GraphQLite workshop

PHP UK CONFERENCE

GraphQLite workshop

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

PHP UK
CONFERENCE

« me »

David Négrier
aka:

moufmouf

@david_negrier

joind.in/user/moufmouf

We are hiring!

TheCodingMachine

CTO & co-founder
@TheCodingMachine

PSR-11 co-editor
GraphQLite author

But also Packanalyst, Mouf, TDBM…

http://bit.ly/phpuk-graphqlite Download the workshop!

Setting up your environment

PHP UK CONFERENCE

Setting up the test project

- Download

  http://bit.ly/phpuk-graphqlite

- Or copy the project from a USB key (better!)

Our test project

- Today, we will work on a marketplace!

- For this demo, our stack will be:
  - Symfony
  - Doctrine ORM
  - GraphQLite
  - Docker
  - React
  - Apollo

http://bit.ly/phpuk-graphqlite *Download the workshop!*

Our test project

From Github

```
$ docker-compose up
```

⚠ (Beware, there is a 500MB download!)

From a USB key (no network connection)

Copy files

Run
```
$ ./install-images.sh
```
(or `install-images.bat`)

Unzip project:
```
$ unzip graphqlite-demo-phpuk.zip
```

Start project:
```
$ docker-compose up
```

Install Altair GraphQL client

http://bit.ly/phpuk-graphqlite ← *Download the workshop!*

Our test project

- If you downloaded from Github:

```
$ docker-compose up
```

- From a USB key:

Copy files

Run
```
$ ./install-images.sh
```
(or install-images.bat)

Then

```
$ docker-compose up
```

http://bit.ly/phpuk-graphqlite *Download the workshop!*

GraphQLite workshop

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

PHP UK CONFERENCE

GraphQL ?

- GraphQL is a **protocol**

http://bit.ly/phpuk-graphqlite

Download the workshop!

GraphQL ?

- GraphQL is a **<u>protocol</u>**

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

http://bit.ly/phpuk-graphqlite

Download the workshop!

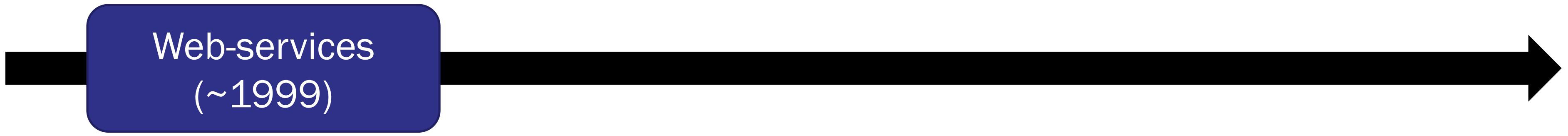
GraphQL ?

- GraphQL is a **protocol**

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

http://bit.ly/phpuk-graphqlite *Download the workshop!*

A bit of history

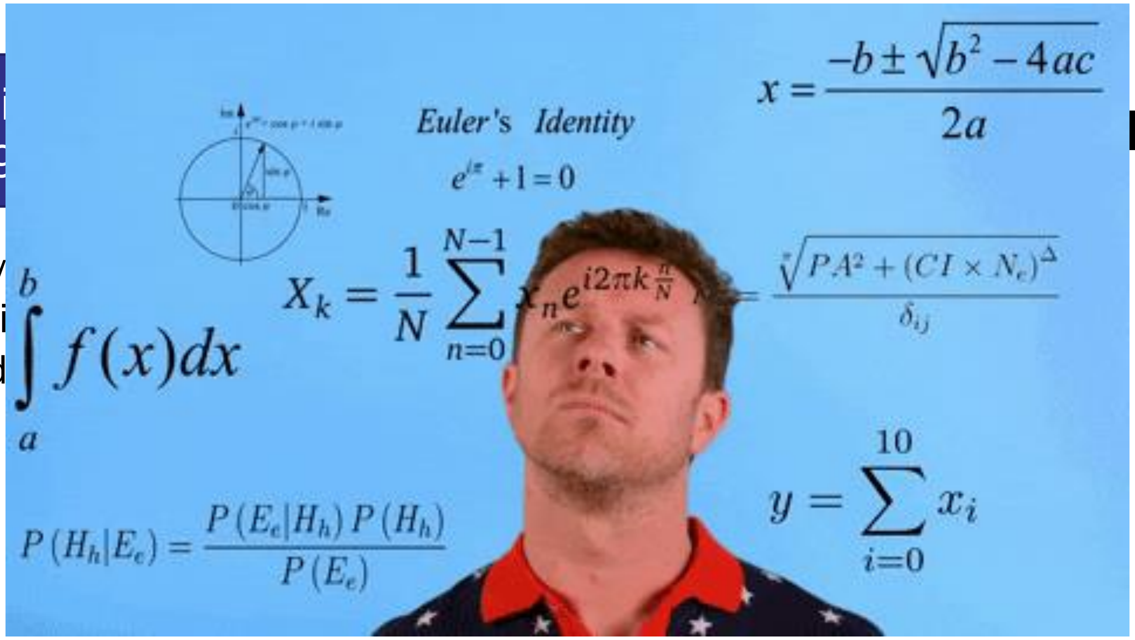


- ✓ Strongly typed
- ✓ Self-describing (WSDL)
- × XML-based

A bit of history

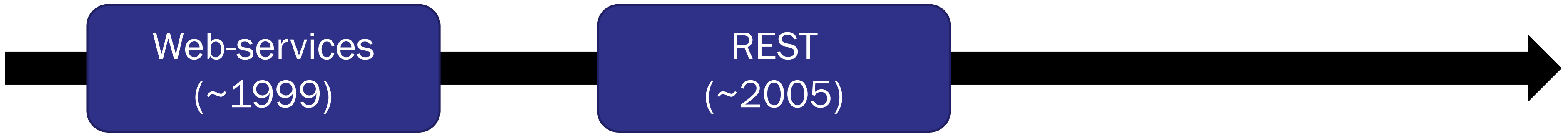
Web-servi
(~1999

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


Euler's Identity
$e^{i\pi}+1=0$
$x=\frac{-b\pm\sqrt{b^2-4ac}}{2a}$
$X_k=\frac{1}{N}\sum_{n=0}^{N-1}x_n e^{i2\pi k\frac{n}{N}}$
$\int_a^b f(x)dx$
$\frac{\sqrt{PA^2+(CI\times N_c)^\Delta}}{\delta_{ij}}$
$y=\sum_{i=0}^{10}x_i$
$P(H_h|E_e)=\frac{P(E_e|H_h)P(H_h)}{P(E_e)}$

http://bit.ly/phpuk-graphqlite Download the workshop!

A bit of history



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

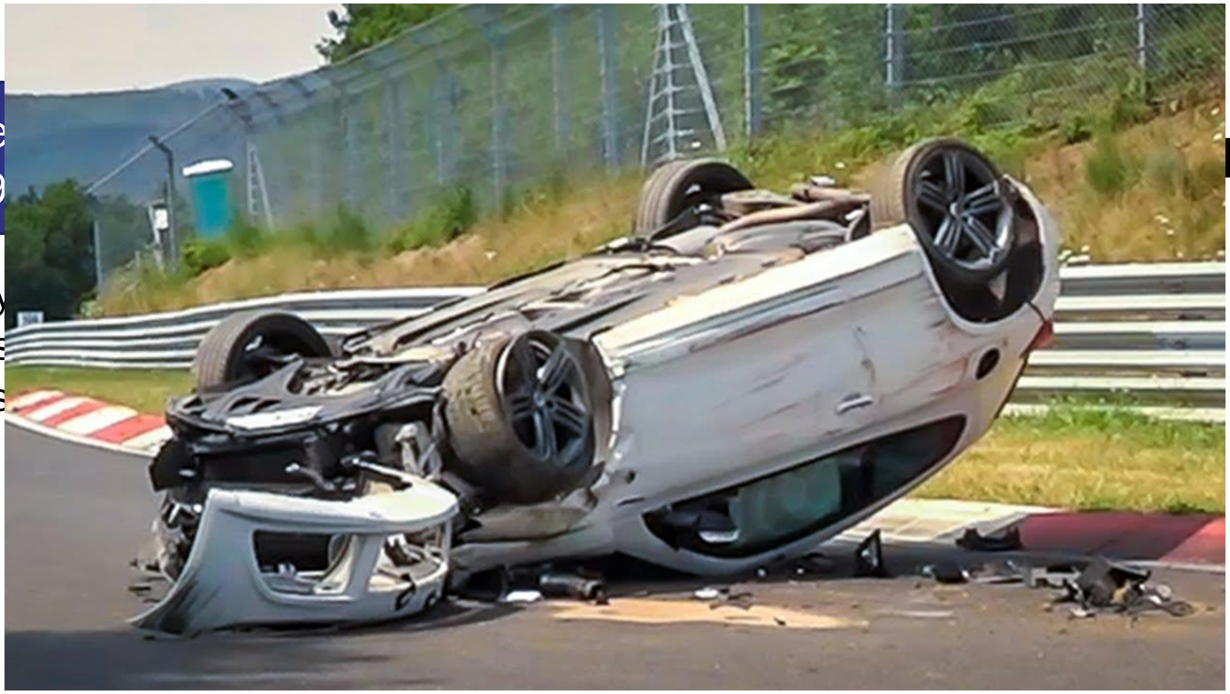
- ✗ Weakly typed
- ✗ Non self-describing (still OpenAPI for doc)
- ✓ Mostly JSON based

http://bit.ly/phpuk-graphqlite ← *Download the workshop!*

A bit of history

Web-se
(~19

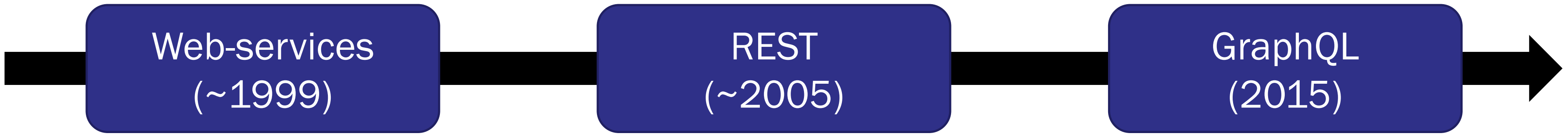
- ✓ Strongly
- ✓ Self-des
- × XML-bas



http://bit.ly/phpuk-graphqlite

Download the workshop!

A bit of history



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

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

- ✓ Strongly typed
- ✓ Self-describing
- ✓ JSON based

- ✓ ✓ + Client driven queries

http://bit.ly/phpuk-graphqlite *Download the workshop!*

A bit of history

SOAP/WSDL REST

GraphQL

http://bit.ly/phpuk-graphqlite

Download the workshop!

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*.

Why GraphQL?

PHP UK
CONFERENCE

What problem does GraphQL solve?

- Your API often <u>changes</u>

- Common problems:
  - You develop a new feature but your API does not exactly respond to your needs.
  - Each time you consume your API on the frontend, you need to change it in the backend

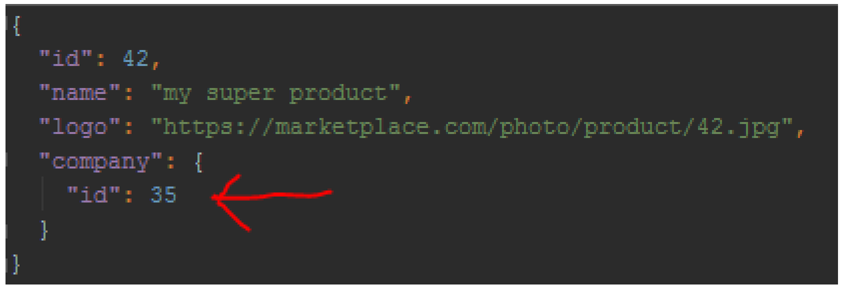
Download the workshop!

What problem does GraphQL solve?

- *For instance*: you are developing a marketplace. You need a page to display a product, along some company information.

