

# SPORE

Specification to a **P**ortable **R**est  
**E**nvironment

franck cuny

<linkfluence>

I'm here thanks to the

**Mongueurs de Perl**

<http://www.mongueurs.net/>



# Who am I ?

- [franck](#)
- [lumberjaph.net](#)
- [github.com/franckcuny](#)
- [search.cpan.org/~franckc/](#)
- [twitter.com/franckcuny](#)

# REST - at last, something that's not crap

- more and more APIs are REST(ful)
- it's easy to write API and Client to interact with it
- it's stateless!
- REST is here to stay (and it's a good thing!)

# REST - creating a request

Most of the time, to create a request :

- you define a FORMAT
  - using HTTP Headers (Content-Type, Accept, ...)
  - adding a prefix to the request (.json, .xml, .yml, ...)
- you give a path to a RESOURCE
- you set some parameters

# REST - the request ; why it's boring

it's always the same code to write:

- set the format
- path to resource
- authentication
- (de)serialization of the request/response
- check parameters
- check the HTTP result (is my request successful ?)

# SPORE - why SPORE exists

- I don't want to write this kind of code over and over again
- I can use Classes with Roles/Mixin, but I still have to think about the code
- I want something easy to maintain, and that I can use with multiple languages ( <3 Acmeism)

# Net::HTTP::API - a first try

```
package TwitterAPI;
```

```
use Net::HTTP::API;
```

```
net_api_declare twitter => (  
  api_base_url  => 'http://search.twitter.com/',  
  api_format    => 'json',  
  api_format_mode => 'append',  
);
```

```
net_api_method search => (  
  method  => 'GET',  
  path    => '/search',  
  params  => [qw/q lang local page/],  
  required => [qw/q/],  
);
```

```
1;
```

# Net::HTTP::API - a first try

- use meta programming to declare an API (will see later)
- thanks to this I can write really quickly a client to an API
- but it's still code ...
- ... and I still have to maintain it

# Net::HTTP::API - what can be improved

- use a data description language (JSON, YAML)
- rely more on meta programming
- simplify API (don't try to fit everything in one implementation)

# SPORE

Spore is a **specification** for **describing REST API** that can be parsed and used automatically by **client implementations** to communicate with the described API.

# SPORE - describing a REST API

**The first part defines a way to describe a REST API in a  
description file**

**SPORE** - a specification to build client

**The second part defines how the various implementations  
should process the description to produce clients**

# SPORE - how to describe the Twitter API

## **public\_timeline:**

**method:** GET

**path:** /statuses/public\_timeline.:format

## **optional\_params:**

- trim\_user
- include\_entities

## **required\_params:**

- format

## **expected\_status:**

- 200

# SPORE - a specification for client

a way to define how to query an API

- HTTP method
- path to a resource
- a format
- a list of (optional or required) parameters
- does this method requires authentication ?
- a list of expected HTTP status for the response

# SPORE - things I stole from others

- **Rack / WSGI / Plack** are **awesome**
- one of the key feature for their adoption were the middlewares
- and the fact that the implementation relies on a simple specification

# SPORE - steals from others

so I stole ...

- CGI's %ENV
- WSGI's response [\$status, [\$headers], \$content]
- middlewares

# SPORE - CGI-like %ENV

contains keys like

- **REQUEST\_METHOD**
- **SCRIPT\_NAME**
- **PATH\_INFO**
- **QUERY\_STRING**
- ...

