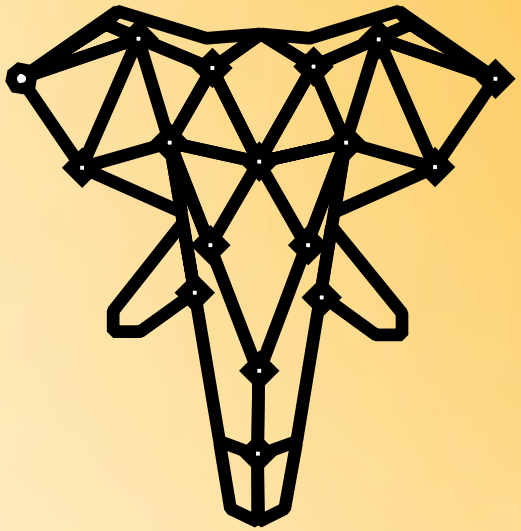


Building a GraphQL API in PHP with GraphQLite

« me »

David Négrier
aka:

moufmouf

@david_negrier

joind.in

joind.in/user/moufmouf

We are hiring!

TheCodingMachine
TCM://

CTO & co-founder
@TheCodingMachine

PHP enthusiast
PSR-11 co-editor
GraphQLite author
WorkAdventure lead

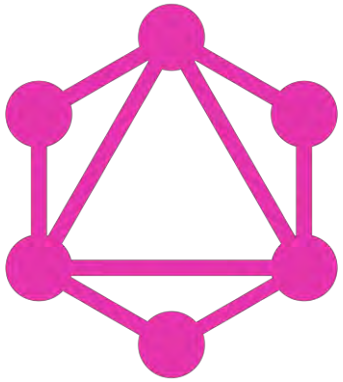
But also Packanalyst, Mouf, TDBM…

- What is GraphQL?
- Why GraphQL?
- GraphQL type system
- The GraphQL ecosystem in PHP
- GraphQLite

IPC

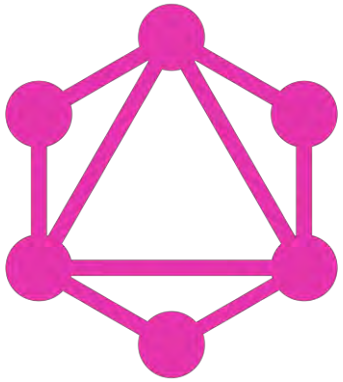
International PHP Conference

GraphQL ?



- GraphQL is a **protocol**

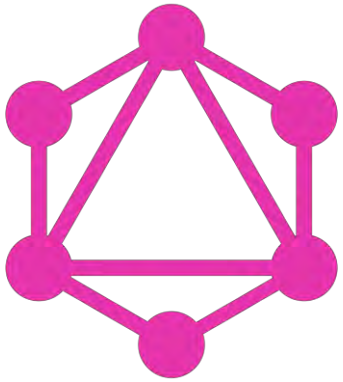
GraphQL ?



- GraphQL is a **protocol**

- It is **not**:
  - A fancy new database
  - A database query language like SQL

GraphQL ?



- GraphQL is a **protocol**
- GraphQL is a challenger to those other protocols:
  - REST
  - Web-services (SOAP/WSDL based)

A bit of history

Web-services
(~1999)

- ✓ Strongly typed
- ✓ Self-describing (WSDL)
- ✗ XML-based

A bit of history

Web-servi
(~1999

- ✓ Strongly ty
- ✓ Self-descri
- ✗ XML-based

A bit of history

Web-services (~1999)

- ✓ Strongly typed
- ✓ Self-describing (WSDL)
- ✗ XML-based

REST (~2005)

- ✗ Weakly typed
- ✗ Non self-describing (still OpenAPI for doc)
- ✓ Mostly JSON based

A bit of history

Web-se
(~19

- ✓ Strongly
- ✓ Self-des
- ✗ XML-bas

A bit of history

Web-services (~1999)

- ✓ Strongly typed
- ✓ Self-describing (WSDL)
- ✗ XML-based

REST (~2005)

- ✗ Weakly typed
- ✗ Non self-describing (still OpenAPI for doc)
- ✓ Mostly JSON based

GraphQL (2015)

- ✓ Strongly typed
- ✓ Self-describing
- ✓ JSON based
- ✓ ✓ + Client driven queries

A bit of history

SOAP/WSDL REST

GraphQL

GraphQL ?

It is developed by Facebook and was first used in the Facebook API.

It is now an open protocol backed by the *GraphQL foundation*.

- What is GraphQL?
- Why GraphQL?
- GraphQL type system
- The GraphQL ecosystem in PHP
- GraphQLite

IPC

International PHP Conference

What problem does GraphQL solve?

With a REST API

Dan
PO

« Hey! We need to display the company details in the product page »

Bob
Back dev

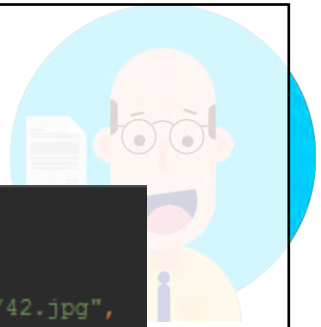
Alice
Front dev

What problem does GraphQL solve?

REST API

/api/product/42

```
{
  "id": 42,
  "name": "my super product",
  "logo": "https://marketplace.com/photo/product/42.jpg",
  "company": {
    "id": 35
  }
}
```



Dan
PO

Bob
Back dev

Alice
Front dev

« Oops, the API endpoint for products does not have the info I want »

What problem does GraphQL solve?

