that will be allowed to the API calls. In this case, the caller will be able to *Query*, *PutItem* and *UpdateItem* of the DynamoDB table referred **(line 69)** .

3. This code belongs in your OpenAPI definition file `openapi.yaml`

   Review the code and then **copy/paste** it into the `openapi.yaml` file

```yaml
openapi: "3.0.1"
info:
  title: "module-3-dynamodb-integration"
  description: "API Gateway example for Amazon DynamoDB Integration"
  version: "1.0"
paths:
  /resource:
    get:
      responses:
        "200":
          description: "200 response"
          content:
            application/json:
              schema:
                $ref: "#/components/schemas/Empty"
      x-amazon-apigateway-integration:
        type: "aws"
        credentials:
          Fn::Sub: "${RestApiRole.Arn}"
        httpMethod: "POST"
        uri: "arn:aws:apigateway:${AWS::Region}:dynamodb:action/Query"
        responses:
          default:
            statusCode: "200"
            responseTemplates:
              application/json: "#set($inputRoot = $input.path('$'))\n{\n    \"Names\"\
                : [\n              #foreach($elem in $inputRoot.Items) {\n                \
                id\": \"$elem.id.S\",\n                \"name\": \"$elem.name.S\",\n \
                \           \"lastName\": \"$elem.lastName.S\",\n                \"email\"\
                : \"$elem.email.S\"\n              }#if($foreach.hasNext),#end\n\t#end\n\
                \            ]\n}"
        requestTemplates:
          application/json: "{\n    \"TableName\": \"MyTable\",\n    \"KeyConditionExpression\"\
            : \"id = :v1\",\n    \"ExpressionAttributeValues\": {\n      \":v1\"\
            : {\n        \"S\": \"$input.params('id')\"\n      }\n    }\n}"
        passthroughBehavior: "when_no_templates"
  /resource/{id}:
    post:
      parameters:
      - name: "id"
        in: "path"
        required: true
        schema:
          type: "string"
      responses:
        "200":
          description: "200 response"
          content:
            application/json:
              schema:
                $ref: "#/components/schemas/Empty"
      x-amazon-apigateway-integration:
        type: "aws"
        credentials:
          Fn::Sub: "${RestApiRole.Arn}"
        httpMethod: "POST"
        uri: "arn:aws:apigateway:${AWS::Region}:dynamodb:action/PutItem"
        responses:
          default:
            statusCode: "200"
        requestTemplates:
          application/json: "{\n    \"TableName\": \"MyTable\",\n    \"Item\": {\n\
            \t    \"id\": {\n                \"S\": \"$input.params('id')\"\n      \
            \      },\n            \"name\": {\n                \"S\": \"$input.path('$.name')\"\
            \n            },\n            \"lastName\": {\n                \"S\": \"$input.path('$.lastName')\"\
            \n            },\n            \"email\": {\n                \"S\": \"$input.path('$.email')\"\
            \n            }\n    }\n}"
        passthroughBehavior: "when_no_templates"
components:
  schemas:
    Empty:
      title: "Empty Schema"
      type: "object"
```

Lines **(27-57)** are particular important as these are what create the direct service integration for DynamoDB. The x-amazon-apigateway-integration is an OpenAPI extension that allows us to integrate our APIs to AWS backends.

On the `/resource` path, lines **(33-35)** defines the model schema of the *Query* request to be made using the json schema pattern. This will transform the simple pageid path parameter on the GET request to the needed DynamoDB Query API, which requires an HTTP POST. Lines **(26-31)** defines the response template from the *Query*. DynamoDB uses a specific query syntax that is been transformed for json one.

On the `/resource/{id}` path, lines **(33-35)** defines the model schema of the *PutItem* request. This mapping template creates the JSON structure required by the DynamoDB PutItem API. The entire mapping template is static. The three input variables are referenced from the request json using the `$input` variable and each comment is stamped with a unique identifier. Also, the `id` is obtained from the parameters of the request.

Deploy the project

1. To deploy the API Gateway and Kinesis resources to your AWS account, run the following commands from the application root `module-3/dynamodb`, where the `template.yaml` file for the sample application is located:

```
sam build && sam deploy --guided
```

The first time that you run the `sam deploy --guided` command, AWS SAM starts an AWS CloudFormation deployment. In this case, you need to say what are the configurations that you want SAM to have in order to get the guided deployment. You can configure it as below.

- **Stack Name**: `module-3-dynamodb-integration`
- **AWS Region**: Put the chosen region to run the workshop. **e.g.** `us-east-1`
- **Parameter KinesisStreamName**: leave blank
- **Confirm changes before deploy**: Y
- **Allow SAM CLI IAM role creation**: Y
- **Disable rollback**: N
- **Save arguments to configuration file**: Y
- **SAM configuration file** and **SAM configuration environment** leave blank

```
Configuring SAM deploy
======================

        Looking for config file [samconfig.toml] :  Found
        Reading default arguments  :  Success

        Setting default arguments for 'sam deploy'
        =========================================
        Stack Name [module-3-dynamodb-integration]: module-3-dynamodb-integration
        AWS Region [us-west-2]:
        #Shows you resources changes to be deployed and require a 'Y' to initiate deploy
        Confirm changes before deploy [Y/n]: Y
        #SAM needs permission to be able to create roles to connect to the resources in your template
        Allow SAM CLI IAM role creation [Y/n]: Y
        #Preserves the state of previously provisioned resources when an operation fails
        Disable rollback [y/N]: N
        Save arguments to configuration file [Y/n]: Y
        SAM configuration file [samconfig.toml]:
        SAM configuration environment [default]:
```

2. After configuring the deployment, AWS SAM will display assets that will be created. But first, it will automatically upload the template to a temporary bucket it creates. Then, it will ask you to confirm the changes. Type `y` to confirm.

```
CloudFormation outputs from deployed stack
----------------------------------------------------------------------------------------------------------------------
Outputs
----------------------------------------------------------------------------------------------------------------------
Key                 MyApiUrl
Description         URL of the API
Value               https://[redacted]execute-api.us-west-2.amazonaws.com/dev
----------------------------------------------------------------------------------------------------------------------

Successfully created/updated stack - module-3-dynamodb-integration in us-west-2
```

3. Once the deployment has been successful, you will see an '**Outputs**' section that contains the API Gateway invoke URL

