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

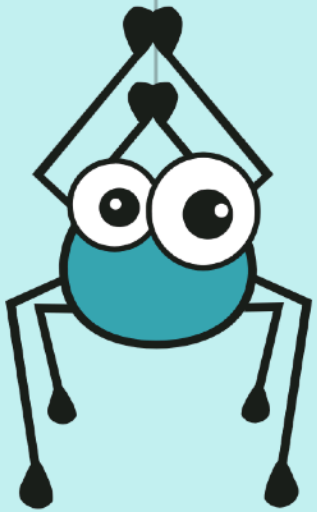
@dunglas

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**

The API Platform framework

SymfonyLive
PARIS 2018
29-30 MARS



API PLATFORM

REST and GraphQL framework to build modern API-driven projects

Download Get started

API Component

Symfony

Schema Gen Component

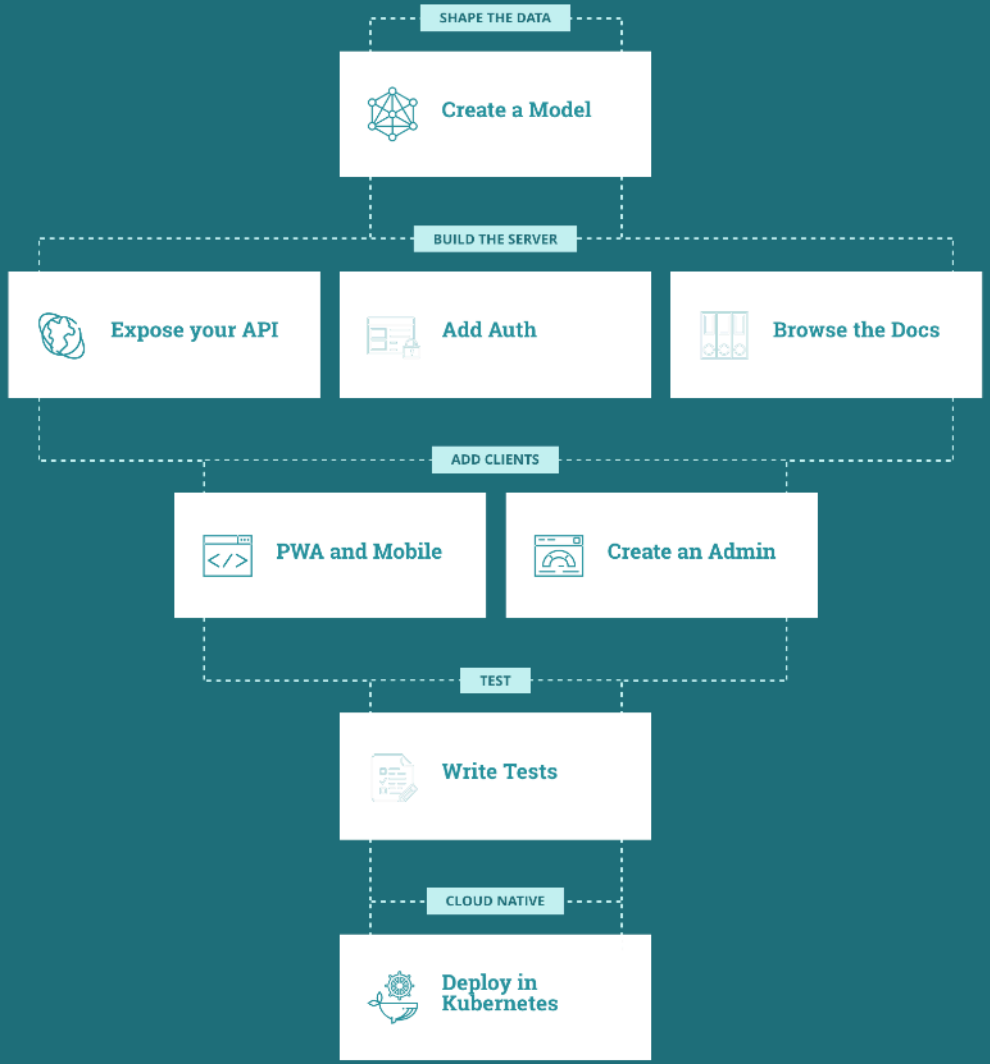
Admin Component

Client Gen Component

React

or

Vue.js


SHAPE THE DATA
Create a Model
BUILD THE SERVER
Expose your API
Add Auth
Browse the Docs
ADD CLIENTS
PWA and Mobile
Create an Admin
TEST
Write Tests
CLOUD NATIVE
Deploy in Kubernetes

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

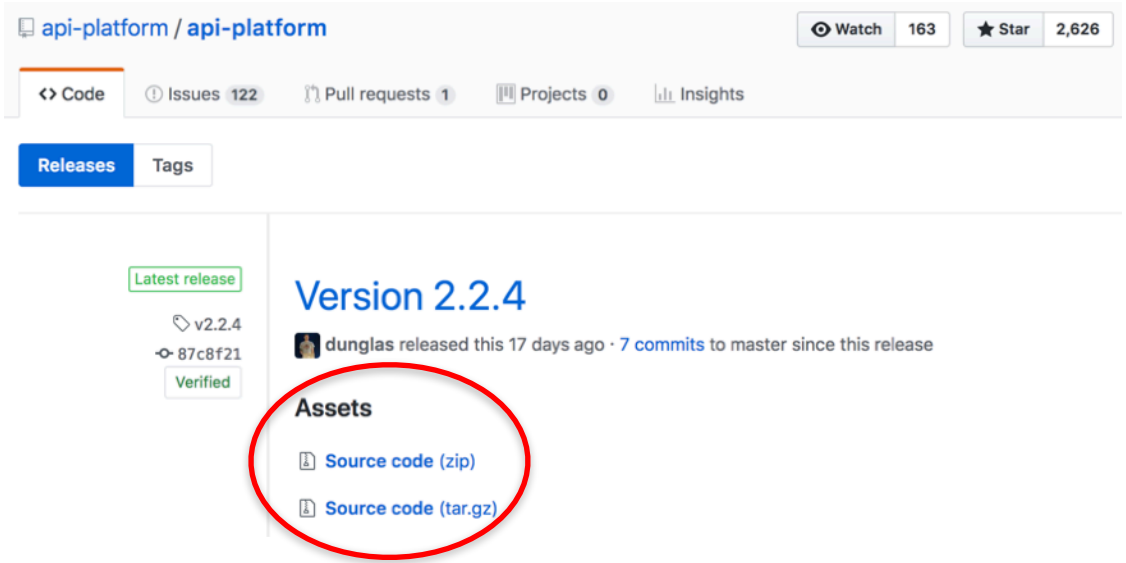
Implement the Relay Global Object Identification Specification ✓ — 6
#1585 by dunglas was merged on 21 Dec 2017

Use IRIs as GraphQL's ids and improve perf ✓ — 18
#1569 by dunglas was merged on 20 Dec 2017 1 of 1

