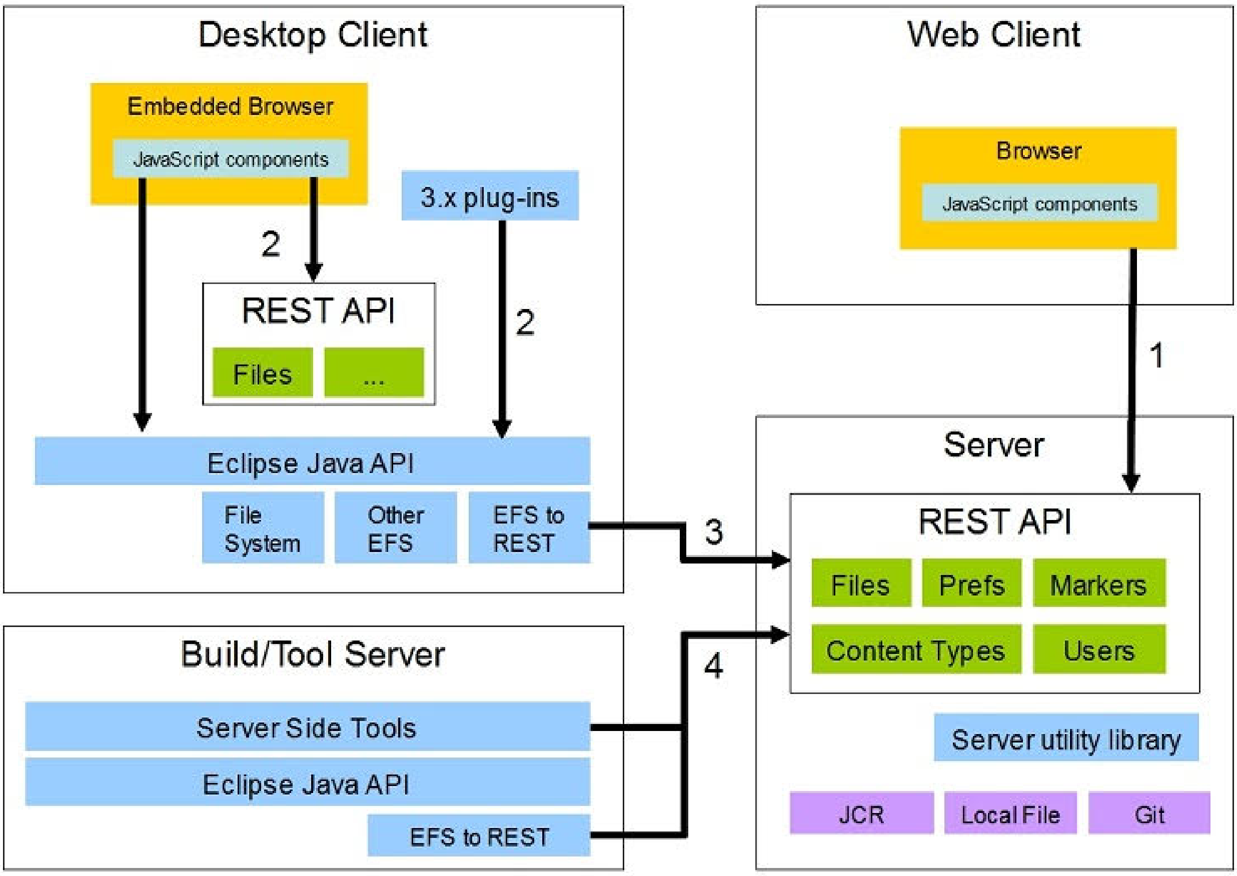
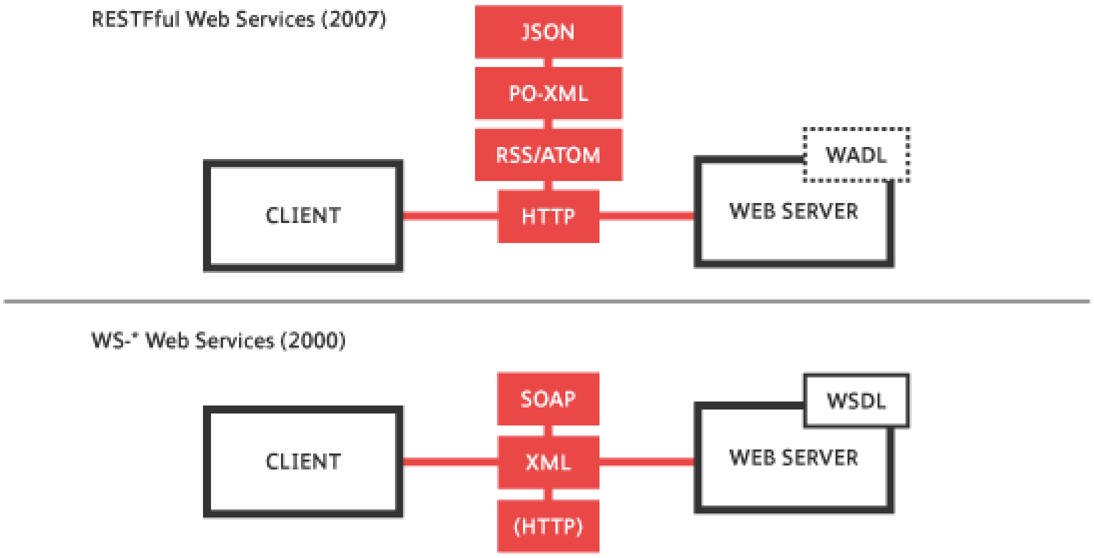
AJAX+REST architecture





