

# Zero code REST with json-server

Mitch Dresdner

# Table of Contents

Summary .....	1
Architecture .....	2
Installation .....	3
Configuring json-server .....	3
db.json database .....	4
Running the json-server .....	5
Interactions with json-server .....	6
HTTPIe Examples .....	7

## *Static JSON API responses with minimal effort*

An end-to-end testing framework

# Summary

Time and again we find ourselves in need of standing up a JSON server for sharing schemas with our clients while our development is in progress or for supporting our own end-to-end testing.

The json-server is a JavaScript application, as a Java and MuleSoft developer i'm not turned off by working with using other application stacks when there's a good fit. To be able to stand up a JSON REST Server with Zero coding makes it a compelling case. While you can add custom JavaScript middleware to enhance it's functionality, for most developers the standard features should suffice.

When you're in need of a fast, easy to use JSON API solution, not many solutions are as quick to get running or as feature-rich as the [json-server](#).

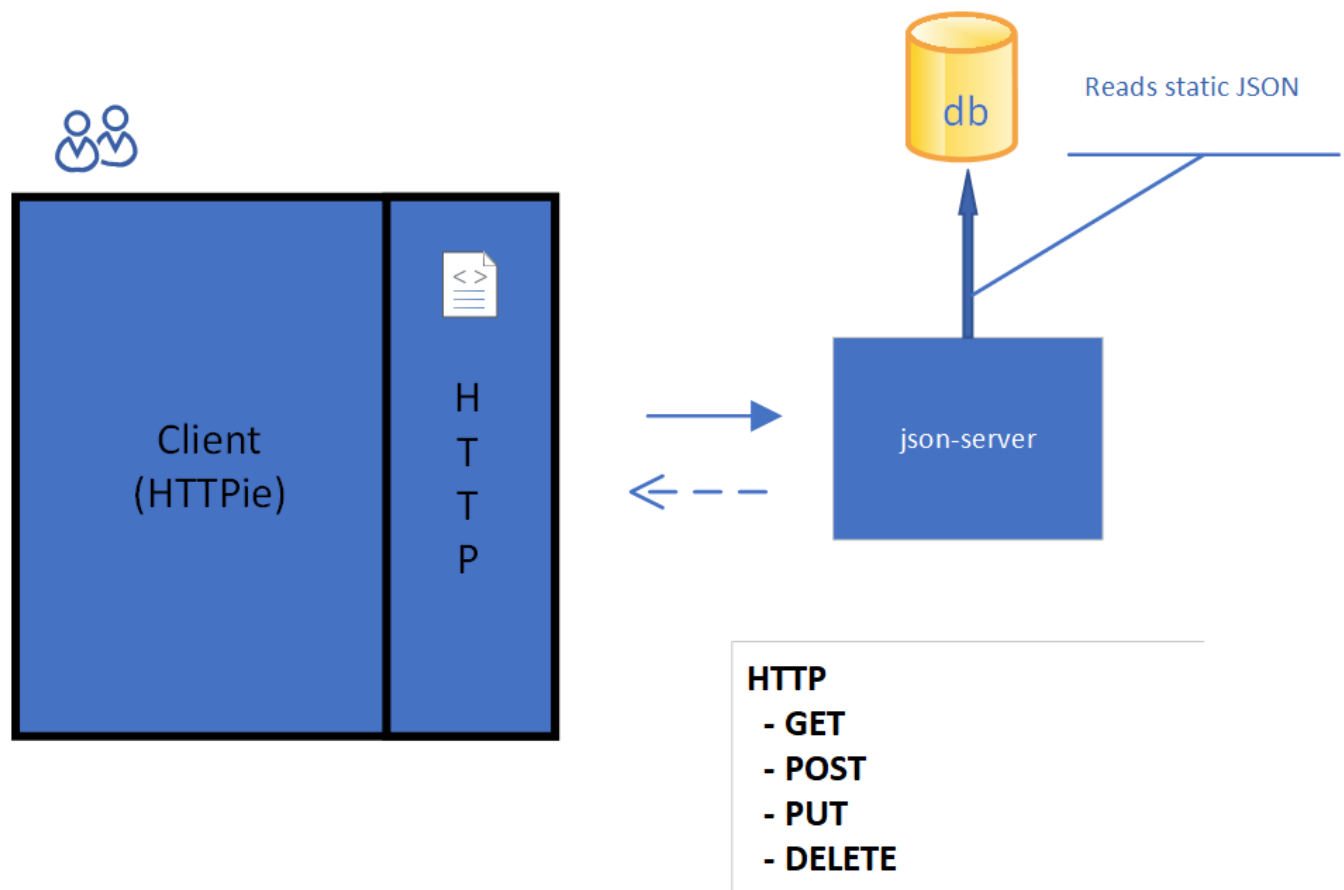
# Architecture

The json-server is the Server side of the Client/Server Use Case, with a Client application making HTTP requests to the Server and the json-server providing determined canned responses or simulated errors.

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.

Another Use Case for json-server is when you're developing a new application which depends on API's which are rate limited. One example of this is the [Open Weather Map](#) api. With the free tier you're limited to a fixed number of calls in a sliding time window. You can capture these kinds of JSON results, add them to json-server database and play them back making unlimited, unbounded requests.

*Client/Server interaction with the json-server*



# Installation

The json-server is a JavaScript application which I hope won't scare off too many Java or Mule developers, certainly it won't scare any polyglots! The installation really is quite simple.

## The Node Package Manager (NPM)

NPM is the default package manager for the JavaScript runtime environment [Node.js](#), it consists of a command line client that interacts with a remote registry. We'll use NPM to download and install **json-server**.

This is as simple installation if you don't already have NPM installed.

Download the latest version [here](#)

With NPM installed, you can perform a global install of json-server. The installation will add json-server to your path and allow you to run it from a shell window.