Install

SymfonyLive
PARIS 2018
29-30 MARS

The Distribution




```
$ docker-compose up
```

Welcome to API Platform

Kévin

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

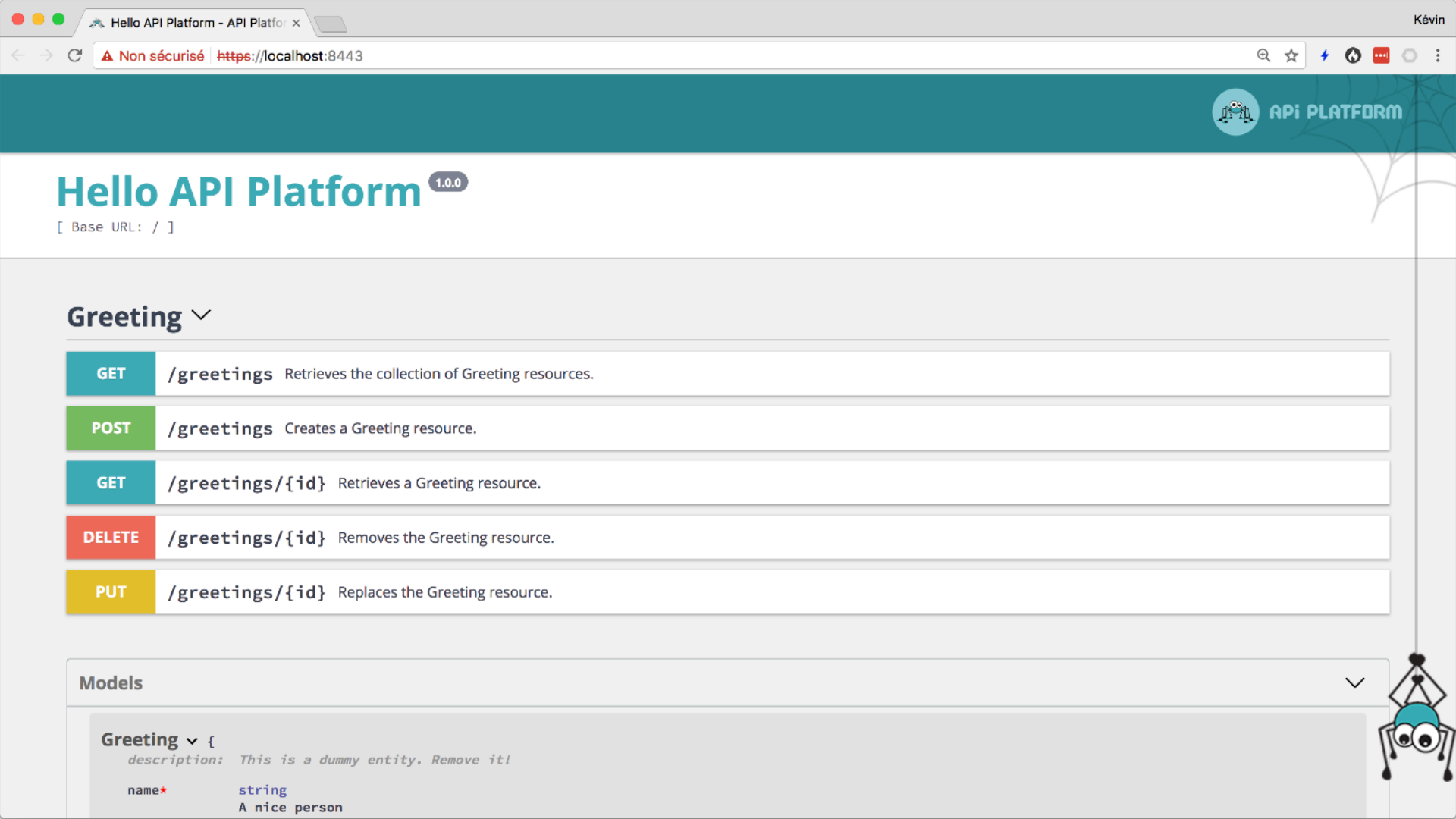
CACHED API

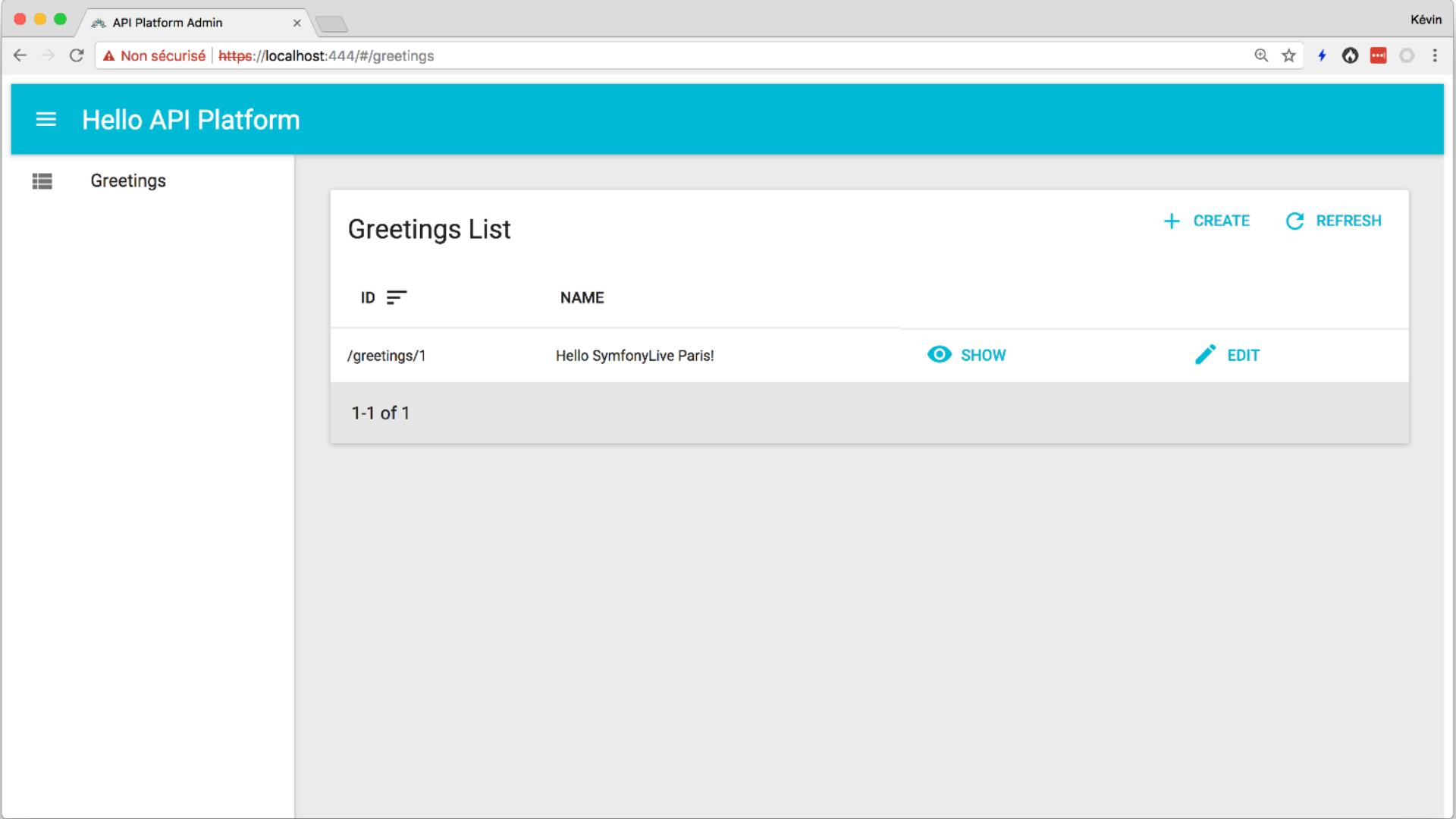
HTTP or HTTPS

ADMIN

HTTP or HTTPS

NEED HELP?


Hello API Platform - API Platfor
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
Kévin
Non sécurisé
https://localhost:444/#/greetings
Hello API Platform
Greetings
Greetings List
CREATE
REFRESH
ID
NAME
/greetings/1
Hello SymfonyLive Paris!
SHOW
EDIT
1-1 of 1

