# Build REST APIs with Symfony2


Eduardo Oliveira
entering @ github

# What we are going to talk about?

- HTTP API
- Serialization
- Documentation
- Forms
- Testing
- HTTP Clients
- API Design

# HTTP API

# HTTP is a request/response protocol

- Request
  - URI and HTTP method
  - Protocol version
  - Entity (headers and body)

- Response
  - Status code
  - Entity (headers and body)

# Request

POST /speakers HTTP/1.1
Host: symfonyday.pt
Content-type: application/json

```
{
  ...
}
```

# Response

HTTP/1.1 201 Created
Location: http://symfonyday.pt/speakers/1
Content-type: application/json

```
{
  id: 1,
  name: “...”
}
```

# Symfony has some tools that helps to build APIs

- Routing
- HTTP Foundation
- Serializer
- Forms

# Is there something more?

# FOSRestBundle

# What does FOSRestBundle do?

- Body decoding
- Multiple output formats
- Exception handling
- Automatic routing
- Content negotiation
- ...

# Setup

```
"friendsofsymfony/rest-bundle": "~1.2",
"jms/serializer-bundle": "~0.13"
```

```
// AppKernel.php
new FOS\RestBundle\FOSRestBundle(),
new JMS\SerializerBundle\JMSSerializerBundle($this),
```

```
# config.yml
sensio_framework_extra:
    view:
        annotations: false

fos_rest:
    format_listener:
        rules:
            -
                path: ^/
                priorities: [json, xml]
                fallback_format: json
                prefer_extension: false
    view:
        view_response_listener: true
```

# Usage

```
esorest_demo_hello:
    path:  /hello/{name}
    defaults: { _controller: esorest_demo.default_controller:indexAction }
```

```
services:
    esorest_demo.default_controller:
        class: ESO\RESTDemoBundle\Controller\DefaultController
```

```
use FOS\RestBundle\Controller\Annotations as Rest;
use Symfony\Component\HttpFoundation\Response;

class DefaultController
{
    /**
     * @Rest\View(statusCode=Response::HTTP_OK)
     */
    public function indexAction($name)
    {
        return array('name' => $name);
    }
}
```

# Serialization

# Symfony supports serialization, but there are alternatives

# JMSSerializerBundle

# What does JMSSerializerBundle do?

- Integrates JMSSerializer lib into Symfony2
- Enables (de)serialization of data
- Highly configurable
- Supports versioning and exclusion strategies

# Usage

```php
class Talk
{
    protected $title;
    protected $startDate;
    protected $abstract;
    protected $speaker;
```

```php
class Speaker
{
    protected $name;
    protected $bio;
    protected $whyMe;
```

```php
class Talks
{
    protected $talks;
```

```php
/**
 * @Rest\View()
 */
public function getTalksAction()
{
    return new Talks(
        array(
            new Talk(
                'Build REST APIs with Symfony2',
                new DateTime("2014-03-08 16:10"),
                'Learn by example how to build a REST API using Symfony 2 ...',
                new Speaker('Eduardo Oliveira', 'bio', '...')
            )
        )
    );
}
```

# Result


```
{"talks": [
    {
        "title": "Build REST APIs with Symfony2",
        "start_date": "2014-03-08T16:10:00+0000",
        "abstract": "Learn by example how to build a REST API using Symfony 2 ...",
        "speaker": {
            "name": "Eduardo Oliveira",
            "bio": "bio",
            "why_me": "..."
        }
    }
]}
```

# Serializer configuration

```
ESO\RESTDemoBundle\Talks:
    exclusion_policy: ALL
    properties:
        talks:
            expose: true
            serialized_name: items
```

```
ESO\RESTDemoBundle\Talk:
    exclusion_policy: ALL
    properties:
        title:
            expose: true
        startDate:
            expose: true
            type: DateTime<'Y-m-d H:i'>
        abstract:
            expose: true
        speaker:
            expose: true
```

```
ESO\RESTDemoBundle\Speaker:
    exclusion_policy: ALL
    properties:
        name:
            expose: true
        bio:
            expose: true
            serialized_name: biography
```

# Result


```
{"items": [
    {
        "title": "Build REST APIs with Symfony2",
        "start_date": "2014-03-08 16:10",
        "abstract": "Learn by example how to build a REST API using Symfony 2 ...",
        "speaker": {
            "name": "Eduardo Oliveira",
            "biography": "bio"
        }
    }
]}
```