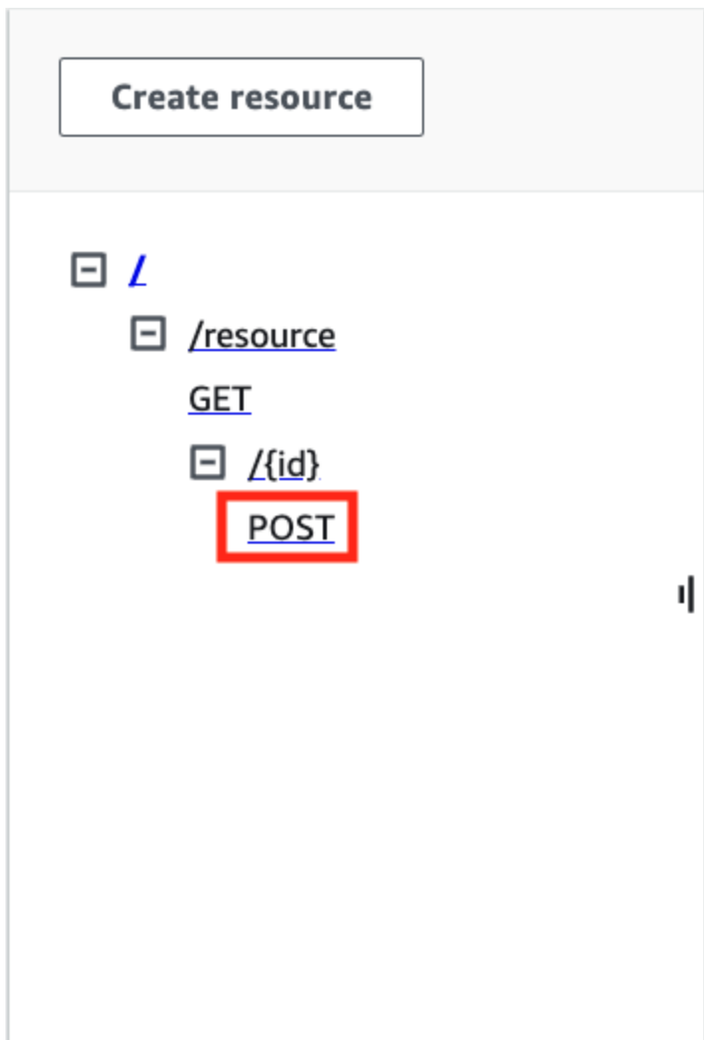
Test the Integration

Now the API has been deployed, the table is in place, its time to test the integration.

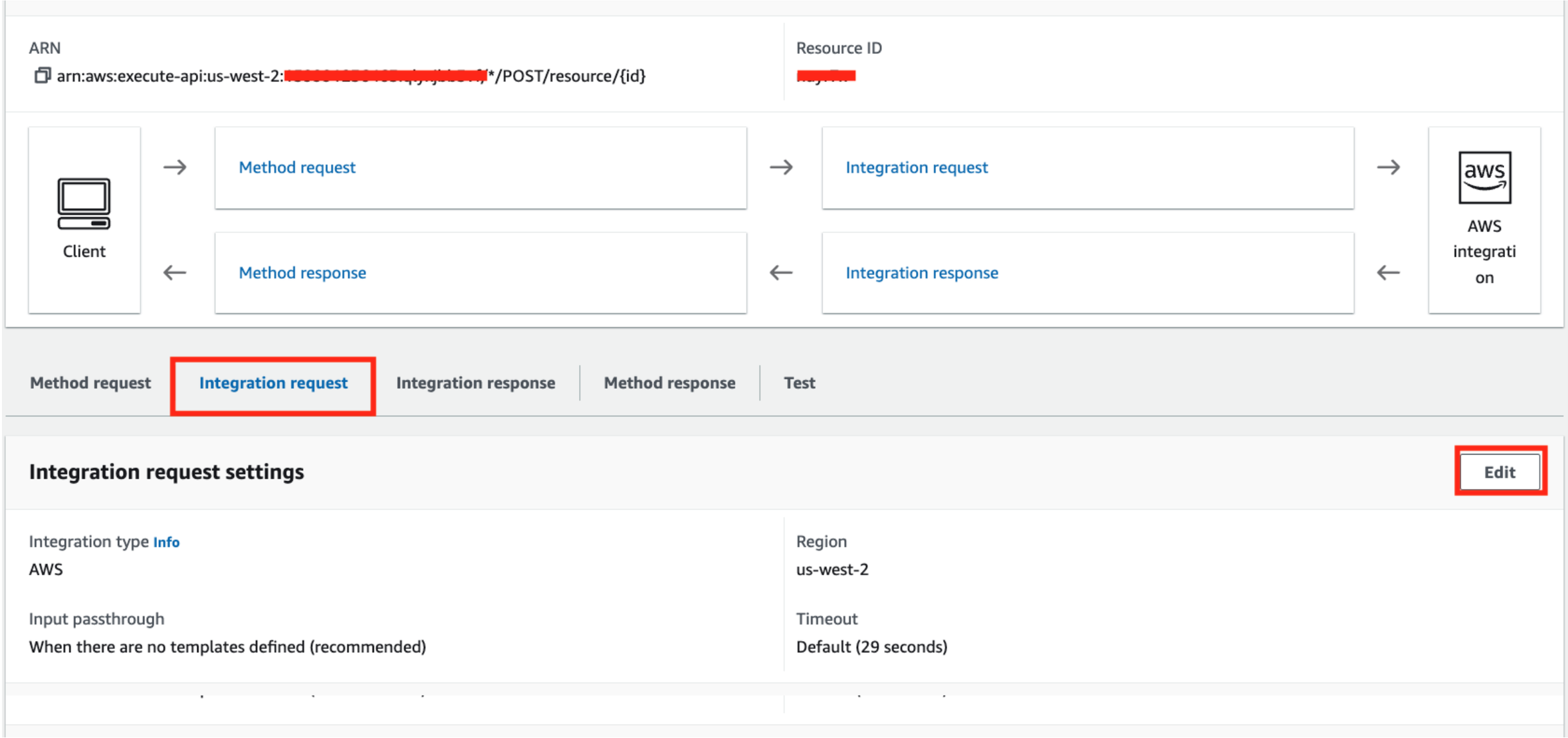
Test DynamoDB Integration using API Gateway console to WRITE an item

First, let's test the **write** integration (*PutItem* api call).

1. Navigate to the API Gateway console and select the API that has been created. It will be named **module-3-dynamodb-integration**.
2. Select the POST method from the resources section to the left of the screen to bring up the API request flow window.

