Rahul Lokurte
Posted on 29 Aug 2021

How to Validate Requests to the AWS API Gateway using CDK

#aws #cdk #apigateway #lambda

Amazon API Gateway is an AWS service that helps to create REST and WebSocket APIs easily. It acts as the first entry point for applications to access data, business logic, or functionality from your backend services. It handles traffic management, authorization and access control, monitoring, and API version management.

If you have written backend services, you must have written lots of request validations and transformation logics in the backend services. When the request comes to the backend service, you validate the request and do the transformations and return the response. Can we avoid the request coming to the backend if it is invalid?. The answer to this question is provided by **AWS API Gateway**.

This blog post will show how to validate the request before it goes to the backend service. We will use the **AWS Lambda function** as a backend service and expose API with querystring parameters and request body validations.

Create API Gateway and Lambda using CDK

we will create an API Gateway and integrate it with Lambda Function using AWS CDK.

Create a new directory on your system.

```
mkdir cdk-apigateway && cd cdk-apigateway
```

We will use cdk init to create a new Javascript CDK project:

```
cdk init --language javascript
```

The cdk init command creates a number of files and folders inside the cdk-apigateway directory to help us organize the source code for your AWS CDK app.

We can list the stacks in our app by running the below command. It will show CdkApigatewayStack.

```
cdk ls
```

Let us install the AWS Lambda and API gateway construct.

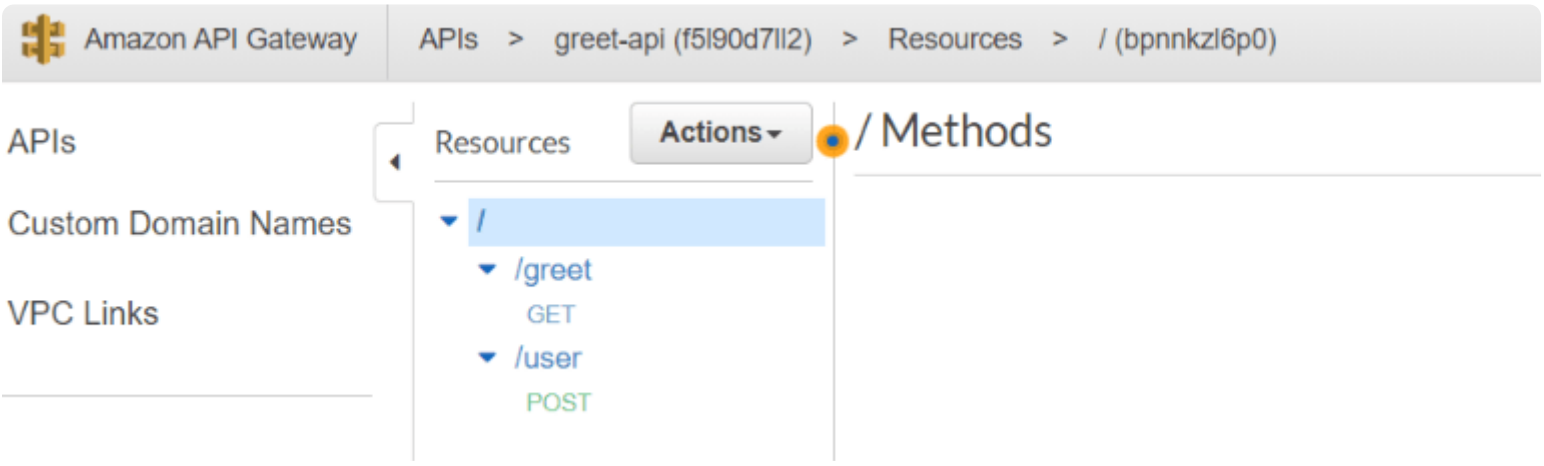
```
npm install @aws-cdk/aws-lambda @aws-cdk/aws-apigateway
```

