

REST vs GraphQL

Illustrated examples using API Platform





Kévin Dunglas

- Founder of Les-Tilleuls.coop
- **Symfony Core Team**
(PropertyInfo, WebLink, Serializer, autowiring, PSR-7 Bridge...)
- **API Platform** creator

  @dunglas





Les-Tilleuls.coop

- Self-managed company since 2011
- 100% owned by employees
- 25 people, 97% growth in 2016
- Hiring in Paris and Lille: jobs@les-tilleuls.coop



React



mongoDB.



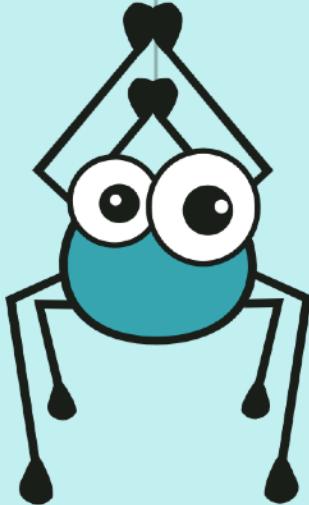
elastic



The API Platform framework



Symfony Live
PARIS 2018
29-30 MARS



API PLATFORM

REST and GraphQL framework to build modern
API-driven projects



Download



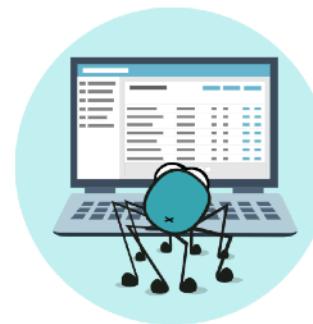
Get started



API Component



Schema Gen
Component

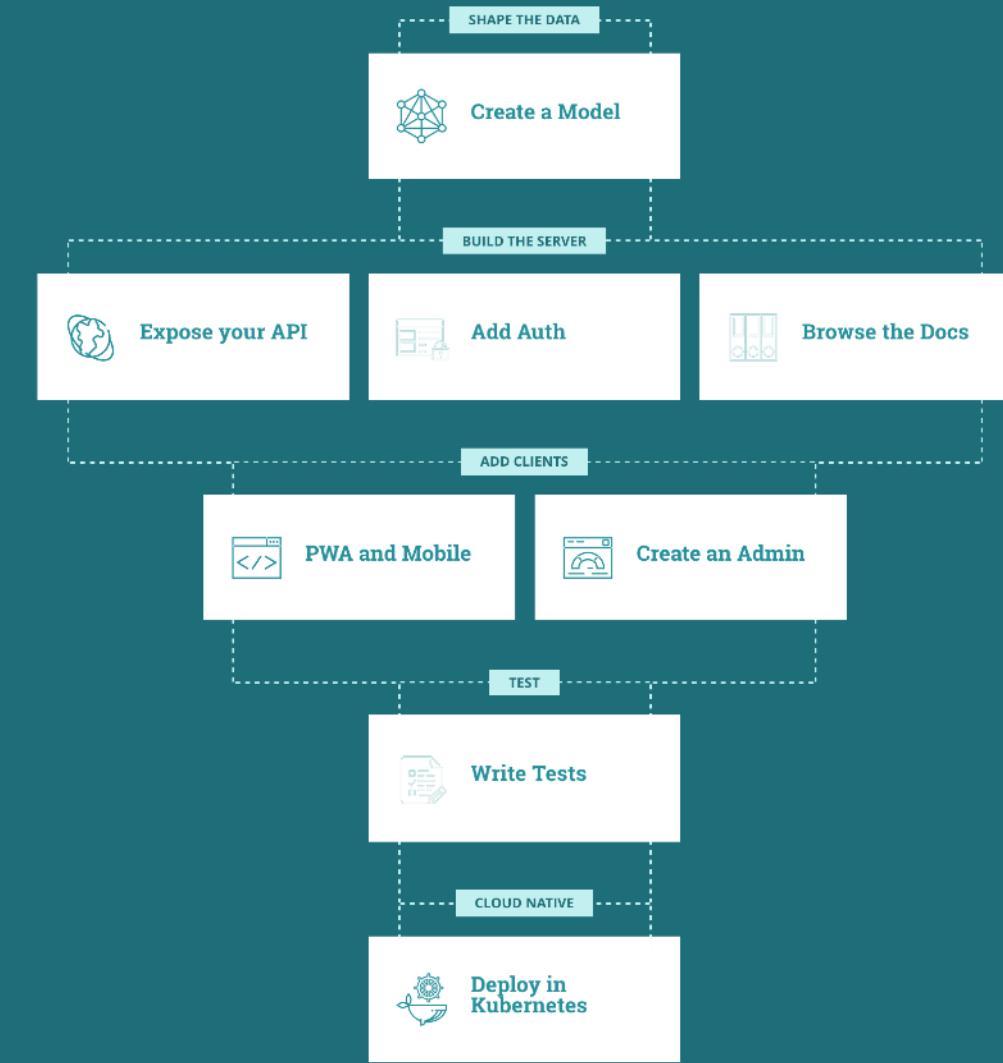


Admin
Component



Client Gen
Component

React
or
Vue.js





Supported Formats

- JSON-LD+Hydra (default)
- OpenAPI (default)
- GraphQL
- JSONAPI
- HAL + API Problem
- YAML, JSON, CSV, XML

GraphQL Query support #1358

[Edit](#)**Merged**dunglas merged 11 commits into [api-platform:master](#) from [alanpoulain:feature/graphql](#) on 6 Oct 2017

