

REST Workshop

Lab 2

1	LAB SETUP	1
2	INTERACTING WITH THE SWAGGER PETSTORE	2
2.1	ADDING PETS WITH POST /PET	3
2.2	CHECKING PETS WITH GET /PET/{PETID}	6
2.3	FILTERING PETS WITH GET /PET/FINDBYSTATUS	7
2.4	MODIFYING PARTIAL PET DETAILS WITH POST /PET/{PETID}	7
2.5	MODIFYING COMPLETE PET DETAILS WITH PUT /PET/{PETID}	8
3	DOCUMENTING AN API USING OPENAPI 3.0 IN SWAGGER EDITOR	9
3.1	INFO METADATA	11
3.2	SERVER METADATA	11
3.3	TAGS METADATA	12
3.4	PATHS AND OPERATIONS	13
3.5	DATA MODELS AND SCHEMAS	14
3.6	PARAMETERS	17
3.7	REQUEST BODY	19
3.8	RESPONSE BODY	20
4	WORKING WITH AN OPENAPI SPEC IN SWAGGERHUB	22
4.1	TESTING VIA THE MOCK API SERVER	22
4.2	INTEGRATION WITH OTHER SERVICES	23
4.3	SHARING AND COLLABORATION	24
4.4	GENERATING CLIENT-SIDE AND SERVER-SIDE CODE STUBS	24
4.5	EXPORTING THE SPEC	25
4.6	OTHER SWAGGERHUB FEATURES	25
5	FURTHER EXERCISES	25

1 Lab setup

Make sure you have the following items installed or available:

- A free account at SwaggerHub
- A free account at Postman and installed the Postman app
- A free account on RapidAPI and a valid credit card to register during use
- A suitable text editor (Notepad ++)

2 Interacting with the Swagger PetStore

Swagger provides a reasonably realistic mock for a REST API which includes the important ability of performing and persisting POST, PUT and DELETE operations to the local state. So far, you will have seen the majority of public REST APIs that are open for interaction do not accommodate this feature of persisting to local state due to the risk of meaningless or nonsensical modification of app state.

The mock PetStore can be located at:

<https://petstore.swagger.io>

The view that you see here is a Swagger UI-visualization of a REST API generated from an OpenAPI specification.

<https://swagger.io/tools/swagger-ui/>

The UI supports visualization and interaction with the API's resources without having any of the implementation logic in place. This helps facilitate the completion of the design process before the actual coding implementation is performed.

If you scroll down to the bottom of this page, you will see the data models (or schema) for the various objects that the REST API will manipulate.

- The APIResponse provides structure for error information in the event of a client-side error
- The Category and Tag provides structure for two ways of classification for a Pet object
- The Pet object identifies a pet and contains an embedded Category and Tag object
- The Order object contains information for a single order and is linked to the Pet object that the order is for
- The User object contains information for a single user and is not linked to either Pet or Order object

Some key points to notice about the organization of the API endpoints:

- Endpoints are neatly organized into different categories based on the first portion of the path after the domain name (e.g:- /pet /store /user).
- Some endpoints contain further path portions after the first portion (e.g:- /pet/{petId} /store/order user/{username}) which reflects a hierarchical structure in the organization of resources. The { } indicates parameters whose values are supplied by the REST client during invocation of that respective endpoint.
- Many endpoints have multiple REST HTTP methods that apply to them (e.g:- /pet/{petid} supports a GET, POST and DELETE). This is quite common with REST APIs in that you want to be able to perform a set of operations to a single identified resource given by the endpoint URL.
- You may see a particular method and endpoint greyed and crossed out (e.g. GET /pet/findByTags). This indicates that this method and endpoint are deprecated (i.e. they were available in an older version of the API but are no longer available in the current version).

The base URL is given as: `https://petstore.swagger.io/v2`

The complete URL for an API endpoint is obtained by concatenating the path portion of the API endpoint to the base URL, for e.g.:

<https://petstore.swagger.io/v2/pet/3>

We will use Postman to make requests to this REST API.

Create a new collection called: Swagger Petstore Requests

Create a new environment called: Petstore Environment

Add this to it:

Variable: BaseURL

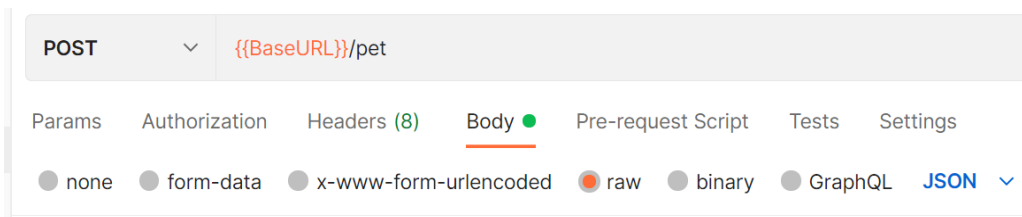
Initial value: `https://petstore.swagger.io/v2`

You will use this variable to simplify constructing your requests to this REST API.