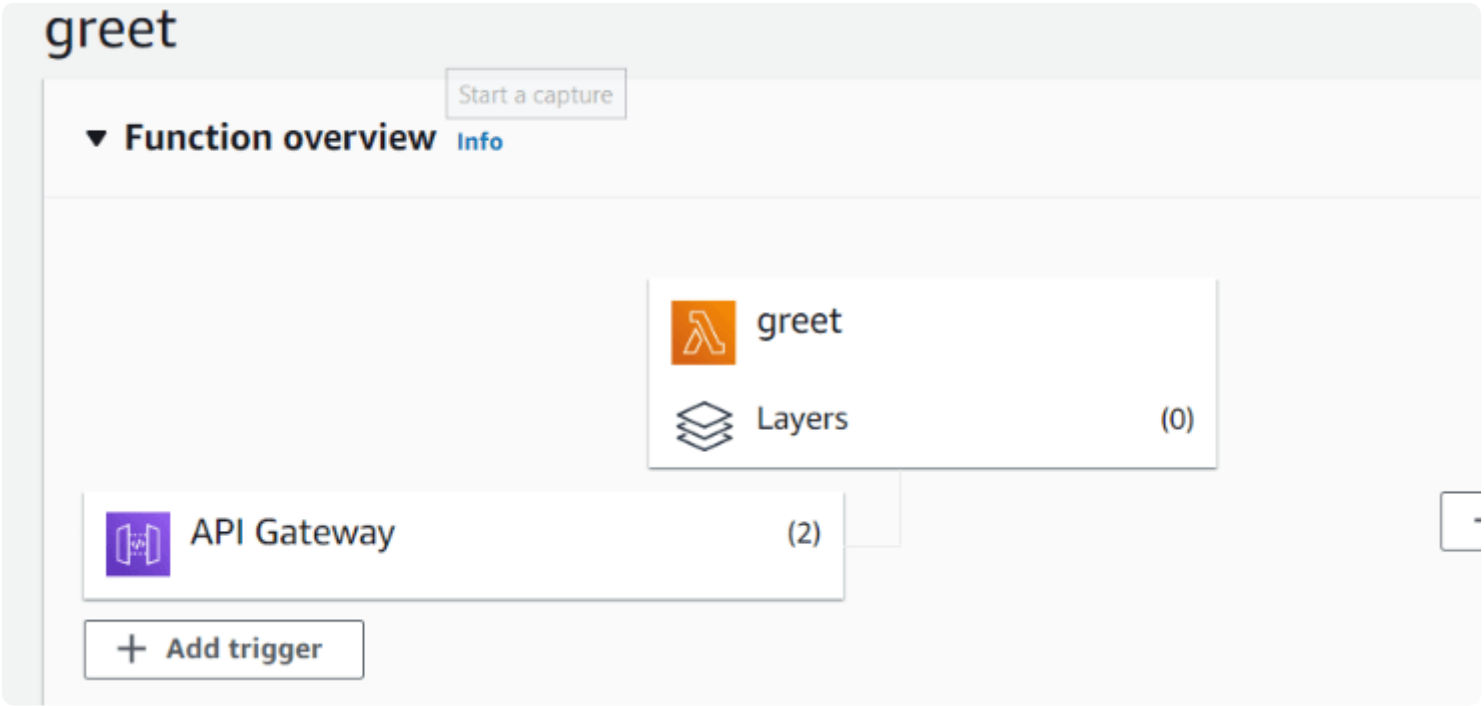
Edit the file **lib/cdk-apigateway-stack.js** to create an API Gateway and AWS lambda resource as shown below.

```
const cdk = require("@aws-cdk/core");
const apigateway = require("@aws-cdk/aws-apigateway");
const lambda = require("@aws-cdk/aws-lambda");

class CdkApigatewayStack extends cdk.Stack {
  /**
   *
   * @param {cdk.Construct} scope
   * @param {string} id
   * @param {cdk.StackProps=} props
   */
  constructor(scope, id, props) {
    super(scope, id, props);

    const greet = new lambda.Function(this, "GreetHandler", {
      runtime: lambda.Runtime.NODEJS_14_X,
      code: lambda.Code.fromAsset("lambda"),
      handler: "greet.handler",
      functionName: "greet",
    });

    const greetApi = new apigateway.LambdaRestApi(this, "GreetLambda", {
      handler: greet,
      proxy: false,
      restApiName: "greet-api",
    });
```

```
    const greetApiIntegration = new apigateway.LambdaIntegration(greet);
    const items = greetApi.root.addResource("greet");

    const users = greetApi.root.addResource("user");
    items.addMethod("GET", greetApiIntegration);
    users.addMethod("POST", greetApiIntegration);
  }
}

module.exports = { CdkApigatewayStack };
```

Now, we can deploy them using `cdk deploy` and then verify resources are created in AWS Console.