REST Architectural Constraints

REST stands for **Re**presentational **S**tate **T**ransfer, a term coined by Roy Fielding in 2000. It is an **architecture style** for designing loosely coupled applications over HTTP, that is often used in the development of web services. REST does not enforce any rule regarding how it should be implemented at lower level, it just put high level design guidelines and leave you to think of your own implementation.

In my last employment, I designed RESTful APIs for telecom major company for 2 good years. In this post, we will be sharing my thoughts apart from normal design practices. You may not be agree with me on few points, and that's perfectly OK. I will be happy to discuss anything from you with open mind.

Let's start with standard design specific stuff to clear what 'Roy Fielding' wants us to build. Then we will discuss my stuff which will be more towards finer points while you design your RESTful APIs.

Architectural Constraints

REST defines **6 architectural constraints** which make any web service – a true RESTful API.

1. Uniform interface
2. Client–server
3. Stateless
4. Cacheable
5. Layered system
6. Code on demand (optional)

Uniform interface

As the constraint name itself applies, you MUST decide APIs interface for resources inside system which are exposed to API consumers and follow religiously. A resource in system should have only one logical URI, and then should provide way to fetch related or additional data. It's always better to **synonymise a resource with a web page**.

Any single resource should not be too large and contain each and everything in it's representation. Whenever relevant, a resource should contain **links (HATEOAS) pointing to relative URIs** to fetch related information.

Also, the resource representations across system should follow certain guidelines such as naming conventions, link formats or data format (xml or/and json).

All resources should be accessible through a common approach such as HTTP GET, and similarly modified using consistent approach.

Once a developer become familiar with one of your API, he should be able to follow similar approach for other APIs.

Client–server

This essentially means that client application and server application MUST be able to evolve separately without any dependency
on each other. Client should know only resource URIs and that’s all. Today, this is normal practice in web development so nothing fancy is required from your side. Keep it simple.

Servers and clients may also be replaced and developed independently, as long as the interface between them is not altered.

Stateless

Roy fielding got inspiration from HTTP, so it reflects in this constraint. Make all client-server interaction stateless. Server will not store anything about latest HTTP request client made. It will treat each and every request as new. No session, no history.

If client application need to be a stateful application for end user, where user logs in once and do other authorized operations thereafter, then each request from the client should contain all the information necessary to service the request – including authentication and authorization details.

No client context shall be stored on the server between requests. Client is responsible for managing the state of application.

Cacheable

In today’s world, caching of data and responses is of utmost important wherever they are applicable/possible. The webpage you are reading here is also a cached version of HTML page. Caching brings performance improvement for client side, and better scope for scalability for server because load has reduced.

In REST, caching shall be applied on resources when applicable and then these resources MUST declare themselves cacheable. Caching can be implemented on server or client side.

Well-managed caching partially or completely eliminates some client–server interactions, further improving scalability and performance.