If this is the first time you're installing NPM you may need to open a new shell to add the new path.

### *Installing the JSON Server*

```
npm install -g json-server  
  
# verify the installation was successful  
json-server -v
```

## Configuring json-server

You will need to decide where you'll keep the JSON schema database, which keeps the schema that will be returned for client requests.

## db.json database

With json-server installed, create a folder where you plan to keep any sample data and project properties.




It's up to you where you would like to put it, I like keeping it on my Google Drive so I can reuse it on different machines, keeping it in Git is another good solution.

```
mkdir json-server
cd json-server
mkdir json
```

In the **json** folder create this sample **db.json** file.

When you've completed the hierarchy should look like this:

*Folder hierarchy*

▼  json-server	^	<input type="checkbox"/> Name	Date modified	Type
 json		<input type="checkbox"/>  db	4/6/2018 3:16 PM	JSON File

```
{
  "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", "postId": 1 },
    { "id": 1, "body": "the directions need to be clearer", "postId": 2 },
    { "id": 3, "body": "I received 3 different packages of wood chips", "postId": 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](#).

From within the json-server folder, we'll run a quick command to print out command line help, run the following command:

*Getting json-server command line help*

```
json-server -h
```

As you can see there's lots of options for changing or overriding the default behaviors.



On a Unix system, change the Windows backslash "\" to a Unix forward slash "/" in the examples that follow.

When we start json-server, the default port it will listen on is 3000. If you prefer a different port you have two options, the first is to use the -p switch passing the new port number. You can also add a config file which you specify the location of using the -c switch. In the examples below we'll be using the defaults.

*Example json-server config file: json-server.json*

```
{  
  "port": 9000  
}
```



With the preliminaries out of the way, lets start json-server and prepare for sending some command line requests.

### *Starting the json-server*

```
json-server --watch json/db.json
```

In this the first example we start the json-server asking it to watch the file **json\db.json** for changes.

### *Beneath the ascii art you should see the following*

```
Loading json\db.json ①
Done

Resources ②
http://localhost:3000/wines
http://localhost:3000/comments
http://localhost:3000/profile

