HTTP Calls

Cory House

@housecor bitnative.com

Here’s the Plan

- Making HTTP Calls
- Mocking HTTP Calls
  - Why mock?
  - Mocking Approaches
  - Generate mock data and mock API

HTTP Call Approaches

Node	Browser	Node & Browser
http request	XMLHttpRequest	

```
let http = new XMLHttpRequest();
http.open("POST", 'http://your-api.com/user', true);
http.setRequestHeader('Content-type', 'text/html;
charset=UTF-8');

http.onreadystatechange = function() {

    if (http.readyState == 4) {

        if (http.status == 200) {

            onSuccess(JSON.parse(http.responseText));

        } else {

            onError(http);

        }

    }

};

http.onerror = onError;

http.send(text);
```

◀ Plain XMLHttpRequest

HTTP Call Approaches

Node	Browser	Node & Browser
http	XMLHttpRequest	
request	jQuery	
	Framework-based	
	Fetch	

Home | News | Sep 19, 2016 - 2 new features added | Compare browsers | About

Can I use fetch ? Settings

1 result found

Fetch - LS

A modern replacement for XMLHttpRequest.

Global	60.92% + 0.14% =	61.05%
U.S.A.	54.95% + 0.22% =	55.17%

Current aligned | Usage relative | Show all

IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Chrome for Android
			49						
		47	51			9.2		4.4	
8	13	48	52	9.1		9.3		4.4.4	
11	14	49	53	10	39	10	all	52	51
		50	54	TP	40				
		51	55		41				

Browser Polyfill: github.com/github/fetch

Isomorphic polyfill: github.com/matthew-andrews/isomorphic-fetch

```
var request = new Request('http://your-api.com/user', {

  method: 'GET',

  mode: 'cors',

  headers: new Headers({

    'Content-Type': 'text/html; charset=UTF-8'
  })

});



fetch(request).then(onSuccess, onError);
```

◀ With Fetch

HTTP Call Approaches

Node	Browser	Node & Browser
http	XMLHttpRequest	isomorphic-fetch
request	jQuery	xhr
	Fetch	SuperAgent
		Axios

```
axios({
    url: ‘http://your-api.com/user',
    method: 'post',
    headers: {
      'Content-type': 'text/html; charset=UTF-8'},
      data: text
      }).then(onSuccess, onError)
```

◀ With Axios

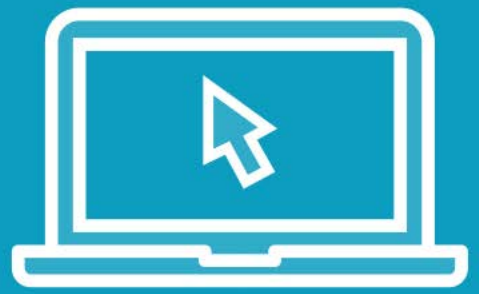
Key: Centralize API Calls

Why Centralize API Calls?

1 Spot

- Configure all calls
- Handle preloader logic
- Handle errors
- Single seam for mocking

Demo



Set up fetch

Centralizing HTTP calls

Why send a polyfill to everyone?

In other words, I was feeling lazy :)

Polyfill.io

Upgrade the web. Automatically.

- **About**
- Browsers and features
- API reference
- Live examples
- Usage stats
- Contributing

Just the polyfills you need for your site, tailored to each browser. Copy the code to unleash the magic:

```
<script src="https://cdn.polyfill.io/v2/polyfill.min.js"></script>
```

Polyfill.io reads the User-Agent header of each request and returns polyfills that are suitable for the requesting browser. Tailor the response based on the features you're using in your app, and see our live examples to get started quickly.

Tweets by @polyfillio

Polyfill.io @polyfillio

We've pushed a 2nd RC of 3.12.0. If you haven't yet tested 3.12 and you use our CDN, test now! github.com/Financial-Time…

3.12.0 release · Issue #858 · Financial-Times/polyfill-se…

We've pushed version 3.12.0-0 to QA and it has been published to NPM under the next label. To install this package from NPM you can run npm i polyfill-service@next.

github.com

08 Sep

This site runs version `3.11.0` of the service. To get notified when we release new versions or push release candidates to our public QA environment, follow us on twitter at @polyfillio.