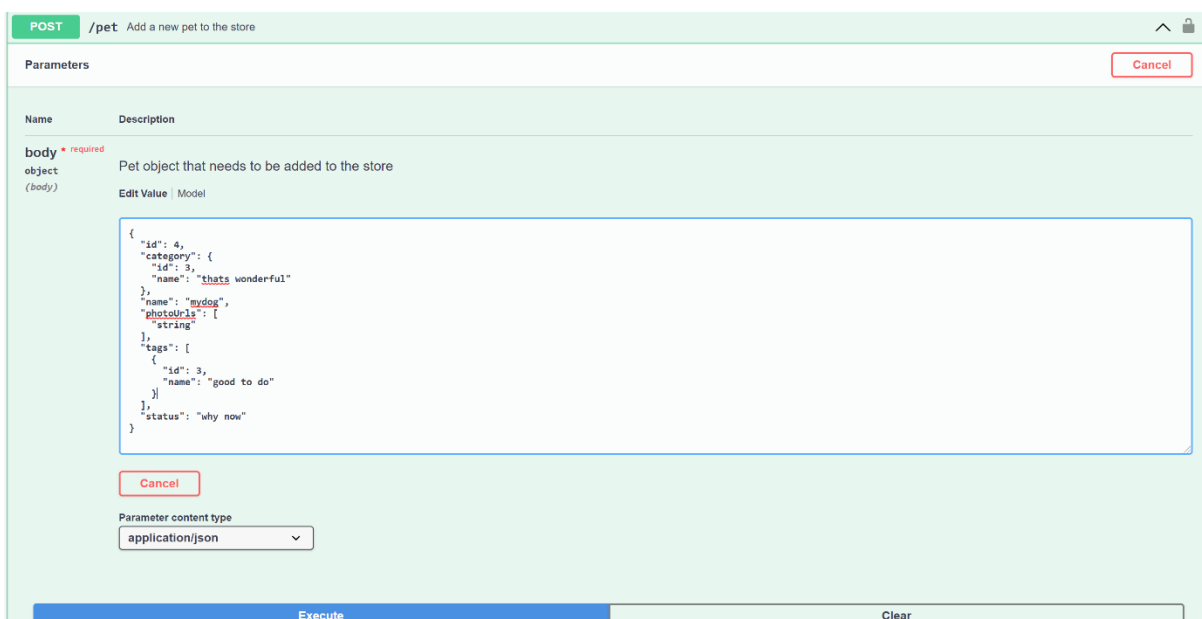
2.1 Adding pets with POST /pet

WE are going to add 6 new pets to the store using the `POST /pet` endpoint.

Set up Postman for these requests as shown below. The contents of each of these 6 pet objects should be entered in the Body field with JSON selected in raw format, before being sent one at a time. The response for a successful creation should be a 200 OK with the body of the new record returned.



You can also perform these POST operations directly from within the UI itself by clicking on the Try it Out button, then entering the required information to send (if any) and then clicking Execute.



The server response will be shown at the bottom after Execute has been clicked on.

Request URL
https://petstore.swagger.io/v2/pet

Server response

Code	Details
200	<p>Response body</p> <pre>{ "id": 4, "category": { "id": 3, "name": "thats wonderful" }, "name": "mydog", "photoUrls": ["string"], "tags": [{ "id": 3, "name": "good to do" }], "status": "why now" }</pre> <p>Response headers</p> <pre>access-control-allow-headers: Content-Type,api_key,Authorization access-control-allow-methods: GET,POST,DELETE,PUT access-control-allow-origin: * content-type: application/json date: Sun,01 Aug 2021 08:12:31 GMT server: Jetty(9.2.9.v20150224)</pre>

Responses

Code	Description
405	Invalid input

If you are working on lab with multiple participants at the same time in workshop, you will all need to use different values for the "id" property in the body options below to avoid conflict, as all your POST data will all be persisted in the same underlying store. I will suggest a simple scheme on how to do this effectively.

IMPORTANT: Note that it may be possible that not all of your POST requests involving these 5 new pets will be persisted so that you can subsequently retrieve them later with different GET requests. This will depend on the allocation that the Swagger Petstore has for dynamic operations for your particular machine/IP address and the frequency of requests from there.