Conversation 117

Commits 11

Files changed 54

+4,457 -26 

alanpoulain commented on 11 Sep 2017 • edited

Contributor



Reviewers



dunglas



Implement the Relay specification for mutations ✓

9

#1597 by dunglas was merged on 22 Dec 2017 • Approved

Implement the Relay Global Object Identification Specification ✓

6

#1585 by dunglas was merged on 21 Dec 2017

Use IRIs as GraphQL's ids and improve perf ✓

18

#1569 by dunglas was merged on 20 Dec 2017 

Install



Symfony Live
PARIS 2018
29-30 MARS

The Distribution

api-platform / api-platform

Code Issues 122 Pull requests 1 Projects 0 Insights

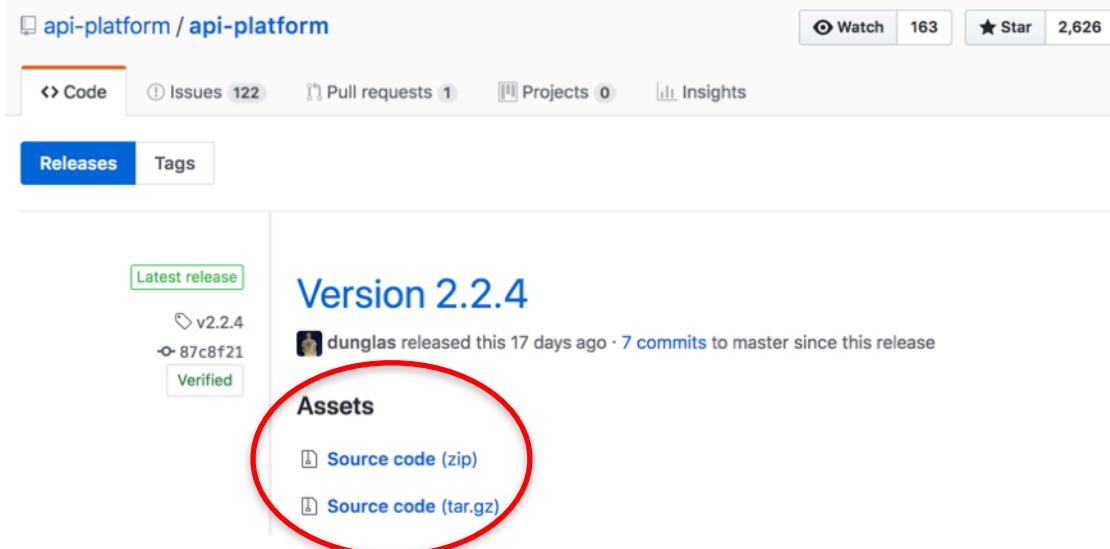
Releases Tags

Latest release v2.2.4 · 87c8f21 · Verified

dunglas released this 17 days ago · 7 commits to master since this release

Assets

Source code (zip) Source code (tar.gz)



\$ docker-compose up

Welcome to API Platform

Non sécurisé | https://localhost

made with ❤ by Les-Tilleuls.coop



API PLATFORM

Welcome to API Platform!

This container will host your **Progressive Web App** ([HTTP](#)). Learn how to create your first API and generate a PWA:

[GET STARTED →](#)

AVAILABLE SERVICES:



API
HTTP or HTTPS

CACHED API
HTTP or HTTPS



ADMIN
HTTP or HTTPS

NEED HELP?

☰ Hello API Platform - API Platfo ✎ Kévin

⚠ Non sécurisé <https://localhost:8443>

API PLATFORM

Hello API Platform 1.0.0

[Base URL: /]

Greeting ▾

GET	/greetings	Retrieves the collection of Greeting resources.
POST	/greetings	Creates a Greeting resource.
GET	/greetings/{id}	Retrieves a Greeting resource.
DELETE	/greetings/{id}	Removes the Greeting resource.
PUT	/greetings/{id}	Replaces the Greeting resource.

Models

Greeting ▾ {
description: This is a dummy entity. Remove it!

name*	string
A nice person	



API Platform Admin

Non sécurisé | https://localhost:444/#/greetings

Hello API Platform

Greetings

Greetings List

ID	NAME	SHOW	EDIT
/greetings/1	Hello SymfonyLive Paris!	 SHOW	 EDIT

1-1 of 1



What's Inside

- All components: API, Schema gen, Admin, Client gen
- PHP and JS Docker images, **Docker Compose env**
- Postgres server
- Varnish, invalidation-based cache mechanism enabled
- **HTTPS** and **HTTP/2** dev reverse proxy
- Helm chart to deploy in **Kubernetes** clusters (GKE...)



Alternative: use Flex

```
$ composer create-project \
symfony/skeleton my-api
$ cd my-api
$ composer req api
$ php -S 127.0.0.1:8000 -t public
```



API Platform

localhost:8000/api

0.0.0

[Base URL: /]

No operations defined in spec!

Available formats: [jsonld](#) [json](#) [html](#)

- Only the API component
- Minimalist

Enabling the GraphQL Support



Symfony Live
PARIS 2018
29-30 MARS



Install GraphQL support

```
$ composer req \
webonyx/graphql-php
```

← → C⚠ Non sécurisé | <https://localhost:8443/graphql?query=%7B%0A...> ☆

GraphQL

PrettifyHistory< Docs

```
1 {  
2   greeting (id: "/greetings/1") {  
3     name  
4   }  
5 }  
6 }
```

```
1 {  
2   data: {  
3     greeting: {  
4       name: "Hello"  
5     }  
6   }  
7 }
```

QUERY VARIABLES

[←](#) [→](#) [C](#)⚠ Non sécurisé | <https://localhost:8443/graphql?query=%7B%0A...> [☆](#)

GraphQL

[Prettify](#)[History](#)[Docs](#)

```
1 {  
2   fr: greeting(id: "/greetings/1") {  
3     name  
4   }  
5  
6   de: greeting(id: "/greetings/3") {  
7     name  
8   }  
9 }  
10 }
```

```
{  
  "data": {  
    "fr": {  
      "name": "Hello"  
    },  
    "de": {  
      "name": "Guten tag"  
    }  
  }  
}
```



GraphQL?



Symfony Live
PARIS 2018
29-30 MARS



HOW TO GRAPHQL



Search tutorials...

GraphQL is the better REST

Javascript > May 17, 2017 > By Michael Paris

REST 2.0 Is Here and Its Name Is GraphQL

[GraphQL](#)[REST \(software architectural style\)](#)

Is GraphQL a REST killer?

Short answer:

GraphQL will do to REST what JSON did to XML.