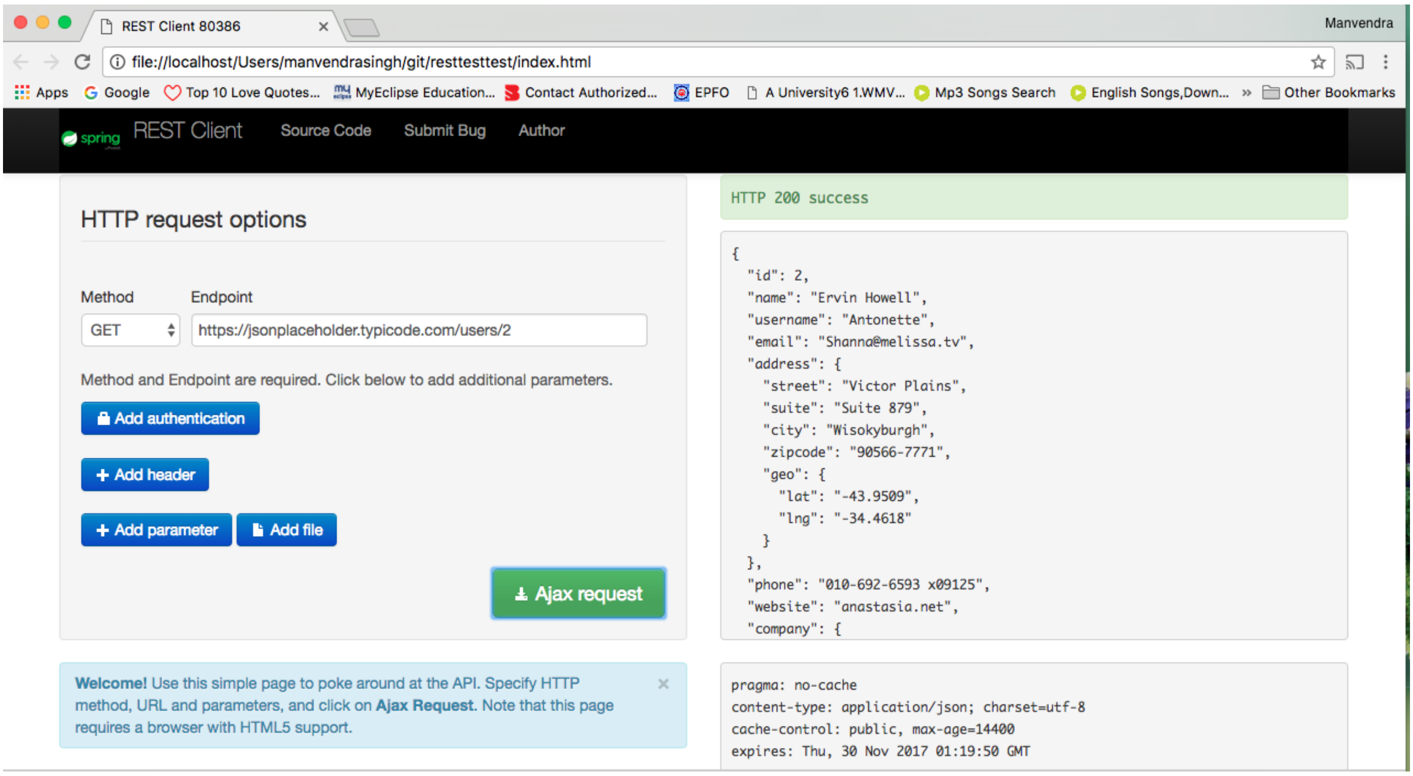
Layered system

REST allow you to use a layered system architecture where you deploy the APIs on server A, and store data on server B and authenticate requests in Server C, for example. A client cannot ordinarily tell whether it is connected directly to the end server, or to an intermediary along the way.

Code on demand (optional)

Well, this constraint is optional. Most of the time you will be sending the static representations of resources in form of XML or JSON. But when you need to, you are free to `return executable code` to support a part of your application e.g. clients may call your API to get a UI widget rendering code. It is permitted.

All above constraints helps you build a true RESTful API and you should follow them. Still, at times you may find yourself violating one or two constraints. Do not worry, you are still making a RESTful API – but not “truely RESTful”.
Notice that all above constraints are most closely related to WWW