# Documentation

# API documentation is important

# NelmioApiDocBundle

# What does NelmioApiDocBundle do?

- Allows to generate documentation for APIs
- Introspection to keep docs up to date
- Integrates with forms
- Integrates with other bundles annotations
- Sandbox

# Setup

```
"nelmio/api-doc-bundle": "2.4.*"
```

```
// AppKernel.php
new Nelmio\ApiDocBundle\NelmioApiDocBundle(),
```

```
# config.yml
nelmio_api_doc: ~
```

```
# routing.yml
NelmioApiDocBundle:
    resource: "@NelmioApiDocBundle/Resources/config/routing.yml"
    prefix:   /api/doc
```

# Usage

```
* @ApiDoc(
*     resource=true,
*     description="Returns all the talks",
*     output="ESO\RESTDemoBundle\Talks",
*     statusCodes={
*         200="Success, talks on body",
*     }
* )
*
```

# Result

/talks

GET /talks — Returns all the talks

**Documentation** Sandbox

## Documentation

Returns all talks for the SymfonyDay PT 2014.

## Return

| Parameter | Type | Versions | Description |
|---|---|---|---|
| items[] | array of objects (Talk) | * | |
| items[][title] | string | * | |
| items[][start_date] | DateTime | * | |
| items[][abstract] | string | * | |
| items[][speaker] | object (Speaker) | * | |
| items[][speaker][name] | string | * | |
| items[][speaker][biography] | string | * | |

## Status Codes

| Status Code | Description |
|---|---|
| 200 | Success, talks on body |

# Forms

# What does Symfony Forms do?

- Makes easier to deal with input data
- Integrates well with Symfony validator
- Together with entities can act as “contract” with the “world”

# Usage 1/2

```php
class SpeakerType extends AbstractType
{
    /**
     * @SuppressWarnings(PHPMD.UnusedFormalParameter)
     */
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder->add('name', 'text', array('description' => 'The name of speaker'))
                ->add('bio', 'text', array('description' => 'Speaker biography'))
                ->add('whyMe', 'text', array('description' => 'Speaker motives to be choose'));
    }

    public function setDefaultOptions(OptionsResolverInterface $resolver)
    {
        $resolver->setDefaults(
            array(
                'data_class' => 'ESO\RESTDemoBundle\Speaker',
                'csrf_protection' => false,
                'empty_data' => function (FormInterface $form) {
                        return new Speaker(
                            $form->get('name')->getData(),
                            $form->get('bio')->getData(),
                            $form->get('whyMe')->getData()
                        );
                    }
            )
        );
    }
```

# Usage 2/2

```yaml
services:
    esorest_demo.default_controller:
        class: ESO\RESTDemoBundle\Controller\DefaultController
        arguments:
            serializer: "@form.factory"

    esorest_demo.speaker_form:
        class: ESO\RESTDemoBundle\SpeakerType
        tags:
            - { name: form.type, alias: speaker_create }
```