Samer Buna

Follow

Author for Pluralsight and Lynda and curator of jsComplete.com

Jul 24, 2017 · 16 min read

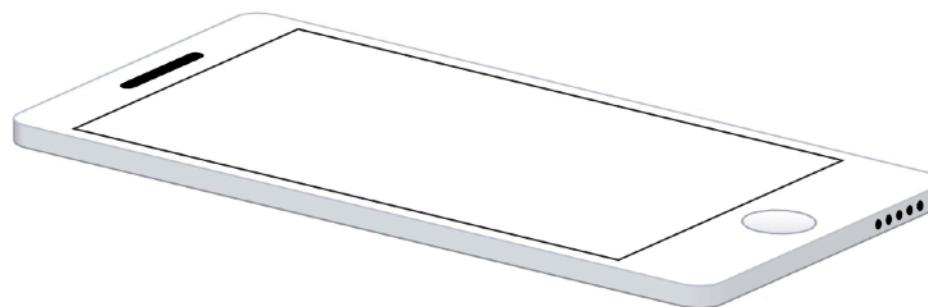
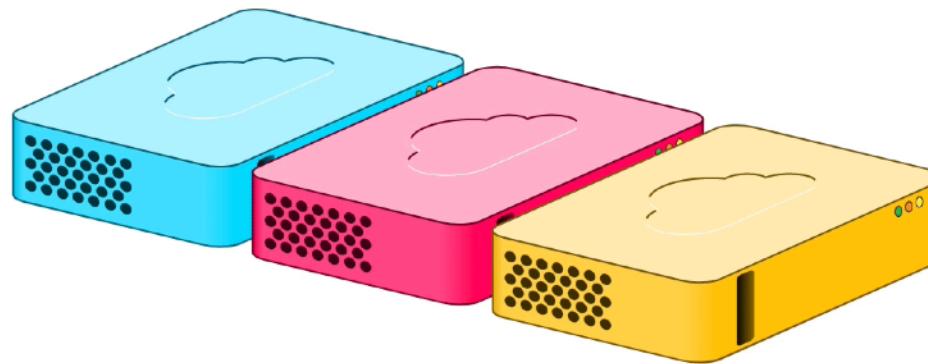
REST APIs are REST-in-Peace APIs. Long Live GraphQL.

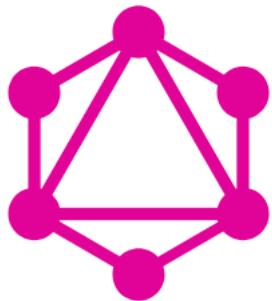


GraphQL

- A data query language
- Specifically designed for web APIs...
- ...as an alternative to REST
- Open **format** with a formal specification
- Perfect as an API gateway (aggregate data from services)
- Released by **Facebook** in 2015
- Service oriented







GraphQL

Describe your data

```
type Project {  
  name: String  
  tagline: String  
  contributors: [User]  
}
```

Ask for what you want

```
{  
  project(name: "GraphQL") {  
    tagline  
  }  
}
```

Get predictable results

```
{  
  "project": {  
    "tagline": "A query language for APIs"  
  }  
}
```



Main Features

- queries
- mutations
- subscriptions (✗ not supported in PHP)
- type system, introspection
- errors (limited)
- multiple queries in one request (parallelization)



Server Specification

- De-facto standard for GraphQL servers
- Created by **Facebook** for the Relay library
- **Node** (super type, like *Object* in Java)
- **Object Identification**: unique identifier across types
- **Connections**: structure for pagination
- **Mutations** format



Example: the GitHub API



Symfony Live
PARIS 2018
29-30 MARS

```
1▼ query ContributionsOfEmployees($company: String!) {  
2▼   organization(login: $company) {  
3    id  
4    name  
5    members(first: 100) {  
6      edges {  
7        node {  
8          id  
9          login  
10         name  
11        contributedRepositories(first: 100, privacy: PUBLIC) {  
12          edges {  
13            node {  
14              id  
15              name  
16            }  
17          }  
18        }  
19      }  
20    }  
21  }  
22}  
23}
```

QUERY VARIABLES	
1	{
2	"company": "coopTilleuls"
3	}

```
{  
  "data": {  
    "organization": {  
      "id": "MDEy0k9yZ2FuaXphdGlvbjExNjExOTg=",  
      "name": "Les-Tilleuls.coop",  
      "members": {  
        "edges": [  
          {  
            "node": {  
              "id": "MDQ6VXNlcjU3MjI0",  
              "login": "dunglas",  
              "name": "Kévin Dunglas",  
              "contributedRepositories": {  
                "edges": [  
                  {  
                    "node": {  
                      "id": "MDEwOlJlcG9zaXRvcnk0NTgwNTg=",  
                      "name": "symfony"  
                    }  
                  },  
                  {  
                    "node": {  
                      "id": "MDEwOlJlcG9zaXRvcnk1MjE10DM=",  
                      "name": "symfony-docs"  
                    }  
                  }  
                ]  
              }  
            }  
          }  
        ]  
      }  
    }  
  }  
}
```

Defining REST



Symfony Live
PARIS 2018
29-30 MARS

REpresentational State Transfer

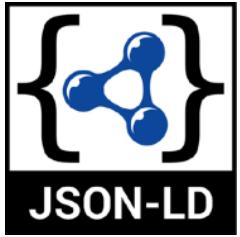
- An **architectural style** for web services
- Defined in Roy Fielding's thesis in 2000
- Leverage the **HTTP protocol**
- **True REST APIs are hypermedia (HATEOAS)**
- Resource oriented



W3C Standards



Symfony Live
PARIS 2018
29-30 MARS



JSON for Linked Data

- Map JSON properties to an **ontology**
- Perfect for **hypermedia APIs**
- **W3C standard** (2014), as HTML, CSS...
- Compatible with existing **semantic web** standards and tools (RDF, OWL, SPARQL, triple stores, Apache Jena...)



schema.org

- Global and open **vocabulary**
- Define **types**, properties and enumerations
- ~600 **types**: Person, Place, Organization, Event...
- Extensible
- Preferred encoding: **JSON-LD**
- W3C community group started by Google, Microsoft, Yahoo and Yandex





Hydra

- Format for interoperable, hypermedia APIs
- Built on top of JSON-LD, compatible with schema.org
- In-band API documentation
- Resources, properties, operations, collections, templated links (filters), pagination, errors
- Draft of a **potential** W3C specification (community group)



Example: Google Knowledge Graph API



Symfony Live
PARIS 2018
29-30 MARS

GET ▾

[https://kgsearch.googleapis.com/v1/entities:search?indent=true&query=Tim Berners-Lee&fields=@context,@type,itemListElement](https://kgsearch.googleapis.com/v1/entities:search?indent=true&query=Tim%20Berners-Lee&fields=@context,@type,itemListElement)

```
10  "@type": "ItemList",
11  "itemListElement": [
12  {
13      "@type": "EntitySearchResult",
14      "result": {
15          "@id": "kg:/m/07d5b",
16          "name": "Tim Berners-Lee",
17          "@type": [
18              "Thing",
19              "Person"
20          ],
21          "description": "Computer scientist",
22          "image": {
23              "contentUrl": "http://t2.gstatic.com/images?q=tbn:AN
24              "url": "https://commons.wikimedia.org/wiki/File:Sir_
25          },
26          "detailedDescription": {
27              "articleBody": "Sir Timothy John Berners-Lee OM KBE
28              "url": "https://en.wikipedia.org/wiki/Tim_Berners-Le
29              "license": "https://en.wikipedia.org/wiki/Wikipedia:
30          },
31          "url": "http://www.w3.org/People/Berners-Lee/"
32      },
33      "resultScore": 1259.341187
34 }
```