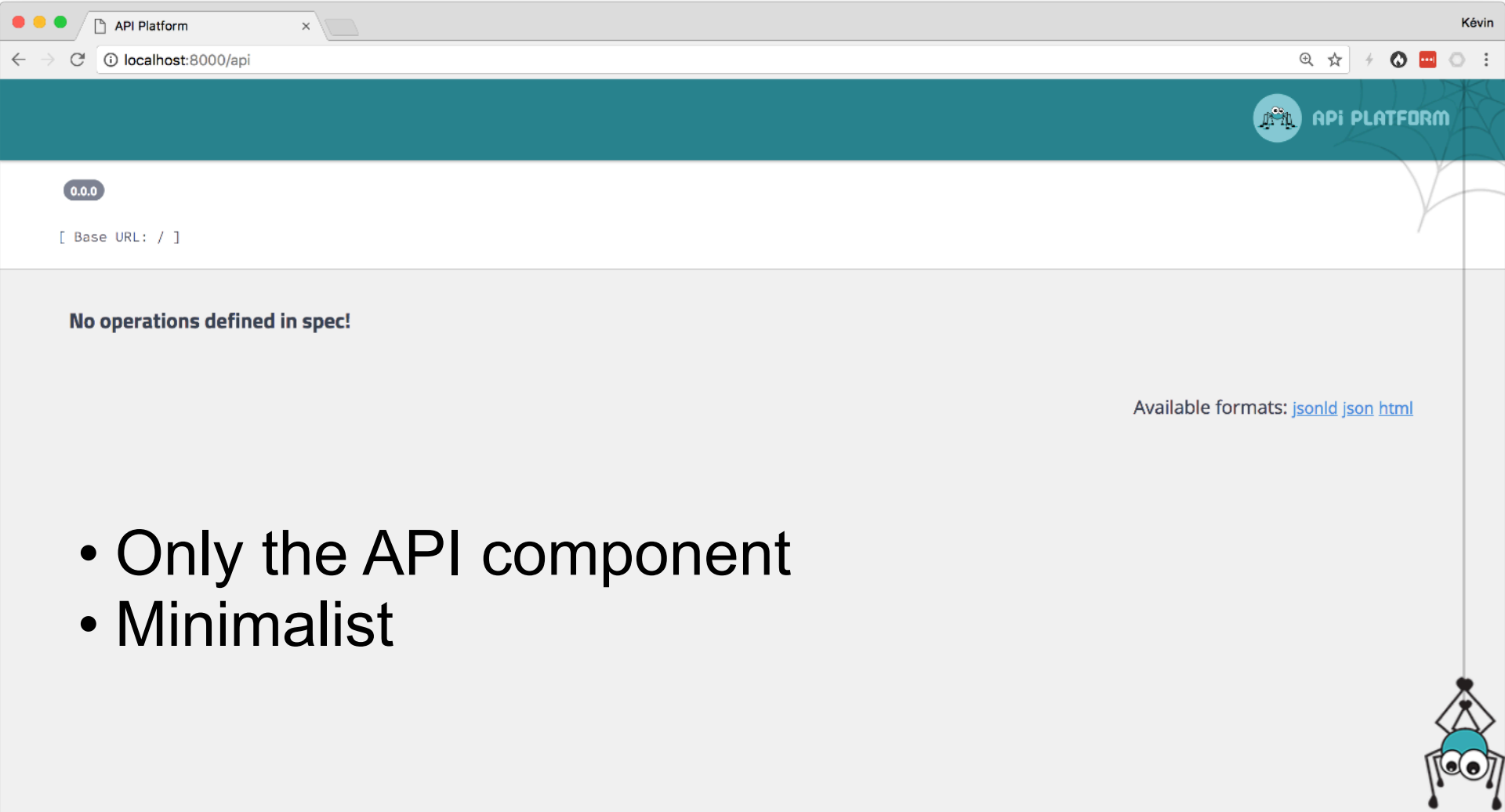
What's Inside

- **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
```



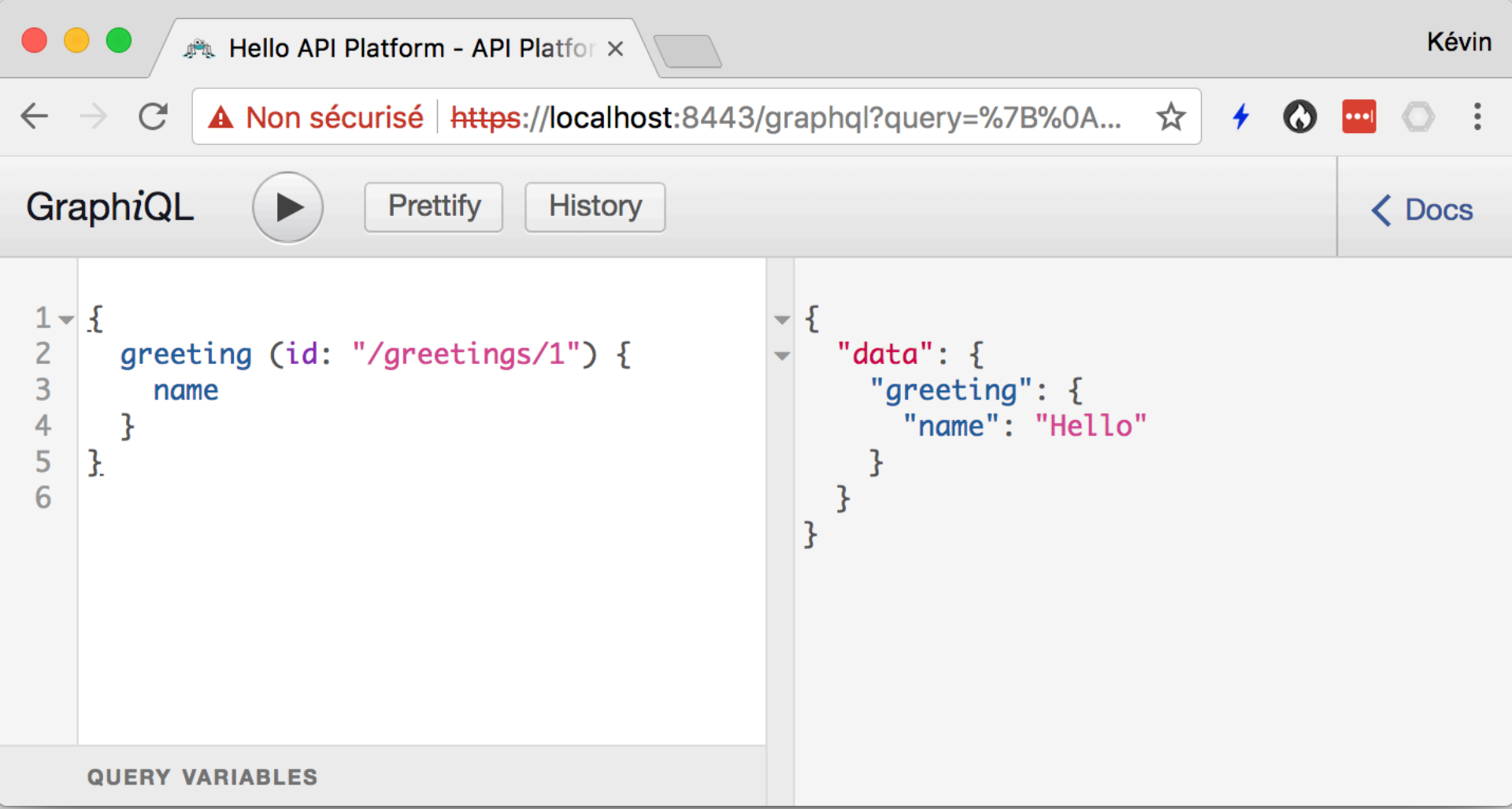
- Only the API component
- Minimalist

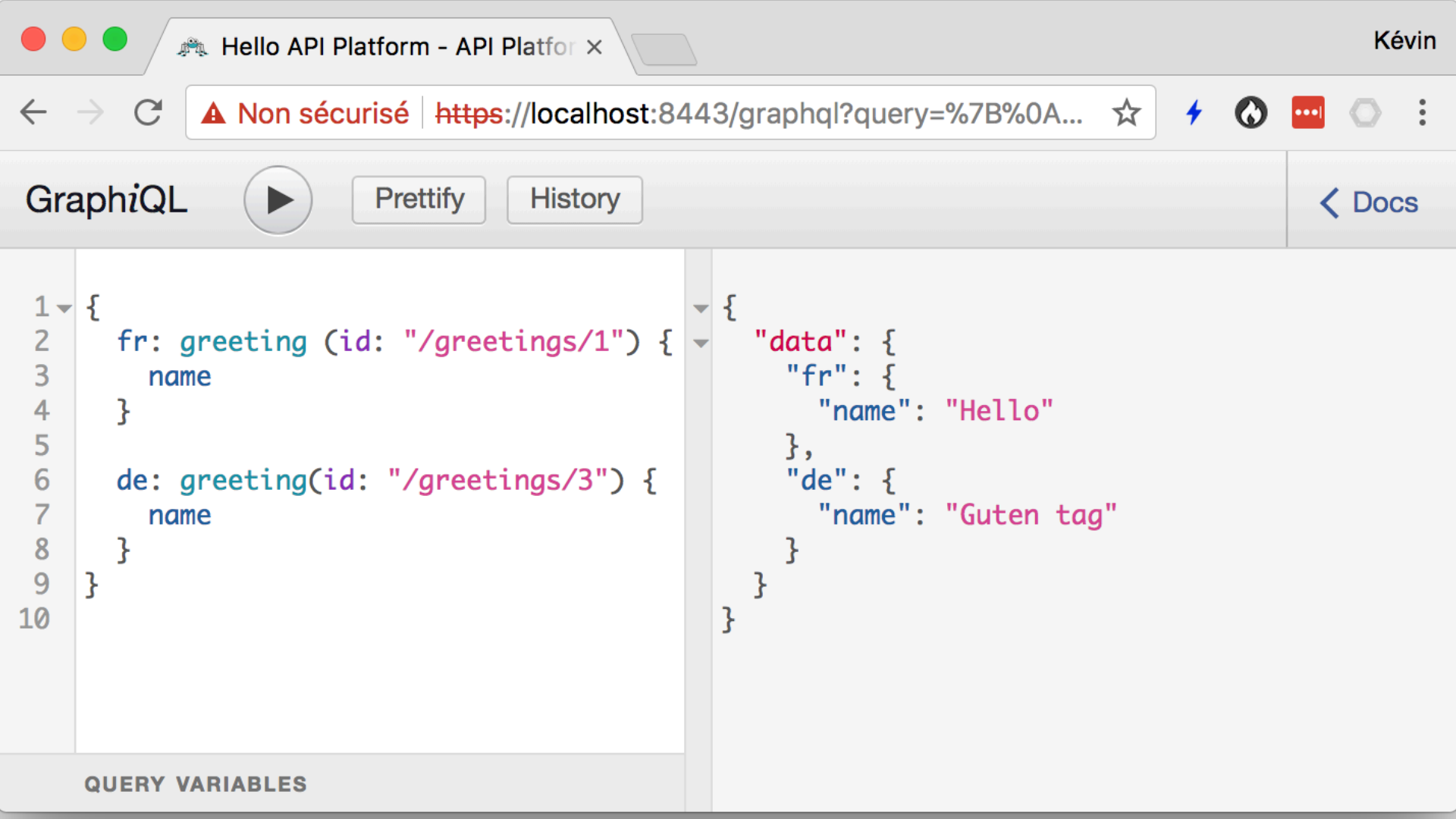
Enabling the GraphQL Support

SymfonyLive
PARIS 2018
29-30 MARS

Install GraphQL support

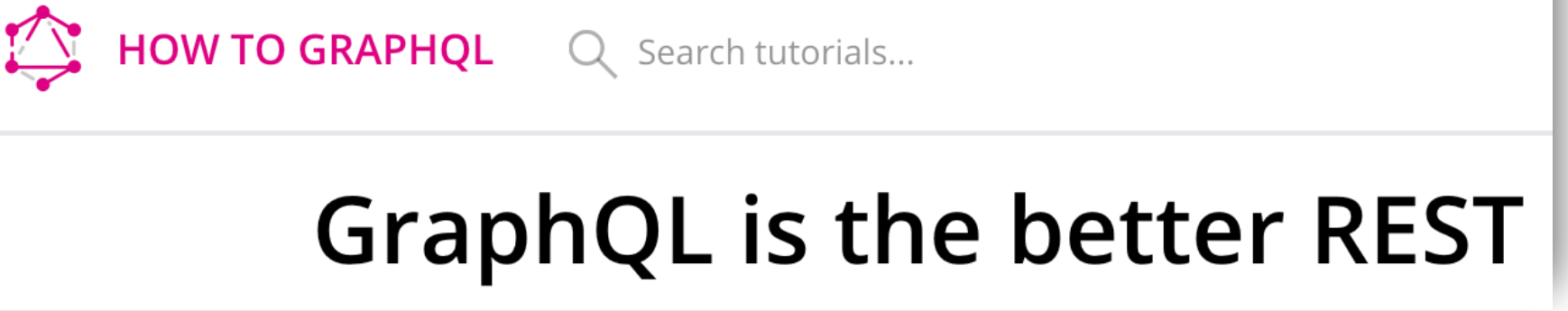