Home ③
http://localhost:3000
```

- ① Database file **json\db.json** loaded successfully
- ② URI's for JSON resources which were loaded
- ③ The URI for the default internal website (you can change this)

## HTTPIe Examples

To install HTTPIe for the examples we'll be working with, you can download it using the this link - [HTTPIe Download](#).

Feel free to use [Postman](#) or curl from a [Git bash](#) terminal shell on Windows if you'd prefer. You should be able to adapt the HTTPIe examples accordingly.

HTTPIe is a [curl](#) like command line tool which can be used from Unix and Windows. I like it better than curl because it comes loaded with lots of syntactic sugar.

### Basic example of HTTPie usage

```
http localhost:3000/wines/1 ①  
or  
http http://localhost:3000/wines/1 ②
```

① Short form

② Long form

Note that when HTTPie installs it will be called **http**, when you invoke it using the command line, you can use or leave off the **http://** part of the URI, it's your choice.

## Default WebSite

Lets get started by hitting the default website from your browser.

*Use browser to access json-server*

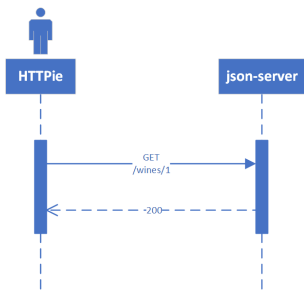
```
http://localhost:3000
```

Under **Resources** you notice that *vintner* has been misspelled as *vintnor*. You fix the typo using your favorite editor to modify the line in **db.json** and save the file. Refreshing the link you notice that the change has already been picked up by json-server.

Providing the **\*--watch** option told the json-server to run in development mode, watching for and reloading changes.

## Making a GET Request

*HTTP GET Requests*



Use *HTTPie*, *curl* or *postman*

```
http localhost:3000/wines/1
```

## GET Requests

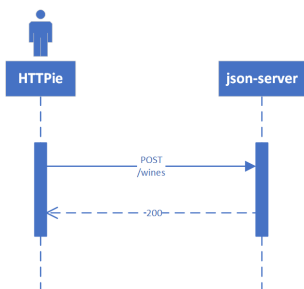
Request	URI	Result
GET	http localhost:3000/wines	All wine entries
GET	http localhost:3000/wines/1	Wine with ID=1
GET	http localhost:3000/wines?price_gte=100	wines with price >= 100
GET	http localhost:3000/wines?id_ne=2	filter id=2

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

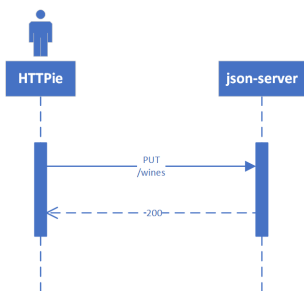
```
http POST localhost:3000/wines id=5 product="TWO BUCK CHUCK" price=2.99 desc="Squeezed rapidly from a delicate, yet unpretentious grape"
```

Request	URI	Result
POST	http localhost:3000/wines ...	All wine entries
GET	http localhost:3000/wines	All wine entries
GET	http localhost:3000/wines?desc_like=grape	All wines with <i>grape</i> 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 localhost:3000/wines/5 product="TWO-ISH BUCK CHUCK" price=2.99 desc="Squeezed rapidly from a delicate, yet pretentious grape"
```

Request	URI	Result
PUT	http localhost:3000/wines	All wine entries
GET	http localhost:3000/wines	All wine entries

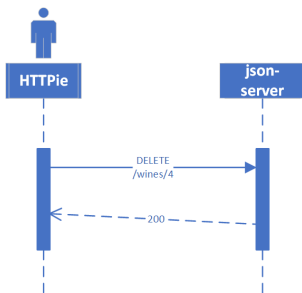


If you don't enter all the fields, PUT will replace with just what you provide.

## 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 localhost:3000/wines/5
```

Request	URI	Result
DELETE	http localhost:3000/wines/5	Deletes wine with ID=5
GET	http localhost:3000/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