Only Send Polyfill to Those Who Need It

```
<script
src="https://cdn.polyfill.io/v2/polyfill.js?features=fetch
"></script>
```

Mocking HTTP

Why Mock HTTP?

- Unit Testing
- Instant response
- Keep working when services are down
- Rapid prototyping
- Avoid inter-team bottlenecks
- Work offline

How to Mock HTTP

Nock

Static JSON

Create development webserver

- api-mock
- JSON server
- JSON Schema faker
- Browsersync
- Express, etc.

Static JSON | JSON Server | JSON Server + JSON Schema Faker | Express, etc

Upfront work →
Realism →
Customization →

Our Plan for Mocking HTTP

1. **Declare our schema:**
   - JSON Schema Faker

2. **Generate Random Data:**
   - faker.js
   - chance.js
   - randexp.js

3. **Serve Data via API**
   - JSON Server

JSON Schema

JSON Schema is a vocabulary that allows you to **annotate** and **validate** JSON documents.

Advantages

JSON Schema

- describes your existing data format
- clear, human- and machine-readable documentation
- complete structural validation, useful for
  - automated testing
  - validating client-submitted data

JSON Hyper-Schema

- describes your existing API - no new structures required
- links (including URI Templates for target URIs)
- forms - specify a JSON Schema for the desired data

Quickstart

The JSON document being validated or described we call the *instance*, and the document containing the description is called the *schema*.

The most basic schema is a blank JSON object, which constrains nothing, allows anything, and describes nothing:

Use JSON Schema along with fake generators to provide consistent and meaningful fake data for your system.

We are looking for **contributors**! If you wanna help us make `jsf` more awesome, simply write us so!

NEW in JSON Schema Faker: store schemas online!

Fake Data Libs Bundled in JSON Schema Faker

faker.js

chance.js

randexp.js

randexp.js

Faker.js Docs

github.com/Marak/faker.js/wiki

marak.github.io/faker.js/index.html

cdn.rawgit.com/Marak/faker.js/master/examples/browser/index.html

faker.js - generate massive amounts of fake data in node.js and the browser

Home Code Issues

Generate Person Example

Locality: English

GENERATE NEW

Name: Robb Prosacco

Date of Birth: 1946-9-3

Street Address: 957 Considine Tunnel

City, State Zip: Elvafurt, TX 62769

Country: United States of America

Phone Number: (377) 036-9373 x28276

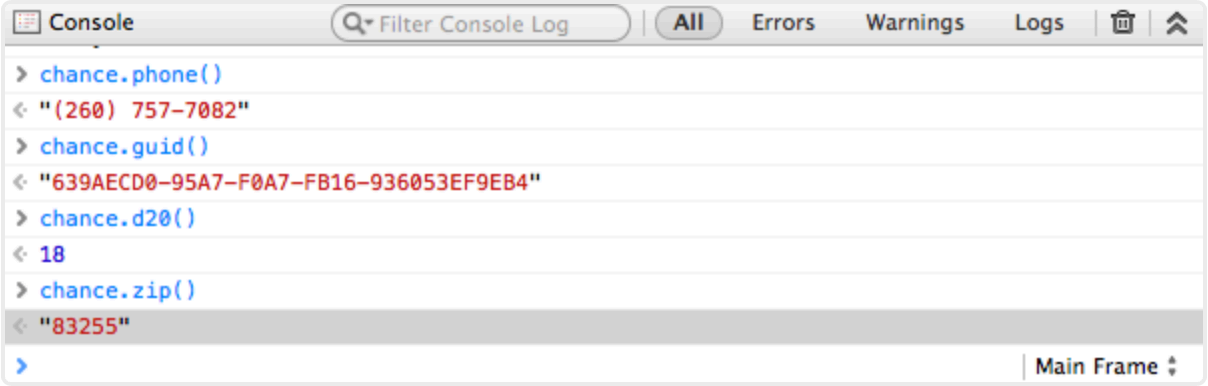
Username: Robb.Prosacco47

Password: g66M5Ay1P6KUAqj

Chance is a minimalist generator of random [1] strings, numbers, etc. to help reduce some monotony particularly while writing automated tests or anywhere else you need anything random.