Alternative Formats



Symfony Live
PARIS 2018
29-30 MARS

Alternative Formats

{ json:api }

A SPECIFICATION FOR BUILDING APIs IN JSON



OpenAPI (formerly Swagger)



{ json:api }

A SPECIFICATION FOR BUILDING APIs IN JSON

- Community format to build **hypermedia** web APIs
- Simpler and easier to implement than Hydra+JSON-LD...
- ...but less powerful (no RDF nor [schema.org](#) compat)
- Pagination, filtering, sparse fieldsets, compound docs, errors...
- Limited type system, no introspection
- Experimental extensions





OpenAPI (formerly Swagger)

- API **doc** format (describe RESTful contracts)
- Path, operations, content negotiation...
- Type system built with **JSON Schema**
- Request/response formats **up to you**: JSON, XML...
- Can *partially* describe JSON-LD and JSONAPI



Schema and Types



Symfony Live
PARIS 2018
29-30 MARS

```
10  /**
11   * An Author.
12   *
13   * @ApiResource(iri="http://schema.org/Person")
14   * @ORM\Entity
15   */
16  class Author
17  {
18      /**
19       * @var string An UUID identifying this author.
20       *
21       * @ORM\Id
22       * @ORM\Column(type="guid")
23       * @Assert\NotBlank
24       * @Assert\Uuid
25       */
26      public $id;
27
28      /**
29       * @var string The author's name
30       *
31       * @ApiProperty(iri="http://schema.org/name")
32       * @ORM\Column(type="string")
33       * @Assert\NotBlank
34       */
35      public $name;
36  }
```



```
10  /**
11  * Book
12  *
13  * @ApiResource(iri="http://schema.org/Book")
14  * @ORM\Entity
15  */
16  class Book
17  {
18  /**
19  * @var string An UUID identifying this book.
20  *
21  * @ORM\Id
22  * @ORM\Column(type="guid")
23  * @Assert\NotBlank
24  * @Assert\Uuid
25  */
26  public $id;
27
28 /**
29 * @var string The Book's title.
30 *
31 * @ApiProperty(iri="http://schema.org/name")
32 * @ORM\Column(type="string")
33 * @Assert\NotBlank
34 */
35 public $title;
36
37 /**
38 * @var Author The book's author.
39 *
40 * @ApiProperty(iri="http://schema.org/author")
41 * @ORM\ManyToOne(targetEntity="Author")
42 * @Assert\NotNull
43 */
44 public $author;
45 }
```





```
1 {  
2   __schema {  
3     queryType {  
4       name  
5       kind  
6       fields {  
7         name  
8         type {  
9           name  
10          description  
11        }  
12      }  
13    }  
14  }  
15 }  
16  
  
1 {  
2   "data": {  
3     "__schema": {  
4       "queryType": {  
5         "name": "Query",  
6         "kind": "OBJECT",  
7         "fields": [  
8           {  
9             "name": "node",  
10            "type": {  
11              "name": "Node",  
12              "description": "A node, according to the Relay specification."  
13            }  
14          },  
15          {  
16            "name": "book",  
17            "type": {  
18              "name": "Book",  
19              "description": "Book"  
20            }  
21          },  
22          {  
23            "name": "books",  
24            "type": {  
25              "name": "BookConnection",  
26              "description": "Connection for Book."  
27            }  
28          },  
29          {  
30            "name": "author",  
31          }  
32        ]  
33      }  
34    }  
35  }  
36 }
```

GraphQL Usage

Documentation Explorer

 Search Schema...

A GraphQL schema provides a root type for each kind of operation.

ROOT TYPES

query: Query

mutation: Mutation

< Query

Book

🔍 Search Book...

Book

IMPLEMENTS

Node

FIELDS

id: ID!

_id: String!

An UUID identifying this book.

title: String!

The Book's title.

author: Author

The book's author.



GraphQL Schema

- Describe the data that can be queried
- Expose operations (queries, mutations)
- Make the API **auto-discoverable** (in-band docs)
- Allow to develop **smart clients** (API agnostic)

Hydra Documentation

GET ▾

https://localhost:8443/

Link → <<https://localhost:8443/docs.jsonld>>; rel="http://www.w3.org/ns/hydra/core#apiDocumentation"

```
1 {  
2   "@context": "/contexts/Entrypoint",  
3   "@id": "/",  
4   "@type": "Entrypoint",  
5   "book": "/books",  
6   "author": "/authors"  
7 }
```

GET

https://localhost:8443/docs

```
"hydra:supportedClass": [
{
  "@id": "http://schema.org/Book",
  "@type": "hydra:Class",
  "rdfs:label": "Book",
  "hydra:title": "Book",
  "hydra:supportedProperty": [
    {
      "@id": "http://schema.org/author",
      "@type": "hydra:Link",
      "rdfs:label": "author",
      "domain": "http://schema.org/Book",
      "owl:maxCardinality": 1,
      "range": "http://schema.org/Person"
    },
    {
      "hydra:title": "author",
      "hydra:required": true,
      "hydra:readable": true,
      "hydra:writable": true,
      "hydra:description": "The book's author."
    }
  ],
  "hydra:isContainer": true
}]]
```

```
"hydra:supportedOperation": [
{
  "@type": [
    "hydra:Operation",
    "schema:FindAction"
  ],
  "hydra:method": "GET",
  "hydra:title": "Retrieves Book resource.",
  "rdfs:label": "Retrieves Book resource.",
  "returns": "http://schema.org/Book"
},
{
  "@type": [
    "hydra:Operation",
    "schema:DeleteAction"
  ],
  "hydra:method": "DELETE",
  "hydra:title": "Deletes the Book resource.",
  "rdfs:label": "Deletes the Book resource.",
  "returns": "owl:Nothing"
},
{
  "@type": [
    "hydra:Operation",
    "schema:ReplaceAction"
  ],
  "expects": "http://schema.org/Book",
  "hydra:title": "Updates Book resource."
}]]
```



JSON-LD/Hydra Vocab

- Capabilities similar to GraphQL's Schema
- Leverage RDF and OWL (😊😔)
- Describe API's types and supported REST operations
- Unlike GraphQL:
 - API interoperability at web scale (hypermedia)
 - ability to use open vocabularies (schema.org)