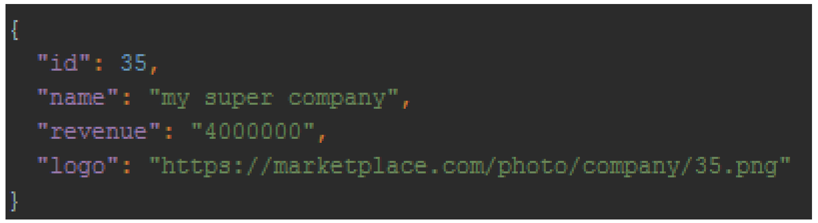
REST

/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"
}
```

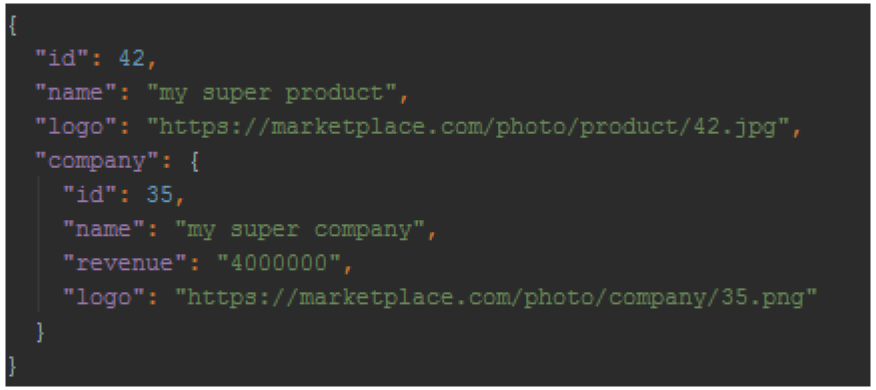
http://bit.ly/phpuk-graphqlite

Download the workshop!

What problem does GraphQL solve?

- Alternative, still REST

/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"
  }
}
```

REST

- But what if some pages don’t need the company details?

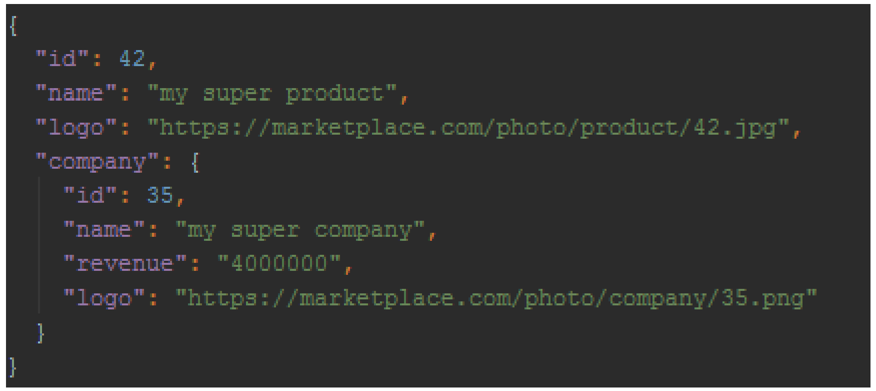
http://bit.ly/phpuk-graphqlite

Download the workshop!

What problem does GraphQL solve?

- Yet another alternative, still REST

/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"
  }
}
```

Flags hell 😳! Probably one flag per consumer of the API

REST

http://bit.ly/phpuk-graphqlite

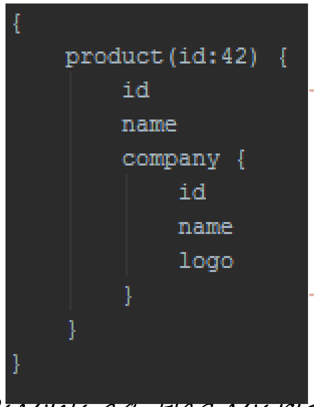
Download the workshop!

What problem does GraphQL solve?

- GraphQL **to the rescue**!

- GraphQL is a *paradigm shift*.
- The **client** asks for the list of fields it wants.

GET /graphql?query=



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

Single endpoint

The name of the query is « products »

List of fields requested

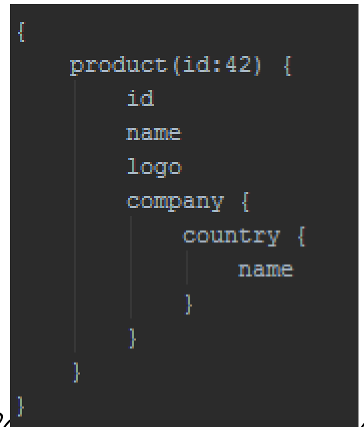
http://bit.ly/phpuk-graphqlite

Download the workshop!

What problem does GraphQL solve?

- GraphQL **to the rescue**!
- Another request of the same query with a different set of fields

GET /graphql?query=



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

No need to change the code on the server-side! All this data in one API call!

http://bit.ly/phpuk-graphqlite

Download the workshop!

GraphQLite workshop

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

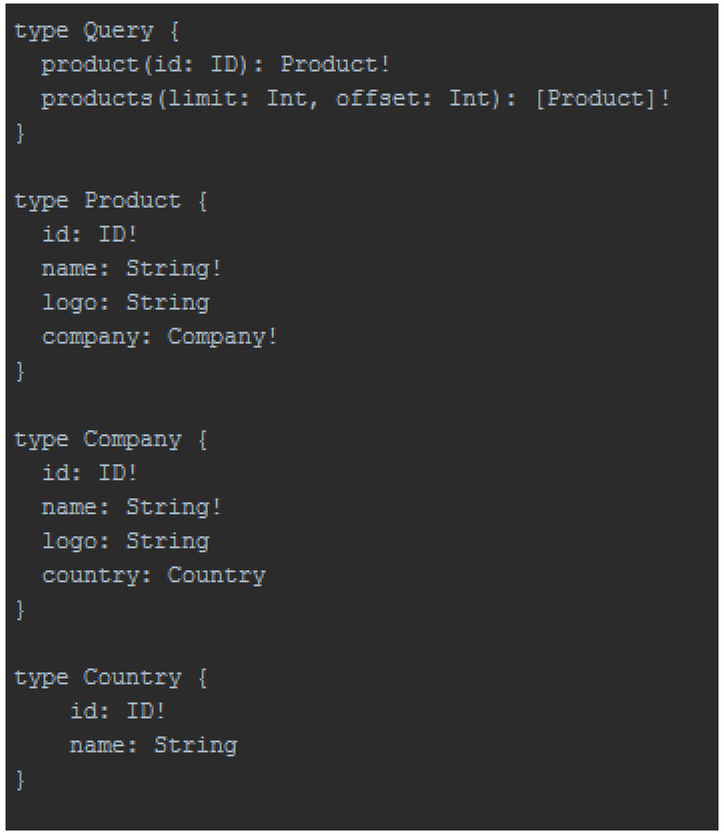
PHP UK CONFERENCE

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
}
```

http://bit.ly/phpuk-graphqlite *Download the workshop!*

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
}
```

http://bit.ly/phpuk-graphqlite *Download the workshop!*

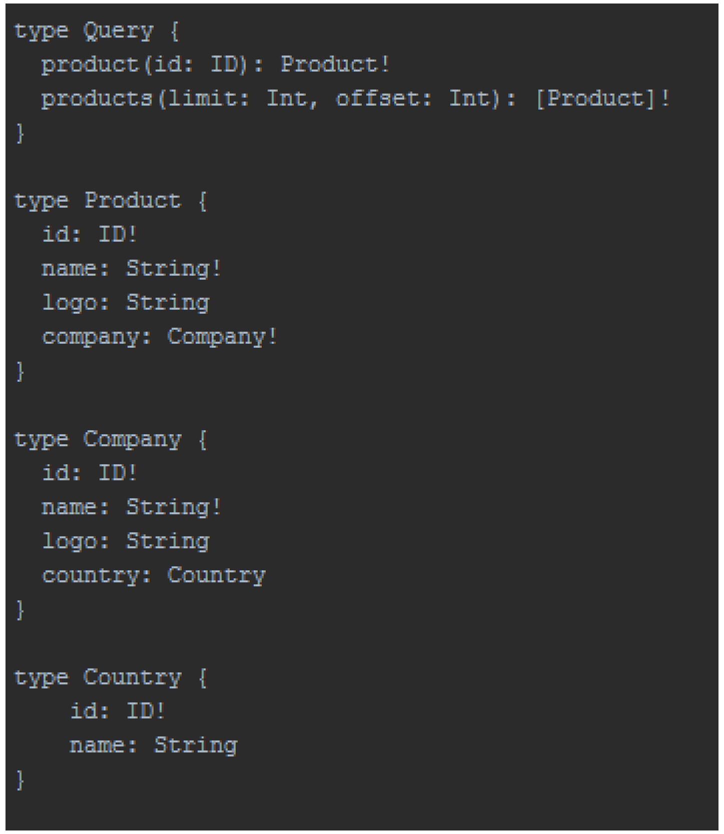
Types

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.



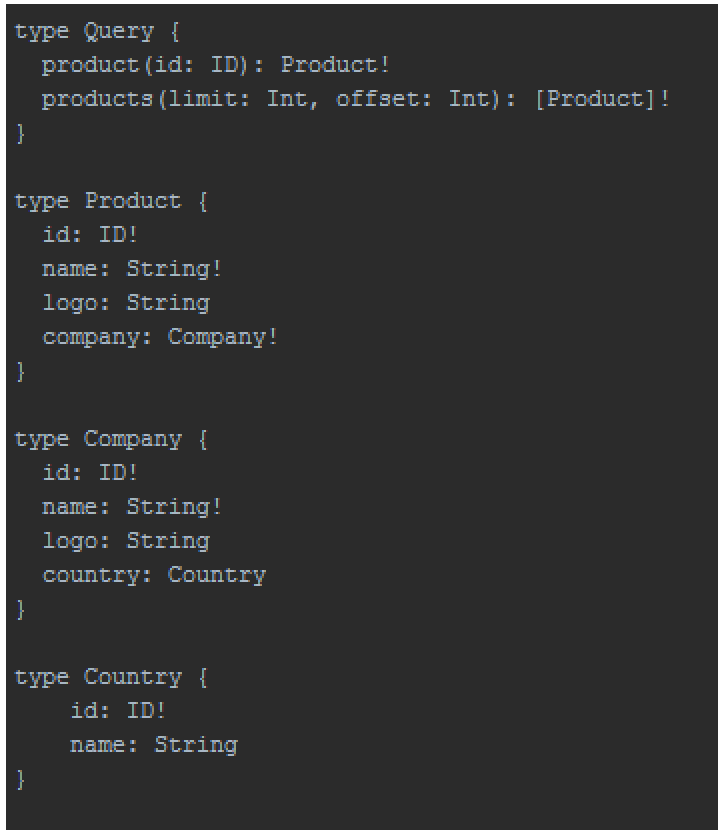
```
type Query {
  product(id: ID): Product!
  products(limit: Int, offset: Int): [Product]!
}

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)



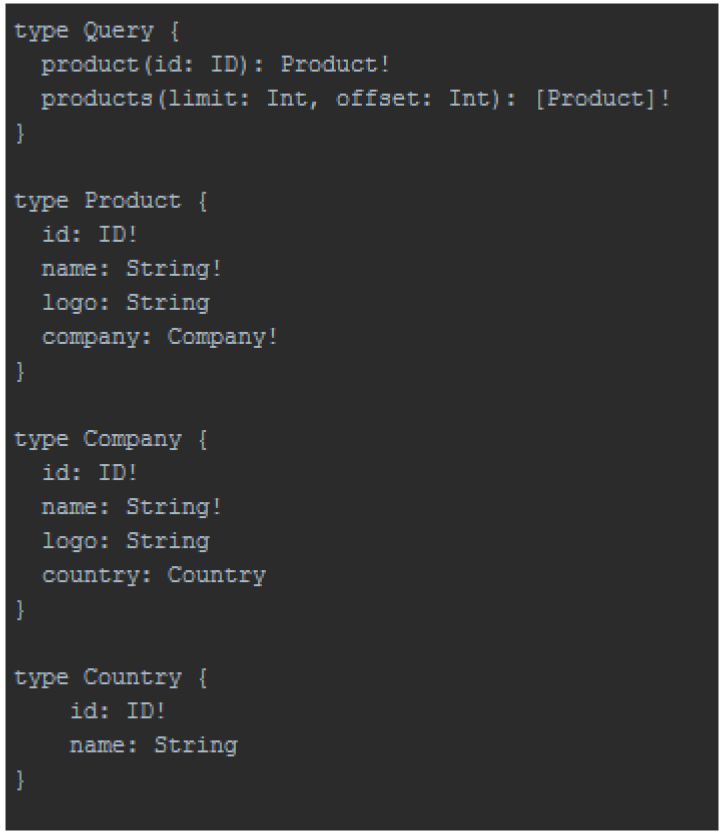
```
type Query {
  product(id: ID): Product!
  products(limit: Int, offset: Int): [Product]!
}

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
}
```