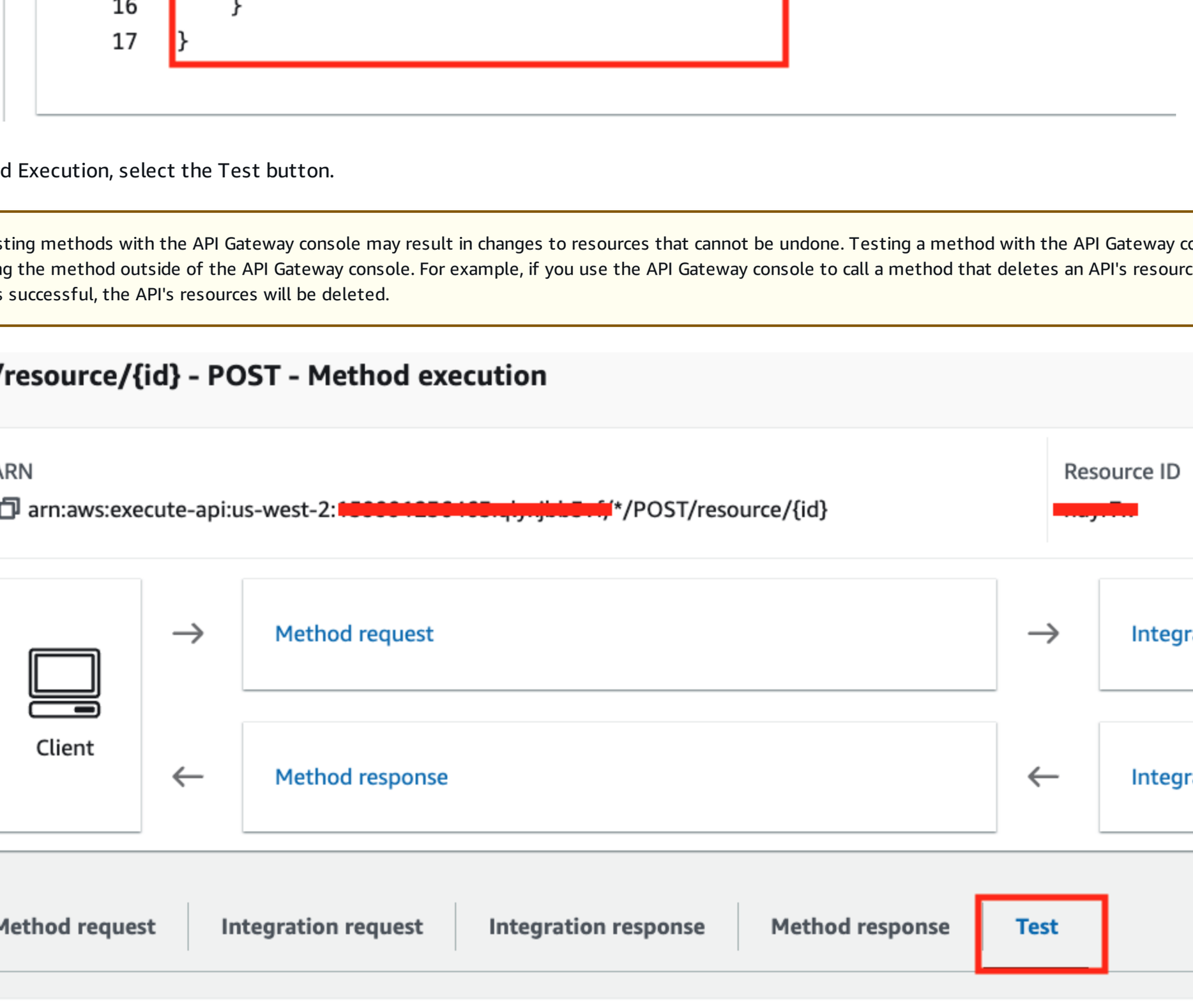


3. In the *Method Execution*, click in **Integration Request** and then **Edit**.



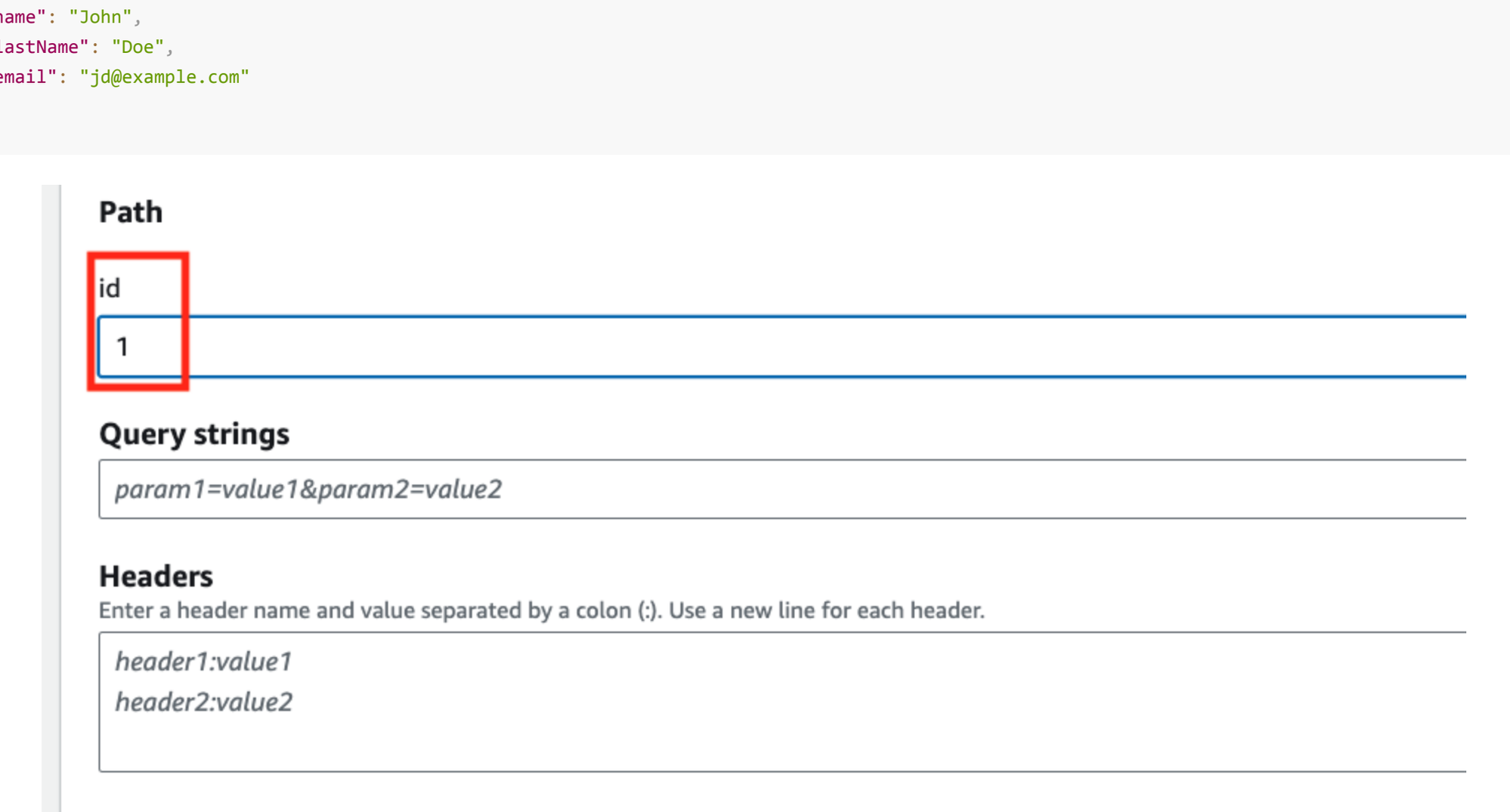
4. Review the **Integration Request** configuration.

5. Click on **Save**. In **Integration Request** tab go down to the **Mapping Template** to see the input transformation request template:



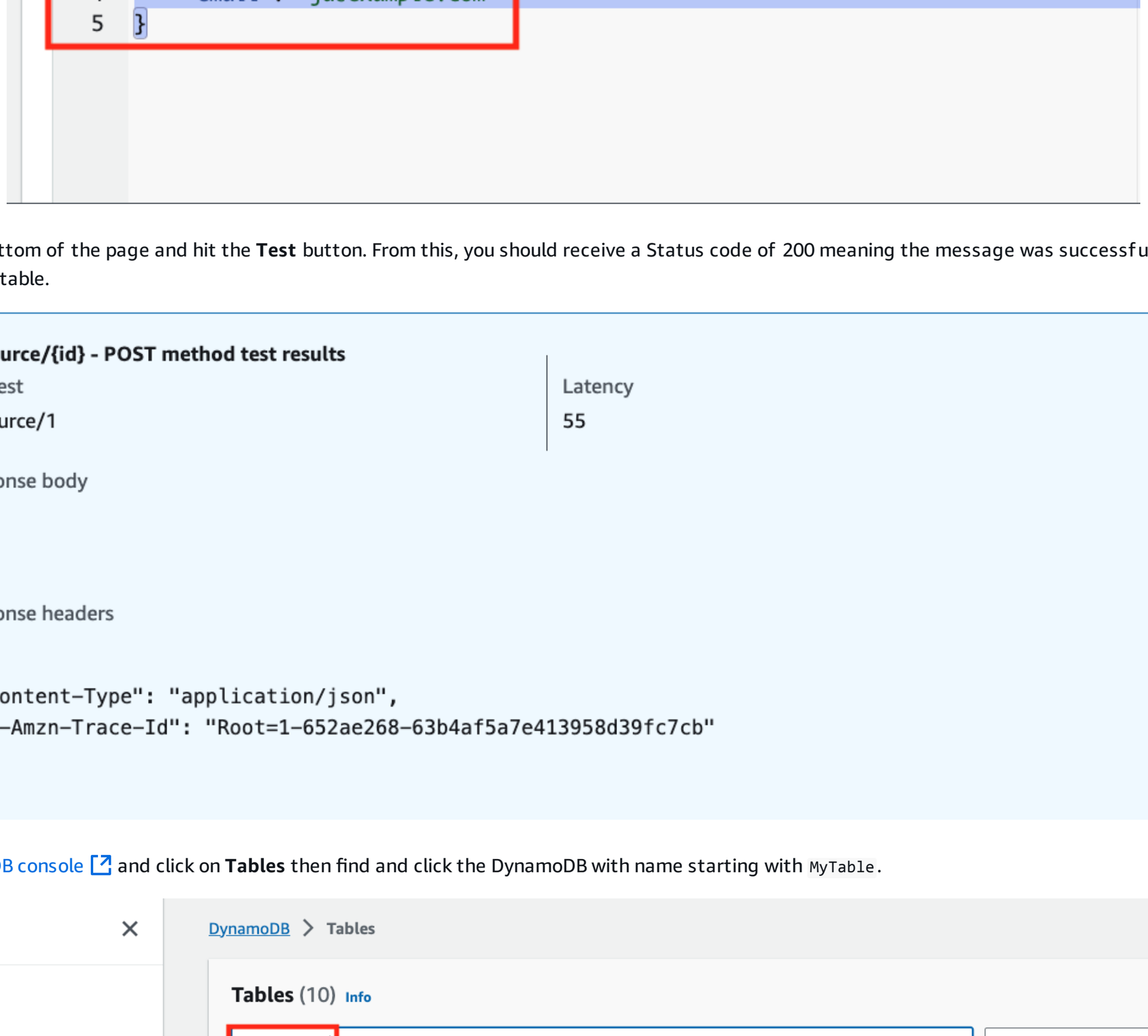
6. Go back to Method Execution, select the Test button.

Warning! Testing methods with the API Gateway console may result in changes to resources that cannot be undone. Testing a method with the API Gateway console is the same as calling the method outside of the API Gateway console. For example, if you use the API Gateway console to call a method that deletes an API's resources, if the method call is successful, the API's resources will be deleted.

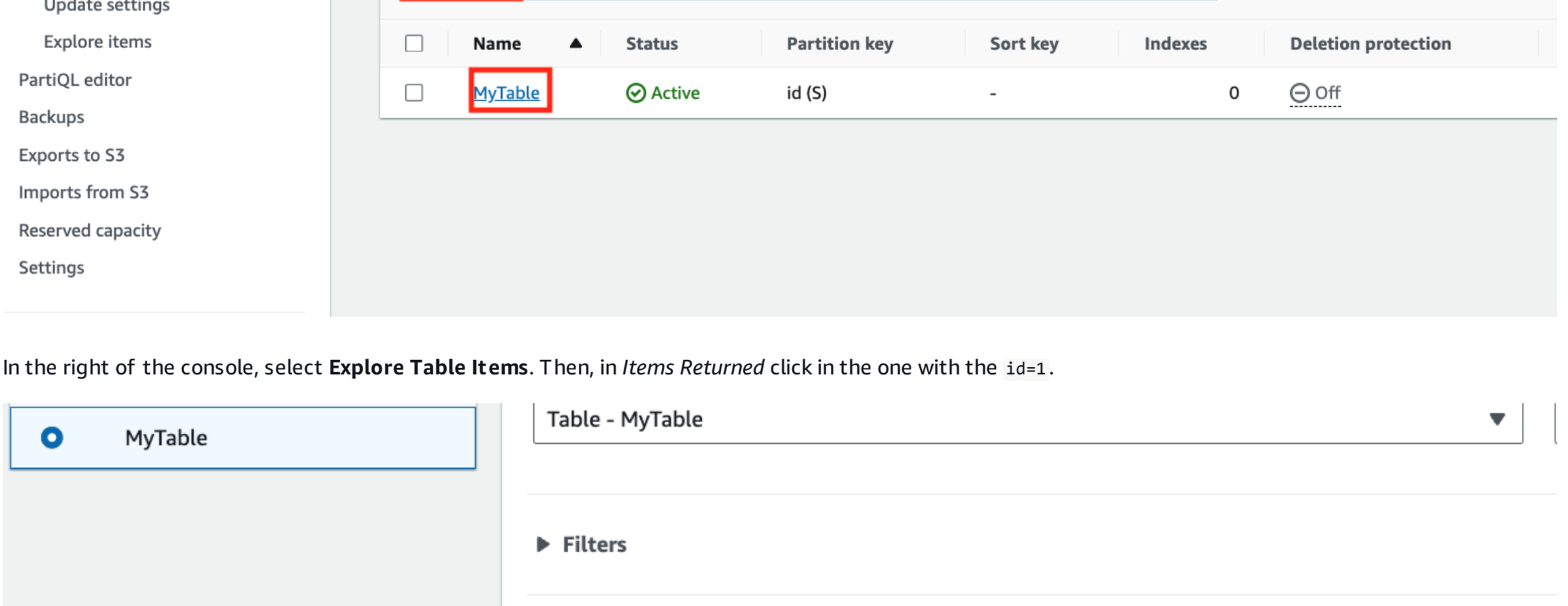


7. This will open the window that allows you to test invocations to your backend. In our case, we want to send a message to the DynamoDB table that we created early. Fill the *id* field and also fill the request body with the following json.