Try performing these operations from inside Postman and inside the UI as well.

Pet #1

```
{
  "id": 100,
  "category": {
    "id": 1,
    "name": "dog"
  },
  "name": "fido",
  "photoUrls": [
    "someURL"
  ],
  "tags": [
    {
      "id": 1,
      "name": "cute"
    }
  ],
  "status": "available"
}
```

Pet #2

```
{
  "id": 200,
  "category": {
    "id": 2,
    "name": "cat"
  },
  "name": "molly",
  "photoUrls": [
    "someURL"
  ],
  "tags": [
    {
      "id": 1,
      "name": "cute"
    }
  ],
  "status": "pending"
}
```

Pet #3

```
{
  "id": 300,
  "category": {
    "id": 2,
    "name": "cat"
  },
  "name": "Toby",
  "photoUrls": [
    "someURL"
  ],
  "tags": [
    {
      "id": 2,
      "name": "ugly"
    }
  ],
  "status": "sold"
}
```

Pet #4

```
{
  "id": 400,
  "category": {
    "id": 1,
    "name": "dog"
  },
  "name": "Ruby",
  "photoUrls": [
    "someURL"
  ],
  "tags": [
    {
      "id": 1,
      "name": "cute"
    }
  ],
  "status": "available"
}
```

Pet #5

```
{
  "id": 500,
  "category": {
    "id": 2,
    "name": "cat"
  },
  "name": "Oscar",
  "photoUrls": [
    "someURL"
  ],
  "tags": [
    {
      "id": 1,
      "name": "pending"
    }
  ],
  "status": "popular"
}
```

2.2 Checking pets with GET /pet/{petId}

To check that all 5 objects have been stored successfully, you can issue a GET request to these URLs ending with the `petID`, e.g.

```

{{BaseURL}}/pet/100
...
{{BaseURL}}/pet/500

```

Again, note that it may be possible that not all of your POST requests involving these 5 new pets will be persisted. If you are not able to retrieve a particular object based on its `petID`, issue another POST for that same `petID` to see whether you can persist its content.

As before, you can perform this GET request operation from inside the Swagger UI or from Postman.

2.3 Filtering pets with GET /pet/findByStatus

You can filter the pets returned by status by issuing a GET request to:

```

{{BaseURL}}/pet/findByStatus

```

with `status` used as a query parameter whose value can either be: `available`, `pending` or `sold`.

Query Params	
KEY	VALUE
<input checked="" type="checkbox"/> status	available

As before, you can perform this GET request operation from inside the Swagger UI or from Postman.

2.4 Modifying partial pet details with POST /pet/{petID}

You can modify some of the details for an existing pet by issuing a POST request to:

```

{{BaseURL}}/pet/{petID}

```

The only values you can change are the name and the status (can either be: `available`, `pending` or `sold`). The format for this should be: `x-www-form-urlencoded`

An example of a setup:

KEY	VALUE
<input checked="" type="checkbox"/> name	thor
<input checked="" type="checkbox"/> status	pending

As before, you can perform this POST request operation from inside the Swagger UI or from Postman.

You can then verify that this modification has taken place by issuing a GET request to the same `petID`

```
{{BaseURL}}/pet/100
```

Note: you may have to perform the POST request several times before it takes effect: again this depends on the traffic limit restriction that Swagger is imposing on your machine/IP address.

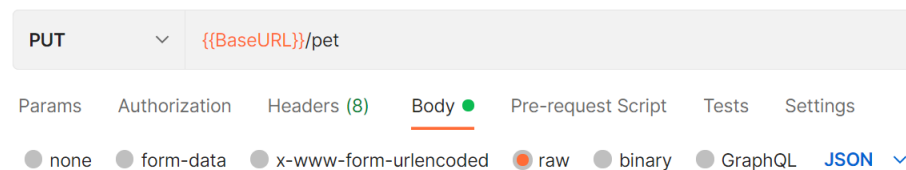
Technically speaking, since this request is also modifying a part of the resource, it should have been done with a PATCH instead of a POST.

2.5 Modifying complete pet details with PUT /pet/{petID}

Finally, we can choose to modify all of the details for an existing pet by issuing a PUT request to:

```
{{BaseURL}}/pet
```

and sending a JSON body with the new details for an existing pet identified by its `petID`. For e.g. assuming that `petID 400` already exists, we can modify its details with:



```
{
  "id": 400,
  "category": {
    "id": 3,
    "name": "bird"
  },
  "name": "Lola",
  "photoUrls": [
    "someURL"
  ],
  "tags": [
    {
      "id": 1,
      "name": "cute"
    }
  ],
  "status": "sold"
}
```

As before, you can perform this PUT request operation from inside the Swagger UI or from Postman.