http://bit.ly/phpuk-graphqlite *Download the workshop!*

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
}
```

Download the workshop!

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).

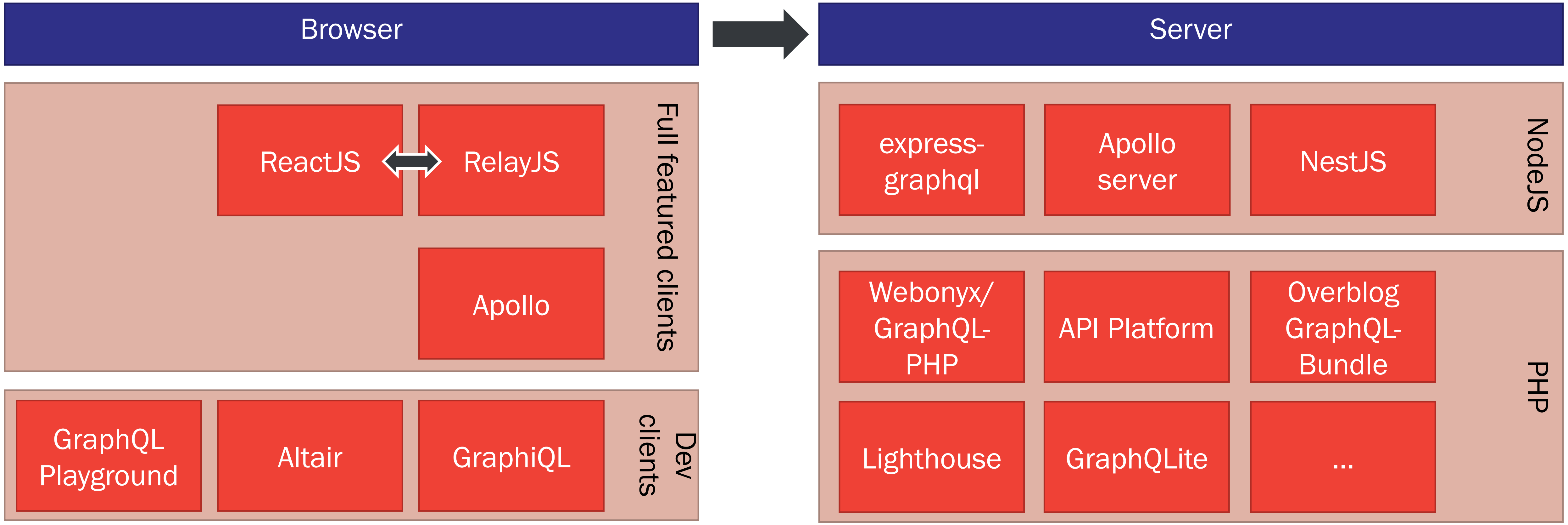
GraphQL can also do **mutations** (to change the state of the DB)

GraphQLite workshop

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

PHP UK
CONFERENCE

Ecosystem (a small part of...)



http://bit.ly/phpuk-graphqlite Download the workshop!

Zoom on GraphQL in PHP

Core library

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

Wrapper library

- High level
- Opiniated
- Easy to use

http://bit.ly/phpuk-graphqlite *Download the workshop!*

Zoom on GraphQL in PHP

Core library

Wrapper library

- webonyx/graphql-php
  - De-facto standard in PHP
- Youshido/GraphQL
  - ☹ Abandonned ☹
- Railt
  - In active development, no solid doc yet

http://bit.ly/phpuk-graphqlite

Download the workshop!

Zoom on GraphQL in PHP

Core library

Wrapper library

- API Platform (Symfony)
- Overblog GraphQL Bundle (Symfony)
- Lighthouse (Laravel)
- getpop/graphql (Wordpress)
- drupal/graphql (Drupal)
- ... and now **GraphQLite**

http://bit.ly/phpuk-graphqlite

Download the workshop!

Zoom of Webonyx/GraphQL-PHP

Define a type

```
$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]
}
```

http://bit.ly/phpuk-graphqlite

Download the workshop!

Zoom of 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'];
            }
        ],
    ],
]);
```

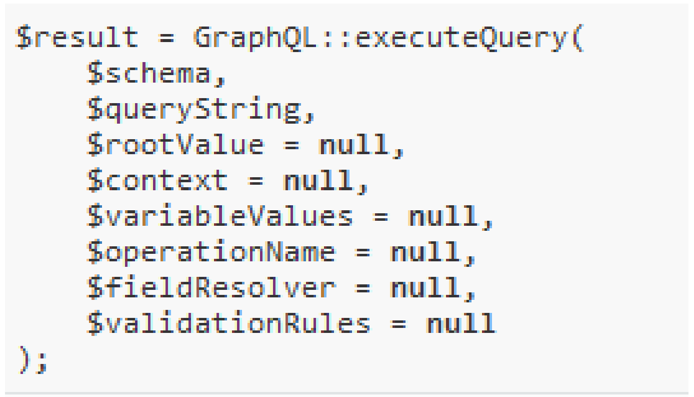
This code will generate this query:

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

Download the workshop!

Zoom of Webonyx/GraphQL-PHP

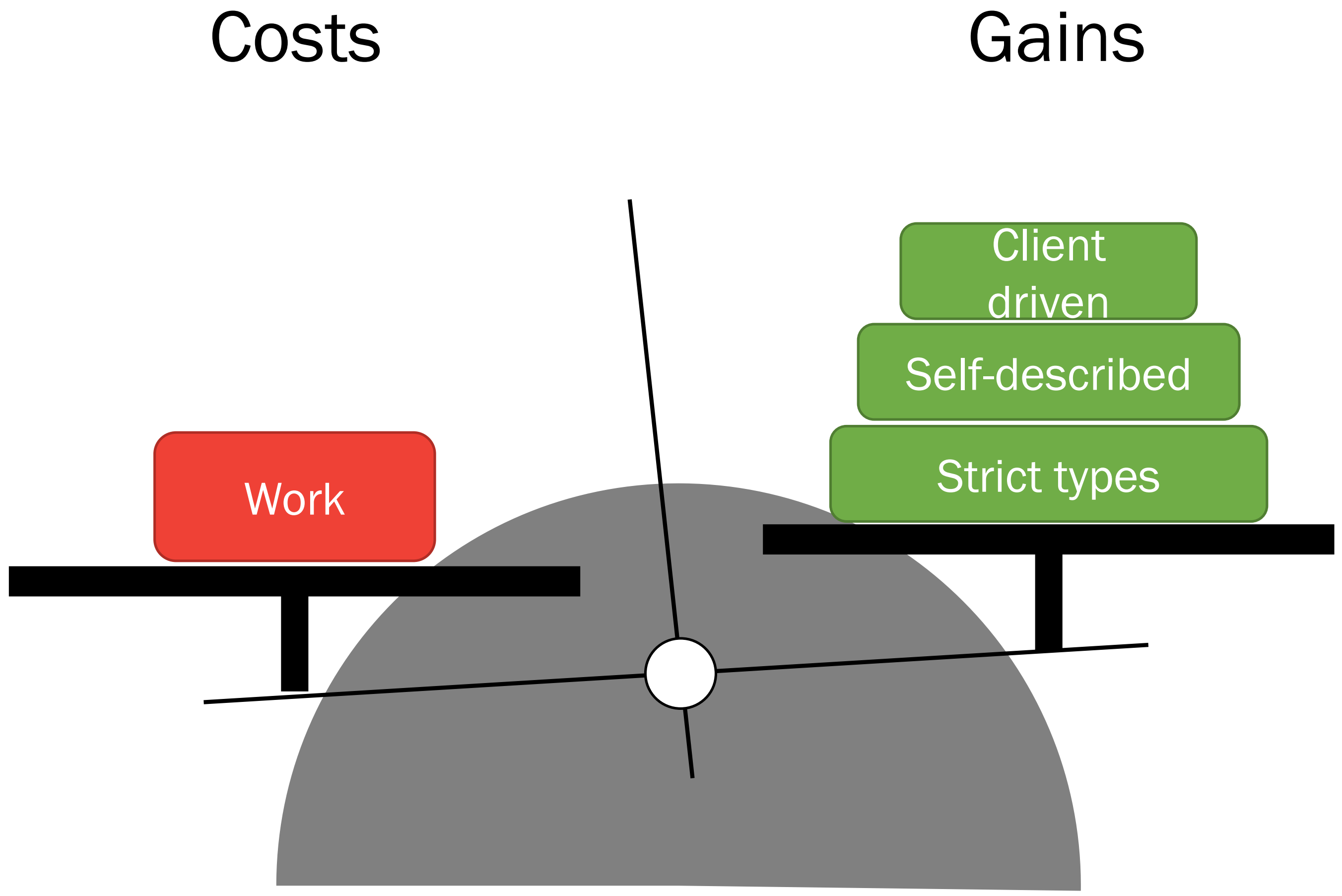
Actually resolving a query



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

http://bit.ly/phpuk-graphqlite *Download the workshop!*

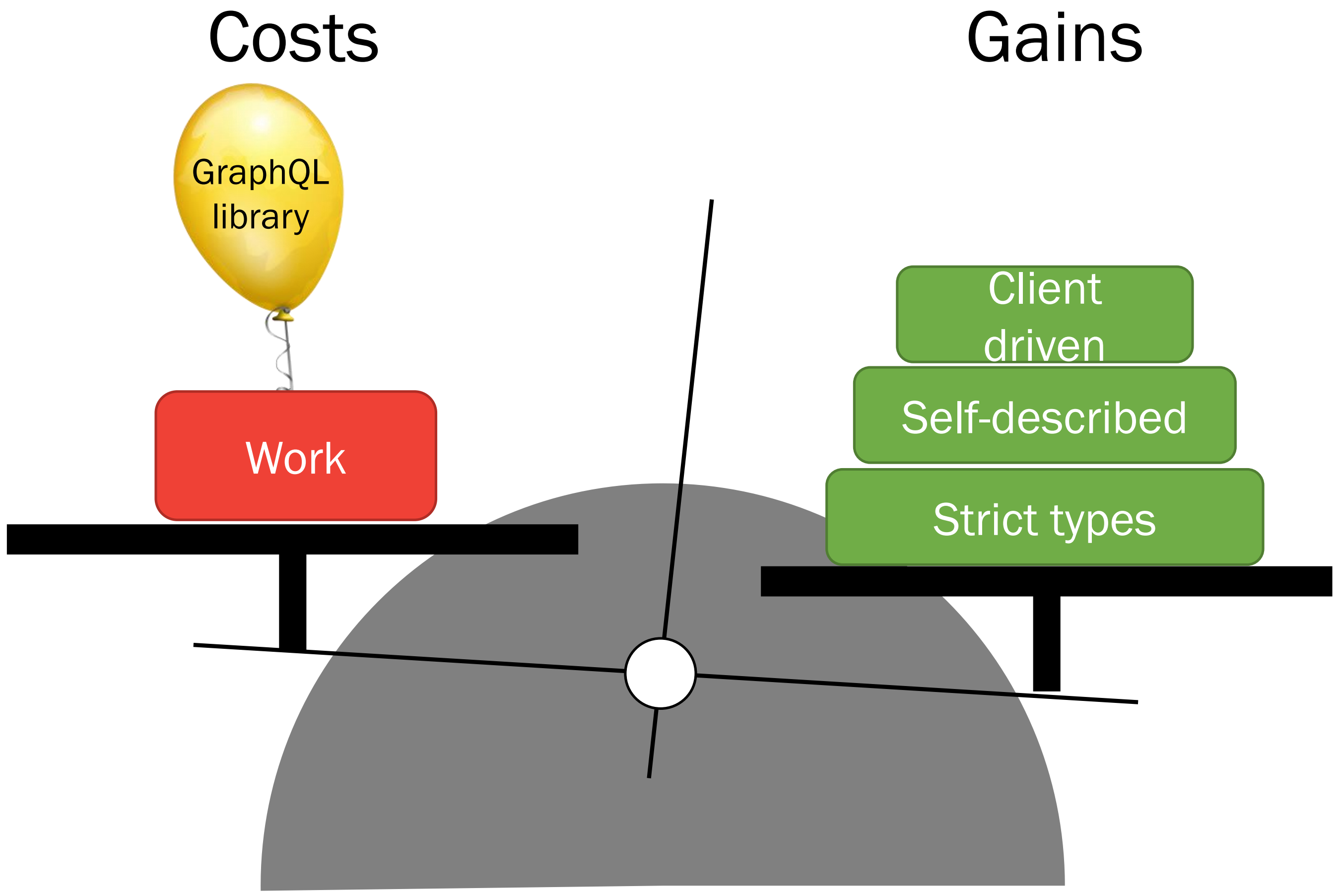
Costs VS benefits



http://bit.ly/phpuk-graphqlite

Download the workshop!

You need a wrapper library



http://bit.ly/phpuk-graphqlite

Download the workshop!

Strategies

Schema-first	Code-first
• Design the GraphQL schema first • Find a way to link it to your code	• Design your domain code • Generate the schema from the code

http://bit.ly/phpuk-graphqlite

Download the workshop!