Request Validation

API Gateway can perform the basic validation. When the validation fails, API Gateway immediately fails the request, returns a 400 error response to the calling application. The AWS API Gateway contains three types of validators:

1. **Validate body** - This validator will only validate the body of the request.
2. **Validate query string parameters and headers** - This validator will only validate the

parameters coming in the request.

3. **Validate body, query string parameters, and headers** - This validator will validate both body and parameters of the request.

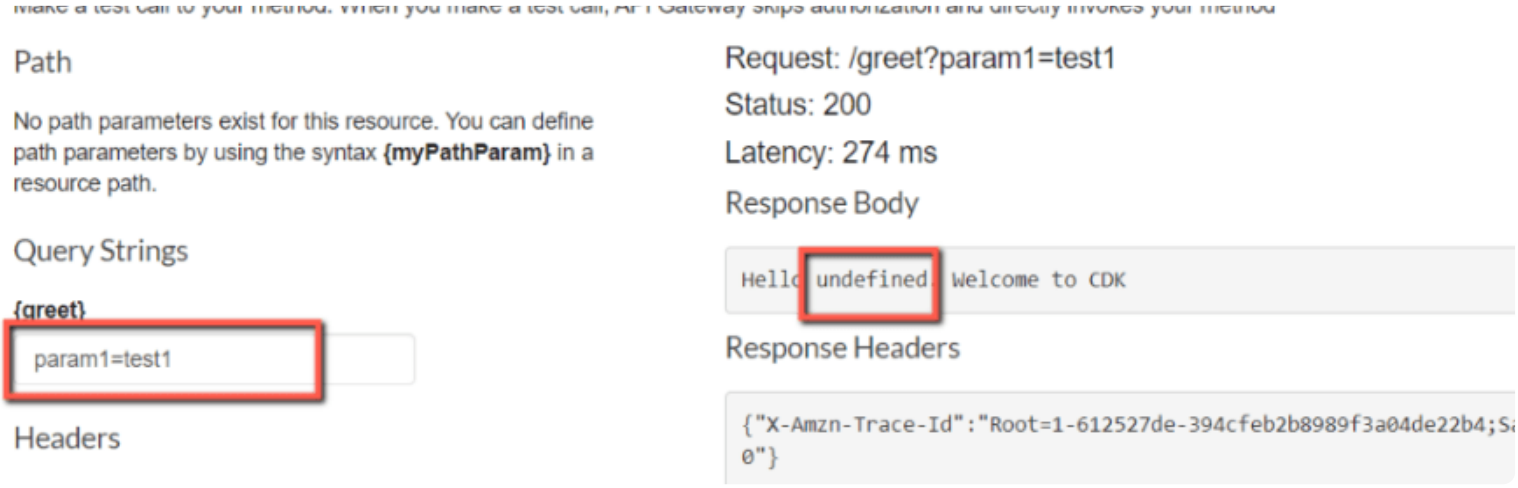
Validate query string parameters

Let us see, how to validate the query string parameter.

If you see the below lambda code, it is expecting us to pass a query string named **greetName**

```
if (event.httpMethod === "GET") {
    let response = {
      statusCode: 200,
      body: `Hello ${event.queryStringParameters.greetName}. Welcome to CDK`,
    };
    return response;
  }
```

If we do not add the validations at the API Gateway level, the request goes to the lambda function and we get **undefined** for the variable **greetName** in the response.

