# **HTTP Calls**



**Cory House** 

@housecor reactjsconsulting.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 XMLHttpRequest** http request



```
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 XMLHttpRequest** http **jQuery** request Framework-based **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 request XMLHttpRequest jQuery Fetch isomorphic-fetch
xhr
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:)





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 <u>User-Agent</u> header of each request and returns polyfills that are suitable for the requesting browser. <u>Tailor the response</u> based on the features you're using in your app, and see our <u>live examples</u> 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

This site runs version **3.11.0** of the service. To get notified when we release new versions or push release candidates to our <u>public QA</u> 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?

Ha! They thought it was safe to use non idempotent GET requests!



**Unit Testing** 

**Instant response** 

Keep working when services are down

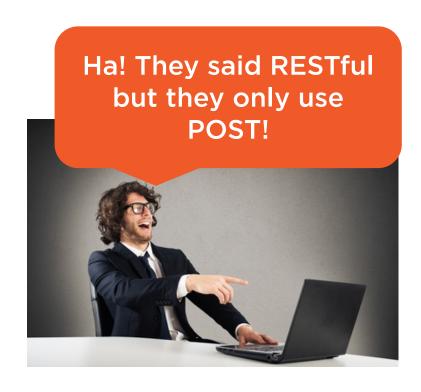
Rapid prototyping

**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



#### Declare our schema:

- JSON Schema Faker

#### 2. Generate Random Data:

- faker.js
- chance.js
- randexp.js

#### 3. Serve Data via API

JSON Server













Docs Examples Software

### 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
  - o validating client-submitted data

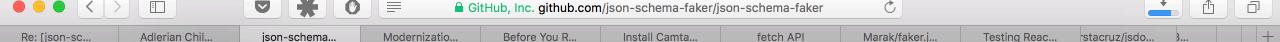
#### JSON Hyper-Schema

- 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





randexp.js

faker.js

chance.js

randexp.js

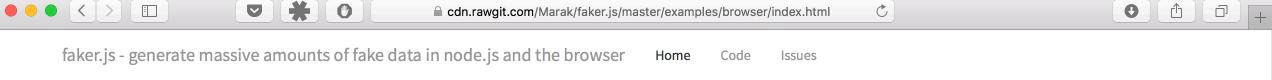


# Faker.js Docs

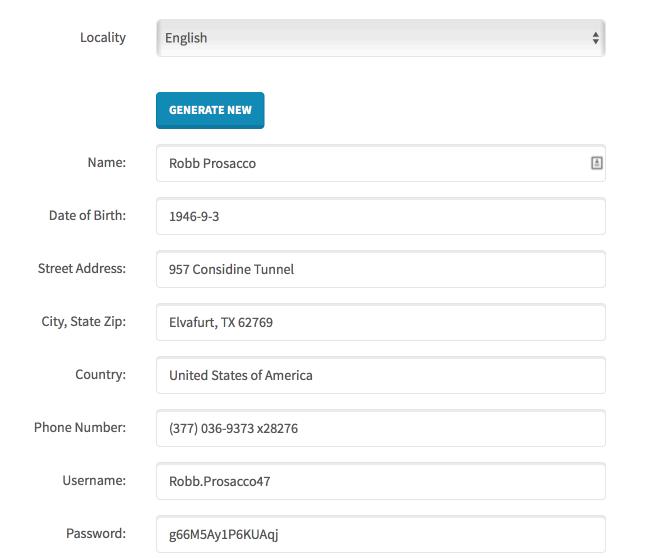


github.com/Marak/faker.js/wiki marak.github.io/faker.js/index.html





### Generate Person Example





#### Chance.js (1.0.3)

- ▶ <u>Download</u>
- ▶ <u>Todo</u>
- ▶ Change Log
- ▶ <u>Acknowledgements</u>

#### <u>Usage</u>

- ▶ <u>bower</u>
- ▶ <u>browser</u>
- → <u>cli</u>
- component
- ▶ <u>node</u>
- ▶ requirejs
- ▶ <u>seed</u>
- ▶ <u>function</u>

#### **Basics**

- ▶ <u>bool</u>
- ▶ <u>character</u>
- ▶ floating
- ▶ <u>integer</u>
- ▶ <u>natural</u>
- ▶ <u>string</u>

#### <u>Text</u>

- ▶ paragraph
- ▶ <u>sentence</u>
- ▶ syllable
- ▶ <u>word</u>

#### <u>Person</u>

- ▶ <u>age</u>
- ▶ birthday
- <u>cf</u> <u>cpf</u>
- ▶ <u>first</u>
- N gender

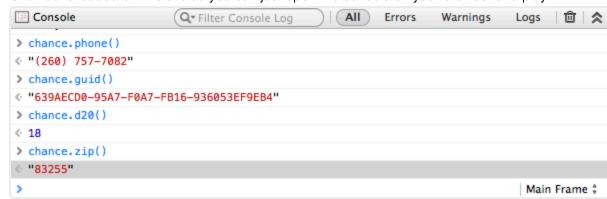


**Chance** is a minimalist generator of random [1] strings, numbers, etc. to help reduce some monotony particularly while writing automated tests or anywhere else you need anything random.

C

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!