Strategies

Schema-first

- Overblog GraphQL Bundle
- Lighthouse
- Railt

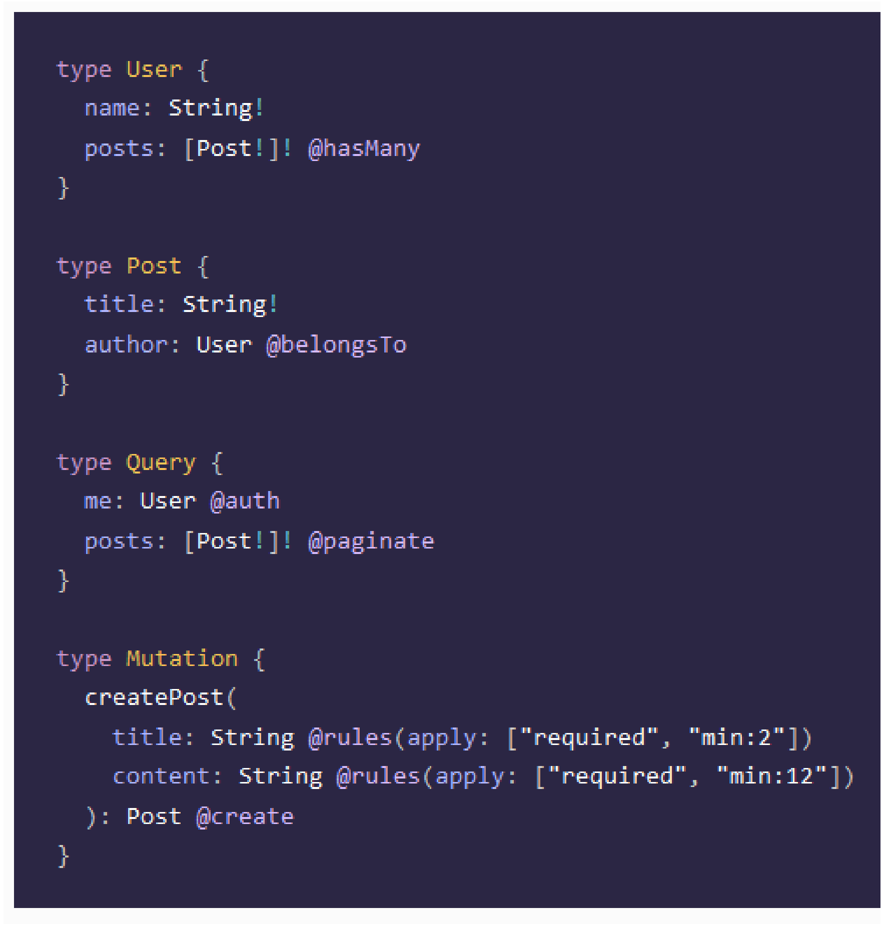
Code-first

- getpop/graphql
- API Platform
- GraphQLite

http://bit.ly/phpuk-graphqlite

Download the workshop!

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 Eloquent directly

Notes:

- Very tied to Eloquent
- Has support for subscriptions

http://bit.ly/phpuk-graphqlite ← *Download the workshop!*

Code-first: API Platform (Symfony)

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

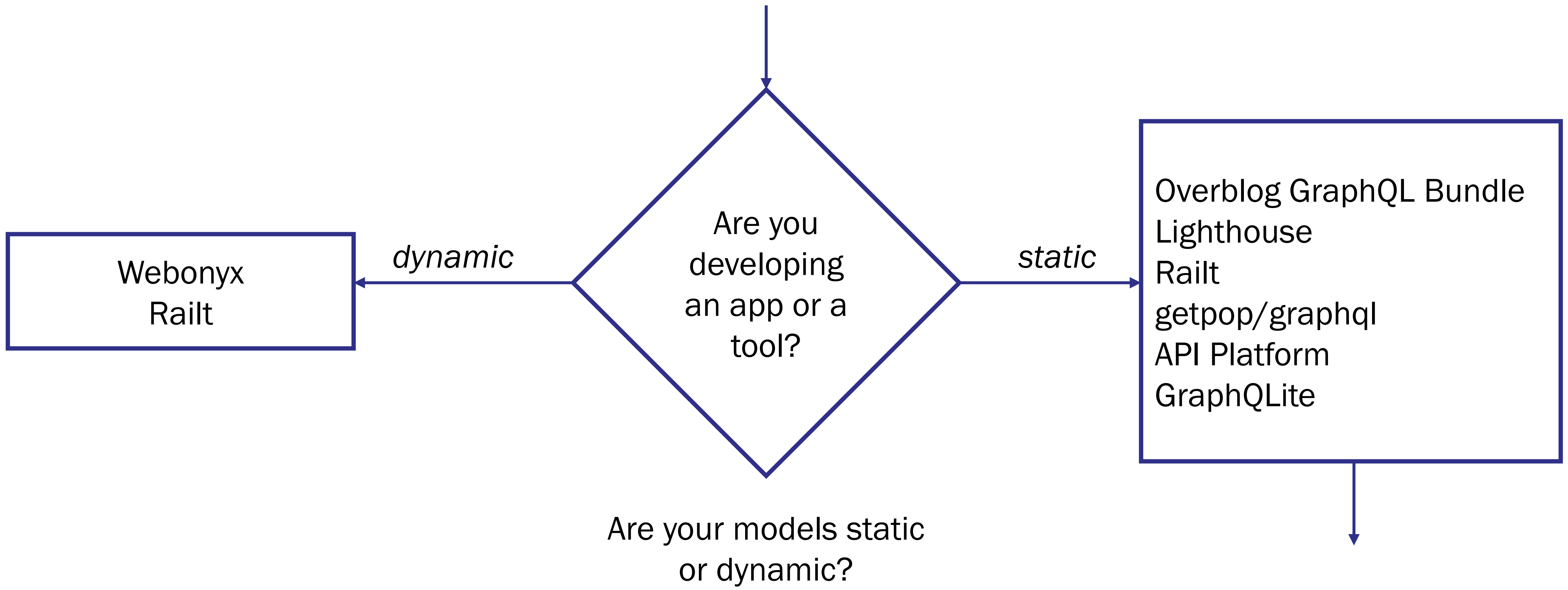
- Annotate your classes
- The GraphQL schema is generated from the annotations
- “REST” philosophy at the core of API Platform

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
- Support for subscriptions is coming

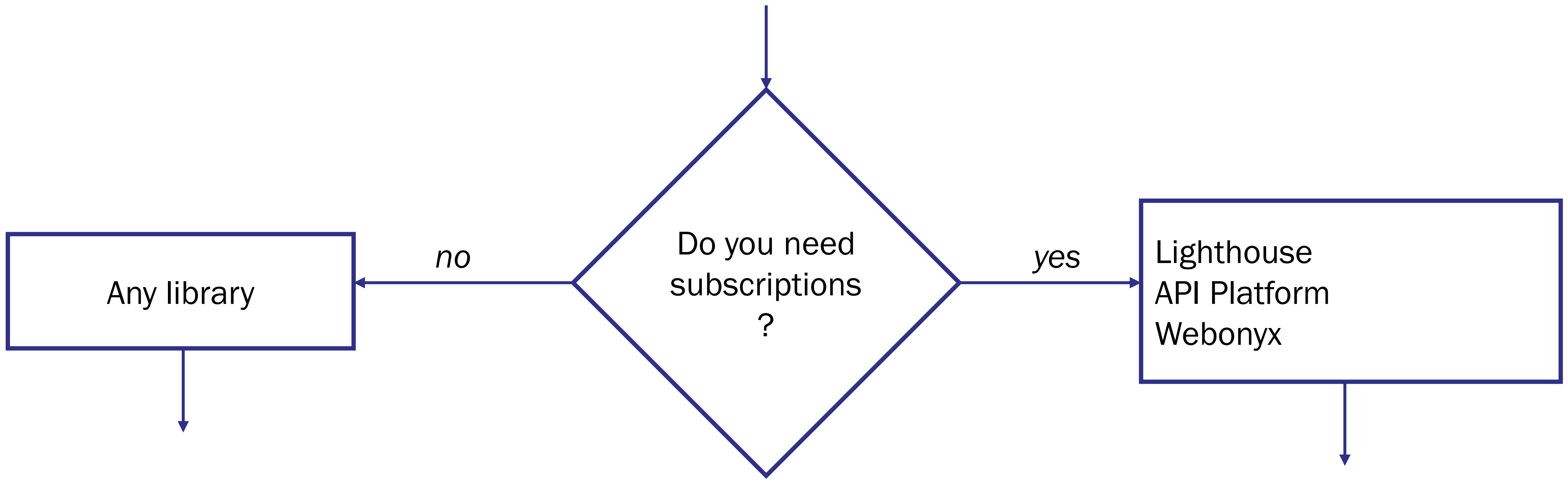
http://bit.ly/phpuk-graphqlite *Download the workshop!*

Picking a GraphQL library
(disclaimer: probably biased view)

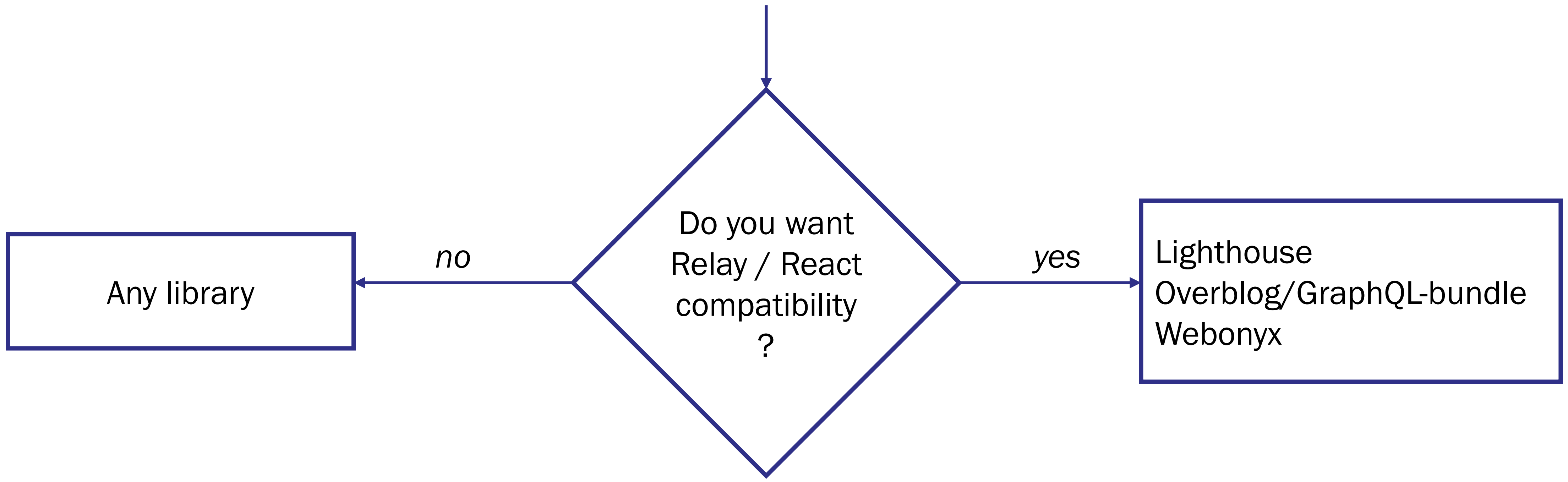


http://bit.ly/phpuk-graphqlite *Download the workshop!*

Picking a GraphQL library
(disclaimer: probably biased view)



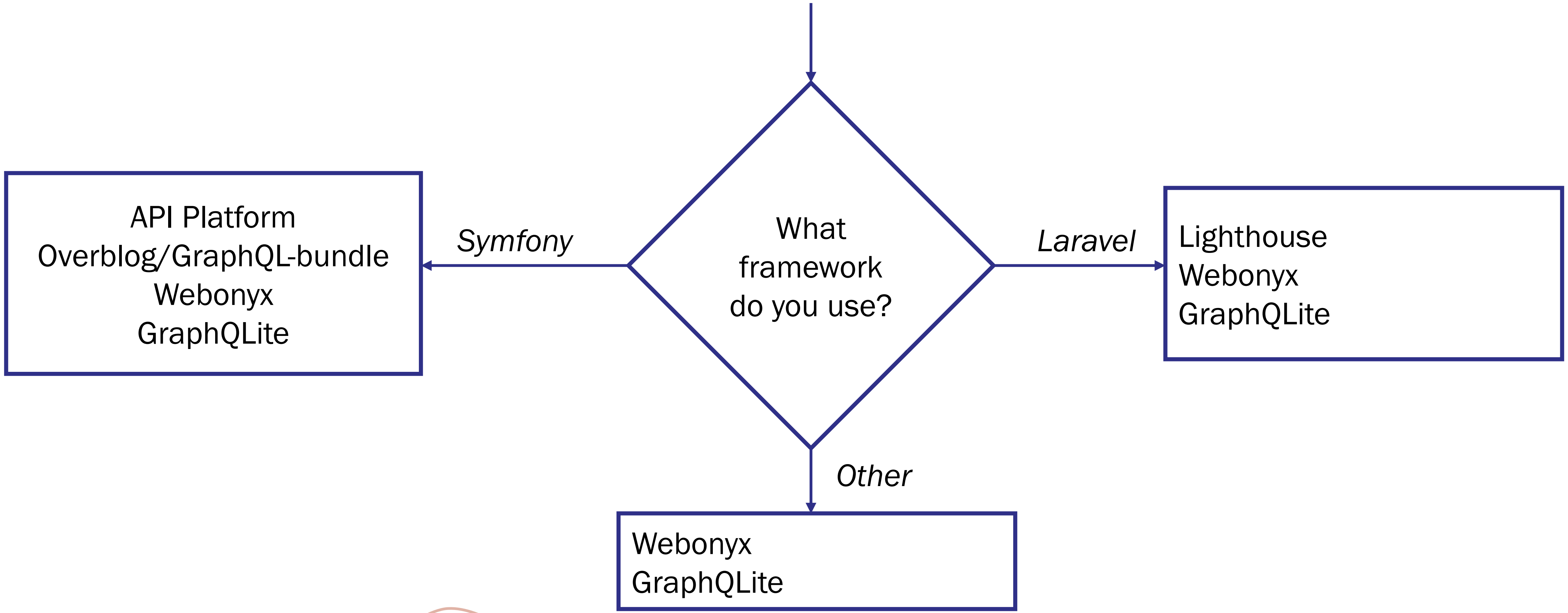
Picking a GraphQL library
(disclaimer: probably biased view)



http://bit.ly/phpuk-graphqlite

Download the workshop!