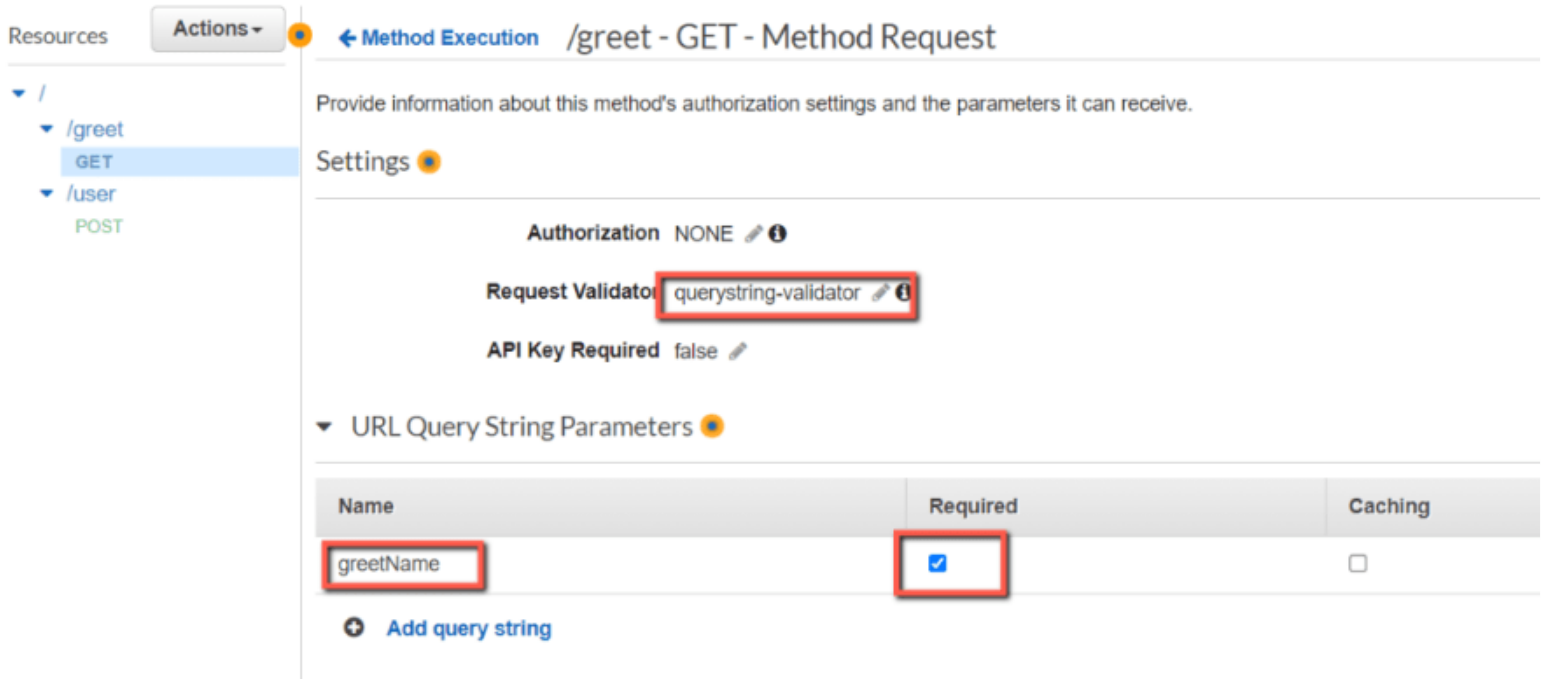


Now, let us edit **lib/cdk-apigateway-stack.js** to add the request validation for the query string parameter to the API Gateway using CDK as shown below.