```
$ composer req \
  webonyx/graphql-php
```


Hello API Platform - API Platfor
Kévin
Non sécurisé
https://localhost:8443/graphql?query=%7B%0A...
GraphiQL
Prettify
History
Docs
{
greeting (id: "/greetings/1") {
name
}
}
QUERY VARIABLES
{
"data": {
"greeting": {
"name": "Hello"
}
}
}


Hello API Platform - API Platfor
Kévin
Non sécurisé
https://localhost:8443/graphql?query=%7B%0A...
GraphiQL
Prettify
History
Docs
{
fr: greeting (id: "/greetings/1") {
name
}
de: greeting(id: "/greetings/3") {
name
}
}
{
"data": {
"fr": {
"name": "Hello"
},
"de": {
"name": "Guten tag"
}
}
}
QUERY VARIABLES

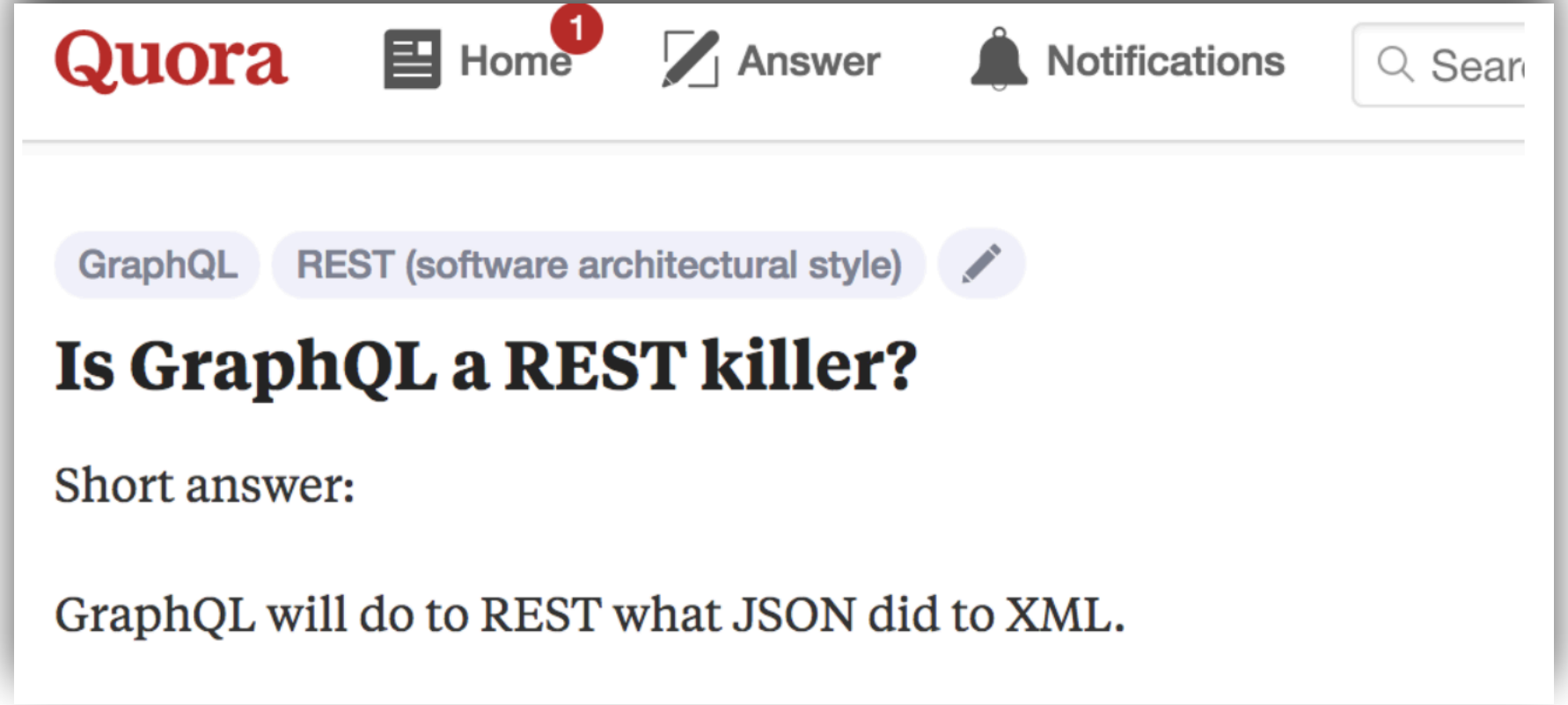
GraphQL?

SymfonyLive
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

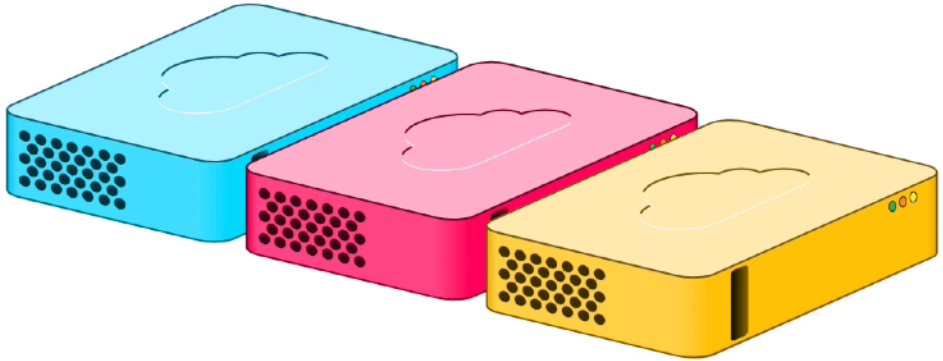

Quora
Home
1
Answer
Notifications
Sear
GraphQL
REST (software architectural style)
Is GraphQL a REST killer?
Short answer:
GraphQL will do to REST what JSON did to XML.

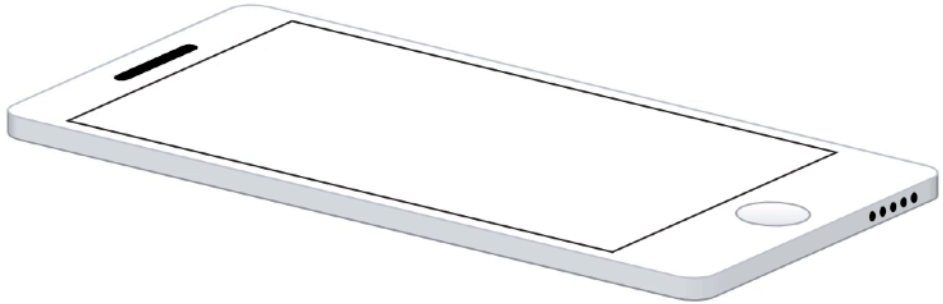
Samer Buna Follow
Author for Pluralsight and Lynda and curator of jsComplete.com
Jul 24, 2017 · 16 min read

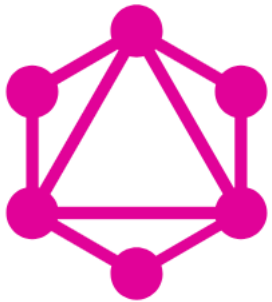
REST APIs are REST-in-Peace APIs. Long Live GraphQL.

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

SymfonyLive
PARIS 2018
29-30 MARS