As a further exercise of your own, come up with some sample data of your own similar to the example shown above to test the `/user` and `/store` REST operations available.

3 Documenting an API using OpenAPI 3.0 in Swagger Editor

Documenting our REST API before we start implementing it in code confers two important advantages:

- a) It helps us think through carefully the design choices we need to make before implementing (at which point it is harder to reverse a particular decision).
- b) It helps provide clear documentation to clients (internal and external) who will interact with the REST API on a regular basis. REST APIs which are clearly and accurately documented with regards to what they can achieve are far more likely to gain traction and adoption amongst the clients who consume them.
- c) A mock API server can be auto-generated from the spec that allows developers who are working on client-side logic to consume the API to work in parallel with the developers who are working on the business logic for the REST API. The REST API specs thus becomes a fixed contract between the two groups of developers to allow coding implementation to proceed in parallel.

A simple tutorial can be found for this specs can be found at:

<https://support.smartbear.com/swaggerhub/docs/tutorials/openapi-3-tutorial.html>

More details on the key topics can be found on the left pane from this main link:

<https://swagger.io/docs/specification/about/>

The complete detailed specification is at:

<https://swagger.io/specification/>

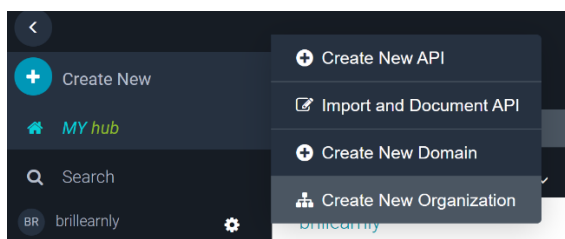
We can use the Swagger editor to document our REST API based on the OpenAPI 3.0 specs:

<https://swagger.io/tools/swagger-editor/>

Login to your SwaggerHub account here.

Create a new organization to house your APIs under. Give it a unique name of your choice.

You can use any valid email address to associate with it.



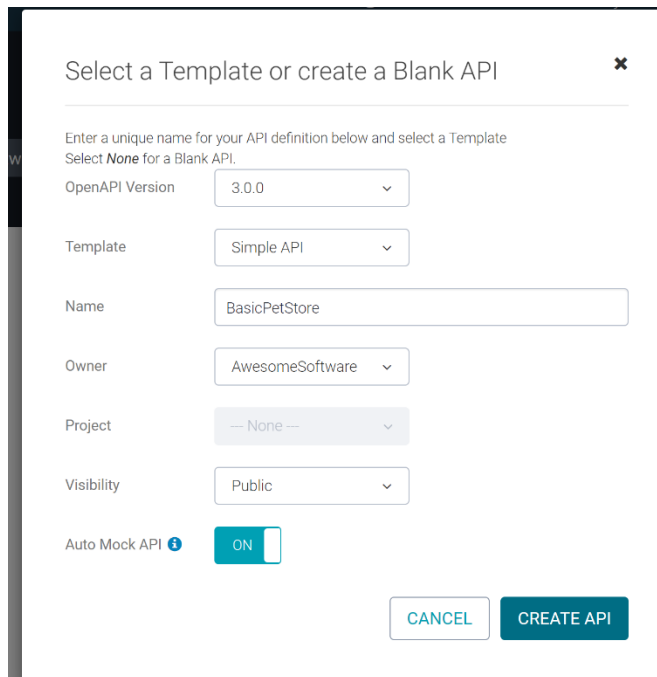
<https://support.smartbear.com/swaggerhub/docs/ui/overview.html>

<https://support.smartbear.com/swaggerhub/docs/apis/creating-api.html>

In the newly created organization, select Create API.

In the dialog box, type in for the Name: `BasicPetStore`

For the rest of the fields, set them as follows:



You will be now be transitioned to the SwaggerHub Code Editor view where you can now construct the description for your API using YAML.

<https://support.smartbear.com/swaggerhub/docs/ui/editor.html>

You start off with a template which you will work from in order to create your API description. On the left hand side, you have a set of icons which you can use to arrange the view in the main window.



You can choose to hide specific views, as well as switch between Visual and Code editor views. The Code editor shows the actual YAML description of the API, and will be the view that we work with most of the time.

The Visual Editor provides a visualization of the YAML description in a more intuitive form. You can use this to enter the relevant values for the specific YAML key-value pairs.