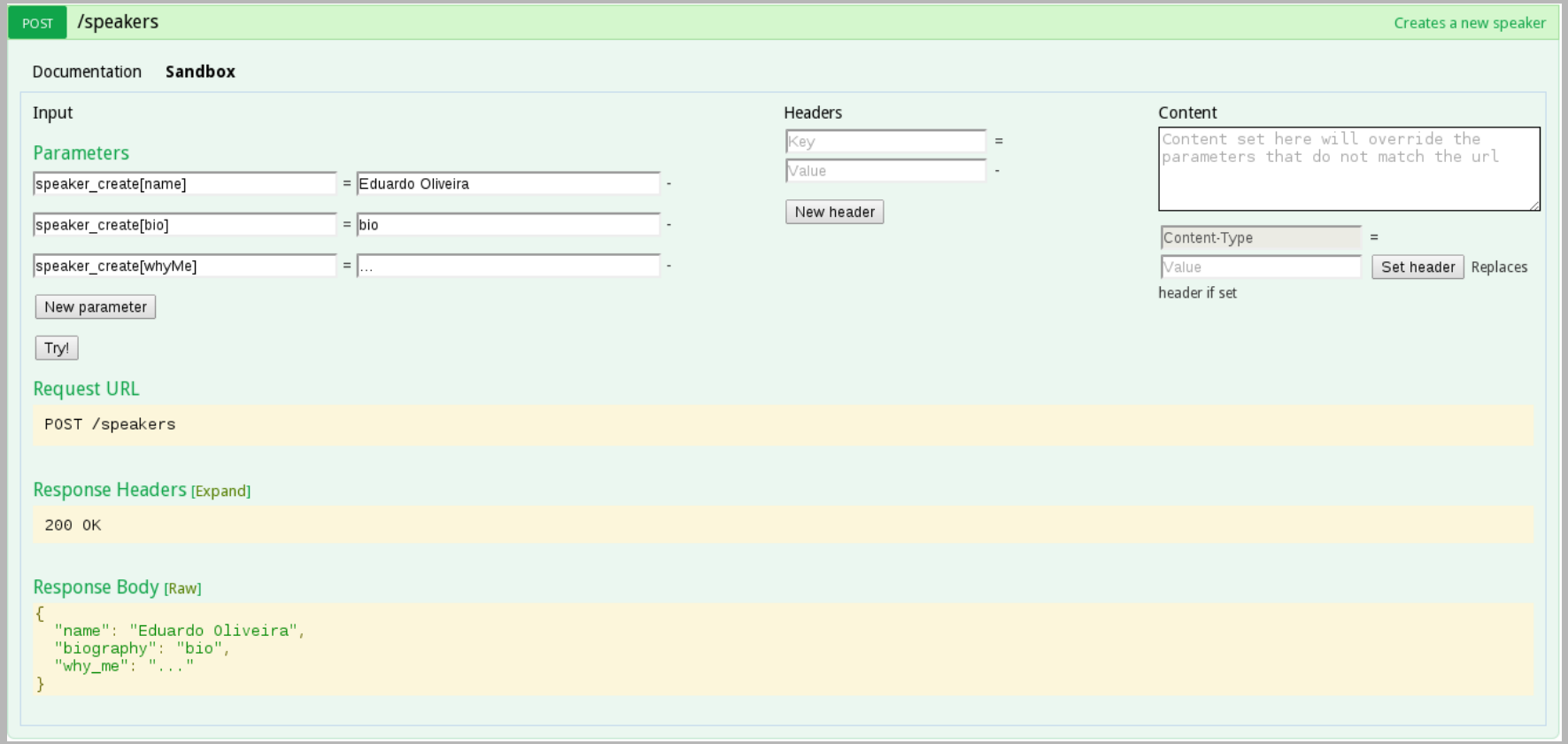
```php
public function createSpeakerAction(Request $request)
{
    $form = $this->formFactory->create('speaker_create');
    $form->handleRequest($request);

    if (!$form->isSubmitted()) {
        throw new BadRequestHttpException('Speaker create form not submitted');
    }

    if (!$form->isValid()) {
        return $form;
    }

    return $form->getData();
}
```

# Result


POST
/speakers
Creates a new speaker
Documentation
Sandbox
Input
Parameters
speaker_create[name]
Eduardo Oliveira
speaker_create[bio]
bio
speaker_create[whyMe]
...
New parameter
Try!
Headers
Key
Value
New header
Content
Content set here will override the parameters that do not match the url
Content-Type
Value
Set header
Replaces header if set
Request URL
POST /speakers
Response Headers [Expand]
200 OK
Response Body [Raw]
{
"name": "Eduardo Oliveira",
"biography": "bio",
"why_me": "..."
}

# Testing

# Some possibilities to test APIs

- Unit testing
  - Test lower layers with unit tests
- Functional testing
  - Test the whole API with Symfony client

# Functional test

```php
use Symfony\Bundle\FrameworkBundle\Test\WebTestCase;
use Symfony\Component\HttpFoundation\Response;

class DefaultControllerTest extends WebTestCase
{
    protected function assertJsonResponse(Response $response)
    {
        $this->assertTrue(
            $response->headers->contains('Content-Type', 'application/json'),
            $response->headers
        );
        $this->assertJson($response->getContent());
    }

    public function testGetTalks()
    {
        $client = static::createClient();
        $client->request('GET', 'talks');

        /** @var Response $response */
        $response = $client->getResponse();

        $this->assertJsonResponse($response);
        $content = json_decode($response->getContent(), true);

        $this->assertEquals('Build REST APIs with Symfony2', $content['items'][0]['title']);
    }
}
```

# HTTP Clients

# Even if you don’t write APIs, probably you need to consume other APIs

# Buzz

“Buzz is a lightweight PHP 5.3 library for issuing HTTP requests.”

```php
use Buzz\Browser;

class DefaultControllerTest extends WebTestCase
{
    public function testTemp()
    {
        $browser = new Browser();
        $response = $browser->get('http://restdemo.dev/app_dev.php/talks');

        echo $response->getContent();
    }
```