REST API

/api/product/42

```
{
  "id": 42,
  "name": "my super product",
  "logo": "https://marketplace.com/photo/product/42.jpg",
  "company": {
    "id": 35
  }
}
```

/api/company/35

```
{
  "id": 35,
  "name": "my super company",
  "revenue": "4000000",
  "logo": "https://marketplace.com/photo/company/35.png"
}
```

Back dev

Solution 1

Alice
Front dev

« Let's make an additional request to the company endpoint »

What problem does GraphQL solve?

This first solution is not ideal.

We don’t have enough data in the first request.
This is called **under-fetching.**

We need to do 2 requests so we increase the **latency** of our application.

What problem does GraphQL solve?

Solution 2

Dan
PO

Bob
Back dev

« Hey, can you change this API for me? »

Alice
Front dev

What problem does GraphQL solve?

REST API

/api/product/42

```
{
  "id": 42,
  "name": "my super product",
  "logo": "https://marketplace.com/photo/product/42.jpg",
  "company": {
    "id": 35,
    "name": "my super company",
    "revenue": "4000000",
    "logo": "https://marketplace.com/photo/company/35.png"
  }
}
```

Bob
Back dev

« Hey, can you change this API for me? »

Alice
Front dev

Solution 2

What problem does GraphQL solve?

This second solution is not ideal.

Some users of the API may not need all the data we are returning. While this solves the problem for Alice, other existing users will get useless additional data.

This is called **over-fetching**.

What problem does GraphQL solve?

Solution 3

Dan
PO

Bob
Back dev

« Hey, can you change this API for me, with a flag? »

Alice
Front dev

What problem does GraphQL solve?

REST API

/api/product/42?**with_company=true**

```
{
  "id": 42,
  "name": "my super product",
  "logo": "https://marketplace.com/photo/product/42.jpg",
  "company": {
    "id": 35,
    "name": "my super company",
    "revenue": "4000000",
    "logo": "https://marketplace.com/photo/company/35.png"
  }
}
```

Bob
Back dev

« Hey, can you change this API for me, with a flag? »

Alice
Front dev

Solution 3

What problem does GraphQL solve?

This third solution is better…

… but it leads to **“flags hell”**.

Each consumer of the API will request its own set of flags. Your API is **dependent on its consumers** which is not great.

It requires additional maintenance and effort for the back-end team.

What problem does GraphQL solve?

GraphQL **to the rescue**!

GraphQL is a *paradigm shift*.

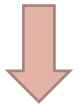
The **client** asks for the list of fields it wants.

What problem does GraphQL solve?

GraphQL **to the rescue**!

GraphQL is a *paradigm shift*.

The **client** asks for the list of fields it wants.



✓ Solves both **over-fetching** and **under-fetching** at the same time!

What problem does GraphQL solve?

With a GraphQL API

Dan
PO

« Hey! We need to display the company details in the product page »

Bob
Back dev

Alice
Front dev

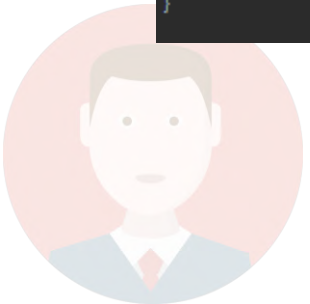
What problem does GraphQL solve?

GraphQL API

GET /graphql?query=

```
{
    product(id:42) {
        id
        name
        company {
            id
            name
            logo
        }
    }
}
```


Dan
PO



Bob
Back dev

Alice
Front dev

« Let me check the current request »

What problem does GraphQL solve?

GraphQL API

GET /graphql?query=

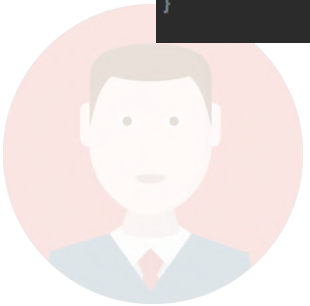
```
{
    product(id:42) {
        id
        name
        company {
            id
            name
            logo
        }
    }
}
```

Single endpoint

The name of the query is « product »

List of fields requested

Dan
PO



Bob
Back dev

« Let me check the current request »

Alice
Front dev

What problem does GraphQL solve?

GraphQL API

GET /graphql?query=

```
{
    product(id:42) {
        id
        name
        logo
        company {
            country {
                name
            }
        }
    }
}
```

Single endpoint

Bob
Back dev

Alice
Front dev

« Let me change the requested list of fields »

What problem does GraphQL solve?

GraphQL API

GET /graphql?query=

```
{
    product(id:42) {
        id
        name
        logo
        company {
            country {
                name
            }
        }
    }
}
```

Single endpoint

Bob
Back dev

Alice
Front dev

What problem does GraphQL solve?

✓ Success!

Alice was able to perform the changes without contacting Bob!

GraphQL reduces the friction between front-end and back-end developers.

What problem does GraphQL solve?

It also shines if you have 2 slightly different APIs (for instance one for the web and one for a mobile app)

Demo time!

https://bit.ly/3IPY6jt

- What is GraphQL?
- Why GraphQL?
- GraphQL type system
- The GraphQL ecosystem in PHP
- GraphQLite

IPC

International
PHP Conference

Types

GraphQL is **strongly typed.**

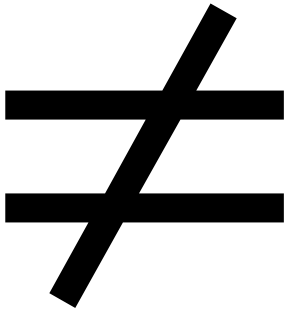
It comes with a « schema language » but this is rarely used while developing.