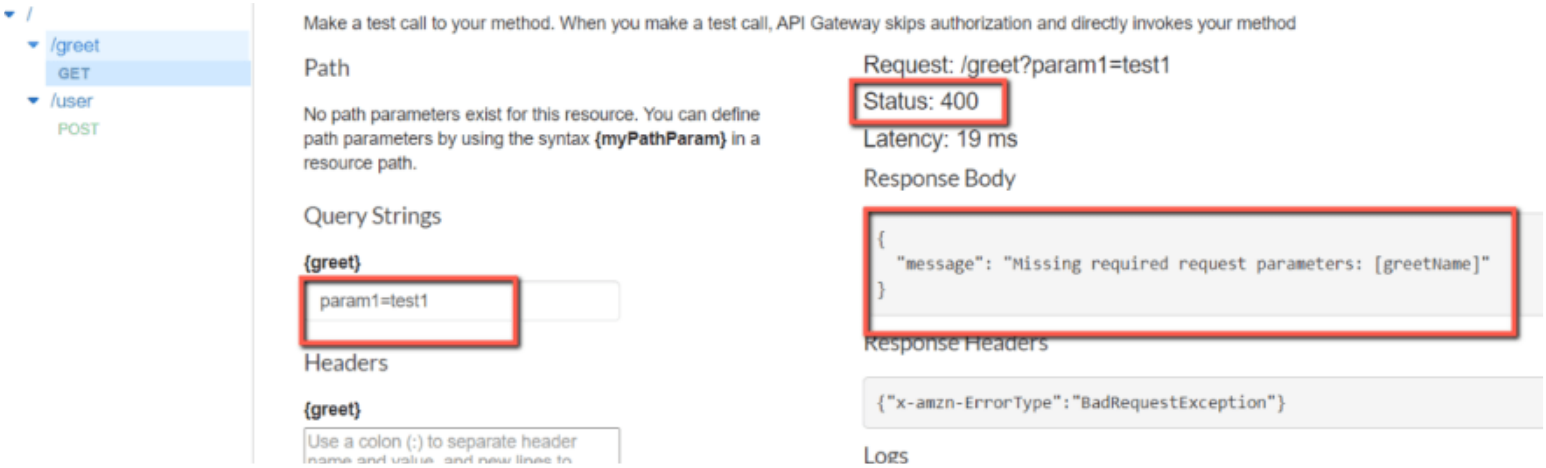
```
items.addMethod("GET", greetApiIntegration, {
      authorizationType: apigateway.AuthorizationType.IAM,
      requestParameters: {
        "method.request.querystring.greetName": true,
      },
      requestValidatorOptions: {
        requestValidatorName: "querystring-validator",
        validateRequestParameters: true,
        validateRequestBody: false,
      },
    });
```

We have added `requestParameters` where we need to specify required request parameters as a key-value pair. A key must match the format `method.request.location.name`, where the location may be `querystring`, `path` or `header` depending on what we want to validate. In our case, it is `querystring` which we want to validate. The name is the valid, unique parameter name. In our case, it is `greetName`. The Value is the boolean, which indicates, if the parameter is required or not. In our case, we want the request to contain a querystring, hence it is set to `true`.

Now deploy the code using the command `cdk deploy` and verify if the validation is added in the AWS console as shown below.



Now, let us send an invalid request, we immediately get an error indicating that the required querystring is not passed in the request.



The same steps can be followed for **pathparameters** and the **headers** validation.

Validate Request body

Let us see, how to validate the body of the request. This is a little bit complex than validating the querystring and headers.

If you see the below lambda code, it is expecting us to have `username, department, departmentName.`

```
if (event.httpMethod === "POST") {
    let request = JSON.parse(event.body);
    let username = request.username;
    let department = request.department;
    let departmentName = department.departmentName;

    let user = { username: username, departmentName: departmentName };
    const response = {
      statusCode: 200,
      body: JSON.stringify(user),
    };
    return response;
  }
```

We need to first create a schema model to validate the request. Let us add the below schema.

```
const greetModel = new apigateway.Model(this, "model-validator", {
      restApi: greetApi,
      contentType: "application/json",
      description: "To validate the request body",
      modelName: "greetmodelcdk",
      schema: {
        type: JsonSchemaType.OBJECT,
        required: ["username"],
        properties: {
          username: { type: "string" },
          department: {
            type: "object",
            properties: {
              departmentName: { type: "string" },
            },
          },
        },
      },
    });
```