# SPORE - CGI-like %ENV

This Hash is built using the meta information from the  
method

# SPORE - implementation (Perl)

## **meta programming everywhere**

- Each API method is an instance of an extended Moose::Meta::Method
- Code generated from the description
- Clean internal API to manipulate API methods
- Easy to add or remove method to the client

# SPORE - implementation (Perl)

```
$class->meta->add_spore_method(  
    "user_timeline",  
    path          => '/statuses/public_timeline.:format',  
    required_params => [qw/format/],  
    optional_params => [qw/trim_user include_entities/],  
    expected_status => [qw/200/],  
);  
  
# if I want to list the methods available for this API  
my @methods = $class->meta->get_all_spore_methods();  
  
my $client = $class->new();  
  
$client->user_timeline();  
  
...
```

# SPORE - middlewares

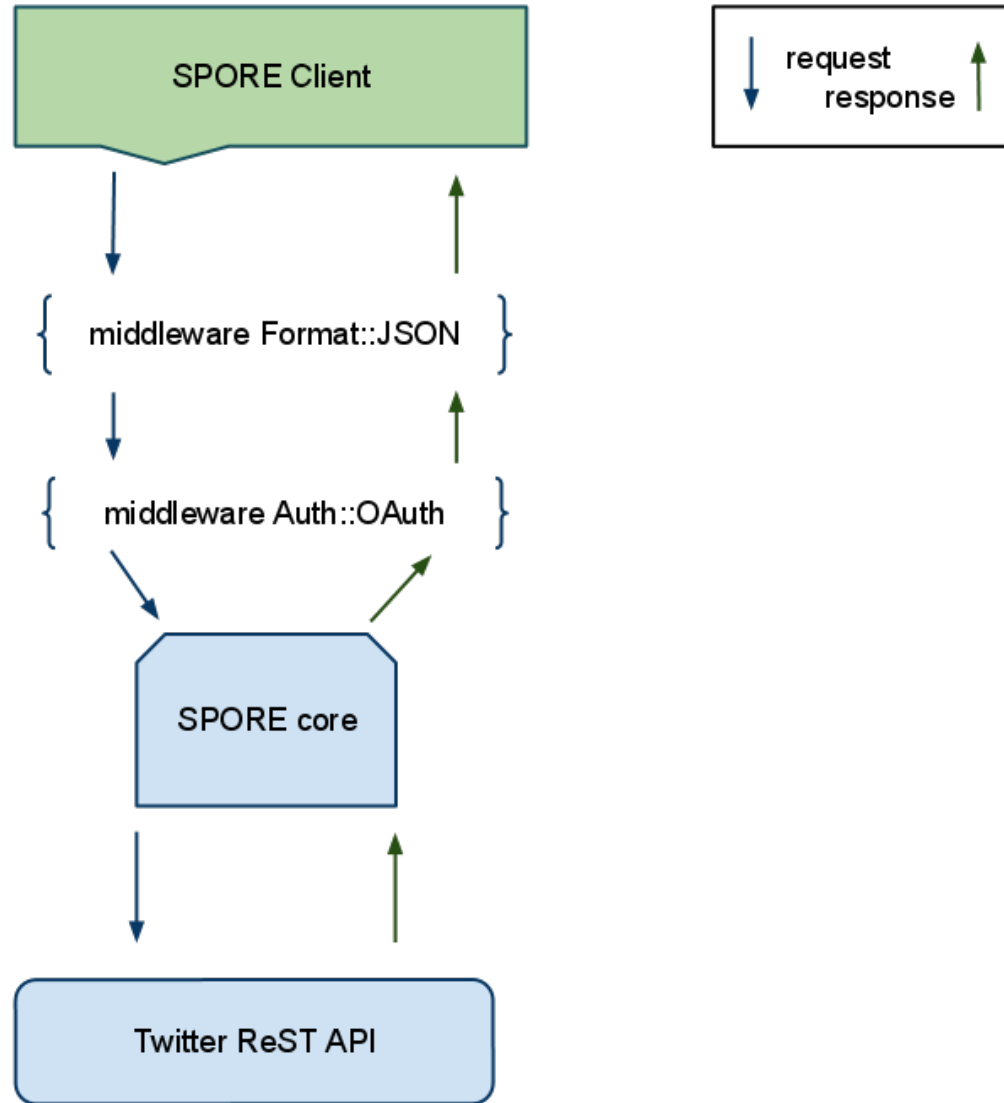
- let's not complicate the specification with corner case details
  - there's NO POSSIBLE WAY to handle all peculiar specifications
- focus on the ones that will always be present