Hello API Platform

Books

Authors

Book #/books/274f584d-8866-4510-8eb4-f23a9bc1e3f5

SHOW

LIST

DELETE

REFRESH

Id *

Title *

Mikhail Bakunin

Karl Marx

SAVE

Alternative: OpenAPI

```
Author ▼ {  
    description: An Author.  
  
    id*          string  
    An UUID identifying this author.  
  
    name*        string  
    The author's name  
}
```

```
Book ▼ {  
    description: Book  
  
    id*          string  
    An UUID identifying this book.  
  
    title*       string  
    The Book's title.  
  
    author*      string  
    The book's author.  
}
```

Data fetching: REST



Symfony Live
PARIS 2018
29-30 MARS

GET ▼

https://localhost:8443/books/274f584d-8866-4510-8eb4-f23a9bc1e3f5

1 ▾ {

```
2   "@context": "/contexts/Book",
3   "@id": "/books/274f584d-8866-4510-8eb4-f23a9bc1e3f5",
4   "@type": "http://schema.org/Book",
5   "id": "274f584d-8866-4510-8eb4-f23a9bc1e3f5",
6   "title": "The Paris Commune and the Idea of the State",
7   "author": "/authors/1e820f57-5be9-4eb2-99e7-94a9b2c53053"
```

8 ▾ }

GET ▼

https://localhost:8443/authors/1e820f57-5be9-4eb2-99e7-94a9b2c53053

new request

1 ▾ {

```
2   "@context": "/contexts/Author",
3   "@id": "/authors/1e820f57-5be9-4eb2-99e7-94a9b2c53053",
4   "@type": "http://schema.org/Person",
5   "id": "1e820f57-5be9-4eb2-99e7-94a9b2c53053",
6   "name": "Mikhail Bakunin"
7 }
```



Data fetching: GraphQL



Symfony Live
PARIS 2018
29-30 MARS



```
{  
  book(id: "/books/274f584d-8866-4510-8eb4-f23a9bc1e3f5") {  
    title  
    author {  
      name  
    }  
  }  
}
```

+ REST compatibility
(API Platform specific)

```
{  
  "data": {  
    "book": {  
      "title": "The Paris Commune and the Idea of the State",  
      "author": {  
        "name": "Mikhail Bakunin"  
      }  
    }  
  }  
}
```

GraphQL

- Declarative and **elegant** query language
- **The client** tells exactly what it needs
- **1 query** (vs N with REST) to retrieve **all** required data
- Solve **under-fetching** and **over-fetching** problem
- Require a **specific query parser**, but returns JSON
- No response/request parity: you cannot post a previously retrieved then updated document



Data fetching: Compound Docs



Symfony Live
PARIS 2018
29-30 MARS

```
/**  
 * @ApiResource(attributes={  
 *     "normalization_context"={  
 *         "groups"={"book_read"}  
 *     }  
 * })  
 */  
class Book  
{  
    public $id;  
  
    /**  
     * @Groups("book_read")  
     */  
    public $title;  
  
    /**  
     * @Groups("book_read")  
     */  
    public $author;  
}
```

```
/** An Author. ....*/  
class Author  
{  
    public $id;  
  
    /**  
     * @Groups("book_read")  
     */  
    public $name;  
}
```

GET ▾

<https://localhost:8443/books/274f584d-8866-4510-8eb4-f23a9bc1e3f5>

```
1 {  
2   "@context": "/contexts/Book",  
3   "@id": "/books/274f584d-8866-4510-8eb4-f23a9bc1e3f5",  
4   "@type": "http://schema.org/Book",  
5   "title": "The Paris Commune and the Idea of the State",  
6   "author": {  
7     "@id": "/authors/1e820f57-5be9-4eb2-99e7-94a9b2c53053",  
8     "@type": "http://schema.org/Person",  
9     "name": "Mikhail Bakunin"  
10    }  
11 }
```

- ▶ Solve the N requests problem
- ▶ The server is still rigid: the client's request must be known
- ▶ Don't solve the under/over-fetching problem

Data fetching: Sparse Fieldsets



Symfony Live
PARIS 2018
29-30 MARS

```
12 /**
13 * @ApiFilter(PropertyFilter::class)
14 *
15 * @ApiResource(iri="http://schema.org/Book")
16 * @ORM\Entity
17 */
18 class Book
19 {
20     /** @var string An UUID identifying this book. ...*/
21     public $id;
22
23     /** @var string The Book's title. ...*/
24     public $title;
25
26     /** @var Author The book's author. ...*/
27     public $author;
28 }
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46 }
```

GET

https://localhost:8443/books/274f584d-8866-4510-8eb4-f23a9bc1e3f5?properties[]="title&properties[]="author&properties[author][]"=name

Key	Value
properties[]	title
properties[]	author
properties[author][]	<pre>1 "title": "The Paris Commune and the Idea of the State", 2 "author": { 3 "@id": "/authors/1e820f57-5be9-4eb2-99e7-94a9b2c53053", 4 "@type": "http://schema.org/Person", 5 "name": "Mikhail Bakunin" 6 } 7 8 9 10 }</pre>

- Solve the N requests problem
- Solve the under/over fetching problem
- Can be subject to problems with RDMS (more on this later)
- Less explicit and elegant than GraphQL but...

[http_build_query](#)

PHP

(PHP 5, PHP 7)

`http_build_query` — Generate URL-encoded query string

Description

```
string http_build_query ( mixed $query_data [, string $numeric_prefix [, string $arg_separator  
[, int $enc_type = PHP_QUERY_RFC1738 ]]] )
```

Generates a URL-encoded query string from the associative (or indexed) array provided.



URLSearchParams

JavaScript

Jump to: [Syntax](#) [Example](#) [Specifications](#) [Browser compatibility](#)

[Web technology for developers](#) > [Web APIs](#) >
[URL](#) > [URLSearchParams](#)

The `searchParams` property of the `URL` interface returns a `URLSearchParams` object allowing access to the GET query arguments contained in the URL.