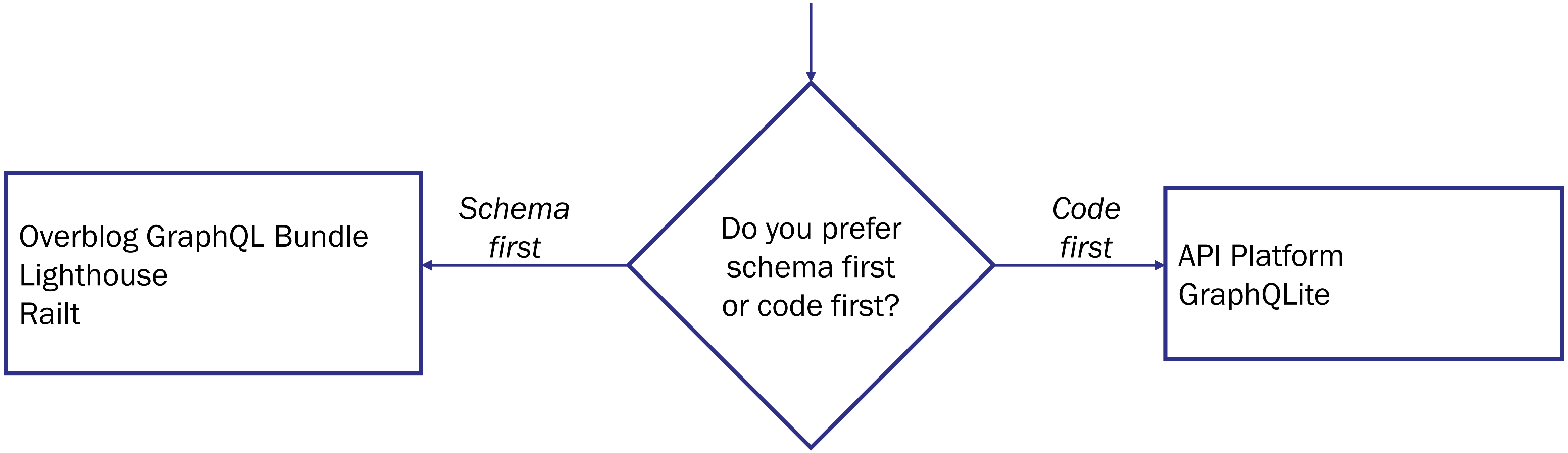
Picking a GraphQL library
(disclaimer: probably biased view)



http://bit.ly/phpuk-graphqlite

Download the workshop!

Picking a GraphQL library
(disclaimer: probably biased view)



http://bit.ly/phpuk-graphqlite Download the workshop!

GraphQLite workshop

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

PHP UK CONFERENCE

The idea

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

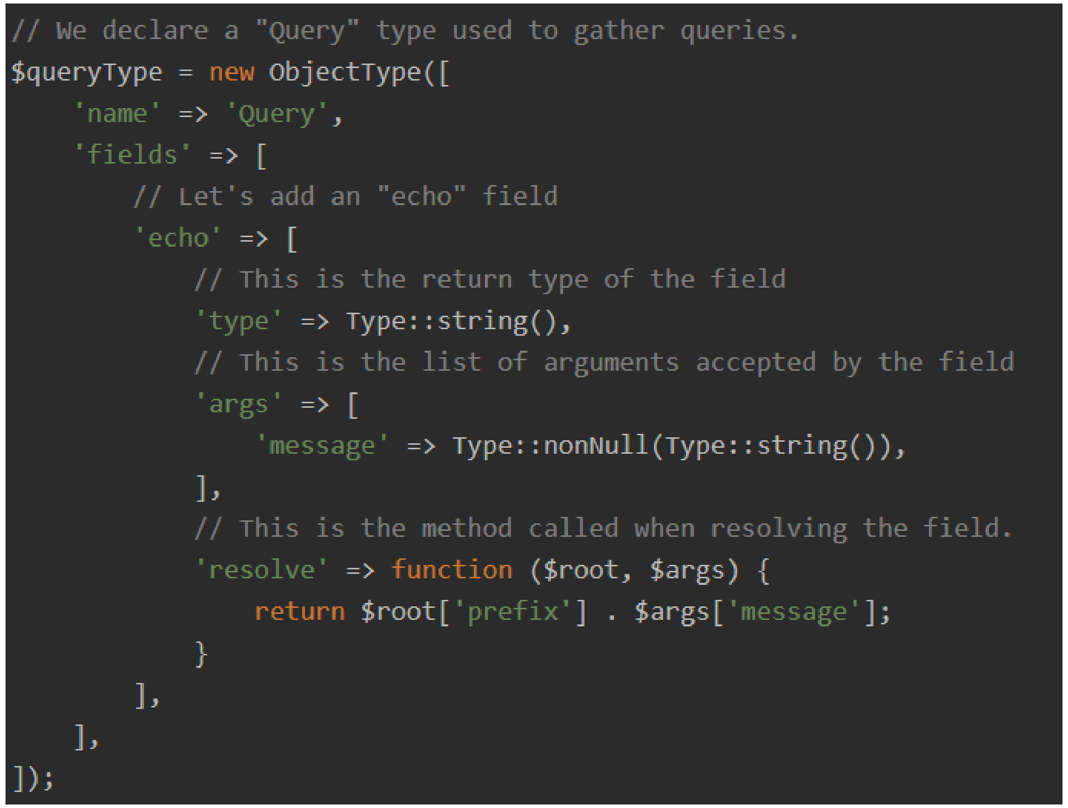
```
query {
  echo(message: "Hello World")
}
```

http://bit.ly/phpuk-graphqlite

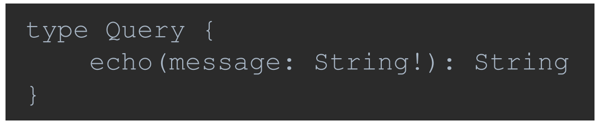
Download the workshop!