```graphql
query ContributionsOfEmloyees($company: String!) {
  organization(login: $company) {
    id
    name
    members(first: 100) {
      edges {
        node {
          id
          login
          name
          contributedRepositories(first: 100, privacy: PUBLIC) {
            edges {
              node {
                id
                name
              }
            }
```

QUERY VARIABLES

```
{
  "company": "coopTilleuls"
}
```

```
{
  "data": {
    "organization": {
      "id": "MDEyOk9yZ2FuaXphdGlvbjExNjExOTg=",
      "name": "Les-Tilleuls.coop",
      "members": {
        "edges": [
          {
            "node": {
              "id": "MDQ6VXNlcjU3MjI0",
              "login": "dunglas",
              "name": "Kévin Dunglas",
              "contributedRepositories": {
                "edges": [
                  {
                    "node": {
                      "id": "MDEwOlJlcG9zaXRvcnk0NTgwNTg=",
                      "name": "symfony"
                    }
                  },
                  {
                    "node": {
                      "id": "MDEwOlJlcG9zaXRvcnk1MjE1ODM=",
                      "name": "symfony-docs"
                    }
```

Defining REST

SymfonyLive
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

SymfonyLive
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

SymfonyLive
PARIS 2018
29-30 MARS

GET https://kgsearch.googleapis.com/v1/entities:search?indent=true&query=Tim Berners-Lee&fields=@context,@type,itemListE


```
{
    "@context": {
        "@vocab": "http://schema.org/",
        "goog": "http://schema.googleapis.com/",
        "EntitySearchResult": "goog:EntitySearchResult
        "detailedDescription": "goog:detailedDescripti
        "resultScore": "goog:resultScore",
        "kg": "http://g.co/kg"
    },
    "@type": "ItemList",
    "itemListElement": [
        {
            "@type": "EntitySearchResult",
            "result": {
                "@id": "kg:/m/07d5b",
                "name": "Tim Berners-Lee",
                "@type": [
                    "Thing",
                    "Person"
                ],
                "description": "Computer scientist",
                "image": {
                    "contentUrl": "http://t2.gstatic.com/images?q=tbn:AN
                    "url": "https://commons.wikimedia.org/wiki/File:Sir_
                },
                "detailedDescription": {
                    "articleBody": "Sir Timothy John Berners-Lee OM KBE
                    "url": "https://en.wikipedia.org/wiki/Tim_Berners-Le
                    "license": "https://en.wikipedia.org/wiki/Wikipedia:
                },
                "url": "http://www.w3.org/People/Berners-Lee/"
            },
            "resultScore": 1259.341187
        }
```

Alternative Formats

SymfonyLive
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

SymfonyLive
PARIS 2018
29-30 MARS

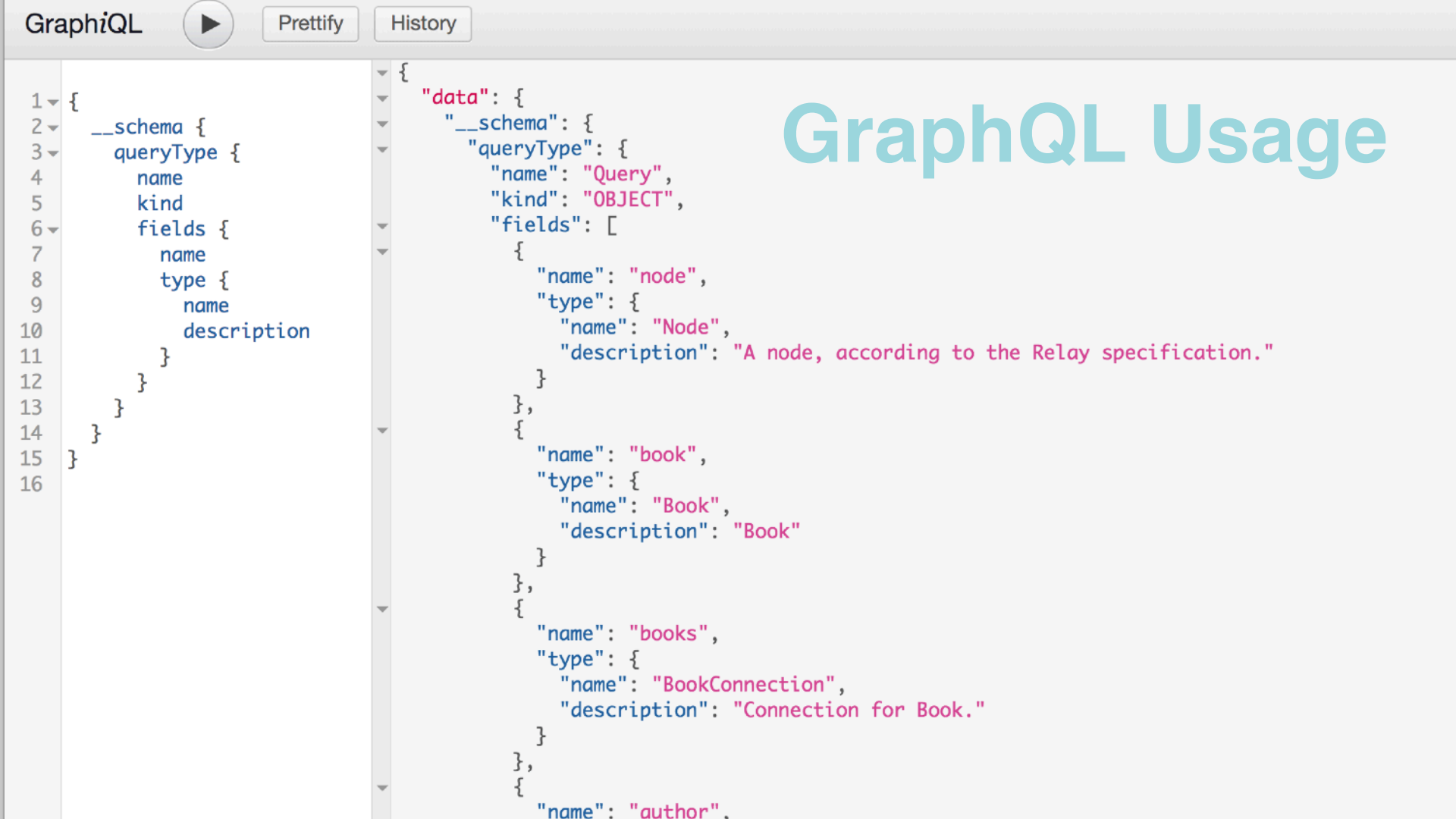
```php
/**
 * An Author.
 *
 * @ApiResource(iri="http://schema.org/Person")
 * @ORM\Entity
 */
class Author
{
    /**
     * @var string An UUID identifying this author.
     *
     * @ORM\Id
     * @ORM\Column(type="guid")
     * @Assert\NotBlank
     * @Assert\Uuid
     */
    public $id;

    /**
     * @var string The author's name
     *
     * @ApiProperty(iri="http://schema.org/name")
     * @ORM\Column(type="string")
     * @Assert\NotBlank
     */
    public $name;
}
```