It is however useful to understand what is going on.

```
type Query {
  product(id: ID): Product!
  products(limit: Int, offset: Int): [Product]!
}

type Product {
  id: ID!
  name: String!
  logo: String
  company: Company!
}

type Company {
  id: ID!
  name: String!
  logo: String
  country: Country
}

type Country {
    id: ID!
    name: String
}
```

Query language

```
product(id: 42) {
  name
  company {
    name
    logo
    country {
      name
    }
  }
}
```



Schema language

```
type Query {
  product(id: ID): Product!
  products(limit: Int, offset: Int): [Product]!
}

type Product {
  id: ID!
  name: String!
  logo: String
  company: Company!
}

type Company {
  id: ID!
  name: String!
  logo: String
  country: Country
}

type Country {
    id: ID!
    name: String
}
```

Types

Note:

- `[Product]` ➔ an **array** of Products
- `String` ➔ a string (or null)
- `String!` ➔ a **non-nullable** string

Hence:

- `[Product!]!` ➔ An array (non-nullable) of products that are also non-nullable.

```
type Query {
  product(id: ID): Product!
  products(limit: Int, offset: Int): [Product]!
}

type Product {
  id: ID!
  name: String!
  logo: String
  company: Company!
}

type Company {
  id: ID!
  name: String!
  logo: String
  country: Country
}

type Country {
    id: ID!
    name: String
}
```

Types

Some « scalar » types:

- `ID`: a unique identifier (~=`String`)
- `String`
- `Int`
- `Float`
- `Boolean`

No support for « Date » in the standard (but custom types are supported by some implementations)

```
type Query {
  product(id: ID): Product!
  products(limit: Int, offset: Int): [Product]!
}

type Product {
  id: ID!
  name: String!
  logo: String
  company: Company!
}

type Company {
  id: ID!
  name: String!
  logo: String
  country: Country
}

type Country {
    id: ID!
    name: String
}
```

Types

Support for “arguments”:

- `product(id: ID!)`
  ➔ the product query requires an “id” field of type “ID” to be passed.

```
type Query {
  product(id: ID): Product!
  products(limit: Int, offset: Int): [Product]!
}

type Product {
  id: ID!
  name: String!
  logo: String
  company: Company!
}

type Company {
  id: ID!
  name: String!
  logo: String
  country: Country
}

type Country {
    id: ID!
    name: String
}
```

Types

Bonus:

- Support for interfaces
- Support for Union types
- Support for “InputType” (to pass complex objects in queries)

Mutations

So far, we mostly talked about **queries** (because this is what is fun in GraphQL).

GraphQL can also do **mutations** (to change the state of the DB)

Subscriptions

The GraphQL protocol also comes with a support for **real-time** communication.

Subscriptions allow a client to be notified when the results of a query change.

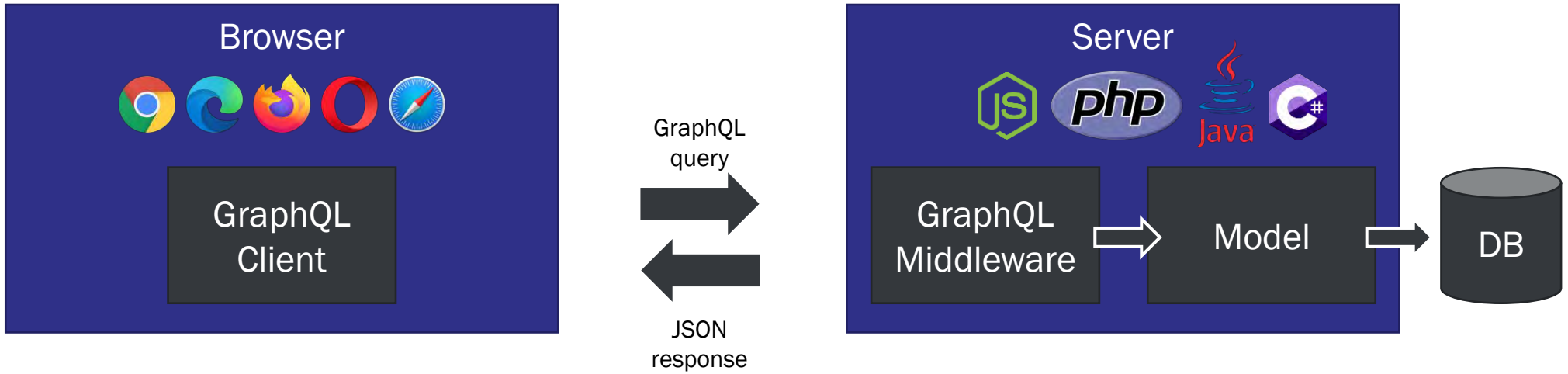
(you need a GraphQL server that supports Subscriptions)

- What is GraphQL?
- Why GraphQL?
- GraphQL type system
- The GraphQL ecosystem in PHP
- GraphQLite

IPC

International PHP Conference

Ecosystem


Browser
GraphQL Client
GraphQL query
JSON response
Server
JS
php
Java
C#
GraphQL Middleware
Model
DB

Ecosystem (a small part of...)

GraphQL Client

- Relay
- Apollo
- + many many others

GraphQL Development Client

- GraphiQL
- Altair
- GraphQL Playground

GraphQL Middleware

- express-graphql
- Nest JS
- Apollo server
- ...

- Webonyx/ GraphQL-PHP
- API Platform
- Lighthouse
- Overblog GraphQL-Bundle
- GraphQLite
- ...