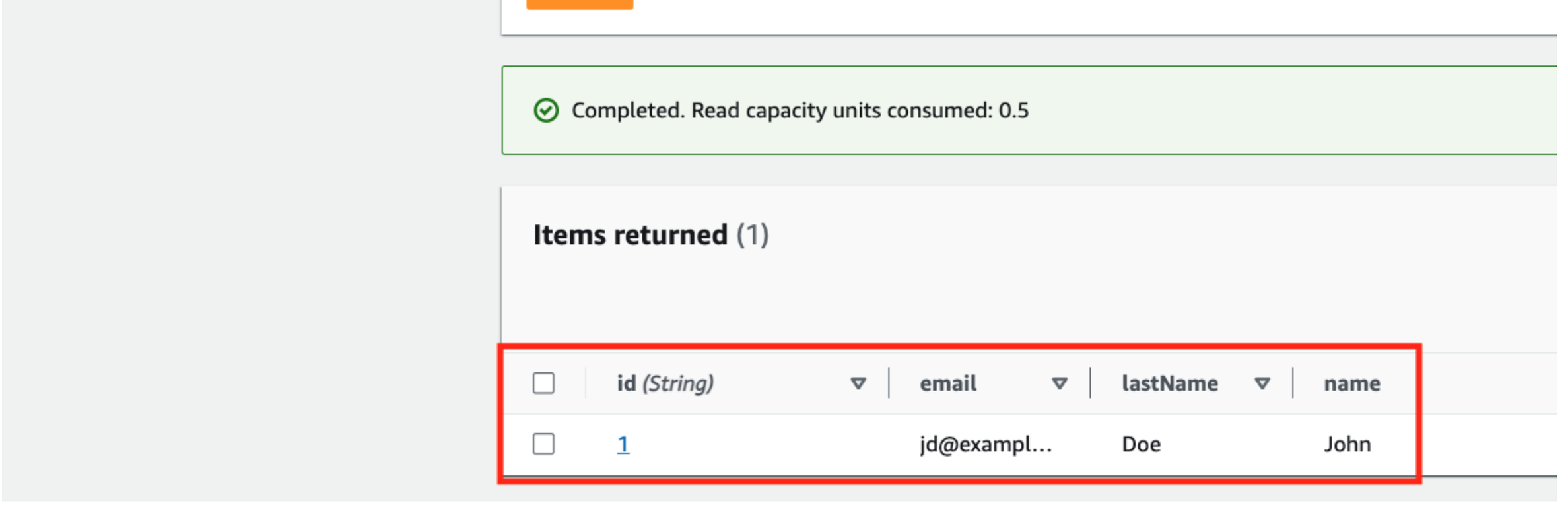
```
{
    "name": "John",
    "lastName": "Doe",
    "email": "jd@example.com"
}
```



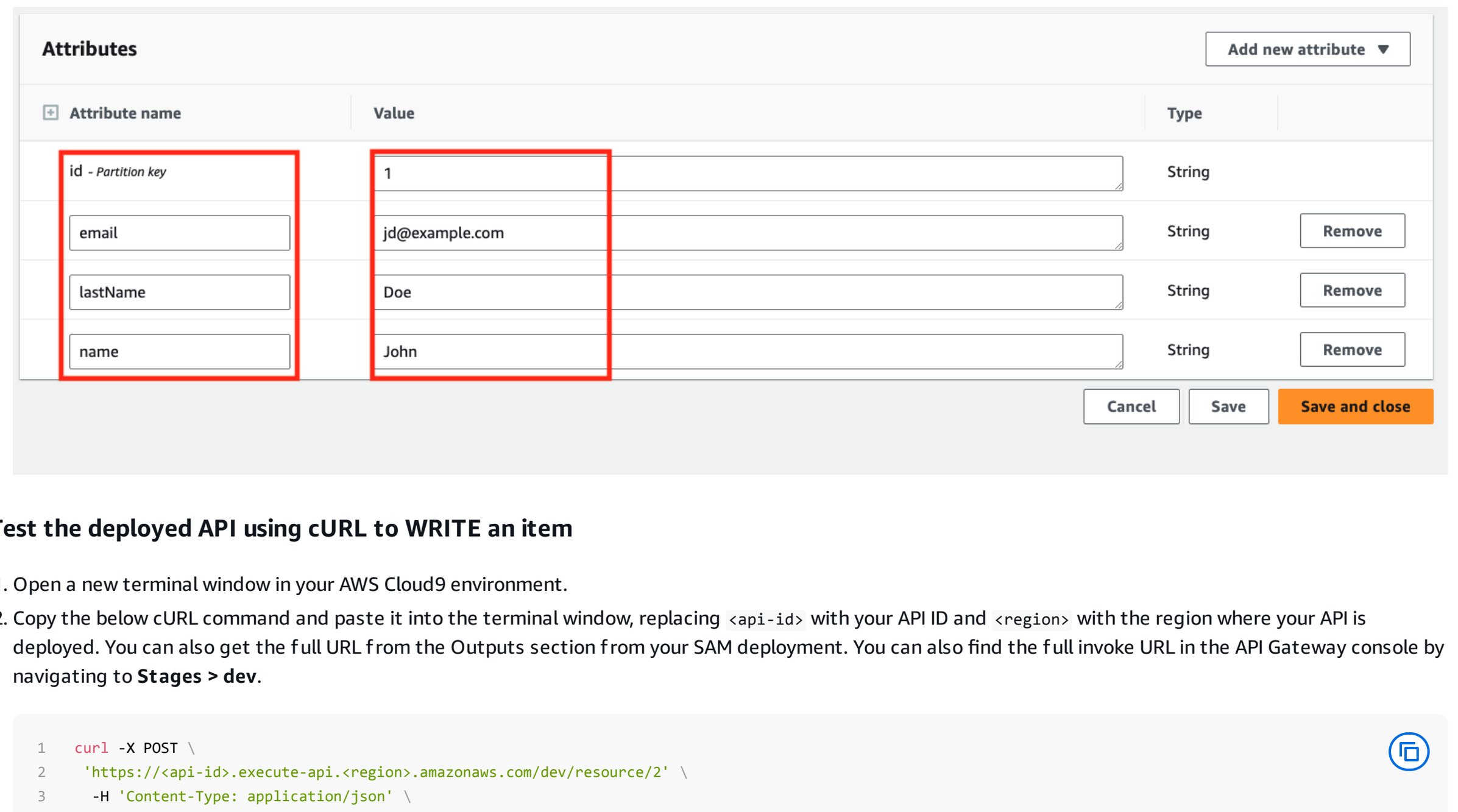
8. Scroll to the bottom of the page and hit the **Test** button. From this, you should receive a Status code of 200 meaning the message was successfully delivered to the DynamoDB table.



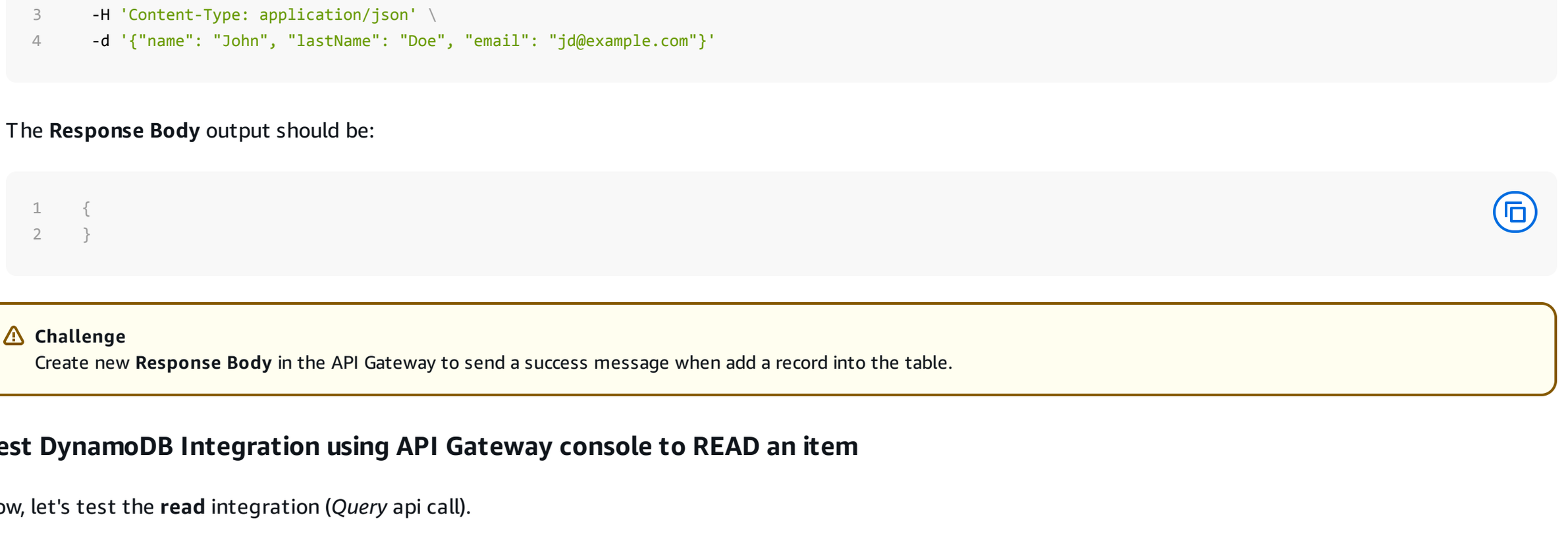
9. Go to DynamoDB console and click on **Tables** then find and click the DynamoDB with name starting with `MyTable`.



