Get a Clue, with json-server

Mitch Dresdner

Table of Contents

An end-to-end testing framework

Summary

In April of this year I made the case for implementing static web sites using the json-server and Zero coding.

While easy to use and practical, i've refactored the article to save some 90 lines of installation steps using Docker and pre-built images courtousy of **Clue** *aka Christian Luck*, so *get a clue* and lets get going!

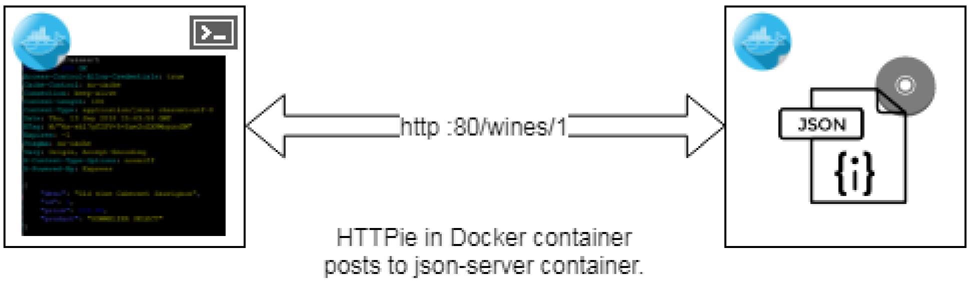
Architecture

The json-server is the Server side of the Client/Server Use Case, with a Client application making REST requests to the Server and the json-server returning canned responses or simulated errors. This is ideal for rapid prototyping of interfaces.

When the json-server starts, it will read a database of static JSON responses which we'll work with later using the HTTPie JSON Client.

This Use Case uses a Dockerized instance of the json-server with it's database mounted externally in a volume share on the local filesystem. The file share is linked on my EC2 instance to an S3 bucket, but you can add it to any convenient location in your file system. Just be sure to adjust the Docker volume mount point accordingly.

Client/Server interaction with the json-server



Installation

The installation really is quite simple! Or, so I stated in the previous article, which hopefully provided a better grounding in each of the steps involved. But now the refactored steps are even easier.

Create HTTPie Docker instance

```
$ # add to your .bashrc file to make alias permanent
$ alias http='docker run -it --rm --net=host clue/httpie'
```

Using your favorite editor, enter the example json data below into the /data/wine.json file.

wine.json example data

```
{
  "wines": [
    { "id": 1, "product": "SOMMELIER SELECT",
      "desc": "Old vine Cabernet Sauvignon", "price": 159.99 },
    { "id": 2, "product": "MASTER VINTNER",
      "desc": "Pinot Noir captures luscious aromas", "price": 89.99 },
    { "id": 3, "product": "WINEMAKER'S RESERVE",
      "desc": "Merlot featuring complex flavors of cherry", "price": 84.99 },
    { "id": 4, "product": "ITALIAN SANGIOVESE",
      "desc": "Sangiovese grape is famous for its dry, bright cherry character",
"price": 147.99 }
  ],
  "comments": [
    { "id": 1, "body": "like the added grape skins", "wineId": 1 },
    { "id": 1, "body": "the directions need to be clearer", "wineId": 2 },
    { "id": 3, "body": "I received 3 different packages of wood chips", "wineId": 1 }
  ],
  "profile": { "name": "vintnor" }
}
```

Running the json-server

With our sample data created lets start playing with the json-server.

Interactions with json-server

In this section we'll starting putting our json-server interactions into practical use.

For a refresher on the usage of **HTTP Verbs** see this DZone HTTP verbs article.

Create json-server Docker container

```
$ # run the json-server
$ docker run -d -p 80:80 --name json-server \
    -v /data/wine.json:/data/db.json \
    clue/json-server
```

① -d json-server runs in the background

② -p host_port_listening_for_request:80 container port

③ -v json_db_on_host:/data.db.json in container

HTTPie Examples

We'll be using the HTTPie Docker container we created an alias for earlier to send JSON messages to the json-server.