Zoom on GraphQL in PHP

GraphQL Middleware

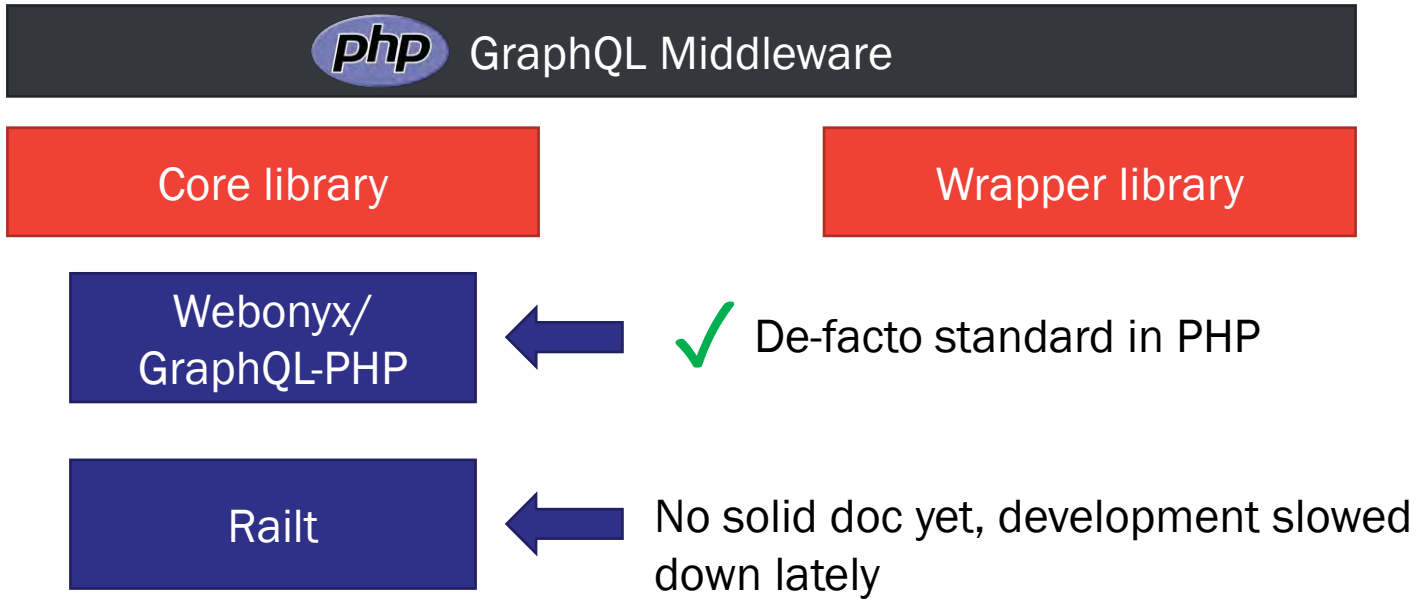
Core library

- Low level
  - Parsing
  - Serving requests
- Powerful
  - Feature complete
- Hard to use (poor DX)