```php
/**
 * Book
 *
 * @ApiResource(iri="http://schema.org/Book")
 * @ORM\Entity
 */
class Book
{
    /**
     * @var string An UUID identifying this book.
     *
     * @ORM\Id
     * @ORM\Column(type="guid")
     * @Assert\NotBlank
     * @Assert\Uuid
     */
    public $id;

    /**
     * @var string The Book's title.
     *
     * @ApiProperty(iri="http://schema.org/name")
     * @ORM\Column(type="string")
     * @Assert\NotBlank
     */
    public $title;

    /**
     * @var Author The book's author.
     *
     * @ApiProperty(iri="http://schema.org/author")
     * @ORM\ManyToOne(targetEntity="Author")
     * @Assert\NotNull
     */
    public $author;
}
```


GraphiQL
Prettify
History
{
__schema {
queryType {
name
kind
fields {
name
type {
name
description
}
}
}
}
}
{
"data": {
"__schema": {
"queryType": {
"name": "Query",
"kind": "OBJECT",
"fields": [
{
"name": "node",
"type": {
"name": "Node",
"description": "A node, according to the Relay specification."
}
},
{
"name": "book",
"type": {
"name": "Book",
"description": "Book"
}
},
{
"name": "books",
"type": {
"name": "BookConnection",
"description": "Connection for Book."
}
},
{
"name": "author",
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)
- Made 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"


```
{
    "@context": "/contexts/Entrypoint",
    "@id": "/",
    "@type": "Entrypoint",
    "book": "/books",
    "author": "/authors"
}
```

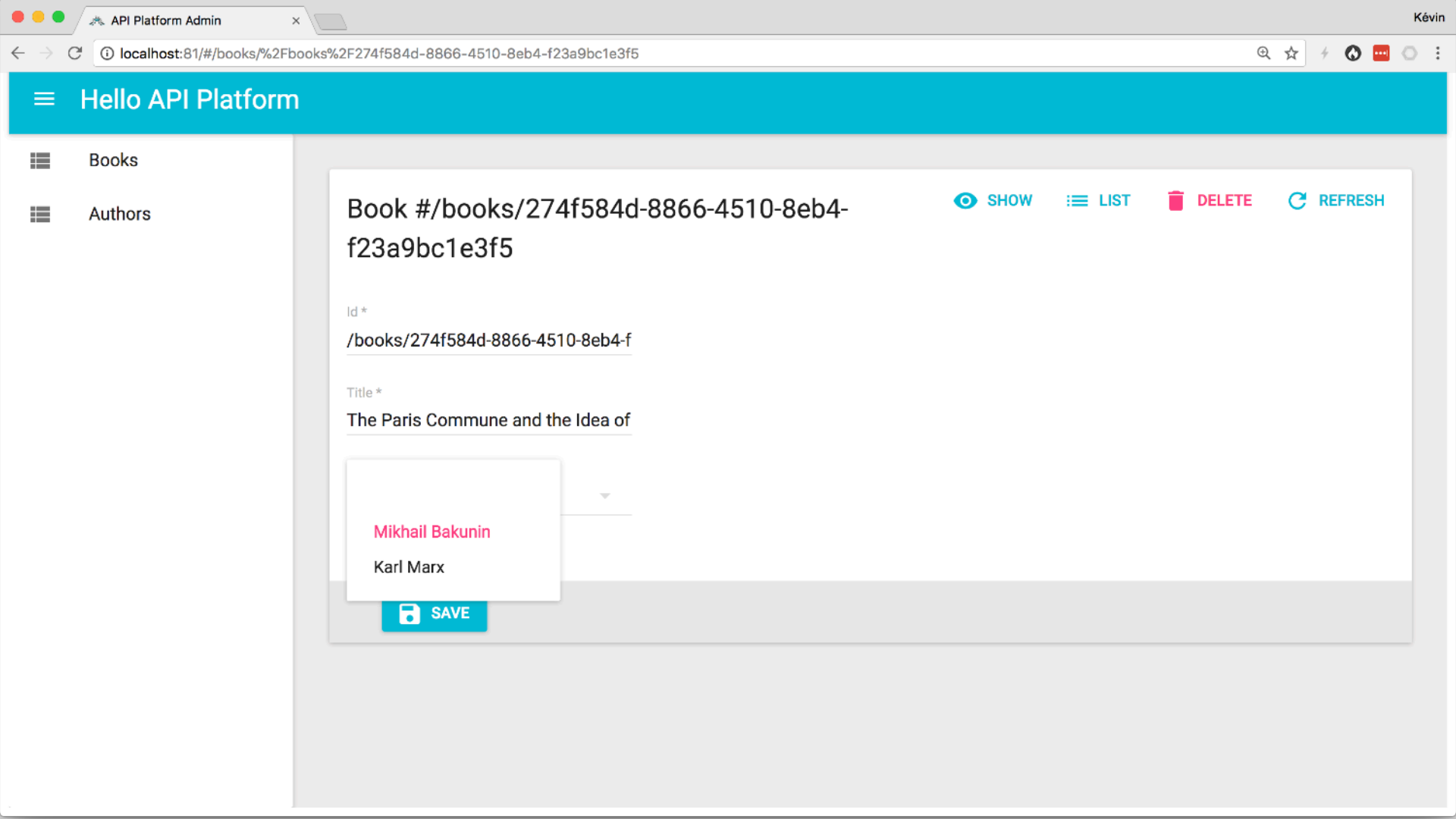
GET https://localhost:8443/docs


```
"hydra:supportedClass": [
    {
        "@id": "http://schema.org/Book",
        "@type": "hydra:Class",
        "rdfs:label": "Book",
        "hydra:title": "Book",
        "hydra:supportedProperty": [
            {…},
            {…},
            {
                "@type": "hydra:SupportedProperty",
                "hydra:property": {
                    "@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."
            }
        ],
```

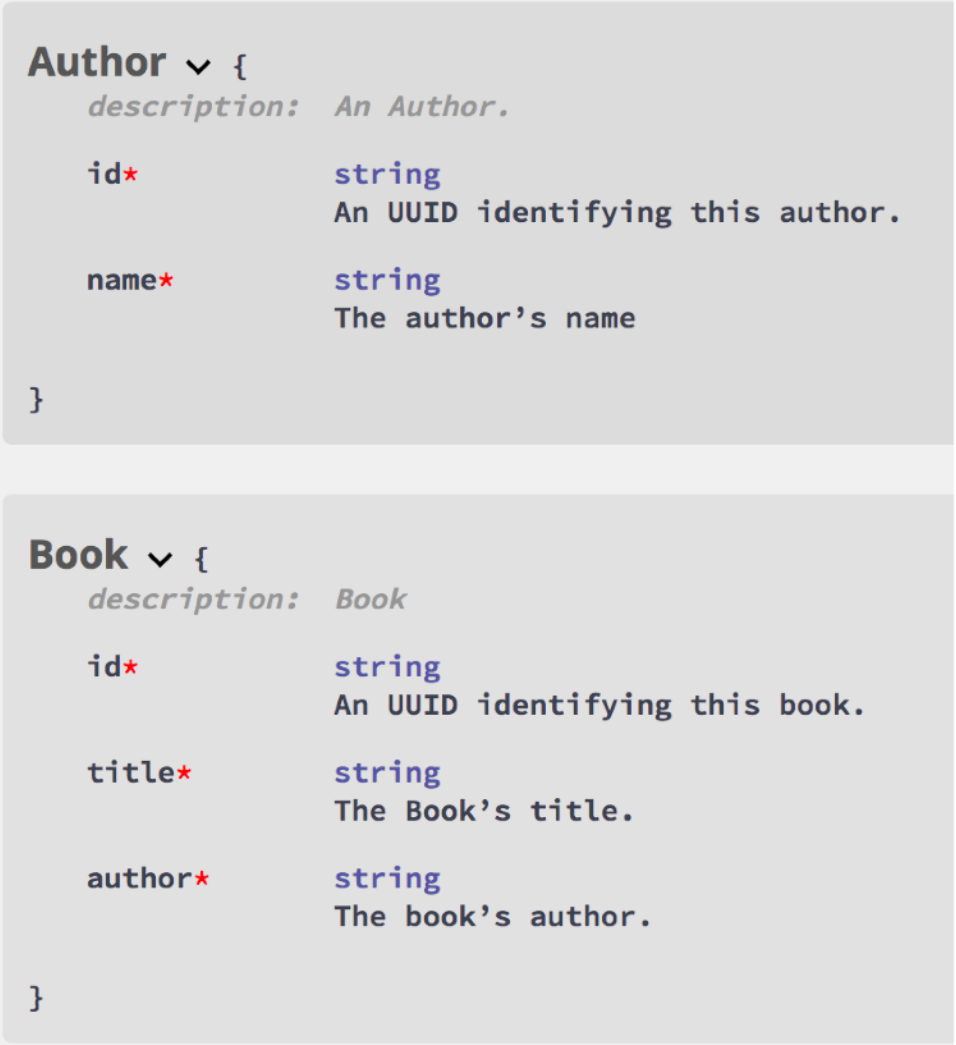
```
"hydra:supportedOperation": [
    {
        "@type": [
            "hydra:Operation",
            "schema:FindAction"
        ],
        "hydra:method": "GET",
        "hydra:title": "Retrieves Book resource.",
        "rdfs:label": "Retrieves Book resource.",
        "returns": "http://schema.org/Book"
    },
    {
        "@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",
```

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)


API Platform Admin
Kévin
localhost:81/#/books/%2Fbooks%2F274f584d-8866-4510-8eb4-f23a9bc1e3f5
Hello API Platform
Books
Authors
Book #/books/274f584d-8866-4510-8eb4-f23a9bc1e3f5
SHOW
LIST
DELETE
REFRESH
Id *
/books/274f584d-8866-4510-8eb4-f
Title *
The Paris Commune and the Idea of
Mikhail Bakunin
Karl Marx
SAVE

Alternative: OpenAPI



Data fetching: REST

SymfonyLive
PARIS 2018
29-30 MARS

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


