Customising Responses

Here, you'll learn how to send a web form as a response for API Gateway!

WE'LL COVER THE FOLLOWING

- Serving static HTML from Lambda functions is a bad idea
- Speed up deployments

API Gateway expects responses in a specific format from Lambda Proxy integrations. The response needs to be a JSON object containing these fields:

- `statusCode` should be a number containing the numeric HTTP response code. The number `200` means OK.
- `body` should be a string representing the response contents.
- `headers` is an optional argument and can contain a map of HTTP response headers.

SAM assumes that you'll use API Gateway to create JSON APIs, so unless the response provides a content type, it assumes that you'll be sending back a serialised JSON object. That's why the sample function formats the response object using `JSON.stringify`:

```
response = {
    'statusCode': 200,
    'body': JSON.stringify({
      message: 'hello world',
    })
  }
```

Line 4 to Line 9 of hello-world/app.js

To show you how to customise responses, you'll make API Gateway send back a web page instead of a JSON object. For example, let's try creating a web form where users will be able to enter their name.

First, let's create a utility function that packages up a response containing some text with the correct headers. In this case, you just need to set the `Content-Type` header to `text/html`.

```
module.exports = function htmlResponse(body) {
  return {
    statusCode: 200,
    body: body,
    headers: {
      'Content-Type': 'text/html'
    }
  };
};
```

code/ch6/hello-world/html-response.js

> ## Serving static HTML from Lambda functions is a bad idea
>
> To keep things simple for now, you'll use Lambda functions to send HTML back to web browsers. This is perfectly fine for quick experiments and tutorials, but is far from optimal in a more general case. It's much cheaper and faster to host static files in S3 and combine that with Lambda functions for dynamic responses. You'll use this combination in Chapter 11.

The main Lambda function file (`app.js`) is now changed to return an HTML form easily:

```
const htmlResponse = require('./html-response');
const formHtml = `
  <html>
  <head>
    <meta charset="utf-8"/>
  </head>
  <body>
    <form method="POST">
      Please enter your name:
      <input type="text" name="name"/>
      <br/>
      <input type="submit" />
    </form>
  </body>
  </html>
`;
```

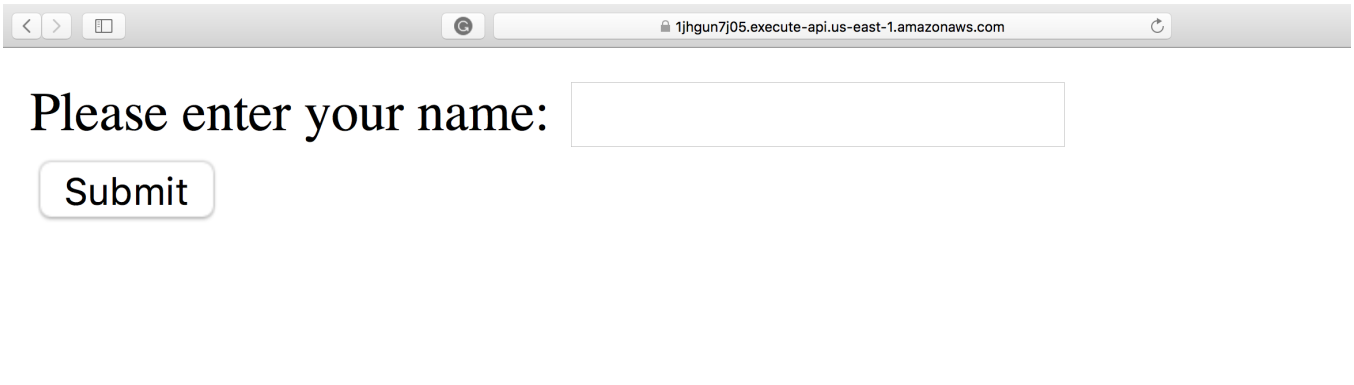
```
exports.lambdaHandler = async (event, context) => {
  console.log(JSON.stringify(event, null, 2));

  return htmlResponse(formHtml);
};
```