Basic example of HTTPie usage

```
http :80/wines/1 ①
or
http http://localhost:80/wines/1 ②
```

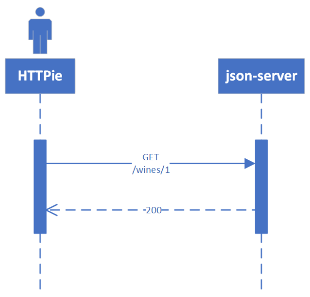
① Short form

② Long form

When you invoke HTTPie using the command line, you can use the *short* form (leave off the **http://localhost** part of the URI), or the *long* form it's your choice.

Making a GET Request

HTTP GET Requests



Use HTTPie, curl or postman

```
http /wines/1
```

GET Requests

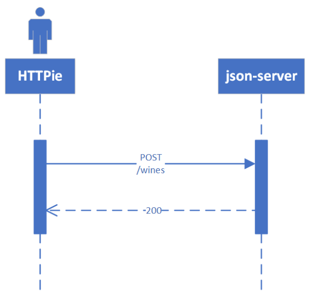
Request	**URI**	**Result**
GET	http :80/wines	All wine entries
GET	http :80/wines/1	Wine with ID=1
GET	http :80/wines?price_gte=100	wines with price >= 100
GET	http :80/wines?id_ne=2	filter id=2
GET	http :80/wines?_embed=comments	embed all comments
GET	http :80/wines/1?_embed=comments	embed comments for ID=1

For more examples see the json-server website

Making a POST Request

With POST we will add a new record to the database.

HTTP POST Requests



Use HTTPie, curl or postman

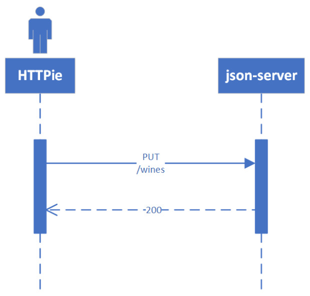
```
http POST :80/wines id=5 product="TWO BUCK CHUCK" price=2.99 desc="Squeezed rapidly
from a delicate, yet unpretentious grape"
```

Request	URI	Result
POST	http POST :80/wines … (see above)	New wine entry with id=5
GET	http :80/wines	All wine entries
GET	http :80/wines?desc_like=grape	All wines with *grape* in desc

Making a PUT Request

In our PUT example we'll make a change to **product** for the record we just added with POST.

HTTP PUT Requests



Use HTTPie, curl or postman

```
http PUT :80/wines/5 product="TWO-ISH BUCK CHUCK" price=2.99 desc="Squeezed rapidly
from a delicate, yet pretentious grape"
```

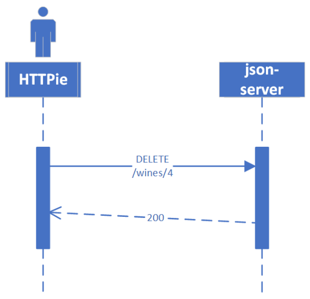
Request	URI	Result
PUT	http PUT :80/wines … (see above)	All wine entries
GET	http :80/wines	All wine entries

If you don’t enter all the fields, PUT will replace with just what you provide.

Finally, a DELETE Request

To complete our example CRUD operations we’ll delete the record with ID=5

HTTP DELETE Requests



Use HTTPie, curl or postman

```
http DELETE :80/wines/5
```

Request	URI	Result
DELETE	http :80/wines/5	Deletes wine with ID=5
GET	http :80/wines	All wine entries

Voila, the record is gone!

There’s lots more you can do with json-server including requests with additional verbs, adding middleware to include new features, enabling complex routing rules, sorting, filtering and much more.

I hope you enjoyed reading this article as much as I have writing it, I’m looking forward to your feedback.

About the Author:

Mitch Dresdner is a Senior Mule Consultant at TerraThink