Chance is open source software and is released under the developer and business-friendly MIT license.

Chance is loaded on this site so you can just open the console on your browser and play!



Tweet

Example usage

```
var jsf = require('json-schema-faker');

var schema = {
  type: 'object',
  properties: {
    user: {
      type: 'object',
      properties: {
        id: {
          $ref: '#/definitions/positiveInt'
        },
        name: {
          type: 'string',
          faker: 'name.findName'
        },
        email: {
          type: 'string',
          format: 'email',
          faker: 'internet.email'
        }
      },
      required: ['id', 'name', 'email']
    }
  },
  required: ['user'],
  definitions: {
    positiveInt: {
      type: 'integer',
      minimum: 0,
      exclusiveMinimum: true
    }
```

Faking values

`jsf` has built-in generators for core-formats, Faker.js and Chance.js are also supported.

You can use **faker** or **chance** properties but they are optional:

```
{
  "type": "string",
  "faker": "internet.email"
}
```

(demo »)

The above schema will invoke `faker.internet.email()`.

Note that both generators has higher precedence than **format**.

You can also use standard JSON Schema keywords, e.g. `pattern`:

```
{
  "type": "string",
  "pattern": "yes|no|maybe|i don't know"
}
```

(demo »)

Advanced usage of faker.js and Chance.js

In following inline code examples the `faker` and `chance` variables are assumed to be created with, respectively:

JSON Schema Faker combines JSON Schema standard with fake data generators, allowing users to generate fake data that conform to the schema.

This application is built using json-schema-faker npm module **version 0.3.6** built with browserify.

JSON Schema

```
{
  "type": "array",
  "minItems": 100,
  "maxItems": 200,
  "items": {
    "type": "integer"
  }
}
```

Sample output

```
[
  35148542,
  -37138934,
  -6267495,
  78032497,
  -59641118,
  2727993,
  -72491060,
  -836538,
  49036025,
  46357723,
  67507885,
  -67953268,
  -18019199,
  -56865883,
  -69997617,
  -41304637,
  40486868
```

Generate sample Save

This repository Search Pull requests Issues Gist

typicode / **json-server**

Watch 495 | Unstar 16,314 | Fork 1,059

Code | Issues 78 | Pull requests 15 | Projects 0 | Pulse | Graphs

Get a full fake REST API with zero coding in less than 30 seconds (seriously)

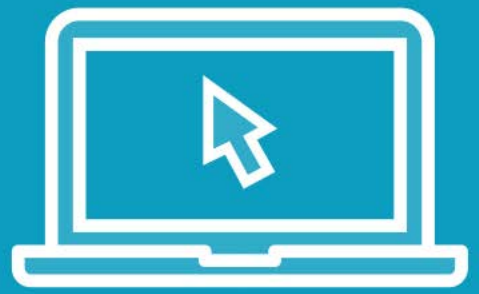
488 commits | **1** branch | **93** releases | **32** contributors | MIT

Branch: **master** | New pull request | Create new file | Upload files | Find file | Clone or download

typicode 0.8.21 — Latest commit e547381 2 days ago

bin	Refactor CLI and add tests	a year ago
src	Add common middlewares for defaults and router	2 days ago
test	lint	19 days ago
.babelrc	Update dependencies and watch	3 months ago
.gitignore	Update dependencies and watch	3 months ago
.travis.yml	Update .travis.yml	10 months ago
CHANGELOG.md	Update CHANGELOG.md	2 days ago
LICENSE	Update LICENSE	10 months ago
README.md	Add pagination	5 days ago
package.json	0.8.21	2 days ago

Demo



Mock HTTP

- JSON Schema Faker
  - faker, chance, regexp
- JSON Server

Wrap Up

Making HTTP Calls

- **Node:** http, request
- **Browser**: XMLHttpRequest, jQuery, fetch
- **Node and Browser**: Isomorphic Fetch, xhr, SuperAgent, Axios

Mocking HTTP Calls

- Nock, Hard coded JSON
- Custom webserver: json-server, JSON Schema Faker, Express, Browsersync

Next up: Project structure & demo apps