# Exercise 7: Final Project

Give yourself an hour or more to create an OAS file from scratch. This time, I will not hold your hand very much. I'll give you bunch of sample requests and responses and related information you might get from a developer's team. This is not a trivial exercise. I suggest having the previous exercise OAS file handy that you can refer to and search this document to see what the values are. You may need to refer back to previous lessons to figure some of this out. Feel free to download the PowerPoint presentations, if that's helpful.

Imagine you are working for a company called MemeMeister. They have an API where you give it an image and some text, and it superimposes the image on the text and returns a new image. This is very useful in creating memes.


There are four APIs: create a meme, get a list of memes and captions, get a meme by ID, and delete a meme. For security, you are using Facebook's OAuth URL, so that you can log in with your Facebook credentials. (In reality, using Facebook to authenticate is more complicated than that.)

Include **description** keys wherever possible so that the documentation is complete.

## General Information

The version is 0.1.0. For the **title**, use "Meme Meister". For description, say "API to create memes."

In general, the API consumes and produces JSON.

The API only has responses if successful (200). Those are the only responses you need to document.

## Security

Security is using OAuth, so you will need to create a security definition. You may want to review how to do OAuth security.

Call the security definition **oauthFacebook**. For the **authorizationUrl**, put **https://dev.facebook.com/oauth/authenticate** (not a real Facebook URL!). The **flow** is **implicit** (meaning that users go to a separate site for authentication). There are two scopes – one for reading memes and one for writing memes:

```
scopes:
  write:memes: Modify memes in your account
  read:memes: Read memes in your account
```

Be sure to write a description that mentions Facebook.

## Creating a new meme

To create a new meme, you make a POST request with a query parameter that contains the captions as query parameters. The body of the post will contain the image. The software will combine these to create a new image, which it returns in the response body.

### Request:

Sample request:

```
POST
https://dev.mememeister.com/v1/meme?topcaption=excellent&bottomcaption=epic%20fail
```

The caption query parameters are both required. In your description for these query parameters, be sure to mention that the string need to be URL-encoded. (This means that spaces should be written as %20, etc.)

The POST body is of type **file** and is an image. The MIME type it consumes are:

- image/jpeg
- image/gif
- image/png

**Hint:** To have the body be a file, use these properties for the parameters:

```
in: formData
required: true
type: file
```

Note that you use **formData** instead of **body** for **in**. In addition to these properties, you should also have a **name** and **description** property. The value of the **name** property isn't important.

The security section should use your security definition and have both scopes (write and read).

### Response:

The response produces MIME type image/jpeg, which means the schema is of type **file**. (You don't need a schema definition for this. See how you did it on the last exercise for the **get** that ended in /image.)

## Listing all memes

This GET request will return a list of meme IDs and captions. No images are returned. If you want to get an image, then you use the ID to make a call to the other GET request.

### Request:

```
GET https://dev.mememeister.com/v1/meme?q=grass
```

The **q** parameter is a search term and is optional. If used, it will filter the responses based on the search. Mention in the description that it's URL-encoded.

Security is OAuth with the read scope only.

### Response:

The 200 code response returns JSON like this:

```
[
  {
    "id": 115,
    "topCaption": "THIS GRASS",
    "bottomCaption": "I'D LIKE TO BURY YOU UNDER IT"
  },
  {
    "id": 543,
    "topCaption": "THE GRASS IS NEVER GREENER",
    "bottomCaption": "ON MARS"
  }
]
```

Create a **definitions** section with a schema for this. Note that what's returned is a JSON object, not a JSON array. This means that when you reference the schema, you need to mention that it's of type array, like this:

```
schema:
  type: array
  items:
    $ref: '#/definitions/memeInfo'
```

## Getting a meme

This GET request will return a JPEG given a meme ID.

### Request:

```
GET https://dev.mememeister.com/v1/meme/543
```

In this example, 543 is the meme ID, which is a required path parameter.

Security is OAuth with the read scope only.

The response is identical to the POST response: it returns a JPEG file.

## Deleting a meme

This DELETE request will delete a meme with the specified ID.

### Request:

```
DELETE https://dev.mememeister.com/v1/meme/543
```

In this example, 543 is the meme ID, which is a required path parameter.

If successful, returns a 204 response, meaning no content.

## Final words

This is a tough exercise. Don't let yourself give up easily! If you get totally confused, you can check my version at http://sdkbridge.com/swagger/Exercise7Answer.yaml.