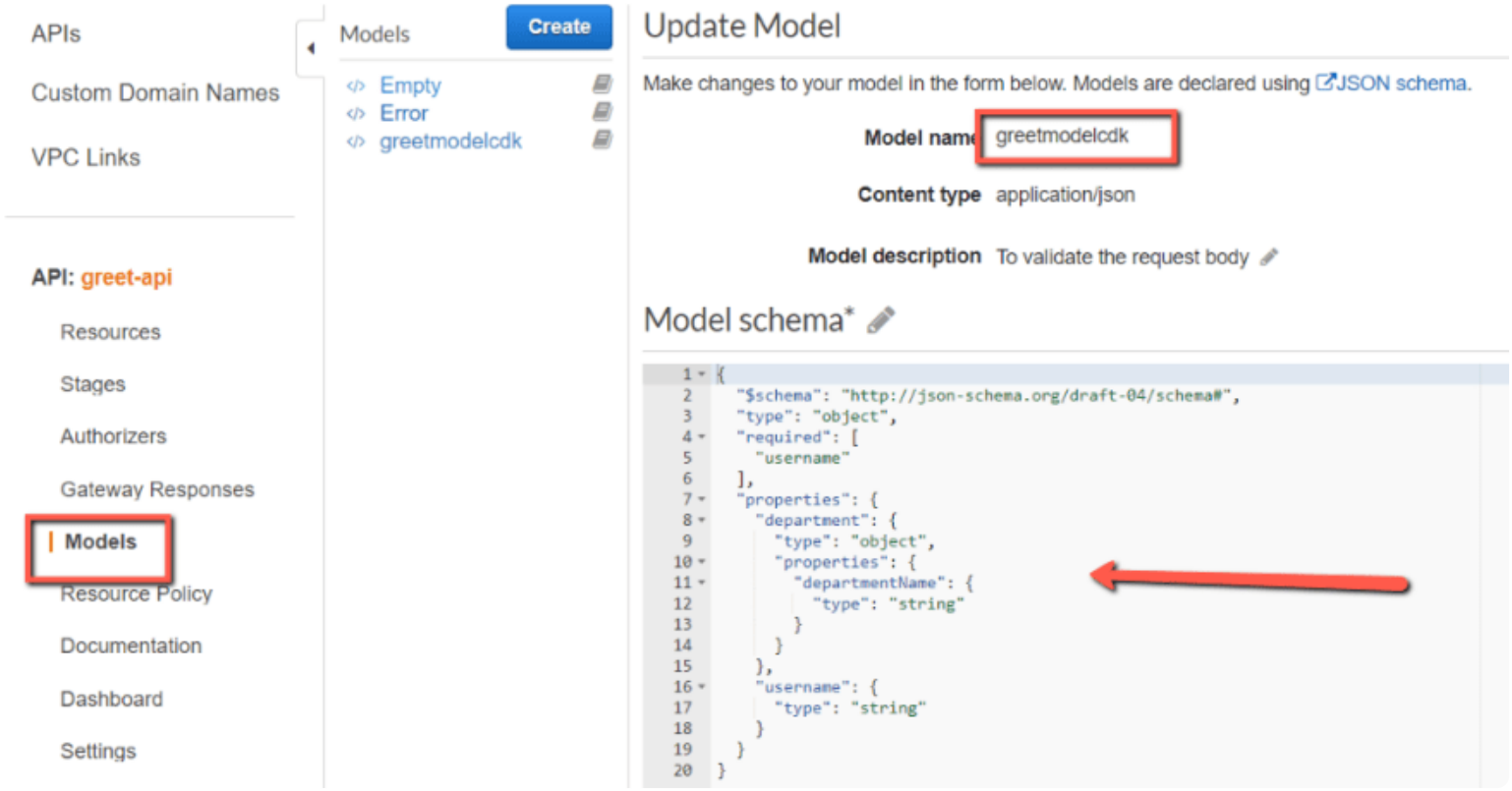
Now, let us edit **lib/cdk-apigateway-stack.js** to add the request validation to the API Gateway using CDK as shown below.

```
users.addMethod("POST", greetApiIntegration, {
      requestValidator: new apigateway.RequestValidator(
        this,
        "body-validator",
        {
          restApi: greetApi,
          requestValidatorName: "body-validator",
          validateRequestBody: true,
```