# Guzzle

“Guzzle is a PHP HTTP client and framework for building RESTful web service clients.”

```php
use Guzzle\Http\Client as GuzzleHttpClient;

class DefaultControllerTest extends WebTestCase
{
    public function testTemp()
    {
        $client = new GuzzleHttpClient('http://restdemo.dev/app_dev.php/');
        $response = $client->get('talks')->send();

        echo $response->getBody();
    }
```

# Guzzle Service Description

“Guzzle allows you to serialize HTTP requests and parse HTTP responses using a DSL called a service descriptions. Service descriptions define web service APIs by documenting each operation …

Guzzle's service descriptions are heavily inspired by Swagger.”

# Usage 1/2

```php
use Guzzle\Service\Client as GuzzleClient;
use Guzzle\Service\Description\ServiceDescription as GuzzleServiceDescription;

class DefaultControllerTest extends WebTestCase
{
    protected function getClient()
    {
        return GuzzleClient::factory(
            array(
                'scheme' => 'http',
                'host' => 'restdemo.dev',
                'basePath' => '/app_dev.php',
                'request.options' => array('headers' => array('Accept' => 'application/json')
                )
            )
        );
    }
```

# Usage 2/2

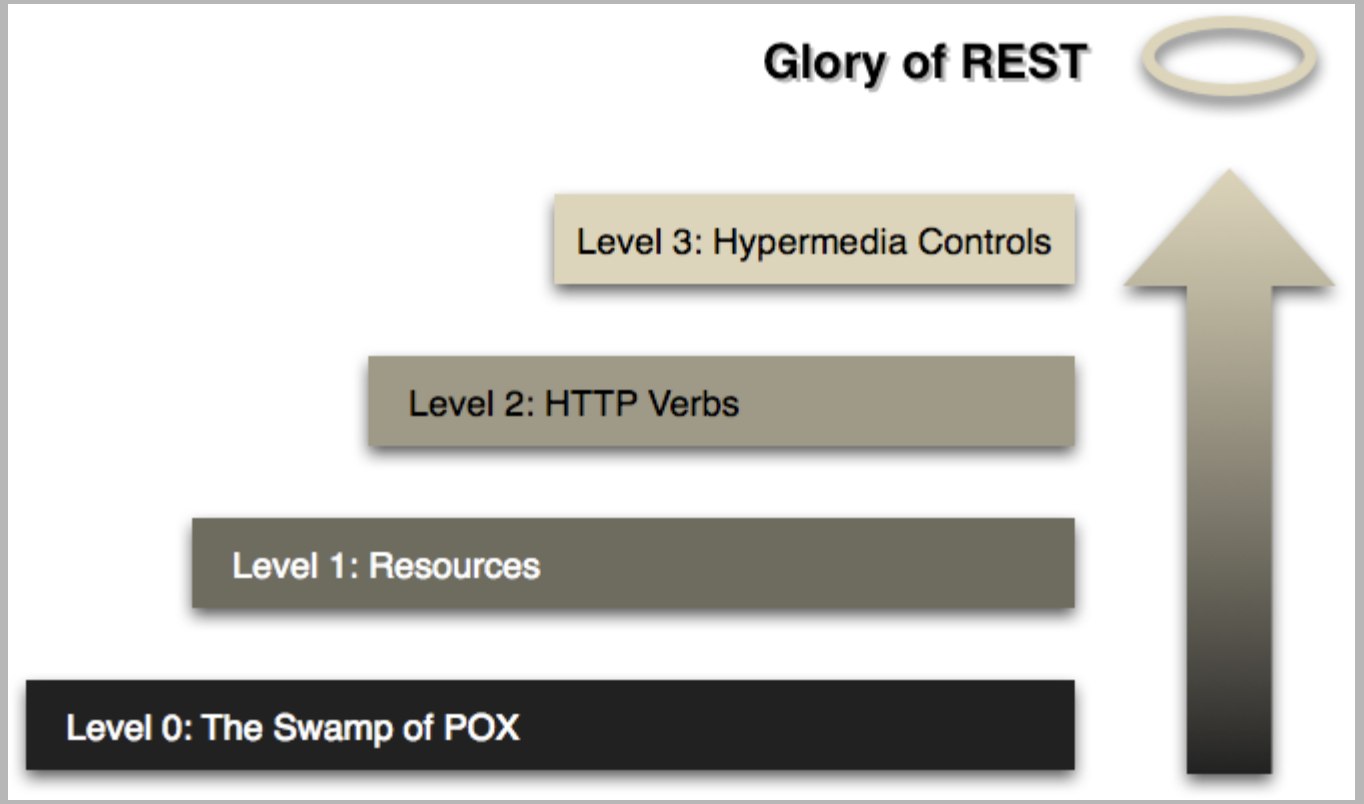
```php
protected function setDescription(GuzzleClient $client)
{
    $client->setDescription(
        new GuzzleServiceDescription(
            array(
                'baseUrl' => '{scheme}://{host}{+basePath}/',
                'operations' => array(
                    'getTalks' => array(
                        'httpMethod' => 'GET',
                        'uri' => 'talks',
                        'parameters' => array(
                            'limit' => array('location' => 'query', 'type' => 'integer')
                        )
                    )
                )
            )
        )
    );
}

public function testTemp()
{
    $client = $this->getClient();
    $this->setDescription($client);

    print_r($client->getCommand('getTalks', array('limit' => 5))->execute());
}
```