Filtering



Symfony Live
PARIS 2018
29-30 MARS

```
12  /* A Book. ...*/
13  class Book
14  {
15      /* @var string An UUID identifying this book. ...*/
16      public $id;
17
18      /*
19      * @ApiFilter(SearchFilter::class, strategy="ipartial")
20      */
21      public $title;
22
23      /* @var Author The book's author. ...*/
24      public $author;
25  }
```

GET ▼

https://localhost:8443/books?title=paris

Key Value

title paris

3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18

```
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
  "concrete": "concrete-book",
  "@id": "/books",
  "@type": "hydra:Collection",
  "hydra:member": [
    {
      "@id": "/books/274f584d-8866-4510-8eb4-f23a9bc1e3f5",
      "@type": "http://schema.org/Book",
      "id": "274f584d-8866-4510-8eb4-f23a9bc1e3f5",
      "title": "The Paris Commune and the Idea of the State",
      "author": "/authors/1e820f57-5be9-4eb2-99e7-94a9b2c53053"
    }
  ],
  "hydra:totalItems": 1,
  "hydra:view": {
    "@id": "/books?title=paris",
    "@type": "hydra:PartialCollectionView"
  },
  "concrete": "concrete-book"
}
```

REST

```
14 "hydra:totalItems": 1,  
15 "hydra:view": {  
16     "@id": "/books?title=paris",  
17     "@type": "hydra:PartialCollectionView"  
18 },  
19 "hydra:search": {  
20     "@type": "hydra:IriTemplate",  
21     "hydra:template": "/books{?title}",  
22     "hydra:variableRepresentation": "BasicRepresentation",  
23     "hydra:mapping": [  
24         {  
25             "@type": "IriTemplateMapping",  
26             "variable": "title",  
27             "property": "title",  
28             "required": false  
29         }  
30     ]  
31 }
```

GraphQL

```
1 {  
2   books(title: "commune") {  
3     edges {  
4       node {  
5         id  
6         author {  
7           name  
8         }  
9       }  
10      }  
11    }  
12  }  
13 }
```

```
{  
  "data": {  
    "books": {  
      "edges": [  
        {  
          "node": {  
            "id": "/books/274f584d-8866-4510-8eb4-f23a9bc1e3f5",  
            "author": {  
              "name": "Mikhail Bakunin"  
            }  
          }  
        }  
      ]  
    }  
  }  
}
```

Alternative:



OData
Open Data Protocol

```
People?$top=2 & $select=FirstName, LastName & $filter=Trips/any(d:d/Budget gt 30)
```

- Alternative REST format
- Describe an advanced query language
- Batch support
- Not implemented (yet) in API Platform



Sorting



Symfony Live
PARIS 2018
29-30 MARS

```
13  /**
14   * @ApiFilter(OrderFilter::class)
15  */
16 class Book
17 {
18     /** @var string An UUID identifying this book. ...*/
25     public $id;
26
27     /** @var string The Book's title. ...*/
35     public $title;
36
37     /** @var Author The book's author. ...*/
44     public $author;
45 }
```

GET

https://localhost:8443/books?order[title]=DESC

Key	Value
order[title]	DESC

REST

```
2     "@context": "/contexts/Book",
3     "@id": "/books",
4     "@type": "hydra:Collection",
5     "hydra:member": [
6         {
7             "@id": "/books/274f584d-8866-4510-8eb4-f23a9bc1e3f5",
8             "@type": "http://schema.org/Book",
9             "id": "274f584d-8866-4510-8eb4-f23a9bc1e3f5",
10            "title": "The Paris Commune and the Idea of the State",
11            "author": "/authors/1e820f57-5be9-4eb2-99e7-94a9b2c53053"
12        },
13        {}
14    ],
15    "hydra:totalItems": 2,
16    "hydra:view": {
17        "@id": "/books?order%5Btitle%5D=DESC",
18        "@type": "hydra:PartialCollectionView"
19    },
20    "hydra:search": {
21        "@type": "hydra:IriTemplate",
22        "hydra:template": "/books{?order[id],order[title]}",
23        "hydra:variableRepresentation": "BasicRepresentation",
24        "hydra:mapping": [
25            {
26                "@type": "IriTemplateMapping",
27                "variable": "order[id]",
28                "property": "id",
29                "required": false
30            },
31            {
32                "@type": "IriTemplateMapping",
33                "variable": "order[title]",
34                "property": "title",
35                "required": false
36            }
37        ]
38    }
39}
```

```
21 "hydra:totalItems": 2,  
22 "hydra:view": {  
23     "@id": "/books?order%5Btitle%5D=DESC",  
24     "@type": "hydra:PartialCollectionView"  
25 },  
26 "hydra:search": {  
27     "@type": "hydra:IriTemplate",  
28     "hydra:template": "/books{?order[id],order[title]}",  
29     "hydra:variableRepresentation": "BasicRepresentation",  
30     "hydra:mapping": [  
31         {  
32             "@type": "IriTemplateMapping",  
33             "variable": "order[id]",  
34             "property": "id",  
35             "required": false  
36         },  
37         {  
38             "@type": "IriTemplateMapping",  
39             "variable": "order[title]",  
40             "property": "title",  
41             "required": false  
42         }  
43     ]  
44 }
```

```
1 {  
2   books(order: {title: "DESC"}) {  
3     edges {  
4       node {  
5         title  
6         author {  
7           name  
8         }  
9       }  
10      }  
11    }  
12  }  
13 }
```

```
{
  "data": {
    "books": {
      "edges": [
        {
          "node": {
            "title": "The Paris Commune and the Idea of the State",
            "author": {
              "name": "Mikhail Bakunin"
            }
          }
        },
        {
          "node": {
            "title": "God and the State",
            "author": {
              "name": "Mikhail Bakunin"
            }
          }
        }
      ]
    }
  }
}
```

GraphQL

Pagination



Symfony Live
PARIS 2018
29-30 MARS

GET

https://localhost:8443/books?page=2

REST

```
1 {  
2   "@context": "/contexts/Book",  
3   "@id": "/books",  
4   "@type": "hydra:Collection",  
5   "hydra:member": [  
6     {  
7       "@id": "/books/00b678a2-54cd-3f37-9381-0cd92acea079",  
8       "@type": "http://schema.org/Book",  
9       "id": "00b678a2-54cd-3f37-9381-0cd92acea079",  
10      "title": "Quibusdam et ab quo voluptatum.",  
11      "author": "/authors/7db54d6a-6a89-3fe2-87ba-c0dd96d4f664"  
12    },  
13    {  
14      "@id": "/books/074cccf9-5d87-3351-8e58-05ad0da5ad9a",  
15      "@type": "http://schema.org/Book",  
16      "id": "074cccf9-5d87-3351-8e58-05ad0da5ad9a",  
17      "title": "Qui neque minus eos.",  
18      "author": "/authors/7db54d6a-6a89-3fe2-87ba-c0dd96d4f664"  
19    },  
20    {  
21      "@id": "/books/08268137-719e-357c-96d5-4c2a2047e9a1",  
22      "@type": "http://schema.org/Book",  
23      "id": "08268137-719e-357c-96d5-4c2a2047e9a1",  
24      "title": "Soluta in dicta molestiae asperiores consequuntur sit repellendus dolorum.",  
25    }]
```

```
217 "hydra:totalItems": 100,  
218 "hydra:view": {  
219     "@id": "/books?page=2",  
220     "@type": "hydra:PartialCollectionView",  
221     "hydra:first": "/books?page=1",  
222     "hydra:last": "/books?page=4",  
223     "hydra:previous": "/books?page=1",  
224     "hydra:next": "/books?page=3"  
225 },
```