```
{
    "@context": "/contexts/Book",
    "@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"
}
```


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


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

Data fetching: GraphQL

SymfonyLive
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

SymfonyLive
PARIS 2018
29-30 MARS

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

    /**
     * @Groups("book_read")
     */
    public $title;

    /**
     * @Groups("book_read")
     */
    public $author;
}
```

```php
/** An Author. ...*/
class Author
{
    /** @var string An UUID id
    public $id;

    /**
     * @Groups("book_read")
     */
    public $name;
}
```

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


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


- 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

```
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. ...*/
27      public $id;
28
29      /** @var string The Book's title. ...*/
36      public $title;
37
38      /** @var Author The book's author. ...*/
45      public $author;
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][]	name


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


- 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.

URL.searchParams

JavaScript

Jump to: Syntax Example Specifications Browser compatibility

Web technology for developers › Web APIs › URL › URL.searchParams

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

Filtering

SymfonyLive
PARIS 2018
29-30 MARS

```
12  /** A Book. ...*/
18  class Book
19  {
20      /** @var string An UUID identifying this book. ...*/
27      public $id;
28
29      /**
30       * @ApiFilter(SearchFilter::class, strategy="ipartial")
31       */
32      public $title;
33
34      /** @var Author The book's author. ...*/
41      public $author;
42  }
```

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

Key	Value
title	paris

REST


```
    "@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"
    },
```

```
    "hydra:totalItems": 1,
    "hydra:view": {
        "@id": "/books?title=paris",
        "@type": "hydra:PartialCollectionView"
    },
    "hydra:search": {
        "@type": "hydra:IriTemplate",
        "hydra:template": "/books{?title}",
        "hydra:variableRepresentation": "BasicRepresentation",
        "hydra:mapping": [
            {
                "@type": "IriTemplateMapping",
                "variable": "title",
                "property": "title",
                "required": false
            }
```

GraphQL

```
{
  books(title: "commune") {
    edges {
      node {
        id
        author {
          name
        }
      }
    }
  }
}
```

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

Alternative:


```
People?$top=2 &amp; $select=FirstName, LastName &amp; $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

SymfonyLive
PARIS 2018
29-30 MARS

```php
/**
 * @ApiFilter(OrderFilter::class)
 */
class Book
{
    /** @var string An UUID identifying this book. ...*/
    public $id;

    /** @var string The Book's title. ...*/
    public $title;

    /** @var Author The book's author. ...*/
    public $author;
}
```

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

Key	Value
order[title]	DESC

REST


```
"@context": "/contexts/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": 2,
"hydra:view": {
    "@id": "/books?order%5Btitle%5D=DESC",
    "@type": "hydra:PartialCollectionView"
},
"hydra:search": {
    "@type": "hydra:IriTemplate",
    "hydra:template": "/books{?order[id],order[title]}",
    "hydra:variableRepresentation": "BasicRepresentation",
    "hydra:mapping": [
        {
            "@type": "IriTemplateMapping",
            "variable": "order[id]",
            "property": "id",
            "required": false
        },
        {
            "@type": "IriTemplateMapping",
            "variable": "order[title]",
            "property": "title",
            "required": false
```

```
    "hydra:totalItems": 2,
    "hydra:view": {
        "@id": "/books?order%5Btitle%5D=DESC",
        "@type": "hydra:PartialCollectionView"
    },
    "hydra:search": {
        "@type": "hydra:IriTemplate",
        "hydra:template": "/books{?order[id],order[title]}",
        "hydra:variableRepresentation": "BasicRepresentation",
        "hydra:mapping": [
            {
                "@type": "IriTemplateMapping",
                "variable": "order[id]",
                "property": "id",
                "required": false
            },
            {
                "@type": "IriTemplateMapping",
                "variable": "order[title]",
                "property": "title",
                "required": false
            }
        ]
    }
```

```
{
  books(order: {title: "DESC"}) {
    edges {
      node {
        title
        author {
          name
        }
      }
    }
  }
}
```

GraphQL


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

Pagination

SymfonyLive
PARIS 2018
29-30 MARS

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

REST


```
{
    "@context": "/contexts/Book",
    "@id": "/books",
    "@type": "hydra:Collection",
    "hydra:member": [
        {
            "@id": "/books/00b678a2-54cd-3f37-9381-0cd92acea079",
            "@type": "http://schema.org/Book",
            "id": "00b678a2-54cd-3f37-9381-0cd92acea079",
            "title": "Quibusdam et ab quo voluptatum.",
            "author": "/authors/7db54d6a-6a89-3fe2-87ba-c0dd96d4f664"
        },
        {
            "@id": "/books/074cccf9-5d87-3351-8e58-05ad0da5ad9a",
            "@type": "http://schema.org/Book",
            "id": "074cccf9-5d87-3351-8e58-05ad0da5ad9a",
            "title": "Qui neque minus eos.",
            "author": "/authors/7db54d6a-6a89-3fe2-87ba-c0dd96d4f664"
        },
        {
            "@id": "/books/08268137-719e-357c-96d5-4c2a2047e9a1",
            "@type": "http://schema.org/Book",
            "id": "08268137-719e-357c-96d5-4c2a2047e9a1",
            "title": "Soluta in dicta molestiae asperiores consequuntur sit repellendus dolorum.",
```

```
    "hydra:totalItems": 100,
    "hydra:view": {
        "@id": "/books?page=2",
        "@type": "hydra:PartialCollectionView",
        "hydra:first": "/books?page=1",
        "hydra:last": "/books?page=4",
        "hydra:previous": "/books?page=1",
        "hydra:next": "/books?page=3"
    },
```

GraphQL

```
{
  books(first: 5, after: "MA==") {
    pageInfo {
      hasNextPage
      endCursor
    }
    edges {
      cursor
      node {
        id
        title
      }
    }
  }
}
```

Relay

```
{
  "data": {
    "books": {
      "pageInfo": {
        "hasNextPage": true,
        "endCursor": "OTk="
      },
      "edges": [
        {
          "cursor": "MQ==",
          "node": { ↔ }
        },
        {
          "cursor": "Mg==",
          "node": {
```

REST: configurable items per page

```php
/**
 * @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

SymfonyLive
PARIS 2018
29-30 MARS

REST

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

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

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

```
mutation {
  updateBook(input: {
    clientMutationId: "random-id-1"
    id: "/books/00b678a2-54cd-3f37-9381-0cd92acea079"
    title: "New Title"
    author: "/authors/896c6153-794e-3e94-b62d-95997c8b60ad"
  }) {
    id
    clientMutationId
  }

  createAuthor(input: {
    clientMutationId: "random-id-2"
    _id: "355c4b4c-79ae-4a1a-8a78-ddf18da7a2c1"
    name: "Peter Kropotkin"
  }) {
    id
    clientMutationId
  }
}
```

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

SymfonyLive
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 "-" "PostmanRuntim
"POST /authors HTTP/1.1" 400 6889 "-" "PostmanRuntim
"PUT /authors/9825df50-8bd6-457e-9ec9-20ddab4bc219 H
"DELETE /authors/9825df50-8bd6-457e-9ec9-20ddab4bc21
"POST /authors/ HTTP/1.1" 404 174079 "-" "PostmanRun
"POST /authors HTTP/1.1" 400 6889 "-" "PostmanRuntim
```