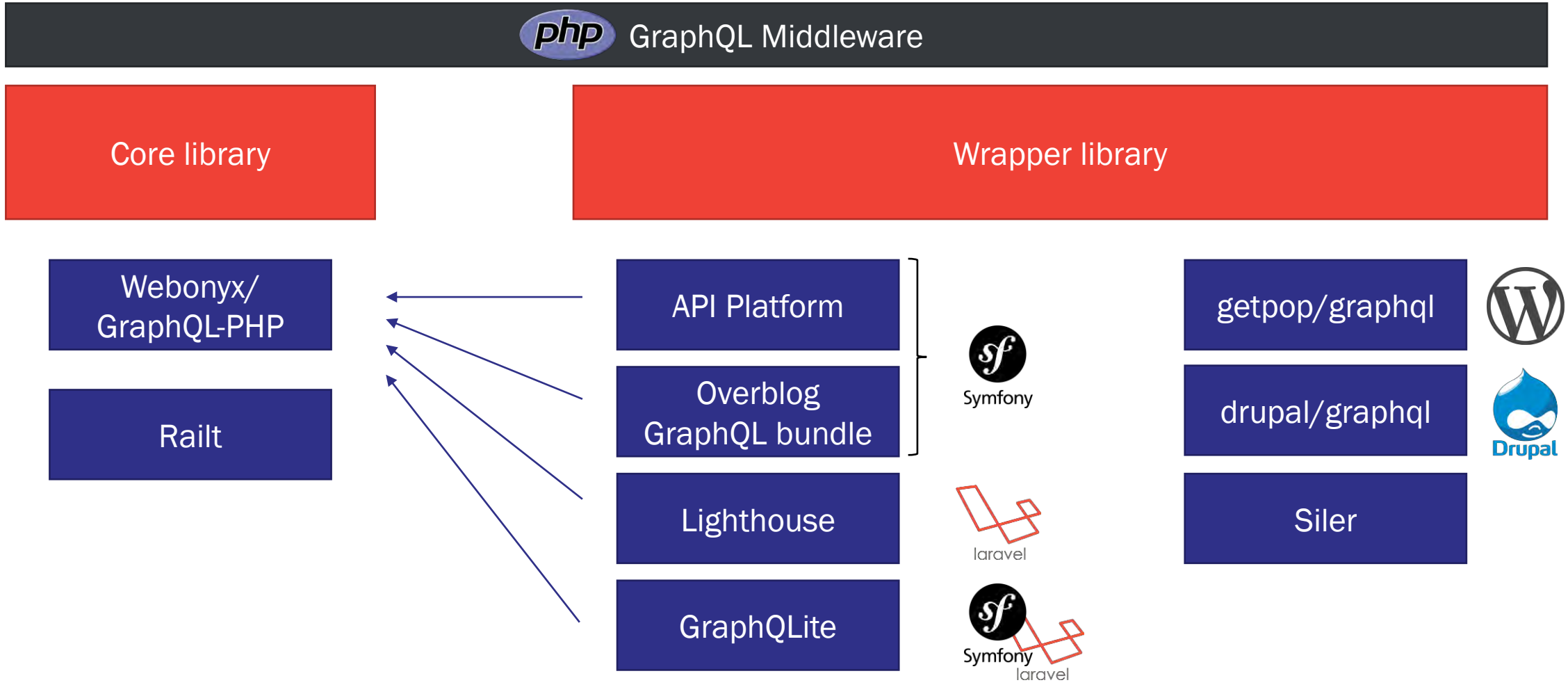
Wrapper library

- High level
- Opiniated
- Easy to use

Zoom on GraphQL in PHP



Zoom on GraphQL in PHP



Zoom on Webonyx/GraphQL-PHP

Define a type

```php
$blogStory = new ObjectType([
    'name' => 'Story',
    'fields' => [
        'body' => Type::string(),
        'author' => [
            'type' => $userType,
            'description' => 'Story author',
            'resolve' => function(Story $blogStory) {
                return DataSource::findUser($blogStory->authorId);
            }
        ],
        'likes' => [
            'type' => Type::listOf($userType),
            'description' => 'List of users who liked the story',
            'args' => [
                'limit' => [
                    'type' => Type::int(),
                    'description' => 'Limit the number of recent likes returned',
                    'defaultValue' => 10
                ]
            ],
            'resolve' => function(Story $blogStory, $args) {
                return DataSource::findLikes($blogStory->id, $args['limit']);
            }
        ]
    ]
]);
```

This code will generate this type:

```
type Story {
  body: String
  author: User
  likes(limit: Int): [User]
}
```

Zoom on Webonyx/GraphQL-PHP

Define a query

```php
$queryType = new ObjectType([
    'name' => 'Query',
    'fields' => [
        'echo' => [
            'type' => Type::string(),
            'args' => [
                'message' => Type::nonNull(Type::string()),
            ],
            'resolve' => function ($root, $args) {
                return $root['prefix'] . $args['message'];
            }
        ],
    ],
]);
```

This code will generate this query:

```
type Query {
  echo(message: String!): String!
}
```

Zoom on Webonyx/GraphQL-PHP

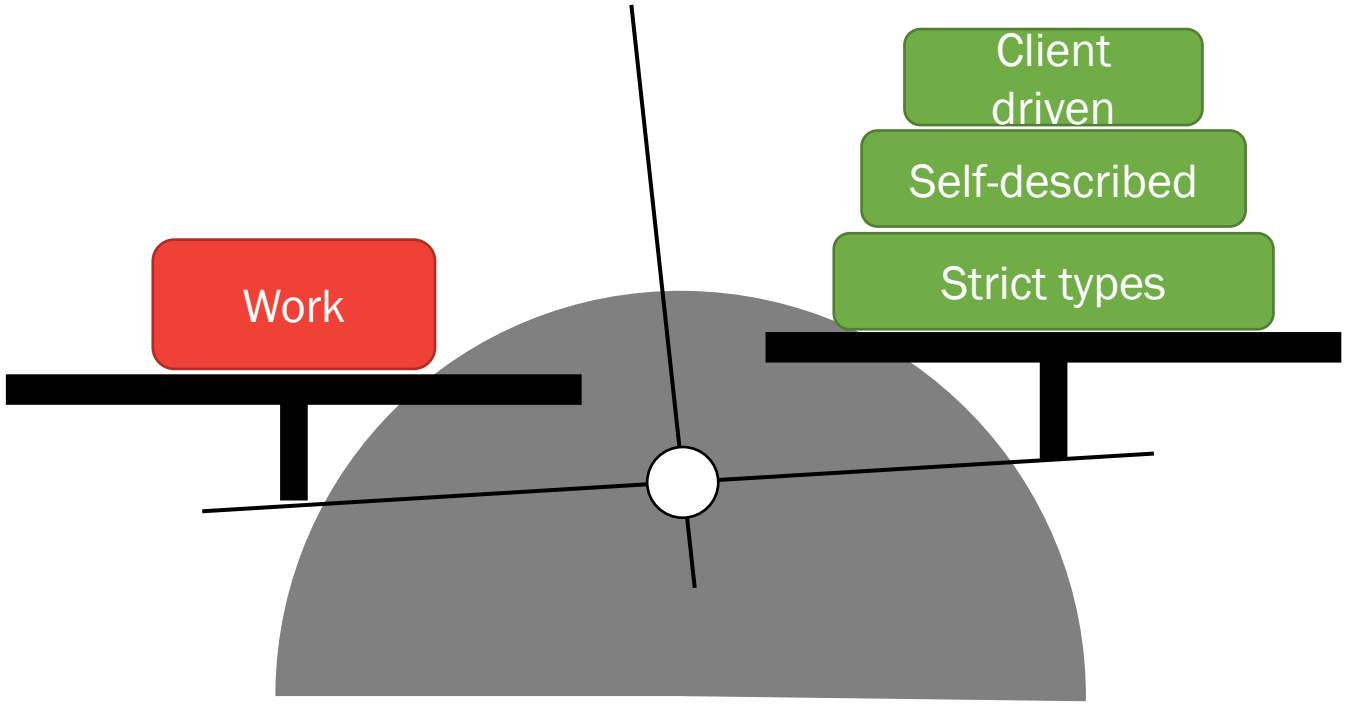
Actually resolving a query

```
$result = GraphQL::executeQuery(
    $schema,
    $queryString,
    $rootValue = null,
    $context = null,
    $variableValues = null,
    $operationName = null,
    $fieldResolver = null,
    $validationRules = null
);
```