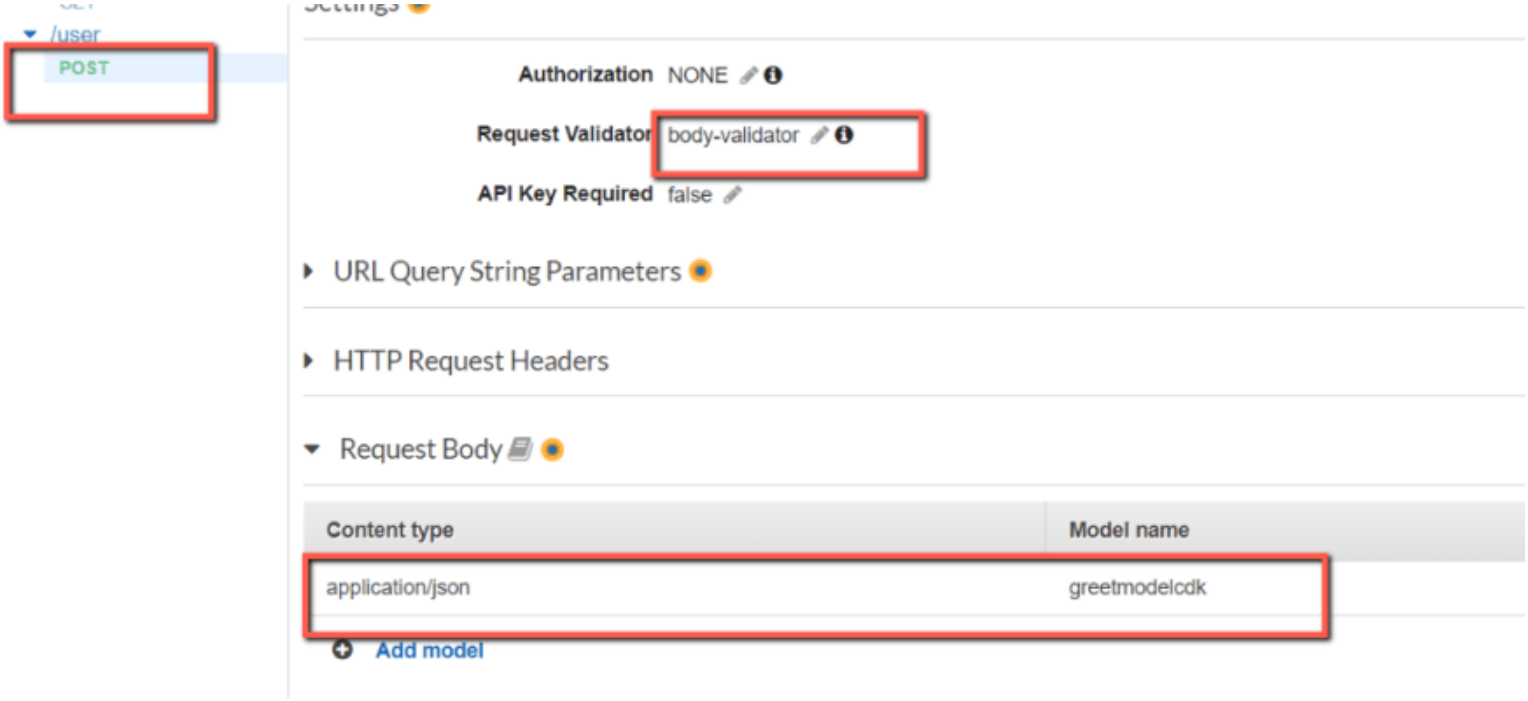
```
    ),
    requestModels: {
      "application/json": greetModel,
    },
  });
```

Let us deploy the code using `cdk deploy` and verify if resources are created in the AWS console.

The model schema is created as shown below.



Also, verify that the API resource has the model schema attached to it as shown below.



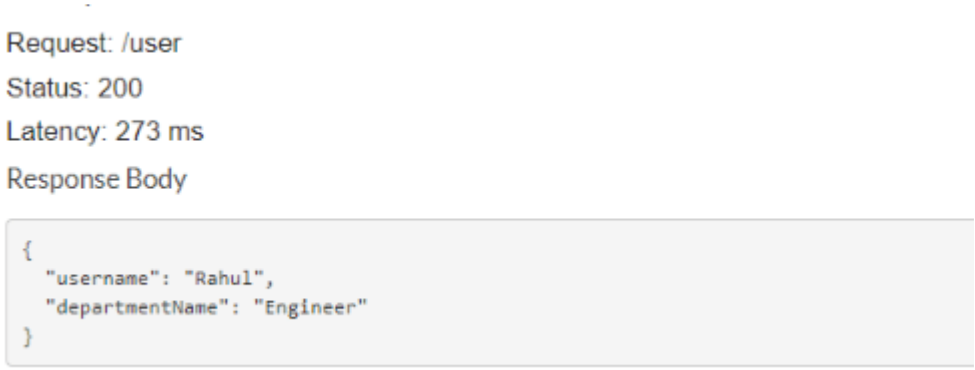
Let us test different scenarios for the request body validation.