GraphQL

ose logs -f api (docker-compose)

```
"POST /graphql HTTP/1.1" 200 241 "-" "PostmanRuntime/7.1.1
"POST /graphql HTTP/1.1" 200 241 "-" "PostmanRuntime/7.1.1
"POST /graphql HTTP/1.1" 200 241 "-" "PostmanRuntime/7.1.1
"POST /graphql HTTP/1.1" 200 241 "-" "PostmanRuntime/7.1.1
"POST /graphql HTTP/1.1" 200 241 "-" "PostmanRuntime/7.1.1
"POST /graphql HTTP/1.1" 200 241 "-" "PostmanRuntime/7.1.1
"POST /graphql HTTP/1.1" 200 241 "-" "PostmanRuntime/7.1.1
"POST /graphql HTTP/1.1" 200 241 "-" "PostmanRuntime/7.1.1
"POST /graphql HTTP/1.1" 200 241 "-" "PostmanRuntime/7.1.1
"POST /graphql HTTP/1.1" 200 241 "-" "PostmanRuntime/7.1.1
"POST /graphql HTTP/1.1" 200 241 "-" "PostmanRuntime/7.1.1
"POST /graphql HTTP/1.1" 200 241 "-" "PostmanRuntime/7.1.1
```

Caching

SymfonyLive
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

SymfonyLive
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

SymfonyLive
PARIS 2018
29-30 MARS

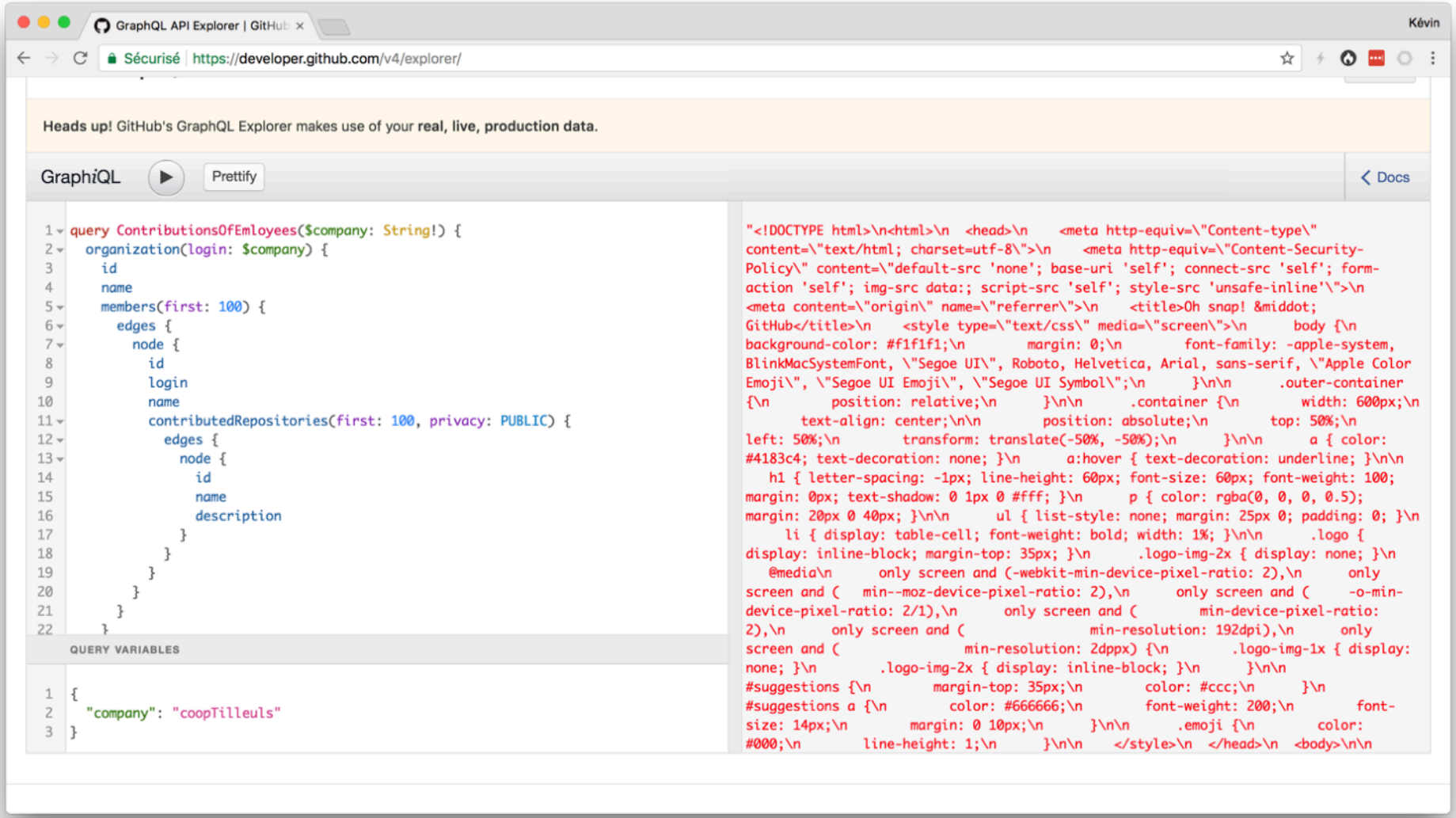
REST: OWASP SCS

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

REST Security Cheat Sheet

GraphQL Security

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


GraphQL API Explorer | GitHub
Kévin
Sécurisé https://developer.github.com/v4/explorer/
Heads up! GitHub's GraphQL Explorer makes use of your real, live, production data.
GraphiQL
Prettify
Docs
query ContributionsOfEmloyees($company: String!) {
organization(login: $company) {
id
name
members(first: 100) {
edges {
node {
id
login
name
contributedRepositories(first: 100, privacy: PUBLIC) {
edges {
node {
id
name
description
QUERY VARIABLES
{
"company": "coopTilleuls"
}
"<!DOCTYPE html>\n<html>\n <head>\n <meta http-equiv=\"Content-type\"
content=\"text/html; charset=utf-8\">\n <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'\">\n
<meta content=\"origin\" name=\"referrer\">\n <title>Oh snap! ·
GitHub</title>\n <style type=\"text/css\" media=\"screen\">\n body {\n
background-color: #f1f1f1;\n margin: 0;\n font-family: -apple-system,
BlinkMacSystemFont, \"Segoe UI\", Roboto, Helvetica, Arial, sans-serif, \"Apple Color
Emoji\", \"Segoe UI Emoji\", \"Segoe UI Symbol\";\n }\n\n .outer-container
{\n position: relative;\n }\n\n .container {\n width: 600px;\n
text-align: center;\n\n position: absolute;\n top: 50%;\n
left: 50%;\n transform: translate(-50%, -50%);\n }\n\n a { color:
#4183c4; text-decoration: none; }\n a:hover { text-decoration: underline; }\n\n
h1 { letter-spacing: -1px; line-height: 60px; font-size: 60px; font-weight: 100;
margin: 0px; text-shadow: 0 1px 0 #fff; }\n p { color: rgba(0, 0, 0, 0.5);
margin: 20px 0 40px; }\n\n ul { list-style: none; margin: 25px 0; padding: 0; }\n
li { display: table-cell; font-weight: bold; width: 1%; }\n\n .logo {
display: inline-block; margin-top: 35px; }\n .logo-img-2x { display: none; }\n
@media\n only screen and (-webkit-min-device-pixel-ratio: 2),\n only
screen and (min--moz-device-pixel-ratio: 2),\n only screen and (-o-min-
device-pixel-ratio: 2/1),\n only screen and (min-device-pixel-ratio:
2),\n only screen and (min-resolution: 192dpi),\n only
screen and (min-resolution: 2dppx) {\n .logo-img-1x { display:
none; }\n .logo-img-2x { display: inline-block; }\n }\n\n
#suggestions {\n margin-top: 35px;\n color: #ccc;\n }\n
#suggestions a {\n color: #666666;\n font-weight: 200;\n font-
size: 14px;\n margin: 0 10px;\n }\n\n .emoji {\n color:
#000;\n line-height: 1;\n }\n\n </style>\n </head>\n <body>\n\n

(No) Versionning

SymfonyLive
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

SymfonyLive
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

SymfonyLive
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

SymfonyLive
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…

api-platform.com

api-platform/api-platform

@ApiPlatform

Internals

SymfonyLive
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