10. In the right of the console, select **Explore Table Items**. Then, in *Items Returned* click in the one with the `id=1`.



11. Now you can see that the item was added correctly.



Test the deployed API using cURL to WRITE an item

1. Open a new terminal window in your AWS Cloud9 environment.
2. Copy the below cURL command and paste it into the terminal window, replacing `<api-id>` with your API ID and `<region>` with the region where your API is deployed. You can also get the full URL from the Outputs section from your SAM deployment. You can also find the full invoke URL in the API Gateway console by navigating to **Stages > dev**.

```
curl -X POST \
 'https://<api-id>.execute-api.<region>.amazonaws.com/dev/resource/2' \
  -H 'Content-Type: application/json' \
  -d '{"name": "John", "lastName": "Doe", "email": "jd@example.com"}'
```

3. The **Response Body** output should be:

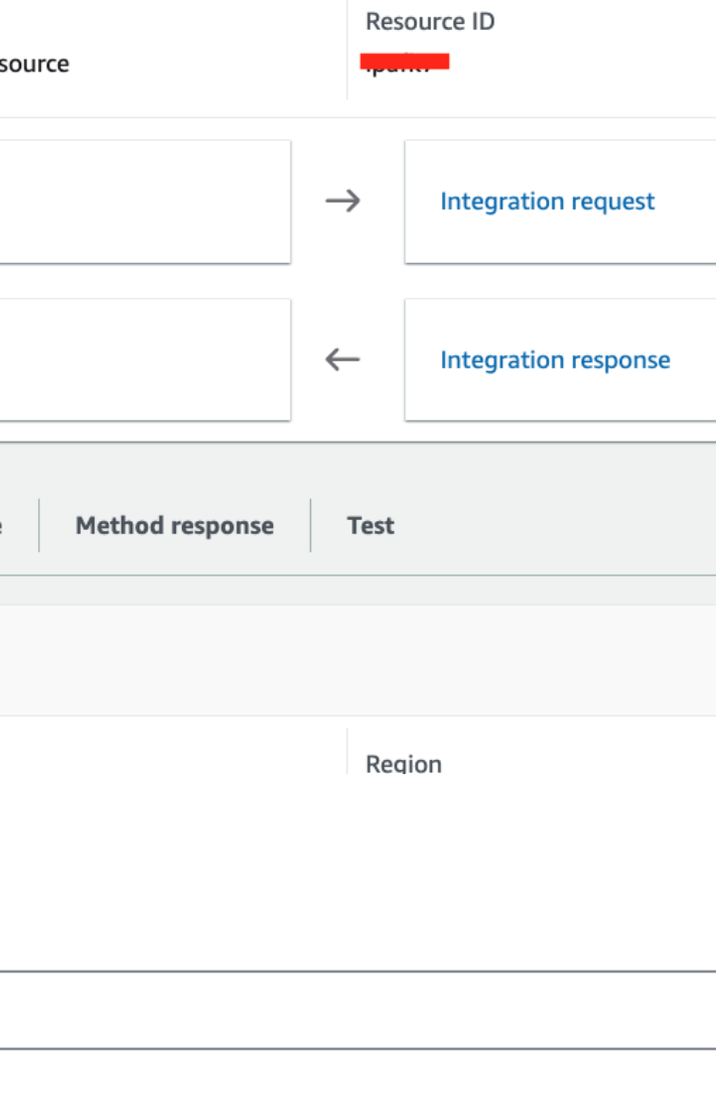
```
{
}
```

Challenge
Create new **Response Body** in the API Gateway to send a success message when add a record into the table.

Test DynamoDB Integration using API Gateway console to READ an item

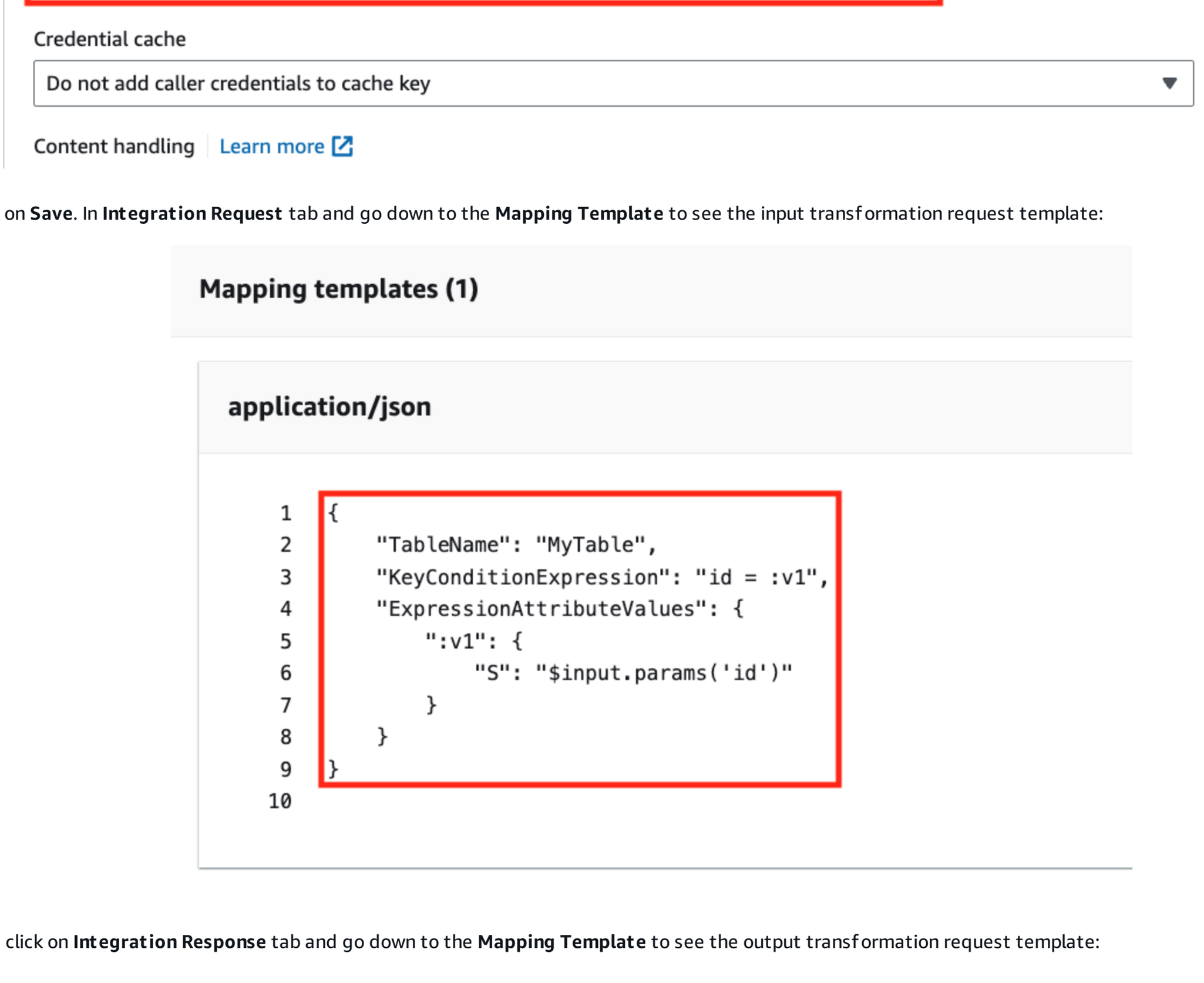
Now, let's test the **read** integration (*Query* api call).