```
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
```

#### **<sup>∞</sup> 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"
```

C

#### (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 faken and shapes variables are assumed to be created with respectively:



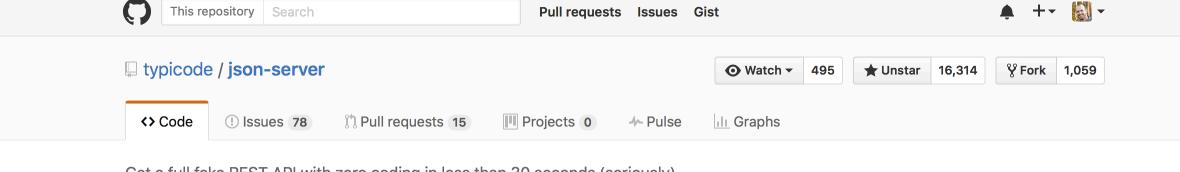
JSON Schema Faker combines JSON Schema standard with fake data generators, allowing users to generate fake data that conform to the schema.

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

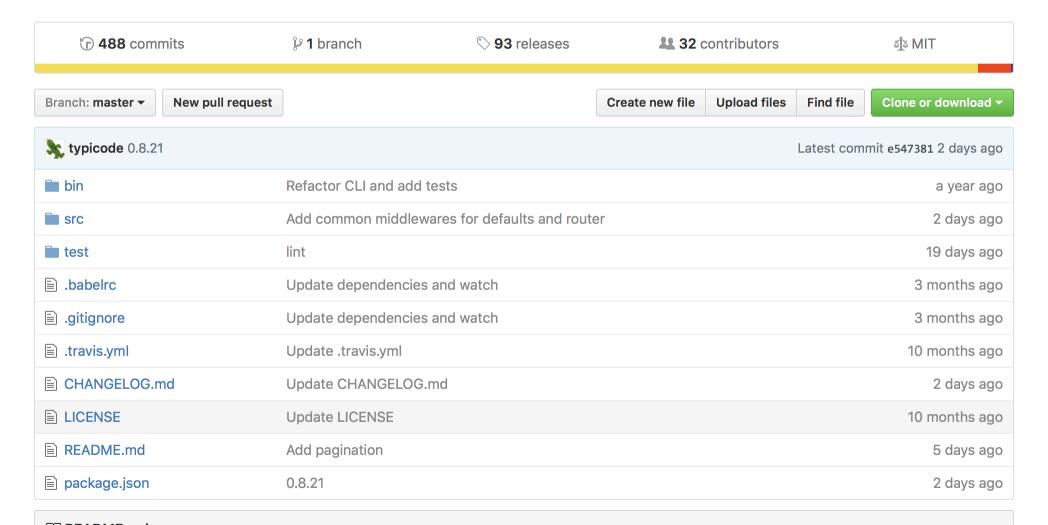
```
Sample output
 1 -
       35148542,
       -37138934.
       -6267495,
 5
      78032497,
       -59641118,
 7
      2727993,
 8
      -72491060,
 9
       -836538,
10
       49036025,
11
       46357723,
12
       67507885,
13
       -67953268
14
       -18019199,
       -56865883,
15
16
       -69997617,
      -41304637,
17
```

Generate sample

Save



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



## 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