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
}
```

http://bit.ly/phpuk-graphqlite

Download the workshop!

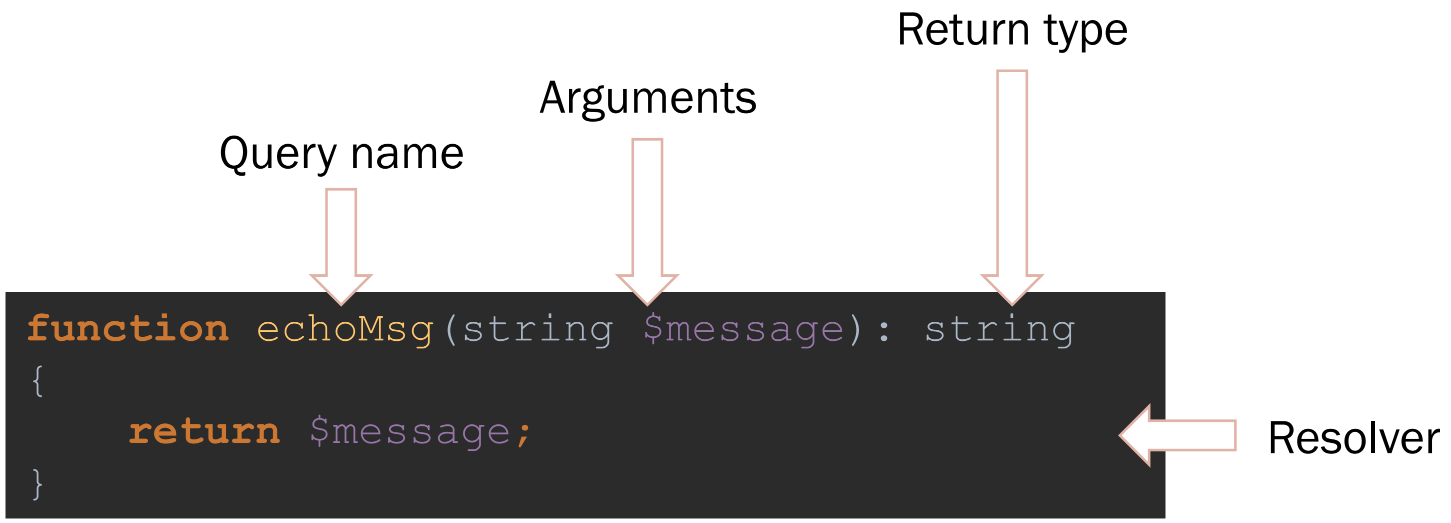
The idea

The same “echo” method in pure PHP

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

The idea

The same "echo" method in pure PHP



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

http://bit.ly/phpuk-graphqlite *Download the workshop!*

The idea

The same “echo” method in pure PHP

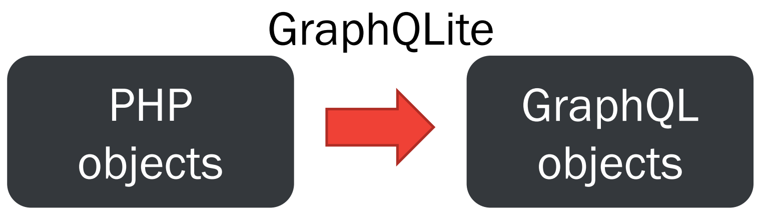
```
/**
 * @Query
 */
function echoMsg(string $message): string
{
    return $message;
}
```

http://bit.ly/phpuk-graphqlite

Download the workshop!

The idea

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



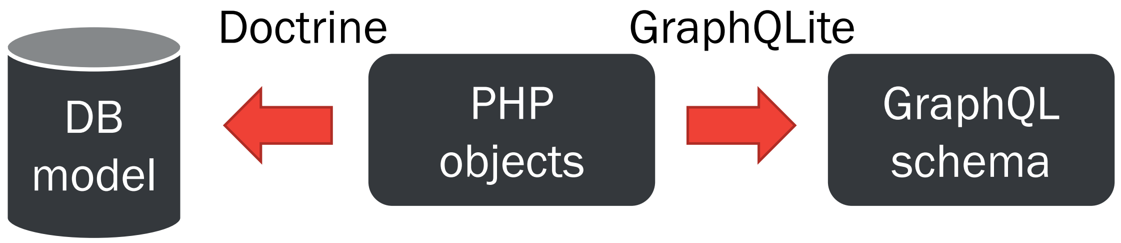
http://bit.ly/phpuk-graphqlite

Download the workshop!

Works well with Doctrine

Bonus:

- It plays nice with Doctrine ORM too



- (it also plays nive with Eloquent and TDBM)

http://bit.ly/phpuk-graphqlite

Download the workshop!

GraphQLite

GraphQLite is:

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

http://bit.ly/phpuk-graphqlite

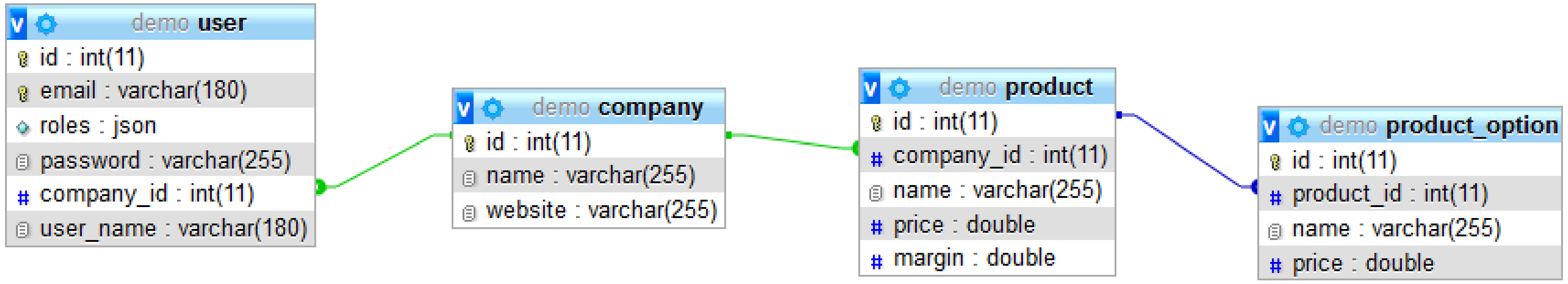
Download the workshop!

Hands on time!

- ▶ Getting started
- ▶ Pagination
- ▶ Authentication / Authorisation
- ▶ Autowiring
- ▶ The front-end side
- ▶ Mutations
- ▶ Performance

PHP UK CONFERENCE

Our playground: a marketplace!



http://bit.ly/phpuk-graphqlite *Download the workshop!*

Our playground

A call to “`docker-compose up`” will:

- Run “`composer install`”
- Initialize the DB model
- Fill the DB model with test data

You can therefore restart the environment at any point to reset the database.

http://bit.ly/phpuk-graphqlite

Download the workshop!

Our playground

Available endpoints:

- http://localhost:81 ➔ Symfony
- http://localhost:82 ➔ PhpMyAdmin
- http://localhost:83 ➔ Svelte front-end
- http://localhost:84 ➔ React front-end

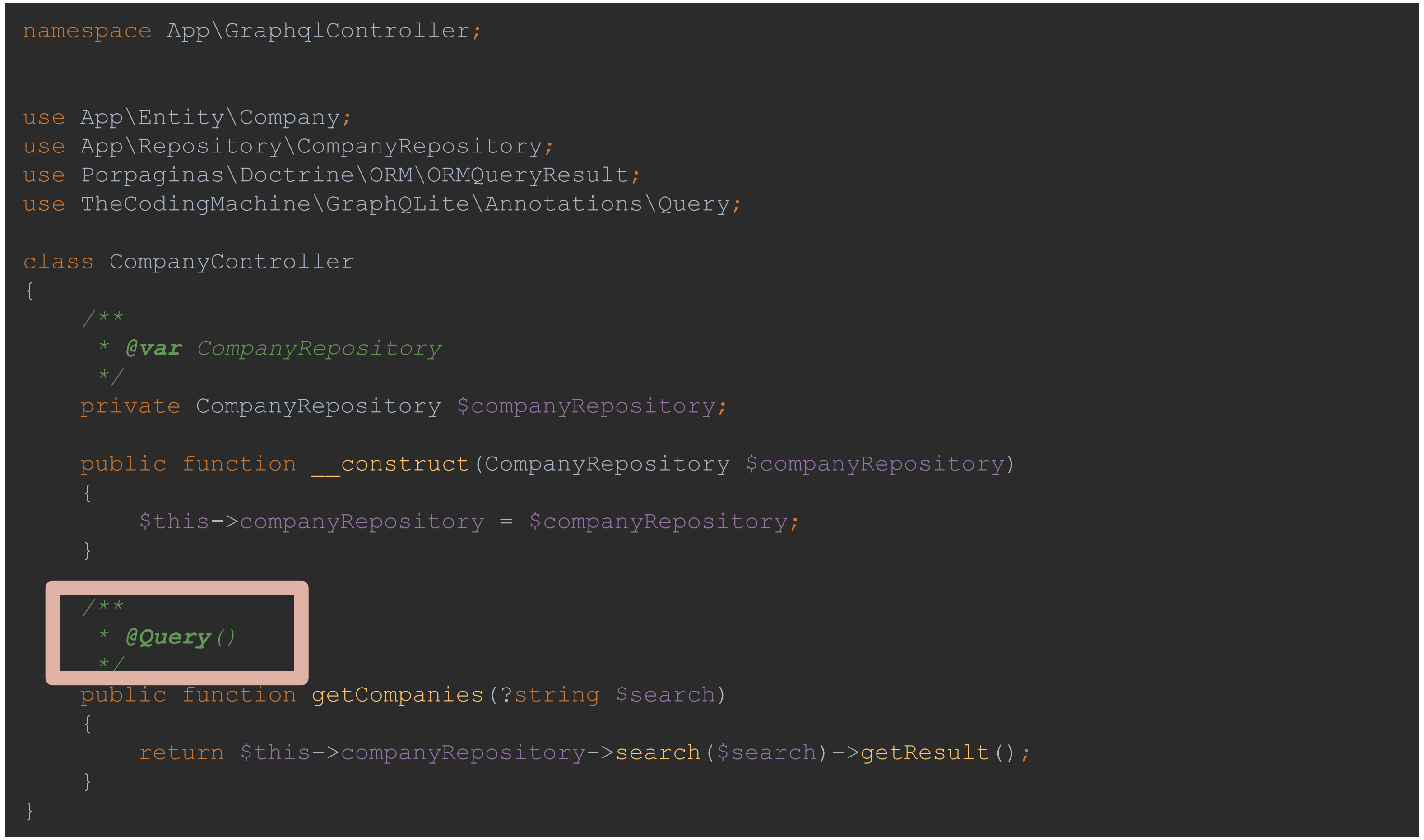
http://bit.ly/phpuk-graphqlite *Download the workshop!*

MEET OUR PO

HE WANTS HIS MARKETPLACE DONE.

FAST.

First query

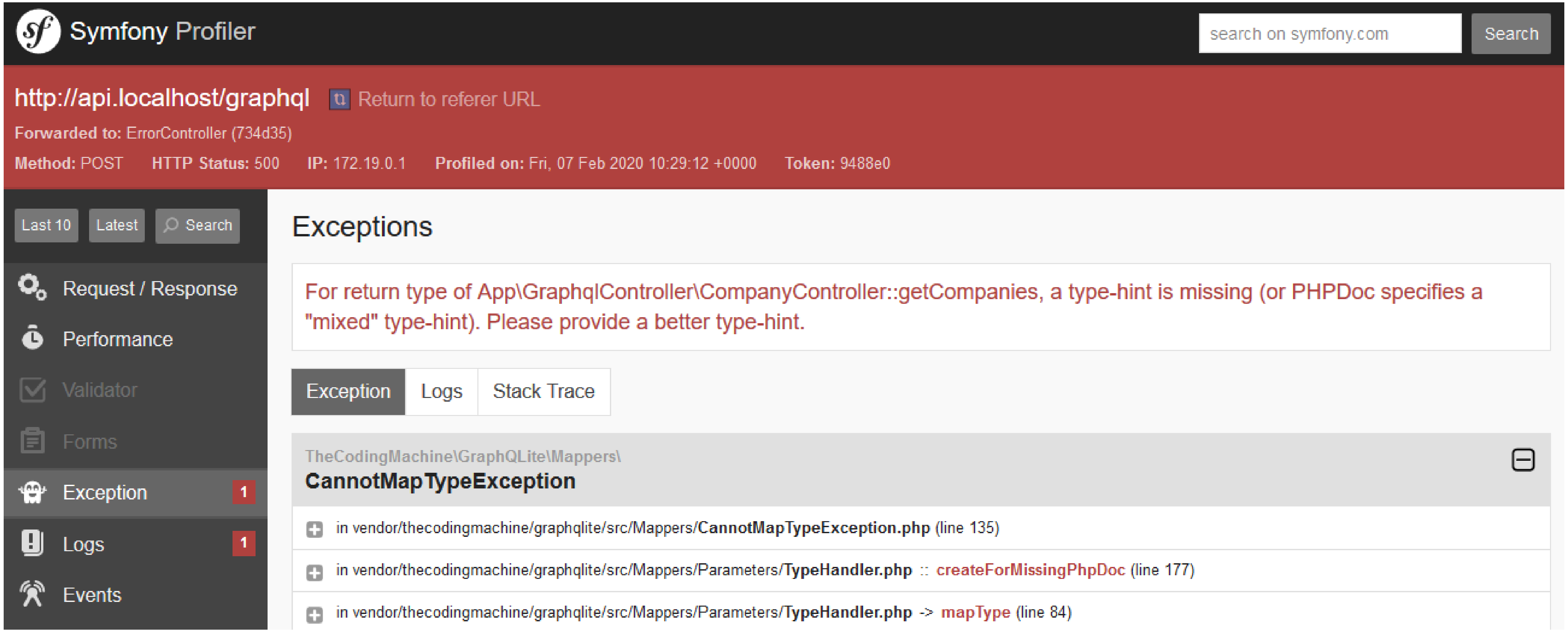


```php
namespace App\GraphqlController;


use App\Entity\Company;
use App\Repository\CompanyRepository;
use Porpaginas\Doctrine\ORM\ORMQueryResult;
use TheCodingMachine\GraphQLite\Annotations\Query;

class CompanyController
{
    /**
     * @var CompanyRepository
     */
    private CompanyRepository $companyRepository;

    public function __construct(CompanyRepository $companyRepository)
    {
        $this->companyRepository = $companyRepository;
    }

    /**
     * @Query()
     */
    public function getCompanies(?string $search)
    {
        return $this->companyRepository->search($search)->getResult();
    }
}
```

http://bit.ly/phpuk-graphqlite

Download the workshop!

First query



Symfony Profiler

search on symfony.com Search

http://api.localhost/graphql Return to referer URL

Forwarded to: ErrorController (734d35)

Method: POST **HTTP Status:** 500 **IP:** 172.19.0.1 **Profiled on:** Fri, 07 Feb 2020 10:29:12 +0000 **Token:** 9488e0

Exceptions

For return type of App\GraphqlController\CompanyController::getCompanies, a type-hint is missing (or PHPDoc specifies a "mixed" type-hint). Please provide a better type-hint.

TheCodingMachine\GraphQLite\Mappers\
CannotMapTypeException

```
in vendor/thecodingmachine/graphqlite/src/Mappers/CannotMapTypeException.php (line 135)
in vendor/thecodingmachine/graphqlite/src/Mappers/Parameters/TypeHandler.php :: createForMissingPhpDoc (line 177)
in vendor/thecodingmachine/graphqlite/src/Mappers/Parameters/TypeHandler.php -> mapType (line 84)
```

http://bit.ly/phpuk-graphqlite

Download the workshop!

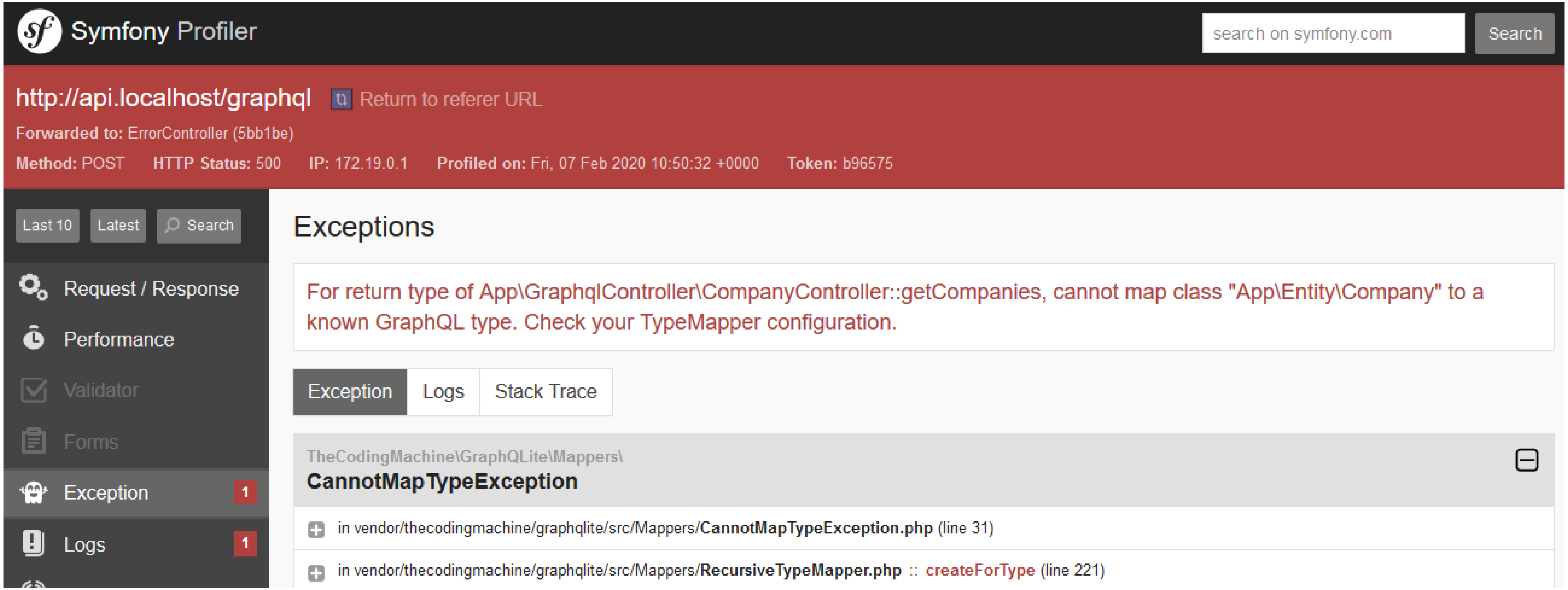
First query