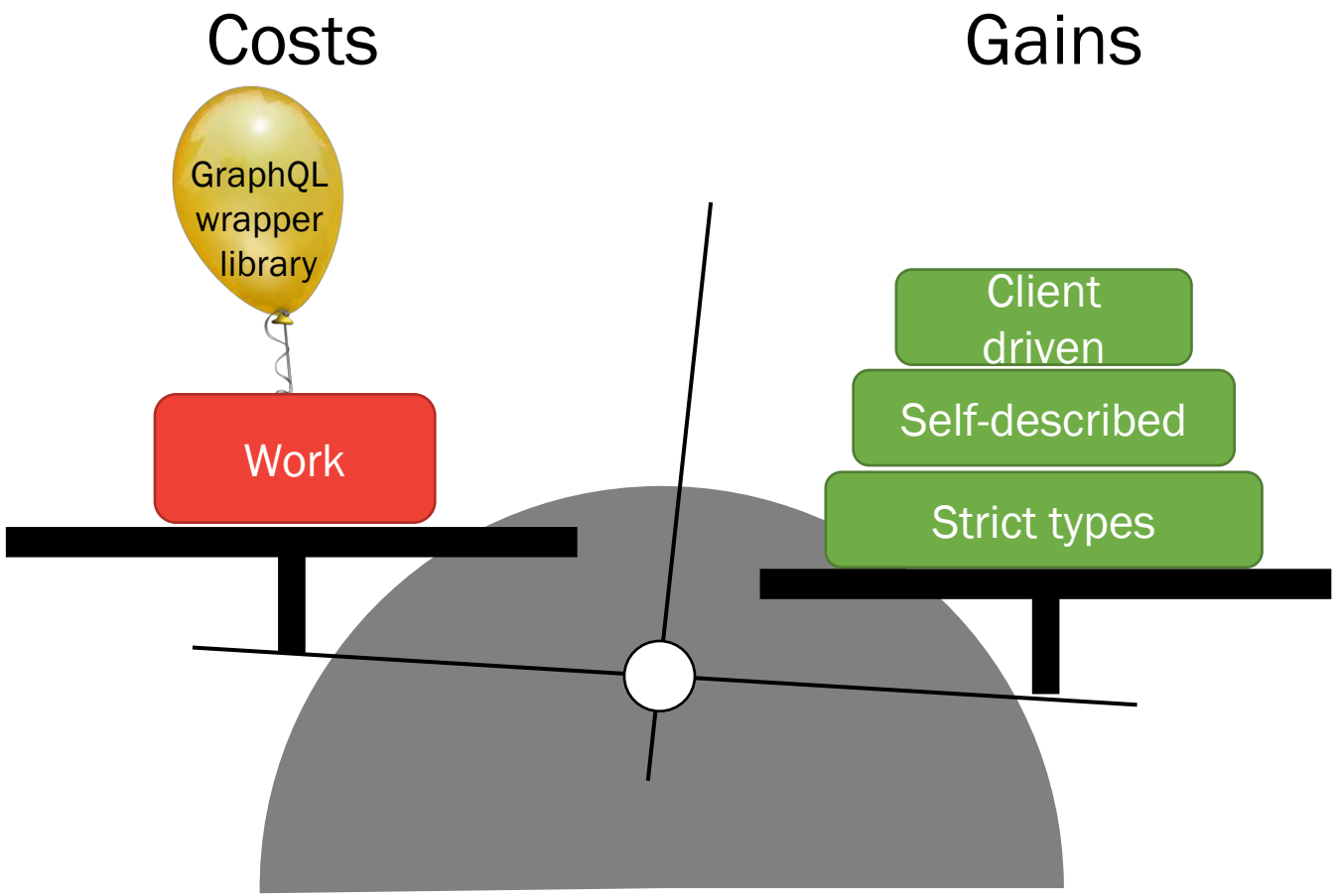
Costs VS benefits

Costs

Gains



You need a wrapper library



Choosing a library

Schema-first

- Design the GraphQL schema first
- Find a way to link it to your code

Code-first

- Design your domain code
- Generate the schema from the code

Choosing a library

Schema-first

- Design the GraphQL schema first
- Find a way to link it to your code

Overblog GraphQL bundle

Lighthouse

Railt

Siler

Code-first

- Design your domain code
- Generate the schema from the code

API Platform

GraphQLite

getpop/graphql

Schema-first: Lighthouse (Laravel)

```
type User {
  name: String!
  posts: [Post!]! @hasMany
}

type Post {
  title: String!
  author: User @belongsTo
}

type Query {
  me: User @auth
  posts: [Post!]! @paginate
}

type Mutation {
  createPost(
    title: String @rules(apply: ["required", "min:2"])
    content: String @rules(apply: ["required", "min:12"])
  ): Post @create
}
```

- Define the GraphQL schema first
- Annotate the schema with “directives”
- The directives are binding the schema to the ORM (Eloquent) directly

Notes:

- Very efficient…
- … but very tied to the ORM (Eloquent)
- Has support for subscriptions

Code-first: API Platform (Symfony)

```
/**
 * @ApiResource(
 *     attributes={
 *         "filters"={"offer.search_filter"}
 *     },
 *     graphql={
 *         "query"={
 *             "filters"={"offer.date_filter"}
 *         },
 *         "delete",
 *         "update",
 *         "create"
 *     }
 * )
 */
class Offer
{
    // ...
}
```