GraphQL

```
1 {  
2   books(first: 5, after: "MA==") {  
3     pageInfo {  
4       hasNextPage  
5       endCursor  
6     }  
7     edges {  
8       cursor  
9       node {  
10         id  
11         title  
12       }  
13     }  
14   }  
15 }
```



```
{  
  "data": {  
    "books": {  
      "pageInfo": {  
        "hasNextPage": true,  
        "endCursor": "0Tk="  
      },  
      "edges": [  
        {  
          "cursor": "MQ==",  
          "node": {  
            "id": "1",  
            "title": "React  
          }  
        },  
        {  
          "cursor": "Mg==",  
          "node": {  
            "id": "2",  
            "title": "Relay  
          }  
        }  
      ]  
    }  
  }  
}
```



REST: configurable items per page

```
 /**
 * @ApiResource(
 *     attributes={"pagination_client_items_per_page=true"}
 * )
 */
class Book
```

GET ▾

<https://localhost:8443/books?page=2&itemsPerPage=5>

Key	Value
page	2
itemsPerPage	5

Changing States



Symfony Live
PARIS 2018
29-30 MARS

REST

PUT ▾

<https://localhost:8443/books/00b678a2-54cd-3f37-9381-0cd92acea079>

```
1 {  
2   "title": "A new title",  
3   "author": "/authors/896c6153-794e-3e94-b62d-95997c8b60ad"  
4 }
```

- Support for compound documents
- Ex: update both a book and an author in 1 request

```
1 - mutation {  
2 -   updateBook(input: {  
3 -     clientMutationId: "random-id-1"  
4 -     id: "/books/00b678a2-54cd-3f37-9381-0cd92acea079"  
5 -     title: "New Title"  
6 -     author: "/authors/896c6153-794e-3e94-b62d-95997c8b60ad"  
7 -   }) {  
8 -     id  
9 -     clientMutationId  
10 -   }  
11 -  
12 -   createAuthor(input: {  
13 -     clientMutationId: "random-id-2"  
14 -     _id: "355c4b4c-79ae-4a1a-8a78-ddf18da7a2c1"  
15 -     name: "Peter Kropotkin"  
16 -   }) {  
17 -     id  
18 -     clientMutationId  
19 -   }  
20 }
```



GraphQL

```
{  
  "data": {  
    "updateBook": {  
      "id": "/books/00b678a2-54cd-3f37-9381-0cd92acea079",  
      "clientMutationId": "random-id-1"  
    },  
    "createAuthor": {  
      "id": "/authors/355c4b4c-79ae-4a1a-8a78-ddf18da7a2c1",  
      "clientMutationId": "random-id-2"  
    }  
  }  
}
```

Logs



Symfony Live
PARIS 2018
29-30 MARS

REST

2. docker-compose logs -f api (docker-compose)

```
"GET /books HTTP/1.1" 200 944 "-" "PostmanRuntime/7.  
"GET /books HTTP/1.1" 200 944 "-" "PostmanRuntime/7.  
"GET /books?order[title]=DESC HTTP/1.1" 200 1035 "-"  
"GET /books?order[title]=ASC HTTP/1.1" 200 1034 "-"  
"GET /authors/1e820f57-5be9-4eb2-99e7-94a9b2c53053 H  
"POST /authors HTTP/1.1" 400 6856 "-" "PostmanRuntime/  
"POST /authors HTTP/1.1" 400 6889 "-" "PostmanRuntime/  
"PUT /authors/9825df50-8bd6-457e-9ec9-20ddab4bc219 H  
"DELETE /authors/9825df50-8bd6-457e-9ec9-20ddab4bc219  
"POST /authors/ HTTP/1.1" 404 174079 "-" "PostmanRuntime/  
"POST /authors HTTP/1.1" 400 6889 "-" "PostmanRuntime/
```

GraphQL

use logs -f api (docker-compose)

Caching



Symfony Live
PARIS 2018
29-30 MARS



API Platform Built-in Cache

- GET responses are generated only 1 time, then served by Varnish
- Responses are tagged with IRIs (URLs) of resources they contain
- When a write operation occurs cached responses containing stale data are purged
- **Support only REST (for now)**



Cache Considerations

- Most GraphQL clients use POST (=no HTTP cache)
- Even with GET: high cache miss rate
- Similar problem for sparse fieldsets
- If most requests are similar:
- A HTTP cache layer limits the over-fetching problem



REST Cache Layers

- Server application cache (ex: Doctrine data cache)
- HTTP server-side cache (ex: Varnish)
- HTTP client-side cache (ex: browser)
- Client application cache (ex: local storage)



GraphQL Cache Layers

- Server application cache (ex: Doctrine data cache)
- ~~HTTP server-side cache (ex: Varnish)~~
- ~~HTTP client-side cache (ex: browser)~~
- Client application cache (ex: local storage)



RDMS Performance



Symfony Live
PARIS 2018
29-30 MARS



RDMS Performance

- API Platform tries to **optimize SQL queries**, (adds relevant joins), for both **REST and GraphQL**
- Dynamic requests (GraphQL, sparse fieldsets) will most likely **not hit the indexes**
 - monitor frequent slow queries
 - use more adapted DB: Neo4J, ElasticSearch...
- With GraphQL, request can quickly become **huge**

Security & Reliability



Symfony Live
PARIS 2018
29-30 MARS

REST: OWASP SCS

- Industry recommendations
- Out of the box implem by API Platform
- Very large history and research

- 1 Introduction
- 2 HTTPS
- 3 Access Control
- 4 JWT
- 5 API Keys
- 6 Restrict HTTP methods
- 7 Input validation
- 8 Validate content types
 - 8.1 Validate request content types
 - 8.2 Send safe response content types
- 9 Management endpoints
- 10 Error handling
- 11 Audit logs
- 12 Security headers
 - 12.1 CORS
- 13 Sensitive information in HTTP requests
- 14 HTTP Return Code
- 15 Authors and primary editors
- 16 Other Cheatsheets



GraphQL Security

- Fewer feedback and history
- Non-obvious security protection:
 - timeout?
 - maximum query depth?
 - maximum query complexity?
- More sensitive to DDOS attacks



Heads up! GitHub's GraphQL Explorer makes use of your real, live, production data.