```php
namespace App\GraphqlController;


use App\Entity\Company;
use App\Repository\CompanyRepository;
use Porpaginas\Doctrine\ORM\ORMQueryResult;
use TheCodingMachine\GraphQLite\Annotations\Query;

class CompanyController
{
    /**
     * @var CompanyRepository
     */
    private CompanyRepository $companyRepository;

    public function __construct(CompanyRepository $companyRepository)
    {
        $this->companyRepository = $companyRepository;
    }

    /**
     * @Query()
     * @return Company[]
     */
    public function getCompanies(?string $search): array
    {
        return $this->companyRepository->search($search)->getResult();
    }
}
```

http://bit.ly/phpuk-graphqlite

Download the workshop!

First query



http://api.localhost/graphql Return to referer URL

Forwarded to: ErrorController (5bb1be)

Method: POST **HTTP Status:** 500 **IP:** 172.19.0.1 **Profiled on:** Fri, 07 Feb 2020 10:50:32 +0000 **Token:** b96575

Exceptions

For return type of App\GraphqlController\CompanyController::getCompanies, cannot map class "App\Entity\Company" to a known GraphQL type. Check your TypeMapper configuration.

TheCodingMachine\GraphQLite\Mappers\
CannotMapTypeException

in vendor/thecodingmachine/graphqlite/src/Mappers/**CannotMapTypeException.php** (line 31)

in vendor/thecodingmachine/graphqlite/src/Mappers/**RecursiveTypeMapper.php** :: createForType (line 221)

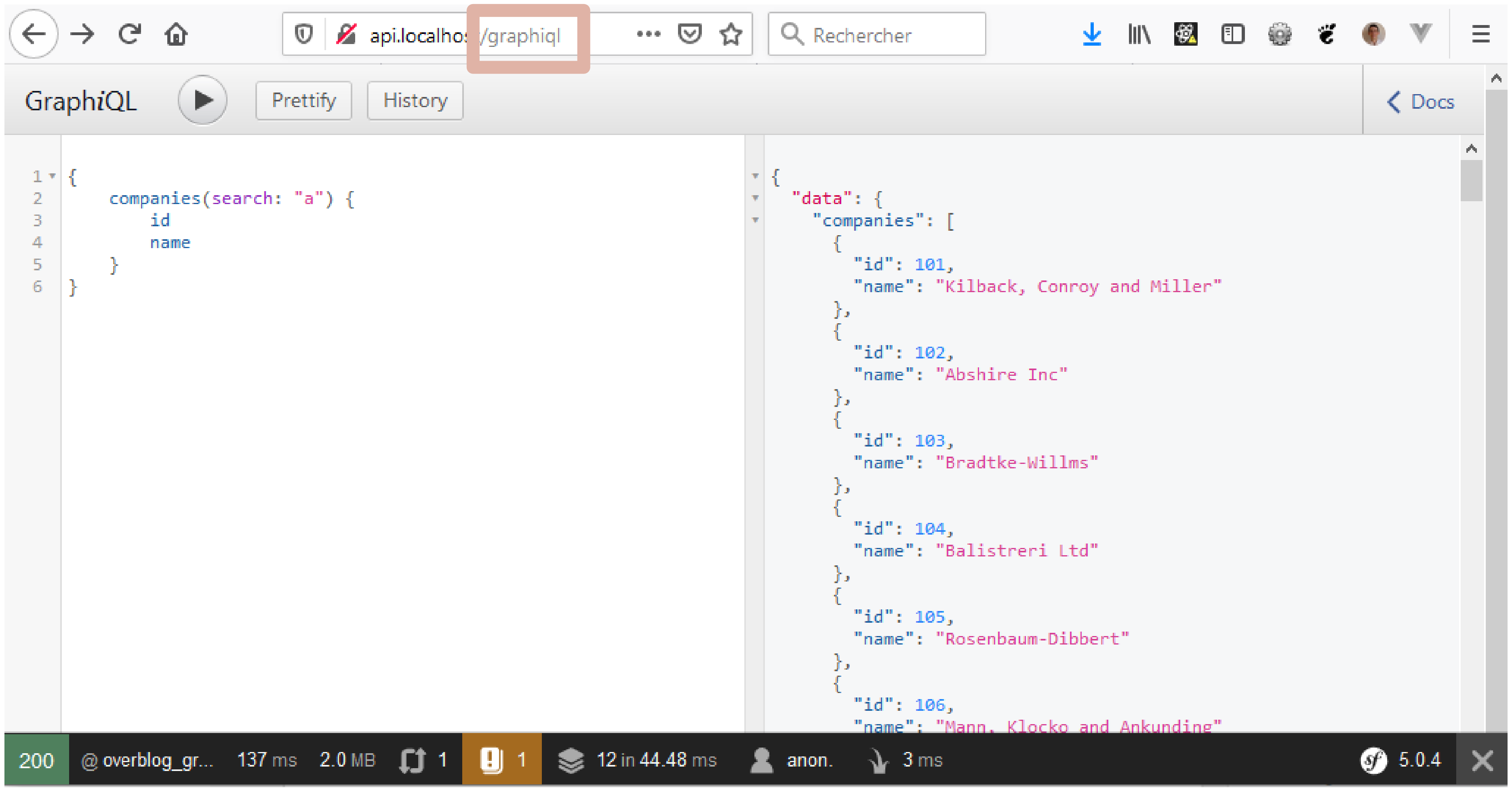
http://bit.ly/phpuk-graphqlite *Download the workshop!*

```php
<?php

namespace App\Entity;

use TheCodingMachine\GraphQLite\Annotations\Field;
use TheCodingMachine\GraphQLite\Annotations\Type;

/**
 * @Type()
 * @ORM\Entity(repositoryClass="App\Repository\CompanyRepository")
 */
class Company
{
    /**
     * @Field()
     */
    public function getId(): ?int
    {
        return $this->id;
    }

    /**
     * @Field()
     */
    public function getName(): ?string
    {
        return $this->name;
    }

    /**
     * @Field()
     */
    public function getWebsite(): ?string
    {
        return $this->website;
    }
```

http://bit.ly/phpuk-graphqlite

Download the workshop!

First query


api.localhost/graphiql

GraphiQL ▶ Prettify History Docs

```
{
    companies(search: "a") {
        id
        name
    }
}
```

```
{
  "data": {
    "companies": [
      {
        "id": 101,
        "name": "Kilback, Conroy and Miller"
      },
      {
        "id": 102,
        "name": "Abshire Inc"
      },
      {
        "id": 103,
        "name": "Bradtke-Willms"
      },
      {
        "id": 104,
        "name": "Balistreri Ltd"
      },
      {
        "id": 105,
        "name": "Rosenbaum-Dibbert"
      },
      {
        "id": 106,
        "name": "Mann, Klocko and Ankunding"
```

200 @ overblog_gr... 137 ms 2.0 MB 1 1 12 in 44.48 ms anon. 3 ms 5.0.4

http://bit.ly/phpuk-graphqlite

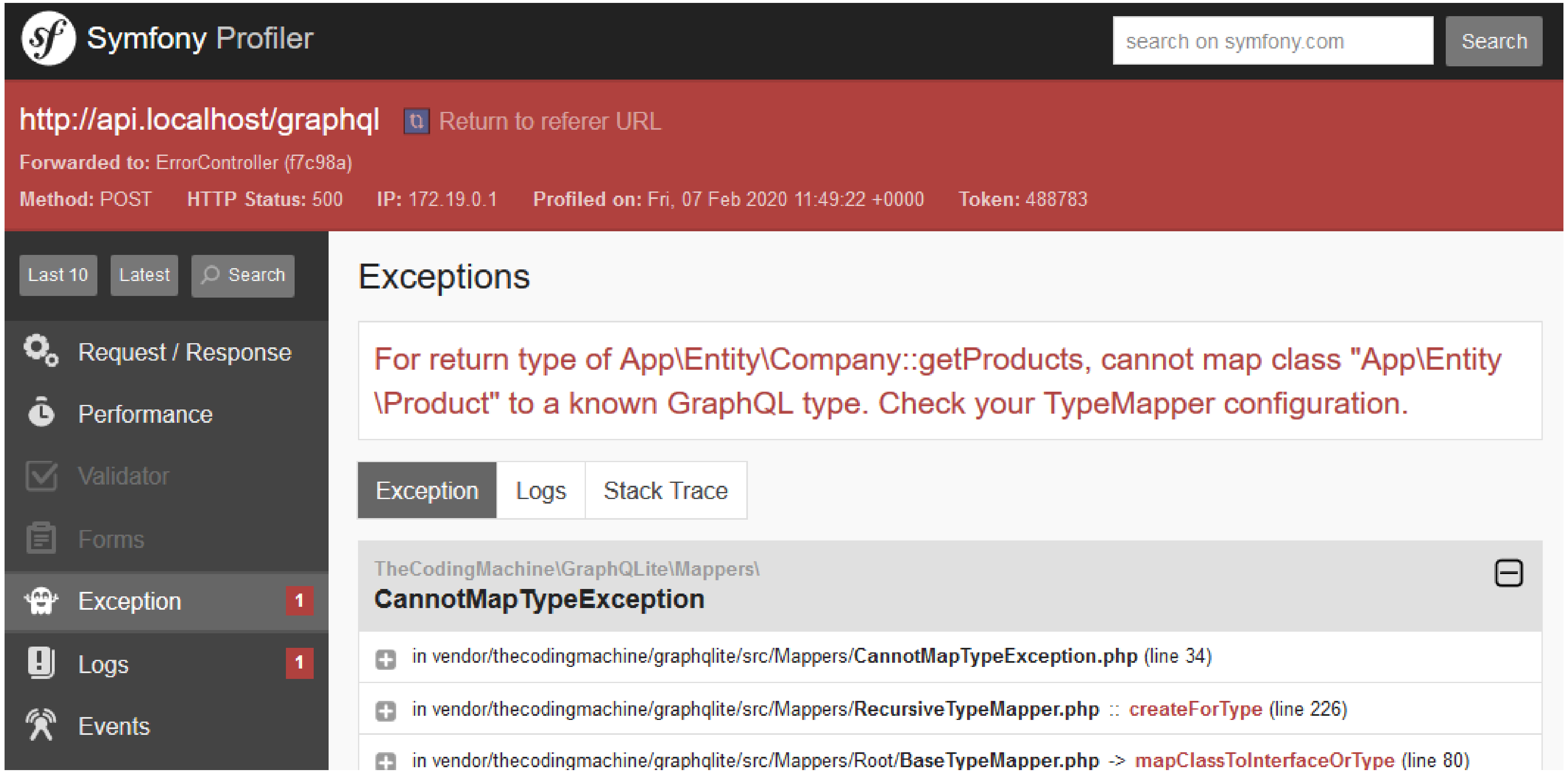
Download the workshop!

First query

