Exercise 4: OAS Continued

Let's add some security, errors, content type, and operation IDs to your example.

You'll probably want to refer to the OAS file I created for the photo album example and used in the lesson you just watched. You can find it at http://sdkbridge.com/swagger/openApiCont.yaml.

Add security

Let's start by adding basic authentication as a security method to one of the operations. In a real example, you would probably add the same security to all operations. The only exception is OAuth, where different operations might require different scopes.

1. Open the Swagger editor at: http://editor.swagger.io/ You should see the YAML from the last exercise, but if not, then you can import it from the file you saved.
2. You can put security anywhere in the operation, but a logical place would be right above **responses**. It should be the same indentation level as **responses**, and it's very simple:

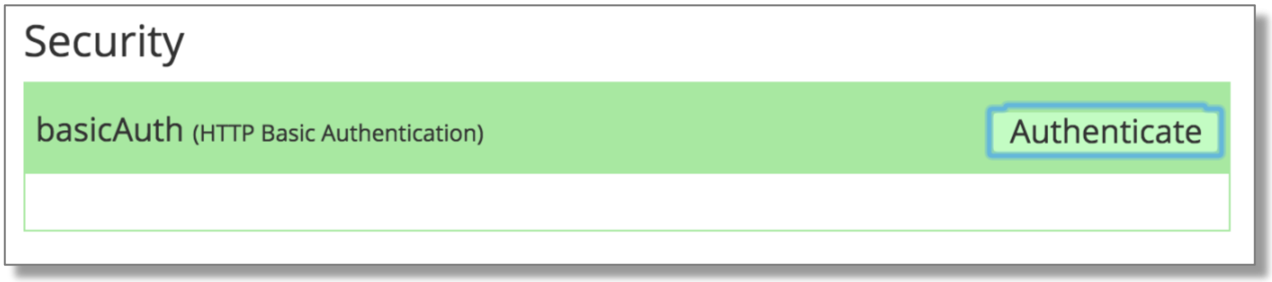
```
security:
  - basicAuth: [ ]
```

I used **basicAuth** as a name for the security type, but you could call it anything, as long as you use the same name in the **securityDefinitions**.

Next, add a **securityDefinitions** key at the bottom of the file. It is at the top level, so has no indentation. Add an item called **basicAuth**, which has a **type** key with value **basic**. Again, it doesn't have to be "basicAuth", but it needs to match what you put in the **security** list.

```
securityDefinitions:
  basicAuth:
    type: basic
```

Look at the right side of the editor, and at the top you'll see a security section. It has the name and it explains that it's using HTTP Basic Authentication. If you click on **Authenticate**, then it will bring up a dialog to ask for username and password. If this were a real API, you would need to do this before clicking on **Try this operation** so that it would have the username and password to send with its requests.



Add an error

Let’s say our API returned JSON in this format when an error occurs:

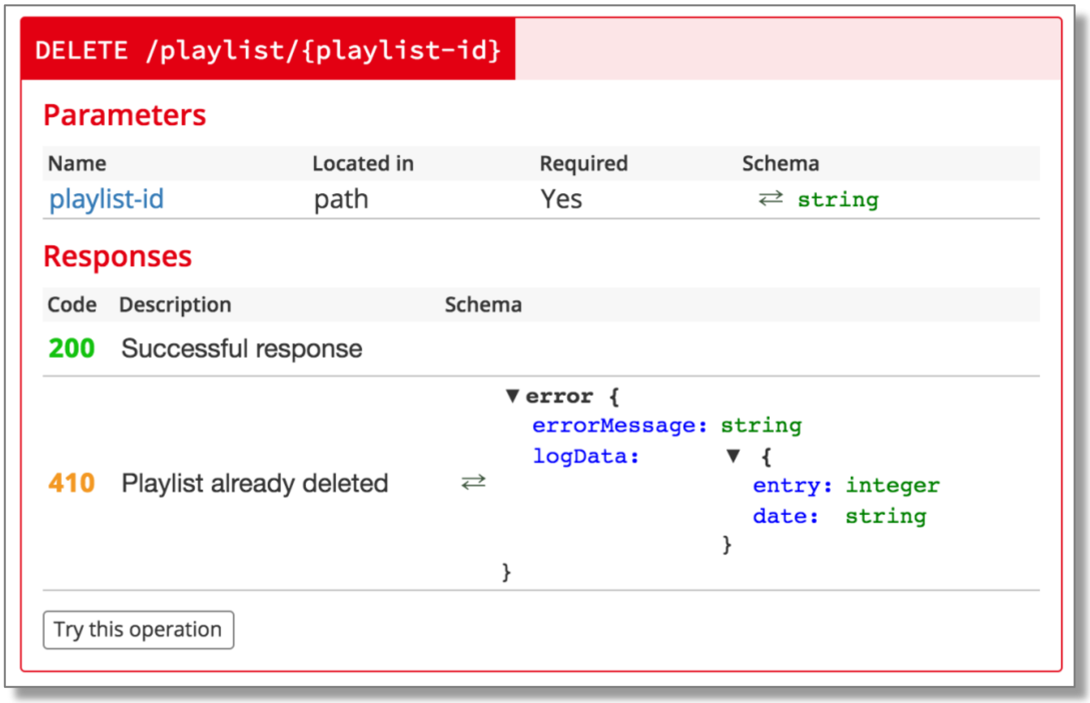
```
{
  "errorMessage": "Playlist with that name already exists",
  "logData": {
    "entry": 3548,
    "date": "2017-10-09 09:40:34"
  }
}
```

In addition to the message, there is data for the server log. A developer could give this information to technical support, and they could find the exact entry in the server log to help debug what is going on.