- Annotate your classes
- The GraphQL schema is generated from the annotations
- “REST” philosophy at the core of API Platform

Notes:

- Great if you want both a REST and a GraphQL API (you code it only once)
- Harder if you want fine grained control on the GraphQL schema
- I has support for subscriptions

Picking a GraphQL library

(disclaimer: probably biased view)

	Type	Framework	Subscriptions	Relay support	Comment
Webonyx	Both	All	✓	✓	Use this if you are building a tool / lib
Lighthouse	Schema-first	laravel	✓	✓	
API Platform	Code-first	Symfony	✓		
Overblog/GraphQL-Bundle	Schema-first	Symfony		✓	
GraphQLite	Code-first	All			
Siler	Schema-first	Siler	✓		
Railt	Schema-first	All			

- What is GraphQL?
- Why GraphQL?
- GraphQL type system
- The GraphQL ecosystem in PHP
- GraphQLite

IPC

International PHP Conference

The idea

Let’s imagine we want to do a simple “echo” query in PHP.

```
query {
  echo(message: "Hello World")
}
```

The idea

Using webonyx/GraphQL-PHP

```php
// We declare a "Query" type used to gather queries.
$queryType = new ObjectType([
    'name' => 'Query',
    'fields' => [
        // Let's add an "echo" field
        'echo' => [
            // This is the return type of the field
            'type' => Type::string(),
            // This is the list of arguments accepted by the field
            'args' => [
                'message' => Type::nonNull(Type::string()),
            ],
            // This is the method called when resolving the field.
            'resolve' => function ($root, $args) {
                return $root['prefix'] . $args['message'];
            }
        ],
    ],
]);
```

```
type Query {
    echo(message: String!): String
}
```

The idea

The same “echo” method in pure PHP

```
function echoMsg(string $message): string
{
    return $message;
}
```

The idea

The same “echo” method in pure PHP

Return type

Arguments

Query name

```
function echoMsg(string $message): string
{
    return $message;
}
```

Resolver

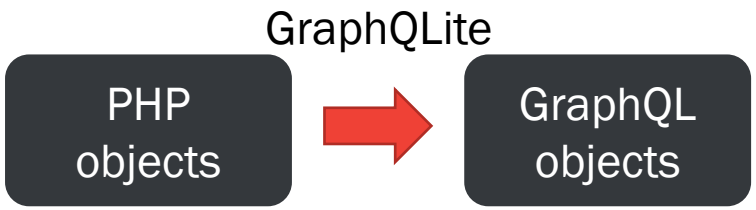
The idea

The same “echo” method in pure PHP

```
#[Query]
function echoMsg(string $message): string
{
    return $message;
}
```

The idea

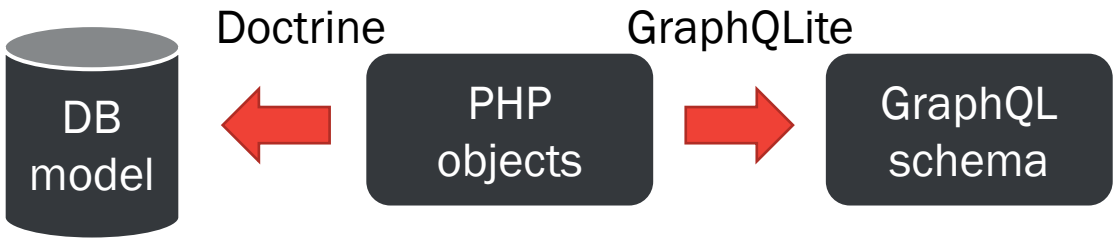
- PHP is **already typed.**
- We should be able to get types from PHP and convert them to a GraphQL schema



Works well with Doctrine

Bonus:

- It plays nice with Doctrine ORM too



- (it also plays nice with Eloquent and TDBM)

GraphQLite

GraphQLite is:

- Framework agnostic
  - Symfony bundle and Laravel package available
- PHP 7.2+
- Based on Webonyx/GraphQL-PHP

Doctrine annotations and PHP 8 attributes

GraphQLite support both “old style” annotations (using doctrine/annotations) and the brand new PHP 8 attributes

With PHP 8 attributes

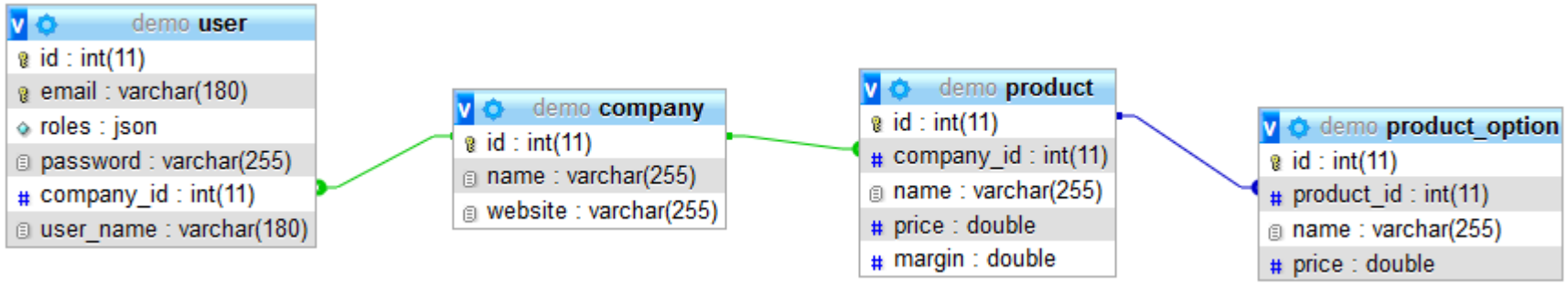
```php
#[Query]
function echoMsg(string $message): string
{
    return $message;
}
```

With Doctrine annotations

```php
/**
 * @Query
 */
function echoMsg(string $message): string
{
    return $message;
}
```

Demo time!