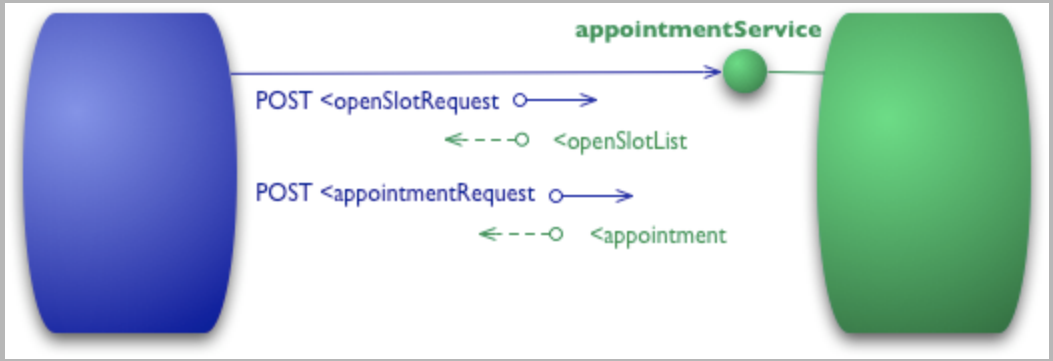
# API Design

# We also need to know how to properly design REST APIs

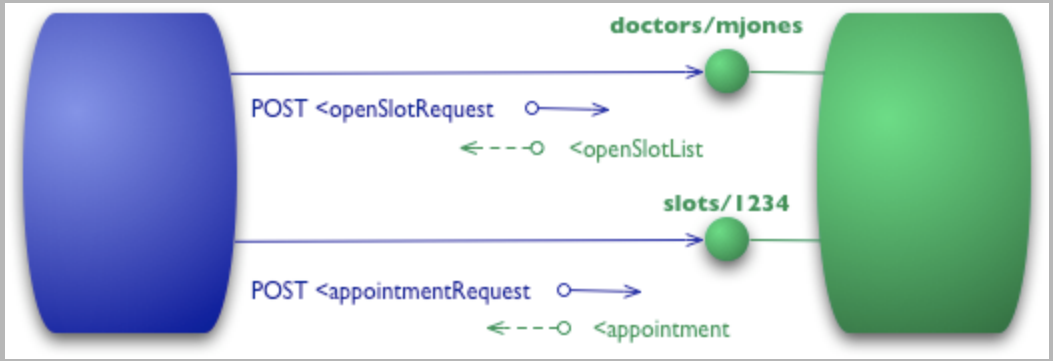



http://martinfowler.com/articles/richardsonMaturityModel.html

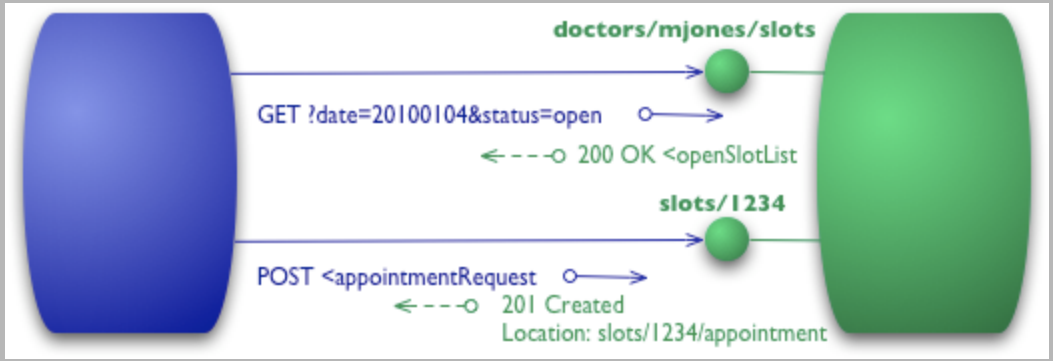
# Level 0 - tunneling mechanism



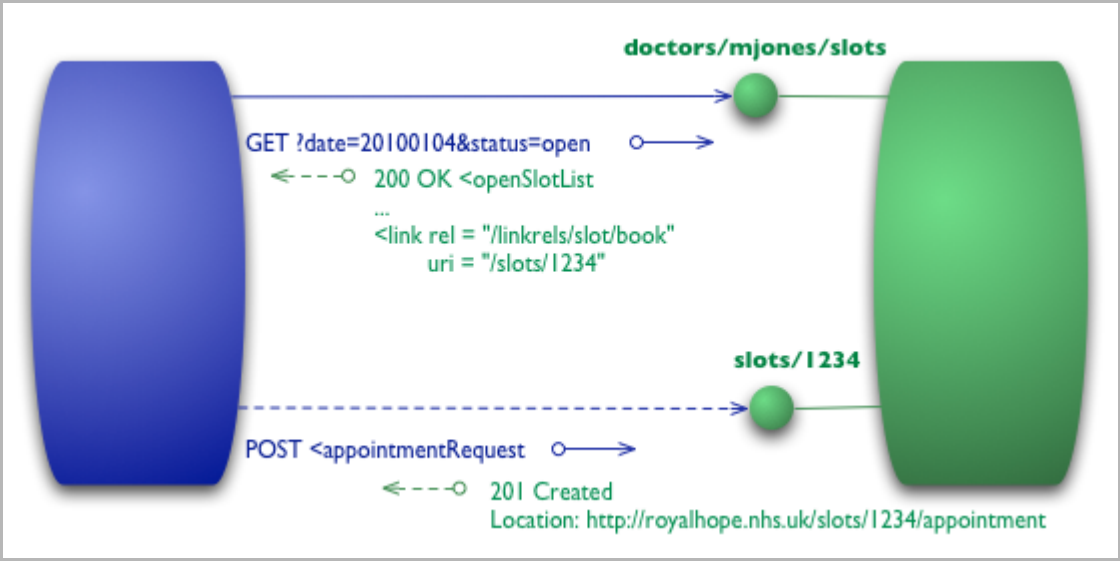
# Level 1 - Resources



# Level 2 - HTTP Verbs



# Level 3 - Hypermedia Controls



# In practice

# We need two URIs per resource

# Verbs are bad
# Nouns are good

# Concrete naming is good

# Plural or singular?
# Plural

# Collection

/speakers

# Element

/speakers/1

POST

GET

PUT

DELETE

...

| Resource | POST (create) | GET (read) | PUT (update) | DELETE (delete) |
|---|---|---|---|---|
| /speakers | new speaker | list speaker | bulk update | delete all speakers |
| /speakers/1 | error | show speaker | replace speaker | delete speaker |

# Errors

HTTP status code
Human readable message

# Format

/speakers.json
/speakers?_format=json
Accept: application/json

# Format

- URIs identify resources
- URIs are format independent

# Additional Resources 1/2

- Blogs
  - REST APIs with Symfony2: The Right Way
    - http://williamdurand.fr/2012/08/02/rest-apis-with-symfony2-the-right-way/
  - Symfony2 REST API: the best way
    - http://welcometothebundle.com/symfony2-rest-api-the-best-2013-way/
    - …

# Additional Resources 2/3

- Presentations
  - https://speakerdeck.com/gordalina/rest-apis-made-easy-with-symfony2
  - https://speakerdeck.com/willdurand/build-awesome-rest-apis-with-symfony2
  - ...

# Additional Resources 3/3

- Bundles/libs
  - https://github.com/FriendsOfSymfony/FOSRestBundle
  - https://github.com/nelmio/NelmioApiDocBundle
  - https://github.com/willdurand/BazingaHateoasBundle
  - https://github.com/willdurand/Negotiation
  - https://github.com/guzzle/guzzle

# Thank you