**so delegate all this to middlewares**

# SPORE - middlewares

- (de)serialization
- authentication
- cache
- proxying
- tests

# SPORE - f\*cking middlewares; how do they work ?



# SPORE - f\*cking middlewares; how do they work ?

- a MW receives a **request** object
- MWs are executed in the order they have been declared
- a MW can bypass the process by returning a **response** object
- a MW can return a **callback** that will be executed when the response is received

# SPORE - create a client

## **create\_repo:**

**method:** POST

**path:** /:format/repos/create

**authentication:** 1

**required\_params:**

- format

## **user\_information:**

**method:** GET

**path:** /:format/user/show/:username

**required\_params:**

- username
- format

# SPORE - create a client (Perl)

```
my $c = Config::GitLike::Git->new(); $c->load;

my $login = $c->get(key => 'github.user');
my $token = $c->get(key => 'github.token');

my $github = Net::HTTP::Spore->new_from_spec('github.json');

$github->enable('Format::JSON');
$github->enable(
    'Auth::Basic',
    username => $login . '/token',
    password => $token,
);

my $res = $github->create_repo(
    format => 'json',
    payload => {name => $name, description => $desc}
);
```

# SPORE - create a client (Lua)

```
require 'Spore' -- http://fperrad.github.com/lua-Spore
```

```
local github = Spore.new_from_spec 'github.json'
```

```
github:enable 'Format.JSON'
```

```
github:enable('Auth.Basic', {  
    username = 'mojombo/token',  
    password = '6ef8395fecf207165f1a82178ae1b984',  
})
```

```
local r = github:user_information({  
    format = 'json', username = 'mojombo'})
```

```
print(r.status)          --> 200
```

```
print(r.headers['x-runtime']) --> 126ms
```

```
print(r.body.user.name)  --> Tom Preston-Werner
```

# SPORE - another exemple ; writing tests

let's say you're writing an application that use an external API.

when you write tests, you may not be able to query this API.

but you still need to test the full process.

here comes the Mock middleware!

# SPORE - another exemple ; writing tests

```
my $mock_server = {  
  '/api/ping' => sub {  
    my $req = shift;  
    $req->new_response( 200, [ 'Content-Type' => 'text/plain' ], 'ok');  
  },  
};
```

```
ok my $client =  
  Net::HTTP::Spore->new_from_spec( 'my_awesome_api.json',  
    api_base_url => 'http://localhost' );
```

```
$client->enable('Mock', tests => $mock_server);
```

```
my $res = $client->ping();  
is $res->header('Content-Type'), 'text/plain';  
is $res->content, 'ok';
```

# SPORE - benefits

- you don't have to write a client anymore
- focus on middlewares
- you don't have to choose between numerous implementations of an API client
- less code to maintain
- easier to test

# SPORE - ecosystem

- <http://lumberjaph.net/misc/2010/09/17/spore.html>
- <http://lumberjaph.net/misc/2010/10/20/spore-update.html>
- <http://github.com/spore>

# SPORE - ports

- <http://github.com/franckcuny/net-http-spore>
- <http://fperrad.github.com/lua-Spore>
- <http://github.com/ngrunwald/clj-spore>
- <http://github.com/elishowk/pyspore>
- <https://github.com/sukria/Ruby-Spore>
- <http://github.com/francois2metz/node-spore>

commercial break - part 1

**Linkfluence is **HIRING****

**we're looking for some **Perl** developers**

**if you want to work in Paris, come talk to me!**

**<http://us.linkfluence.net/>**

commercial break - part 2

## **French Perl Workshop**

<http://journesperl.fr/fpw2011/>

**Paris, France on the 24th and 25th of June**

-

**it's a free event !!**

**Thanks!**

(see you in June)