<https://support.smartbear.com/swaggerhub/docs/ui/visual-editor.html>

In this session, we will attempt to build an API Description from scratch for the Swagger Petstore example that we were working with in the previous lab. We will exclude some of the more advanced components of this specification (such as authentication, links and callbacks) for this lab session.

Navigate to this API and keep it open in a tab for reference during this exercise:

<https://petstore.swagger.io/>

To verify and format the YAML snippets that you are typing in for each section of this lab, you can use either of these online sites:

<https://jsonformatter.org/yaml-validator>

<https://codebeautify.org/yaml-validator>

The basic structure for an OpenAPI 3.0 definition is summarized at:

<https://swagger.io/docs/specification/basic-structure/>

3.1 Info metadata

Every API definition must include the version of the OpenAPI Specification that this definition is based on: the latest version is 3.0

The info section contains API information: title, description (optional), version

<https://swagger.io/docs/specification/api-general-info/>

Replace lines 1 - 14 with this YAML snippet for the `info:` portion of our API description

```
openapi: 3.0.0
info:
  description: |
    This is a sample Petstore server. You can find
    out more about Swagger at
    [http://swagger.io](http://swagger.io) or on
    [irc.freenode.net, #swagger](http://swagger.io/irc/).
  version: "1.0.0"
  title: Swagger Petstore
  termsOfService: 'http://swagger.io/terms/'
  contact:
    email: apiteam@swagger.io
  license:
    name: Apache 2.0
    url: 'http://www.apache.org/licenses/LICENSE-2.0.html'
```

3.2 Server metadata

<https://swagger.io/docs/specification/api-host-and-base-path/>

Insert in the `servers:` metadata description immediately after the `info:` metadata description.

```
servers:
  - description: The production server for the PetStore
    url: https://virtserver.swaggerhub.com/Org-
name/BasicPetStore/1.0.0
  - url: 'https://petstore.swagger.io/v2'
```

Replace *Org-name* with the actual organization name that you selected earlier.

Here there are two server URLs specified:

- The first is an API mocking server that allows you to test it out as well as allow developers who are working on client-side code to start on their work prior to implementation of the actual server logic.

<https://support.smartbear.com/swaggerhub/docs/integrations/api-auto-mocking.html>

- The second is a live or production server hosting an actual implementation of the OpenAPI specification with proper business logic implemented.

You can choose to test again either URL (by selecting from the drop down list) once your API specification is complete and valid - this is explained in more detail later.

Before continuing, we will delete the remaining YAML descriptions below the `servers` metadata. You will obtain an error message because there is a minimum requirement to have a `paths` property, which we will be adding in later.

Errors (1)		
Type ▾	Line	Description
✖	0	should have required property 'paths' missingProperty: paths

3.3 Tags metadata

<https://swagger.io/docs/specification/grouping-operations-with-tags/>

We will add tag metadata for the 3 main categories in the Swagger Petstore: `pet`, `store` and `user`

```
tags:
  - name: pet
    description: Everything about your Pets
  - name: store
    description: Access to Petstore orders
  - name: user
    description: Operations about user
```

3.4 Paths and operations

<https://swagger.io/docs/specification/paths-and-operations/>

We will first add in the top-level paths for the API end points for the `/pet` endpoint after the tag metadata:

```
paths:
  /pet:
  /pet/findByStatus:
  /pet/findByTags:
  '/pet/{petId}':
  '/pet/{petId}/uploadImage':
```

This will result in multiple errors, since the API end point definitions need to have additional properties defined with them. We will start completing these definitions after this.

Last Saved: 7:28:13 am - Aug 2, 2021 5 ✕		
<input checked="" type="checkbox"/> Errors (5)		
Type	Line	Description
✕	28	should be object
✕	29	should be object

Next we expand this definition to add in the corresponding HTTP methods for each of the listed end points:

```
paths:
  /pet:
    post:
    put:
  /pet/findByStatus:
    get:
  '/pet/{petId}':
    get:
    post:
    delete:
  '/pet/{petId}/uploadImage':
    post:
```

Notice now that we have errors flagged for the methods because they also require a complete definition as an object, in particular a response. Notice that a request body is not compulsory.