HTTP request options

Method: GET
Endpoint: https://jsonplaceholder.typicode.com/users/2

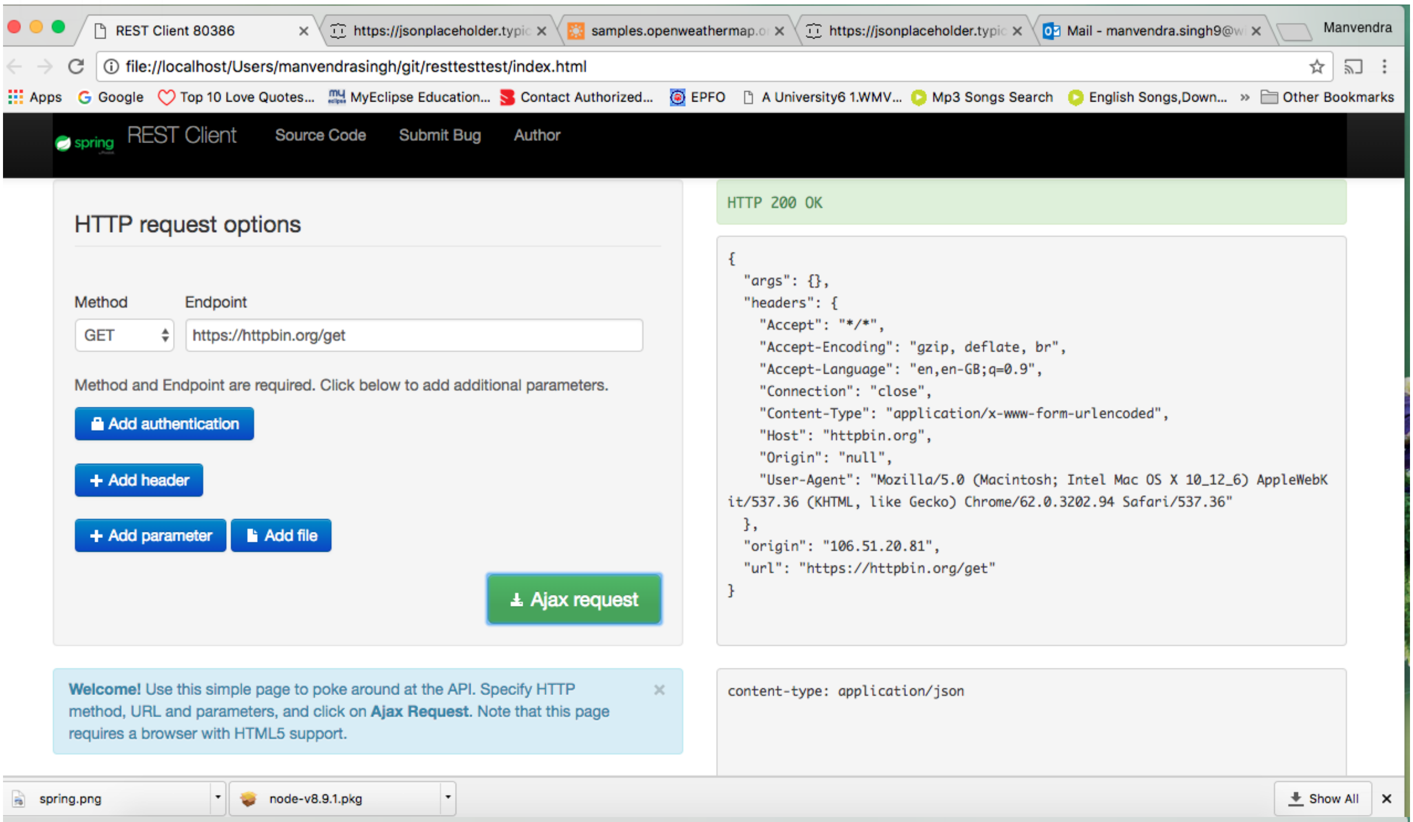
Method and Endpoint are required. Click below to add additional parameters.

Welcome! Use this simple page to poke around at the API. Specify HTTP method, URL and parameters, and click on **Ajax Request**. Note that this page requires a browser with HTML5 support.

HTTP 200 success

```
{
  "id": 2,
  "name": "Ervin Howell",
  "username": "Antonette",
  "email": "Shanna@melissa.tv",
  "address": {
    "street": "Victor Plains",
    "suite": "Suite 879",
    "city": "Wisokyburgh",
    "zipcode": "90566-7771",
    "geo": {
      "lat": "-43.9509",
      "lng": "-34.4618"
    }
  },
  "phone": "010-692-6593 x09125",
  "website": "anastasia.net",
  "company": {
```

```
pragma: no-cache
content-type: application/json; charset=utf-8
cache-control: public, max-age=14400
expires: Thu, 30 Nov 2017 01:19:50 GMT
```