Our playground: a marketplace!



Mutations

Use a mutation to change the state of your application.

Mutations are similar to queries, only the annotation changes.

It means mutations MUST return a value.

```
class ProductController
{
    // …

    #[Mutation]
    public function createProduct(
            string $name,
            float $price,
            int $companyId): Product
    {
        $product = new Product($name,
            $this->companyRepository->
                find($companyId));
        $product->setPrice($price);
        $this->em->persist($product);
        $this->em->flush();
        return $product;
    }
}
```

Authorization

```php
use TheCodingMachine\GraphQLite\Annotations\Right;

class UserController
{
    //…
    /**
     * @return User[]
     */
    #[Query]
    #[Right("ROLE_ADMIN")]
    public function users(?string $search)
    {
        return new $this->userRepository->search($search);
    }
}
```

@Right annotations can be used in a **@Field** too!

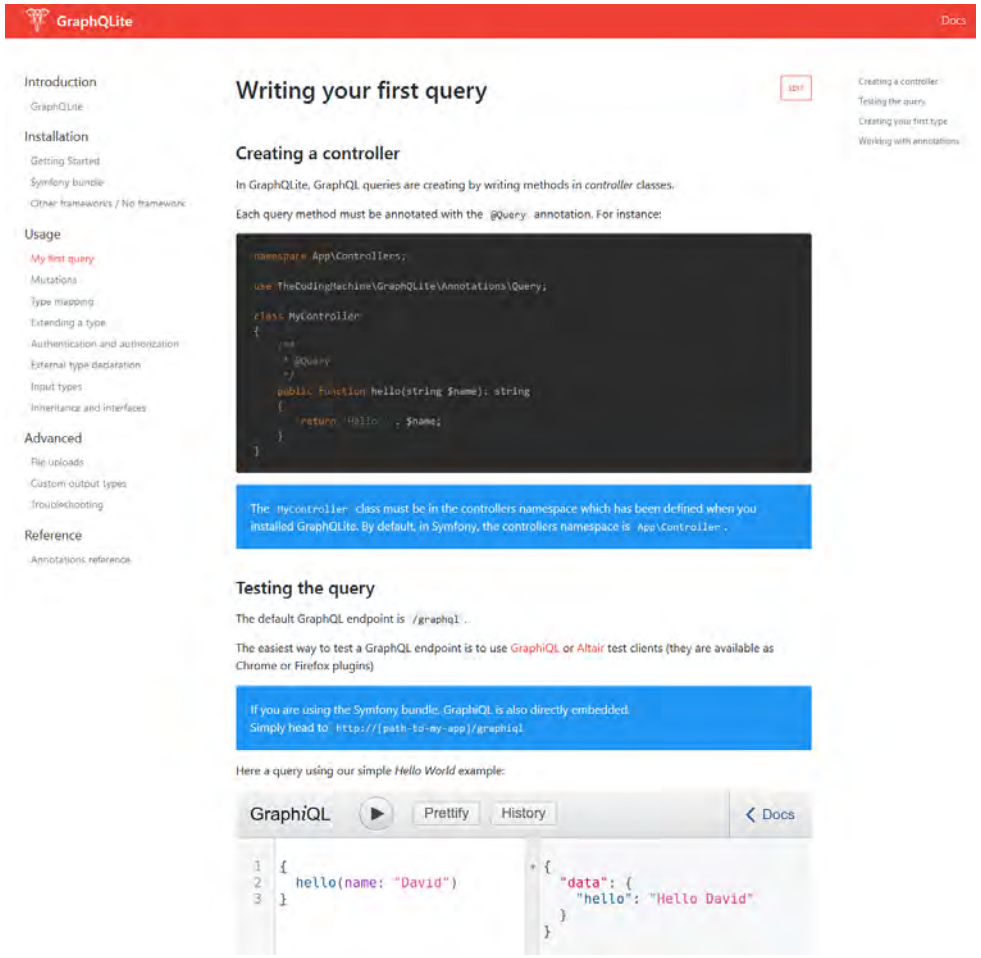
Fine grained authorization

```
#[Type]
class User implements UserInterface, Serializable
{
    //…

    #[Field]
    #[Security("this.getCompany() == user.getCompany()", failWith=null)]
    public function getEmail(): ?string
    {
        return $this->email;
    }
}
```

Autowiring

```
class Product
{
    // …

    #[Field]
    public function getVat(
        #[Autowire] VatServiceInterface $vatService
        ): float
    {
        return $vatService->getVat($this);
    }
}
```

More about GraphQLite



- Pagination
- Authentication
- Security
- Autowiring
- Validation
- Enum support
- File uploads
- Union types
- Declaring a type without annotating the PHP class
- DateTime type mapping
- Inheritance and interfaces

Everything is documented at:

https://graphqlite.thecodingmachine.io

David Négrier

@david_negrier

@moufmouf

https://graphqlite.thecodingmachine.io

Questions?

More cool stuff:

- https://www.thecodingmachine.com/open-source/
- https://thecodingmachine.io

Want to talk to me after the conference?

- I'll be available on https://play.workadventu.re