Finally, we add a tag, summary and an operation ID to complete the definition for each of these methods.

```
paths:
  /pet:
    post:
      tags:
        - pet
      summary: Add a new pet to the store
      operationId: addPet
    put:
      tags:
        - pet
      summary: Update an existing pet
      operationId: updatePet

  /pet/findByStatus:
    get:
      tags:
        - pet
      summary: Finds Pets by status
      description: Multiple status values can be provided with comma
separated strings
      operationId: findPetsByStatus
  '/pet/{petId}':
    get:
      tags:
        - pet
      summary: Find pet by ID
      description: Returns a single pet
      operationId: getPetById
    post:
      tags:
        - pet
      summary: Updates a pet in the store with form data
      operationId: updatePetWithForm
    delete:
      tags:
        - pet
      summary: Deletes a pet
      operationId: deletePet
  '/pet/{petId}/uploadImage':
    post:
      tags:
        - pet
      summary: uploads an image
      operationId: uploadFile
```

3.5 Data models and schemas

<https://swagger.io/docs/specification/components/>

<https://swagger.io/docs/specification/data-models/>

Before we complete the definitions for the methods, we can first define the data model (or schema) for the request and response bodies we expect for the various API endpoints and methods in the `components` section. This allows us to reference the schema from the other sections instead of repeating the same definitions at multiple places in the document specification.

At the bottom of the YAML description, provide the top-level components and schemas properties and then define the specific schema for `ApiResponse`

```
components:
  schemas:
    ApiResponse:
      type: object
      properties:
        code:
          type: integer
          format: int32
        type:
          type: string
        message:
          type: string
```

Below this, we can define the schemas for `Category` and `Tag`. We include the `xml` property for both `Category` and `Tag` so that they can be used as schema for JSON (default) and XML structuring of contents in either request or response bodies.

```
Category:
  type: object
  properties:
    id:
      type: integer
      format: int64
    name:
      type: string
  xml:
    name: Category
Tag:
  type: object
  properties:
    id:
      type: integer
      format: int64
    name:
      type: string
  xml:
    name: Tag
```

Finally, we define the schema for `Pet`, which uses the `Category` and `Tag` schemas

```
Pet:
  type: object
  required:
    - name
    - photoUrls
  properties:
    id:
      type: integer
      format: int64
    category:
      $ref: '#/components/schemas/Category'
    name:
      type: string
      example: doggie
    photoUrls:
      type: array
      xml:
        name: photoUrl
        wrapped: true
      items:
        type: string
    tags:
      type: array
      xml:
        name: tag
        wrapped: true
      items:
        $ref: '#/components/schemas/Tag'
    status:
      type: string
      description: pet status in the store
      enum:
        - available
        - pending
        - sold
  xml:
    name: Pet
```

Notice the use of the following reference notation to Category and Tag within the Pet definition:

```
$ref: '#/components/schemas/Category'
$ref: '#/components/schemas/Tag'
```

Note the use of an `example: doggie` property

<https://swagger.io/docs/specification/adding-examples/>

At this point, we have completed the schema definitions for all the objects that will be used for operations involving all `/pet` endpoints.

Next, we need to define the schema for the content of the request and/or response bodies. This may use any of the objects that we defined earlier by including a references to them. This schema will need to include the appropriate media types for the request and/or response bodies.

<https://swagger.io/docs/specification/media-types/>

Right now, only POST methods will require a body, and this body will contain the specification for a Pet object. So, we can define this request body schema immediately below the schema for Pet:

```
requestBodies:
  Pet:
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/Pet'
      application/xml:
        schema:
          $ref: '#/components/schemas/Pet'
    description: Pet object that needs to be added to the store
    required: true
```

3.6 Parameters

<https://swagger.io/docs/specification/describing-parameters/>

We will start putting in the relevant parameters for the various endpoint methods that require them (not all of them do).

For GET /pet/findByStatus:

```
filter
  parameters:
    - name: status
      in: query
      description: Status values that need to be considered for
      required: true
      explode: true
      schema:
        type: array
        items:
          type: string
          enum:
            - available
            - pending
            - sold
          default: available
```

Note the use of `explode: true` is part of parameter serialization
<https://swagger.io/docs/specification/serialization/>

For GET `/pet/{petId}`:

```
parameters:
  - name: petId
    in: path
    description: ID of pet to return
    required: true
    schema:
      type: integer
      format: int64
```

For POST `/pet/{petId}`:

```
parameters:
  - name: petId
    in: path
    description: ID of pet that needs to be updated
    required: true
    schema:
      type: integer
      format: int64
```

For DELETE `/pet/{petId}`:

```
parameters:
  - name: api_key
    in: header
    required: false
    schema:
      type: string
  - name: petId
    in: path
    description: Pet id to delete
    required: true
    schema:
      type: integer
      format: int64
```

For POST /pet/{petId}/uploadImage:

```
parameters:
  - name: petId
    in: path
    description: ID of pet to update
    required: true
    schema:
      type: integer
      format: int64
```

3.7 Request Body

<https://swagger.io/docs/specification/describing-request-body/>

Next, we can add in the request bodies for the endpoint methods that require them (not all do). We can use references to the schema that we defined earlier for this purpose.