- Valid Request:

- Valid Request:

```
{
  "username": "Rahul",
  "department": {
    "departmentName": "Engineer"
  }
}
```

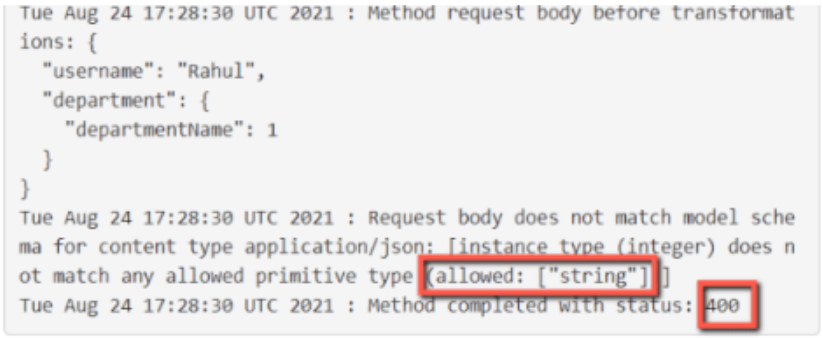
We get a valid response as seen below.



- Invalid type for departmentName:

```
{
  "username": "Rahul",
  "department": {
    "departmentName": 1
  }
}
```

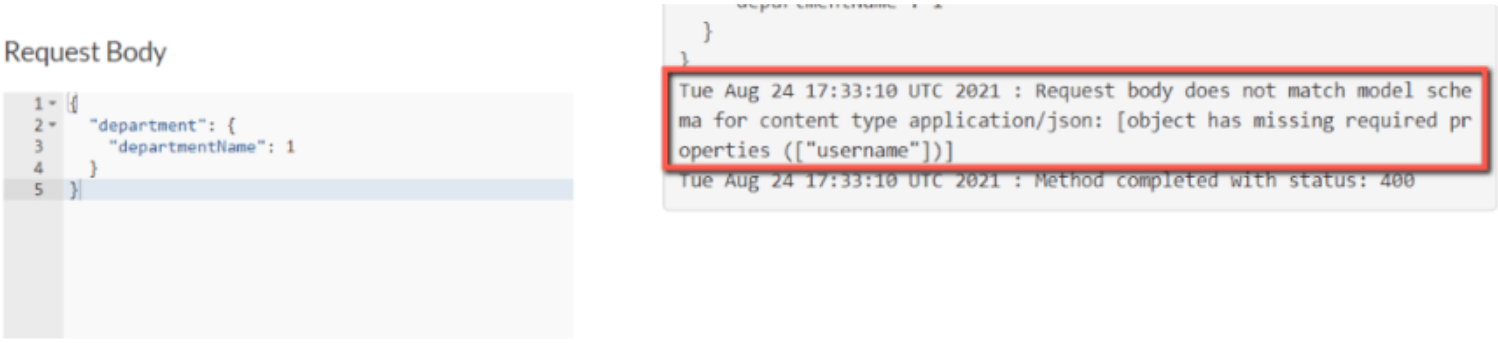
Here, we are passing the **Integer** value for **departmentName**. The schema we defined accepts a **string** for departmentName. So, we get 400 Bad requests with the message as shown below.



- Missing Required Field:

```
{
  "department": {
    "departmentName": 1
  }
}
```

Here, we are missing the field **username**. The schema we defined has a mandatory field **username**. So, we get 400 Bad requests with the message as shown below.



Conclusion

In this blog post, we saw what are request validators in AWS API Gateway. We learned about how to create a validator for querystring and the request body using cloud development kit (CDK).

GitHub Repo: https://github.com/rahulmlokurte/aws-usage/tree/main/aws-cdk/cdk-apigateway

Rahul Lokurte

Passionate programmer eager to learn and teach new technologies

LOCATION
India Bangalore

WORK
Lead Engineer at Dish Networks

JOINED
14 Oct 2018

More from Rahul Lokurte

Ice-cream Flavor Picker using AWS Event Bridge
#eventbridge #aws #stepfunction

AWS Lambda using CDK
#aws #cdk #lambda #serverless