code/ch6/hello-world/app.js

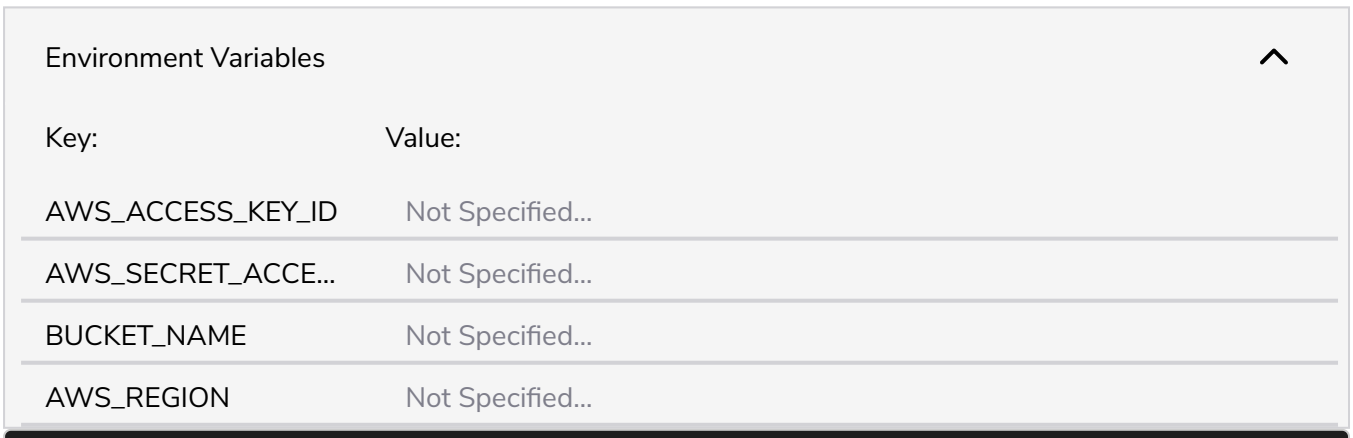
> ## Speed up deployments
>
> You added a deployment preference to the SAM template in the section *Gradual deployments* in Chapter 5. This is great for production-worthy applications, but slows down experimentation unnecessarily for the examples in this chapter. Remove the gradual deployment preference and the associated CloudWatch alert from the template before deploying, so you can test things faster.

Now you will build, package, and deploy the stack in this lesson. Open the API URL again in your browser (you can find it in the Stack outputs), and it should now show a web form, nicely rendered (see figure below).



Adding the `Content-Type` header makes it easy to send back HTML pages from Lambda.

Press the **Run** button to deploy the following application:



```
{
  "body": "{\"message\": \"hello world\"}",
  "resource": "/{proxy+}",

  "path": "/path/to/resource",
  "httpMethod": "POST",
  "isBase64Encoded": false,
  "queryStringParameters": {
    "foo": "bar"
  },
  "pathParameters": {
    "proxy": "/path/to/resource"
  },
  "stageVariables": {
    "baz": "qux"
  },
  "headers": {
    "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8",
    "Accept-Encoding": "gzip, deflate, sdch",
    "Accept-Language": "en-US,en;q=0.8",
    "Cache-Control": "max-age=0",
    "CloudFront-Forwarded-Proto": "https",
    "CloudFront-Is-Desktop-Viewer": "true",
    "CloudFront-Is-Mobile-Viewer": "false",
    "CloudFront-Is-SmartTV-Viewer": "false",
    "CloudFront-Is-Tablet-Viewer": "false",
    "CloudFront-Viewer-Country": "US",
    "Host": "1234567890.execute-api.us-east-1.amazonaws.com",
    "Upgrade-Insecure-Requests": "1",
    "User-Agent": "Custom User Agent String",
    "Via": "1.1 08f323deadbeefa7af34d5feb414ce27.cloudfront.net (CloudFront)",
    "X-Amz-Cf-Id": "cDehVQoZnx43VYQb9j2-nvCh-9z396Uhbp027Y2JvkCPNLmGJHqlaA==",
    "X-Forwarded-For": "127.0.0.1, 127.0.0.2",
    "X-Forwarded-Port": "443",
    "X-Forwarded-Proto": "https"
  },
  "requestContext": {
    "accountId": "123456789012",
    "resourceId": "123456",
    "stage": "prod",
    "requestId": "c6af9ac6-7b61-11e6-9a41-93e8deadbeef",
    "requestTime": "09/Apr/2015:12:34:56 +0000",
    "requestTimeEpoch": 1428582896000,
    "identity": {
      "cognitoIdentityPoolId": null,
      "accountId": null,
      "cognitoIdentityId": null,
      "caller": null,
      "accessKey": null,
      "sourceIp": "127.0.0.1",
      "cognitoAuthenticationType": null,
      "cognitoAuthenticationProvider": null,
      "userArn": null,
      "userAgent": "Custom User Agent String",
      "user": null
    },
    "path": "/prod/path/to/resource",
    "resourcePath": "/{proxy+}",
    "httpMethod": "POST",
    "apiId": "1234567890",
    "protocol": "HTTP/1.1"
  }
}
```

Try submitting the form and you'll get a `403 Not Authorized` response complaining about a missing authentication token. That's because there's no resource in the API Gateway to handle the form submission yet.

Next, you will learn how to troubleshoot gateway integrations.