REST Workshop

Requirements

- Clone the workshop GitHub repository
- Install node and npm on your machine
- Run **`$ npm install -g swagger`** to install Swagger
- Download and install Postman for testing your API (Standalone or Chrome Store)

After you make sure that all requirements are fulfilled, we can start hacking!

Introduction

In this workshop we are going to create a simple REST API for managing a collection of musicians (and their pieces). To achieve this we are going to use Swagger Node and Express and an extremely sophisticated database storage. At the end of this workshop you will have a working REST API that supports creating, updating, deleting and retrieving resources.

Data Storage

In the git repository you will find a file called **storage.js**. This will allow you to manage all artists and pieces in your storage. All data is being stored in a single JSON file (**artists.json**) and can be read to test if operations have been undertaken successfully. The following functions are being provided:

- *createArtist(name, birthdate, deathdate)*: create new artist resource
- *readArtist(id)*: read artist by *id*
- *readAllArtists()*: read all artists
- *deleteArtist(id)*: delete artist
- *updateArtist(id, name, birthdate, deathdate)*: update artist. If a parameter is null, the corresponding value will be left unchanged
- *createPiece(artistId, name, date)*: create a new piece for the artist
- *deletePiece(artistId, pieceId)*: delete piece referenced by *pieceId* from artist referenced by *artistId*
- *updatePiece(artistId, pieceId, name, date)*: update a piece. Similar to *updateArtist(...)*

Tasks

Step -2: create a swagger project

- Run **`$ swagger project create <my-project-name>`** to create a simple Swagger “hello world” project. The project will be created at **./<my-project-name>/**
- During the setup select ‘express’ and press enter
- All following command line commands should be executed inside the project folder.

Step -1: take a look at the dummy project

- Once you have run the commands you will find **api/swagger/swagger.yaml** in your project folder. We recommend you to run **`$ swagger project edit`** to open the awesome editor in your browser.
- Here you can see a default path definition for ‘/hello’ (line 21). This is a dummy path definition. You will have to define another one in a similar way for each API path that you create.
- Next have a look at the file **api/controllers/hello_world.js**. This is the controller file for the ‘/hello’ path and takes care of the actual implementation of the path’s logic.

Step 0: Test the dummy template

- Start the server: run **`$ swagger project start`** in the project root folder
  The server should now be running on http://localhost:10010
- Open your web browser and navigate to http://localhost:10010/hello?name=yourname
- You should get a reply like: "Hello, yourname!"

Step 1: Break the rules

- Modify the swagger.yaml file:
  - Change the type of the ‘name’ parameter from ‘string’ to ‘integer’
  - If the server is running it should automatically restart on changes.
- Now navigate to http://localhost:10010/hello?name=42. It should work.
- Now navigate to http://localhost:10010/hello?name=anyEvilStringHere
  You should get an error message. Swagger validates the types of the supplied parameters against our Swagger definition. Awesome, huh?

Step 2: GET all artists

Now we want to create a new path to get all artists. The API path should be “/artists”.

- Move the **artists.json** (from git) into the root folder of your project
- Create your own controller in **api/controllers/**
- Create a new path in **swagger.yaml** with the url “/artists”
- This path should have the following structure:
  x-swagger-router-controller: **<controller_name>**
  get: structure similar to the existing /hello get.
    - *operationId* is the name of the function that will be executed in your controller.
    - *parameters* are not required for this GET request
    - *responses*: response codes followed by the definitions of the response schemas. A simple string in JSON format should be enough for the start, but Swagger also gives you the option to define the exact structure of your returned objects.
- In your controller, define a function that loads all artists as a list and returns a JSON array from the list. Use the functionality provided by **storage.js**
- Test your functions by navigating to http://localhost:10010/artists
- Try using Postman to send request, it’s really convenient!

Step 3: GET a single artist

In this step we’re going to introduce URI parameters. You have to be able to reference an artist by his id and therefore need a dynamic variable in your URI. To do this we have to:

- Create a new path: /artists/{id}
- Within the path create a new GET method with a parameters entry like this:

```
parameters:
   -  name: id
      in: path
      description: 'artist identifier'
      required: true
      type: int
```

- Your controller method needs to retrieve the id by referencing
  req.swagger.params.id.value ➡ this will contain the value of the id parameter

Step 4: POST, PUT and DELETE artists

Now instead of only focusing on GET request we're going to introduce a few more HTTP methods. You should probably start with DELETE as it's basically the same as a GET request which doesn't return anything besides a status code. The methods you have to implement are the following:

- DELETE artists/:id
- POST artists
- PUT artists/:id

POST and PUT methods also introduce the body parameter, which is pretty much analog to response definitions.

For the ambitious: Introduce Pieces

Be creative! Following functionality could be useful:

- POST musicians/:id/pieces
- GET musicians/:id/pieces
- GET musicians/:id/pieces/:pieceid
- PUT musicians/:id/pieces/:pieceid
- DELETE musicians/:id/pieces/:pieceid