GraphQL



Prettify

< Docs

```
1 query ContributionsOfEmployees($company: String!) {  
2   organization(login: $company) {  
3     id  
4     name  
5     members(first: 100) {  
6       edges {  
7         node {  
8           id  
9           login  
10          name  
11          contributedRepositories(first: 100, privacy: PUBLIC) {  
12            edges {  
13              node {  
14                id  
15                name  
16                description  
17              }  
18            }  
19          }  
20        }  
21      }  
22    }
```

QUERY VARIABLES

```
1 {  
2   "company": "coopTilleuls"  
3 }
```

```
"<!DOCTYPE html><html><head><meta http-equiv=\"Content-type\" content=\"text/html; charset=utf-8\"><meta http-equiv=\"Content-Security-Policy\" content=\"default-src 'none'; base-uri 'self'; connect-src 'self'; form-action 'self'; img-src data:; script-src 'self'; style-src 'unsafe-inline'\"><meta content=\"origin\" name=\"referrer\"><title>Oh snap! &middot; GitHub</title><style type=\"text/css\" media=\"screen\"><body background-color: #f1f1f1; margin: 0; font-family: -apple-system, BlinkMacSystemFont, \"Segoe UI\", Roboto, Helvetica, Arial, sans-serif, \"Apple Color Emoji\", \"Segoe UI Emoji\", \"Segoe UI Symbol\"; ><.outer-container { position: relative; }><.container { width: 600px; text-align: center; }><.a { position: absolute; top: 50%; left: 50%; transform: translate(-50%, -50%); }><.a { color: #4183c4; text-decoration: none; }><.a:hover { text-decoration: underline; }><h1 { letter-spacing: -1px; line-height: 60px; font-size: 60px; font-weight: 100; margin: 0px; text-shadow: 0 1px 0 #fff; }><p { color: rgba(0, 0, 0, 0.5); margin: 20px 0 40px; }><ul { list-style: none; margin: 25px 0; padding: 0; }><li { display: table-cell; font-weight: bold; width: 1%; padding: 0; }><.logo { display: inline-block; margin-top: 35px; }><.logo-img-2x { display: none; }><@media only screen and (-webkit-min-device-pixel-ratio: 2), only screen and (-moz-min-device-pixel-ratio: 2), only screen and (-o-min-device-pixel-ratio: 2/1), only screen and (-min-device-pixel-ratio: 2), only screen and (-min-resolution: 192dpi), only screen and (-min-resolution: 2dppx) {><.logo-img-1x { display: none; }><.logo-img-2x { display: inline-block; }><}><#suggestions { margin-top: 35px; color: #ccc; }><#suggestions a { color: #666666; font-weight: 200; font-size: 14px; margin: 0 10px; }><.emoji { color: #000; line-height: 1; }></style></head><body>
```

(No) Versionning



Symfony Live
PARIS 2018
29-30 MARS



No Versionning

- GraphQL promotes deprecation over real versioning
- Good practice with REST too: **API evolution strategy**
- JSON-LD/Hydra: document deprecation with OWL

- Not supported in API Platform yet (contrib? ❤)

Client-side



Symfony Live
PARIS 2018
29-30 MARS

Client-side

- GraphQL: great, mature low-level libraries for **React**
- Apollo Client (community)
- Relay (Facebook)
- Hydra: younger client-side ecosystem
- api-platform/api-doc-parser: low level doc parser
- api-platform/client-generator: PWA & native app generator (React, React Native, Vue.js)
- api-platform/admin: automatic React admin interface



The Developper Ethical Responsibility



Symfony Live
PARIS 2018
29-30 MARS

Linked Data GraphQL

Origin

Scientific
community

Facebook

Data access

Open data

Information silo

Interop

Global and built-in

Partial

Best suited for

Decentralized
systems
(the world wide
web)

Centralized
systems





Thank You.

CBS
NEWS

* 2017 PRESIDENTIAL INAUGURATION *

CBSN
LIVE

Summary



Symfony Live
PARIS 2018
29-30 MARS

GraphQL

- Awesome, elegant **query language**
- Perfect for: **very dynamic** or different clients
- Real-time support (but not in PHP)
- Ability to query unrelated data easily
- Limited HTTP compat (cache, logs, content negot...)
- Not easy to implement: parser, security, caching, perf...



REST

- Modern formats have similar features than GraphQL
- sparse fieldsets, batch endpoints (ODATA)...
- Mature and robust solution
- Leverage HTTP (cache, content-negotiation, logs, security...)





Use Both?

- Public and open API: REST
- Private API for dynamic clients: GraphQL
- API Platform:
- **Create both with the same code**
- The GraphQL API is **compatible with Linked Data/REST**



Thanks!

Now, the questions...

Internals



Symfony Live
PARIS 2018
29-30 MARS



The Internals

- Design the **API's public data model** as PHP classes
- API Platform creates an **intermediate representation**:
 - resources, types, cardinalities, r/w, description...
- Metadata are used to:
 - **generate the schema** (OpenAPI, Hydra, GraphQL...)
 - **(de)serialize data** in the selected format (JSON-LD, GraphQL...)



The Internals (cont.)

- Interfaces to **access** and **persist** data (**Doctrine**, Mongo, Elastic...):
 - DataProvider
 - DataPersister
- Automatic registration of **REST routes** and GraphQL endpoint
- Serialization and validation rely on **Symfony components**
- REST: no controllers, pass thru a set of **SF kernel event's listeners**
- **Dependency Injection** and service decoration everywhere

NAME	EVENT	PRE & POST HOOKS
AddFormatListener	kernel.request	None
ReadListener	kernel.request	PRE_READ , POST_READ
DeserializeListener	kernel.request	PRE_DESERIALIZE , POST_DESERIALIZE
ValidateListener	kernel.view	PRE_VALIDATE , POST_VALIDATE
WriteListener	kernel.view	PRE_WRITE , POST_WRITE
SerializeListener	kernel.view	None
RespondListener	kernel.view	PRE RESPOND , POST RESPOND
AddLinkHeaderListener	kernel.response	None
ValidationExceptionListener	kernel.exception	None
ExceptionListener	kernel.exception	None