In the **responses** section, create a new key called **error** and give it properties that reflect the JSON above. Hint: in addition to the **errorMessage** key of type **string**, you will need a **logData** key of type **object**, and then another properties section for that.

Next, go to the **delete** operation and add a response for 410 (Gone). This will handle the case where someone tries to delete a playlist that has already been deleted. Have it refer to the error definition. If the editor is showing you errors, just refresh the page and they should disappear (if you’ve written everything correctly).

Now on the right, you should see your 410 response:


DELETE /playlist/{playlist-id}

Parameters

Name	Located in	Required	Schema
playlist-id	path	Yes	⇄ string

Responses

Code	Description	Schema
200	Successful response	
410	Playlist already deleted	⇄ ▼error { errorMessage: string logData: ▼ { entry: integer date: string } }

```
▼error {
  errorMessage: string
  logData:       ▼ {
                   entry: integer
                   date:  string
                 }
}
```

Add content type

For the most part, this API will send and receive data in JSON format, so after the **schemes** section, add keys for **consumes** and **produces** with values of **application/json**.

```
consumes:
  - application/json
produces:
  - application/json
```

Let's imagine that our API can generate a PNG image that it generates for the playlist, using album art from the songs in the playlist. It's going to be a new path:

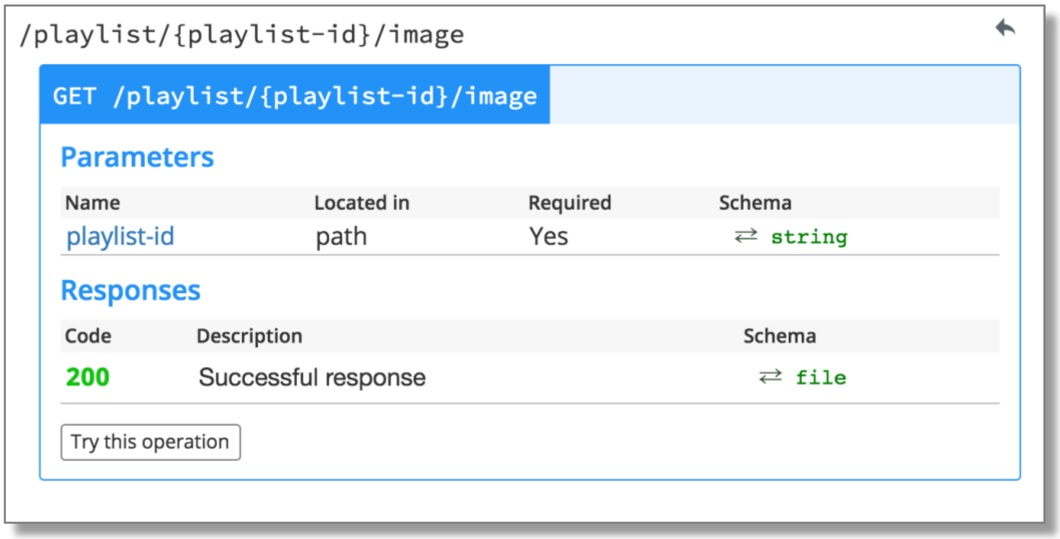
```
GET /playlists/{playlist-id}/image
Response: Image in PNG format
```

1. Add the new path
2. Add a **get** method
3. Copy the parameters from the previous **GET /playlists/{playlist-id}** operation, since this operation has the same parameters (just the path parameter playlist-id).
4. Add a **responses** section that handles the 200 code. Since it's returning a bunch of digital data rather than JSON, the schema type is called **file**.

```
responses:
   200:
    description: Successful response
    schema:
      type: file
```

5. Lastly, add a **produces** section at the same indent level as **responses**. The list should have one value, which is the MIME-type for PNG.
```
produces:
  - image/png
```

When you've done it correctly, the documentation on the right side for the new path should look like this, with "file" listed for the schema:



/playlist/{playlist-id}/image

GET /playlist/{playlist-id}/image

Parameters

Name	Located in	Required	Schema
playlist-id	path	Yes	⇄ string

Responses

Code	Description	Schema
200	Successful response	⇄ file

Add operation ID

Let's add an operation ID to the new image operation, just so you can see what that's like. Under the **get:** add a new line and indent. Then put:

```
operationId: getImage
```

You won't see any changes on the right, but if you've done it correctly, there will be no errors in the editor.

Save

It's a good idea to save your YAML file after each exercise. From the **File** menu, choose **Download YAML**. Save this somewhere where you can easily get to it for the next exercise.

Solution

If you get stuck, you can look at my version of the OAS file: http://sdkbridge.com/swagger/Exercise4Answer.yaml.