For POST /pet/findByStatus

```
requestBody:
  $ref: '#/components/requestBodies/Pet'
```

For PUT /pet

```
requestBody:
  $ref: '#/components/requestBodies/Pet'
```

For POST /pet/{petId}

```
requestBody:
  content:
    application/x-www-form-urlencoded:
      schema:
        type: object
        properties:
          name:
            description: Updated name of the pet
            type: string
          status:
            description: Updated status of the pet
            type: string
```

For POST /pet/{petId}/uploadImage

```
requestBody:
  content:
    application/octet-stream:
      schema:
        type: string
        format: binary
```

3.8 Response body

<https://swagger.io/docs/specification/describing-responses/>

Every endpoint method must have a minimal description of the response body (at least one response, if not more), which is compulsory.

For POST /pet

```
responses:
  '200':
    $ref: '#/components/requestBodies/Pet'
  '405':
    description: Invalid input
```

For PUT /pet

```
responses:
  '200':
    $ref: '#/components/requestBodies/Pet'
  '400':
    description: Invalid ID supplied
  '404':
    description: Pet not found
  '405':
    description: Validation exception
```

For GET /pet/findByStatus

```
responses:
  '200':
    description: successful operation
    content:
      application/json:
        schema:
          type: array
          items:
            $ref: '#/components/schemas/Pet'
      application/xml:
        schema:
          type: array
          items:
            $ref: '#/components/schemas/Pet'
  '400':
    description: Invalid status value
```

For GET /pet/{petId}

```
responses:
  '200':
    description: successful operation
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/Pet'
      application/xml:
        schema:
          $ref: '#/components/schemas/Pet'
  '400':
    description: Invalid ID supplied
  '404':
    description: Pet not found
```

For POST /pet/{petId}

```
responses:
  '200':
    description: successful update
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/ApiResponse'
  '405':
    description: Invalid input
```

For DELETE /pet/{petId}

```
responses:
  '200':
    description: successful delete
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/ApiResponse'
  '400':
    description: Invalid ID supplied
  '404':
    description: Pet not found
```

For POST /pet/{petId}/uploadImage

```
responses:
  '200':
    description: successful operation
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/ApiResponse'
```

At this point, we have provided a nearly complete OpenAPI specification for the /pet endpoint for this REST API (leaving out authentication, links and callbacks). There should no longer be any errors flagged by the code editor at this point in time.

4 Working with an OpenAPI Spec in SwaggerHub

4.1 Testing via the mock API server

You can now proceed to test out the API by using the mock API server:

<https://virtserver.swaggerhub.com/Org-name/BasicPetStore/1.0.0>

This provides sample dummy responses that are always successful regardless of the request being sent out. For e.g. executing a GET /pet/{petId} will always return a fixed response regardless of the value of {petId}

```
{
  "id": 0,
  "category": {
    "id": 0,
    "name": "string"
  },
  "name": "doggie",
  "photoUrls": [
    "string"
  ],
  "tags": [
    {
      "id": 0,
      "name": "string"
    }
  ],
  "status": "available"
}
```

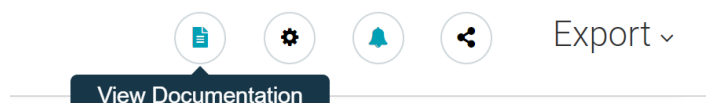
If you wish to test against the second live server URL that you defined: 'https://petstore.swagger.io/v2', you will need to disable credentials for CORS requests first (scroll down to the bottom of the page and click on that link).

Credentials in CORS requests are enabled | [Disable credentials](#) ⓘ

When you are done, make sure you enable credentials again by clicking on the same link.

You can also test the live server at: <https://petstore.swagger.io/v2> using Postman (or some other REST API client) as outlined in the previous lab.

To interact with the Open API spec UI in full browser view, click on the View Documentation icon.



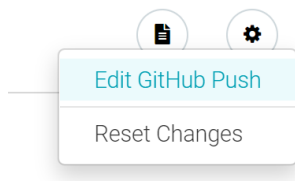
<https://support.smartbear.com/swaggerhub/docs/ui/interactive-documentation.html>

4.2 Integration with other services

OpenAPI specs created in SwaggerHub support integration with a wide variety of other services:

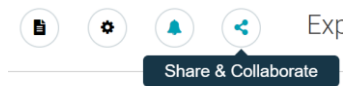
<https://support.smartbear.com/swaggerhub/docs/integrations/index.html>

For e.g. you can push the OpenAPI spec that you have created to an existing repo that you have on a valid GitHub account. This allows you to perform versioning control on future changes that you make to this spec, in a similar way that you would do for a code base.