1. Navigate to the API Gateway console and select the API that has been created. It will be named **module-3-dynamodb-integration**.
2. Select the GET method from the resources section to the left of the screen to bring up the API request flow window.



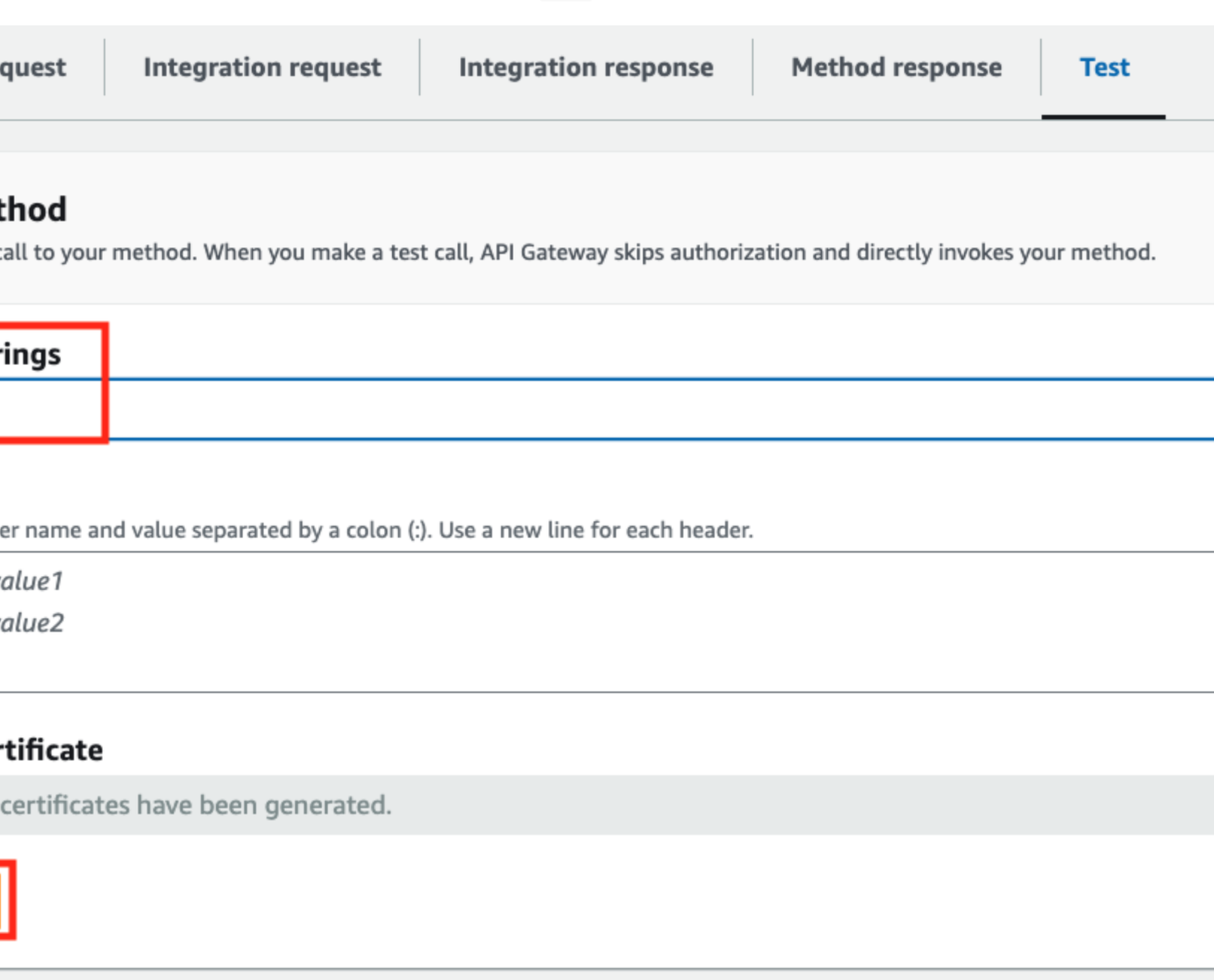
3. In the *Method Execution*, click in **Integration Request** and then **Edit**.