HTTP request options

Method: GET

Endpoint: https://httpbin.org/get

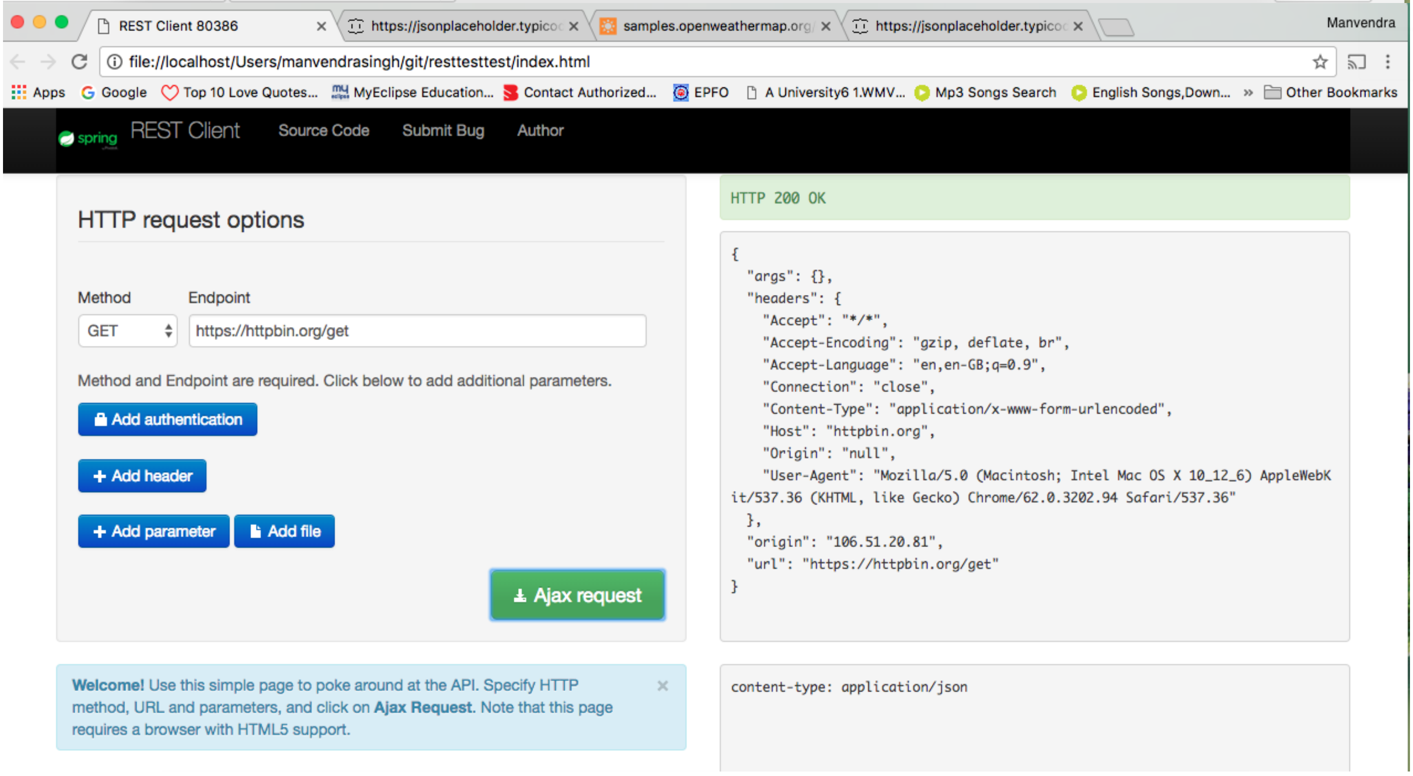
Method and Endpoint are required. Click below to add additional parameters.

Welcome! Use this simple page to poke around at the API. Specify HTTP method, URL and parameters, and click on **Ajax Request**. Note that this page requires a browser with HTML5 support.

```
HTTP 200 OK
```

```
{
  "args": {},
  "headers": {
    "Accept": "*/*",
    "Accept-Encoding": "gzip, deflate, br",
    "Accept-Language": "en,en-GB;q=0.9",
    "Connection": "close",
    "Content-Type": "application/x-www-form-urlencoded",
    "Host": "httpbin.org",
    "Origin": "null",
    "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebK
it/537.36 (KHTML, like Gecko) Chrome/62.0.3202.94 Safari/537.36"
  },
  "origin": "106.51.20.81",
  "url": "https://httpbin.org/get"
}
```

```
content-type: application/json
```



HTTP request options

Method: GET

Endpoint: https://httpbin.org/get

Method and Endpoint are required. Click below to add additional parameters.

Welcome! Use this simple page to poke around at the API. Specify HTTP method, URL and parameters, and click on **Ajax Request**. Note that this page requires a browser with HTML5 support.

```
HTTP 200 OK
```

```
{
  "args": {},
  "headers": {
    "Accept": "*/*",
    "Accept-Encoding": "gzip, deflate, br",
    "Accept-Language": "en,en-GB;q=0.9",
    "Connection": "close",
    "Content-Type": "application/x-www-form-urlencoded",
    "Host": "httpbin.org",
    "Origin": "null",
    "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebK
it/537.36 (KHTML, like Gecko) Chrome/62.0.3202.94 Safari/537.36"
  },
  "origin": "106.51.20.81",
  "url": "https://httpbin.org/get"
}
```

```
content-type: application/json
```