4.3 Sharing and collaboration

You can choose to share the spec with others or invite other individuals with a SwaggerHub account to collaborate with you on further work on the spec. This is conceptually similar to sharing and collaborating on code repos in GitHub.



<https://support.smartbear.com/swaggerhub/docs/collaboration/index.html>

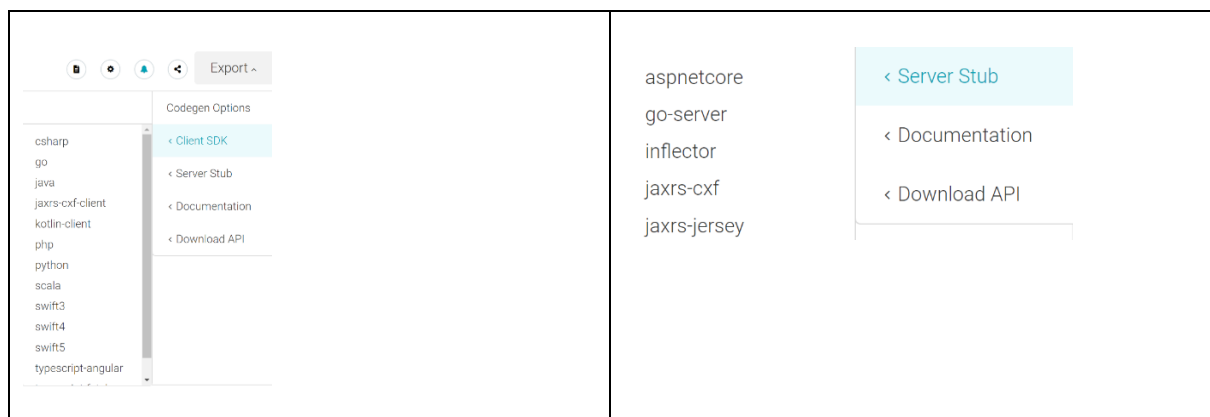
By default, your API spec is public, which means it is visible to everyone who browses through SwaggerHub.

You can view all the existing public API specs created by all SwaggerHub owners on the main search bar, as well as search for a specific type of spec.

<https://support.smartbear.com/swaggerhub/docs/ui/searching.html>

4.4 Generating client-side and server-side code stubs

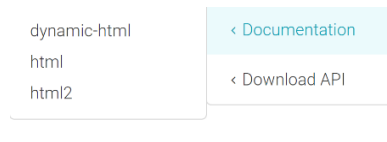
You can generate code stubs for a server that implements the REST API spec as well as a client that is capable of consuming all the operations exposed by that spec. These code stubs can be generated in a variety of frameworks/libraries across multiple languages.



Keep in mind that the code stubs only provide the implementation necessary to handle incoming request bodies of a specific format and return response bodies of a specific format as specified by the API spec. The actual business logic of the client or server will still need to be implemented.

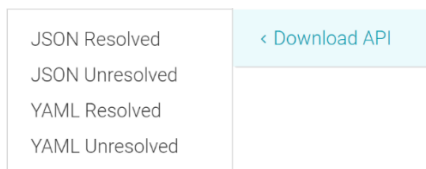
4.5 Exporting the spec

The spec can be exported in HTML for a more browsable form of documentation that does not require a SwaggerHub account or editor to view.



Finally, you can also down the spec in either JSON or YAML format.

<https://support.smartbear.com/swaggerhub/docs/apis/downloading-swagger-definition.html>



This specs can subsequently be imported to create a new API project:

<https://support.smartbear.com/swaggerhub/docs/apis/importing-api.html>

4.6 Other SwaggerHub features

The full range of features available on SwaggerHub are documented at:

<https://support.smartbear.com/swaggerhub/docs/index.html>

5 Further exercises

You can proceed to provide the OpenAPI specification for the `/store` and `/user` endpoint operations in a similar manner just demonstrated using the information provided at:

<https://petstore.swagger.io/>

To check whether your solution is correct, create a new API based on the Petstore template. This provides the complete API spec for the Swagger Petstore.

You can try creating OpenAPI specs for the dummy REST server we worked with in a previous lab:

<https://jsonplaceholder.typicode.com/>

You can also try creating OpenAPI specs for simple REST APIs on RapidAPI or from the other list of public APIs at:

<https://any-api.com/>

<https://mixedanalytics.com/blog/list-actually-free-open-no-auth-needed-apis/>