4. Review the **Integration Request** configuration.

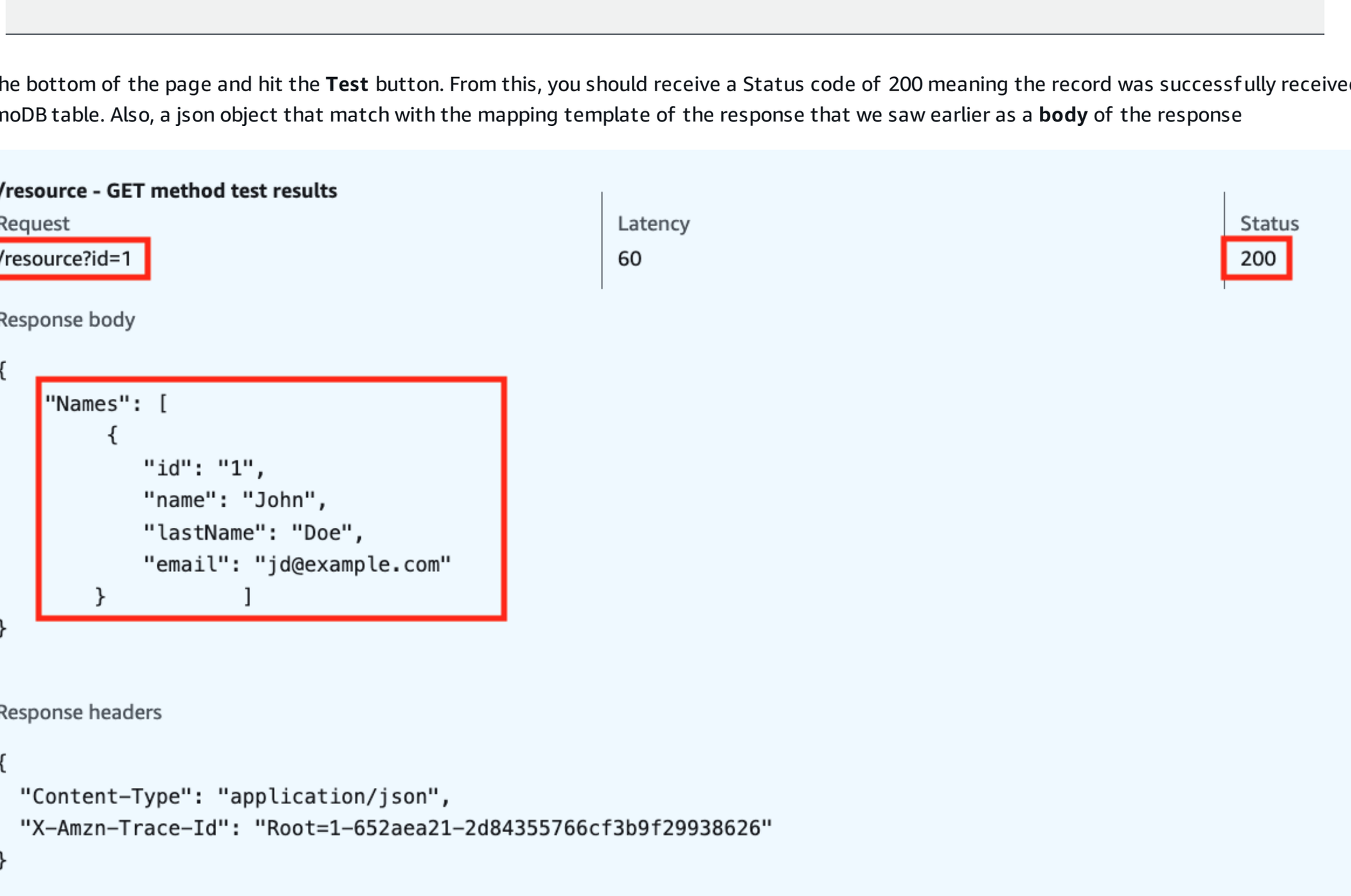


5. Click on **Save**. In **Integration Request** tab and go down to the **Mapping Template** to see the input transformation request template:

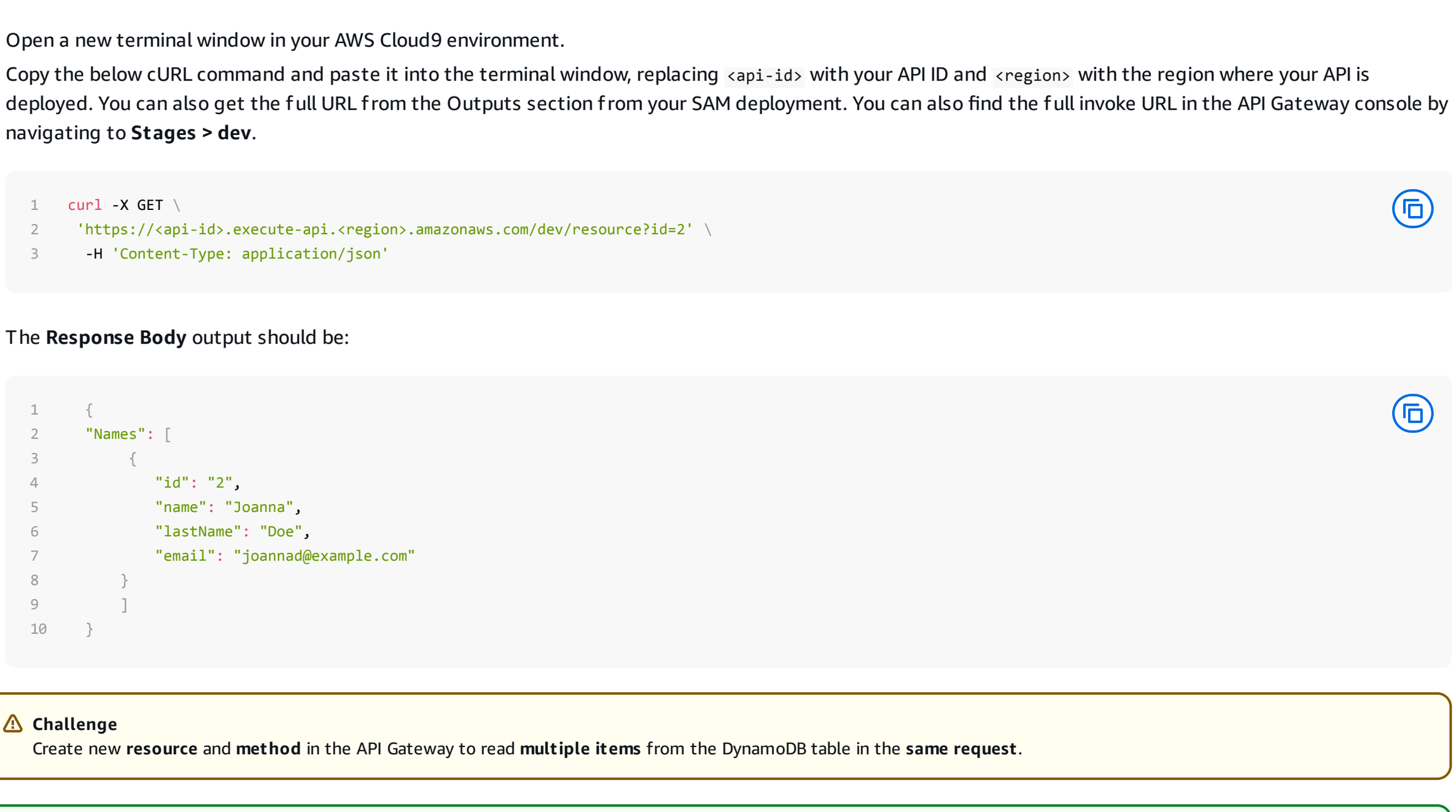
6. Now, click on **Integration Response** tab and go down to the **Mapping Template** to see the output transformation request template:



7. Now, click on **Integration Response** tab. This will open the window that allows you to test invocations to your backend. In our case, we want to get a record from the DynamoDB table that we created early. Fill the **Query String** field with `id=1` and click on the **Test** button.



8. Scroll to the bottom of the page and hit the **Test** button. From this, you should receive a Status code of 200 meaning the record was successfully received to the DynamoDB table. Also, a json object that match with the mapping template of the response that we saw earlier as a **body** of the response



Test the deployed API using cURL to READ an item

1. Open a new terminal window in your AWS Cloud9 environment.
2. Copy the below cURL command and paste it into the terminal window, replacing `<api-id>` with your API ID and `<region>` with the region where your API is deployed. You can also get the full URL from the Outputs section from your SAM deployment. You can also find the full invoke URL in the API Gateway console by navigating to **Stages > dev**.

```
curl -X GET \
 'https://<api-id>.execute-api.<region>.amazonaws.com/dev/resource?id=2' \
  -H 'Content-Type: application/json'
```

3. The **Response Body** output should be:

```
{
"Names": [
    {
      "id": "2",
      "name": "Joanna",
      "lastName": "Doe",
      "email": "joannad@example.com"
    }
    ]
}
```

Challenge
Create new **resource** and **method** in the API Gateway to read **multiple items** from the DynamoDB table in the **same request**.