```php
/**
 * @Type()
 */
class Company
{
    // …

    /**
     * @Field()
     * @return Product[]
     */
    public function getProducts()
    {
        return $this->products;
    }
}
```

http://bit.ly/phpuk-graphqlite Download the workshop!

First query



http://api.localhost/graphql

Forwarded to: ErrorController (f7c98a)

Method: POST **HTTP Status:** 500 **IP:** 172.19.0.1 **Profiled on:** Fri, 07 Feb 2020 11:49:22 +0000 **Token:** 488783

Exceptions

For return type of App\Entity\Company::getProducts, cannot map class "App\Entity\Product" to a known GraphQL type. Check your TypeMapper configuration.

Exception | Logs | Stack Trace

TheCodingMachine\GraphQLite\Mappers\
CannotMapTypeException

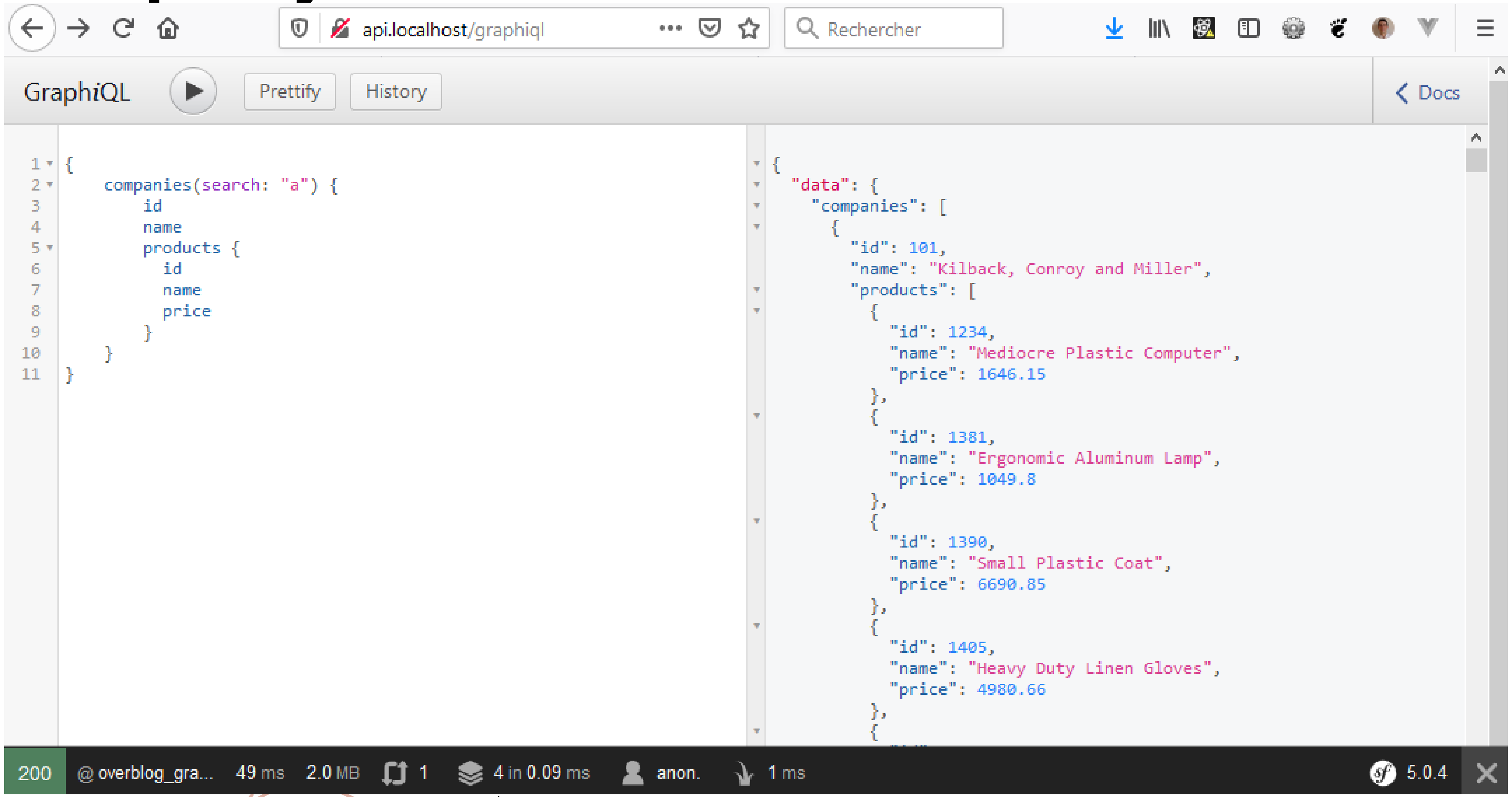
```
in vendor/thecodingmachine/graphqlite/src/Mappers/CannotMapTypeException.php (line 34)
in vendor/thecodingmachine/graphqlite/src/Mappers/RecursiveTypeMapper.php :: createForType (line 226)
in vendor/thecodingmachine/graphqlite/src/Mappers/Root/BaseTypeMapper.php -> mapClassToInterfaceOrType (line 80)
```

http://bit.ly/phpuk-graphqlite Download the workshop!

First query

```php
<?php

namespace App\Entity;

use TheCodingMachine\GraphQLite\Annotations\Field;
use TheCodingMachine\GraphQLite\Annotations\Type;

/**
 * @Type()
 */
class User implements UserInterface
{
    // …

    /**
     * @Field(outputType="ID!")
     */
    public function getId(): int
    {
        return $this->id;
    }

    /**
     * @Field()
     */
    public function getLogin(): string
    {
        return $this->login;
    }
    // …
```

First query



api.localhost/graphiql

GraphiQL | Prettify | History | Docs

```graphql
{
    companies(search: "a") {
        id
        name
        products {
          id
          name
          price
        }
    }
}
```

```json
{
  "data": {
    "companies": [
      {
        "id": 101,
        "name": "Kilback, Conroy and Miller",
        "products": [
          {
            "id": 1234,
            "name": "Mediocre Plastic Computer",
            "price": 1646.15
          },
          {
            "id": 1381,
            "name": "Ergonomic Aluminum Lamp",
            "price": 1049.8
          },
          {
            "id": 1390,
            "name": "Small Plastic Coat",
            "price": 6690.85
          },
          {
            "id": 1405,
            "name": "Heavy Duty Linen Gloves",
            "price": 4980.66
          },
          {
```

200 | @ overblog_gra... | 49 ms | 2.0 MB | 1 | 4 in 0.09 ms | anon. | 1 ms | 5.0.4

http://bit.ly/phpuk-graphqlite

Download the workshop!

MY LIST OF COMPANIES IS HUGE!
GIVE ME PAGINATION! NOW!

Hands on time!

- ▶ Getting started
- ▶ Pagination
- ▶ Authentication / Authorisation
- ▶ Autowiring
- ▶ The front-end side
- ▶ Mutations
- ▶ Performance

PHP UK
CONFERENCE

Pagination using input arguments

Since we can pass any argument in a function, it is quite easy to add a “limit” and an “offset” parameters:

```
/**
 * @Query()
 * @return Company[]
 */
public function getCompanies(
    ?string $search,
    int $limit = 100,
    int $offset = 0)
{
    // ...
}
```

http://bit.ly/phpuk-graphqlite Download the workshop!

Native pagination

Actually, you don’t even have to bother adding pagination as GraphQLite integrates natively with Porpaginas.

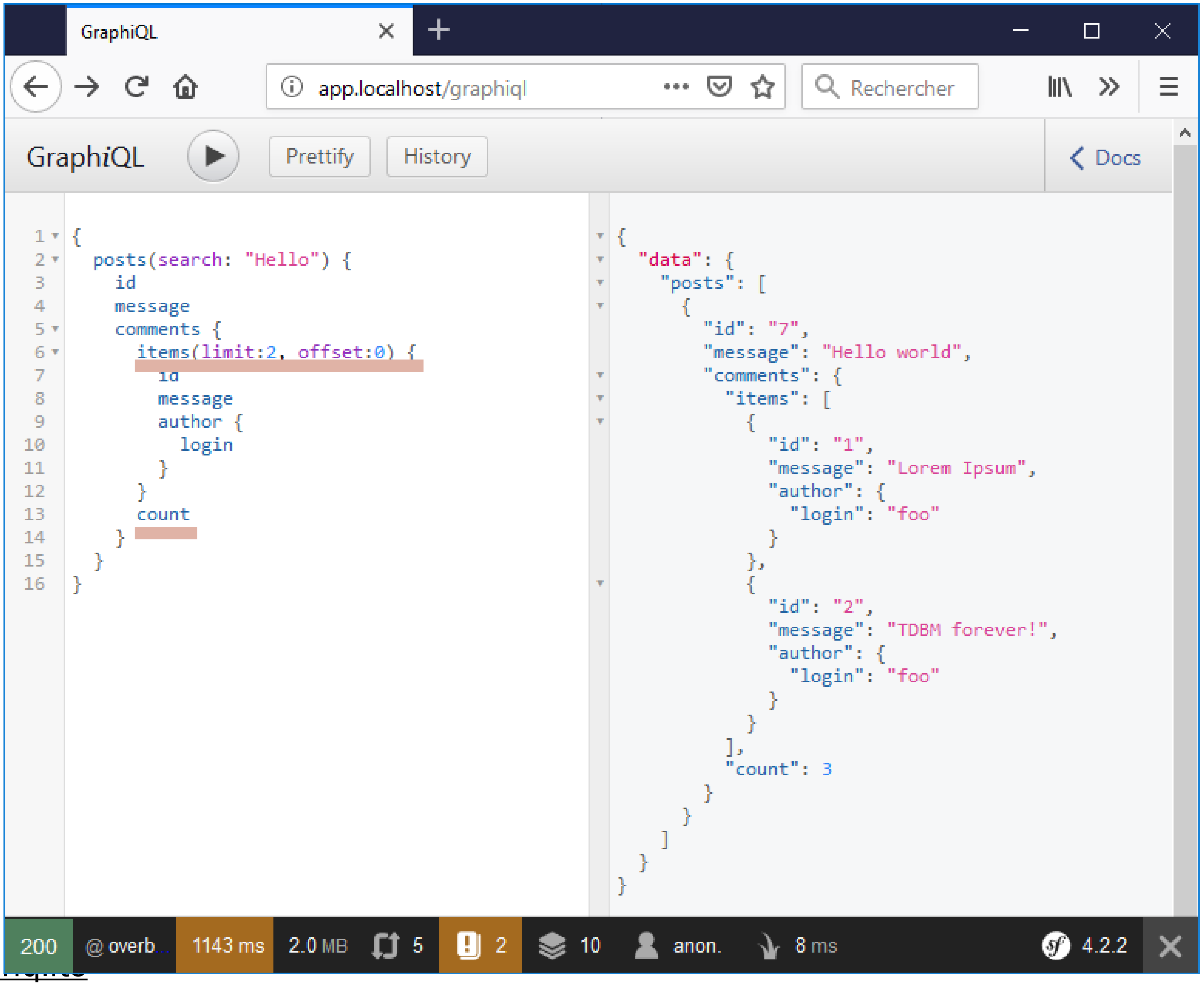
Porpaginas is a generic pagination interface.

(and Porpaginas integrates with Doctrine queries)

Native pagination

```
use Porpaginas\Doctrine\ORM\ORMQueryResult;

class CompanyController
{
    /**
     * @Query()
     * @return ORMQueryResult|Company[]
     */
    public function getCompanies(?string $search): ORMQueryResult
    {
        return new ORMQueryResult($this->companyRepository->search($search));
    }
}
```

http://bit.ly/phpuk-graphqlite Download the workshop!



```
{
  posts(search: "Hello") {
    id
    message
    comments {
      items(limit:2, offset:0) {
        id
        message
        author {
          login
        }
      }
      count
    }
  }
}
```

```
{
  "data": {
    "posts": [
      {
        "id": "7",
        "message": "Hello world",
        "comments": {
          "items": [
            {
              "id": "1",
              "message": "Lorem Ipsum",
              "author": {
                "login": "foo"
              }
            },
            {
              "id": "2",
              "message": "TDBM forever!",
              "author": {
                "login": "foo"
              }
            }
          ],
          "count": 3
        }
      }
    ]
  }
}
```

http://bit.ly/phpuk-grap

Hands on time!

- Getting started
- Pagination
- Authentication / Authorisation
- Autowiring
- The front-end side
- Mutations
- Performance

PHP UK CONFERENCE

Authentication

GraphQLite – Symfony integration comes with 3 operations:

- "me" query
- "login" mutation
- "logout" mutation

You can use those as soon as Symfony security is setup.

```
mutation login {
  login(
      userName:"user1@example.com",
      password:"password") {
    userName
  }
}
```

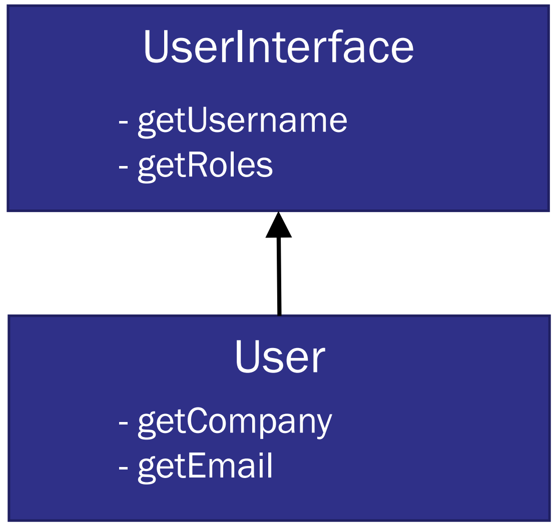
```
query me {
  me {
    userName
    roles
  }
}
```

http://bit.ly/phpuk-graphqlite ← Download the workshop!

Authentication

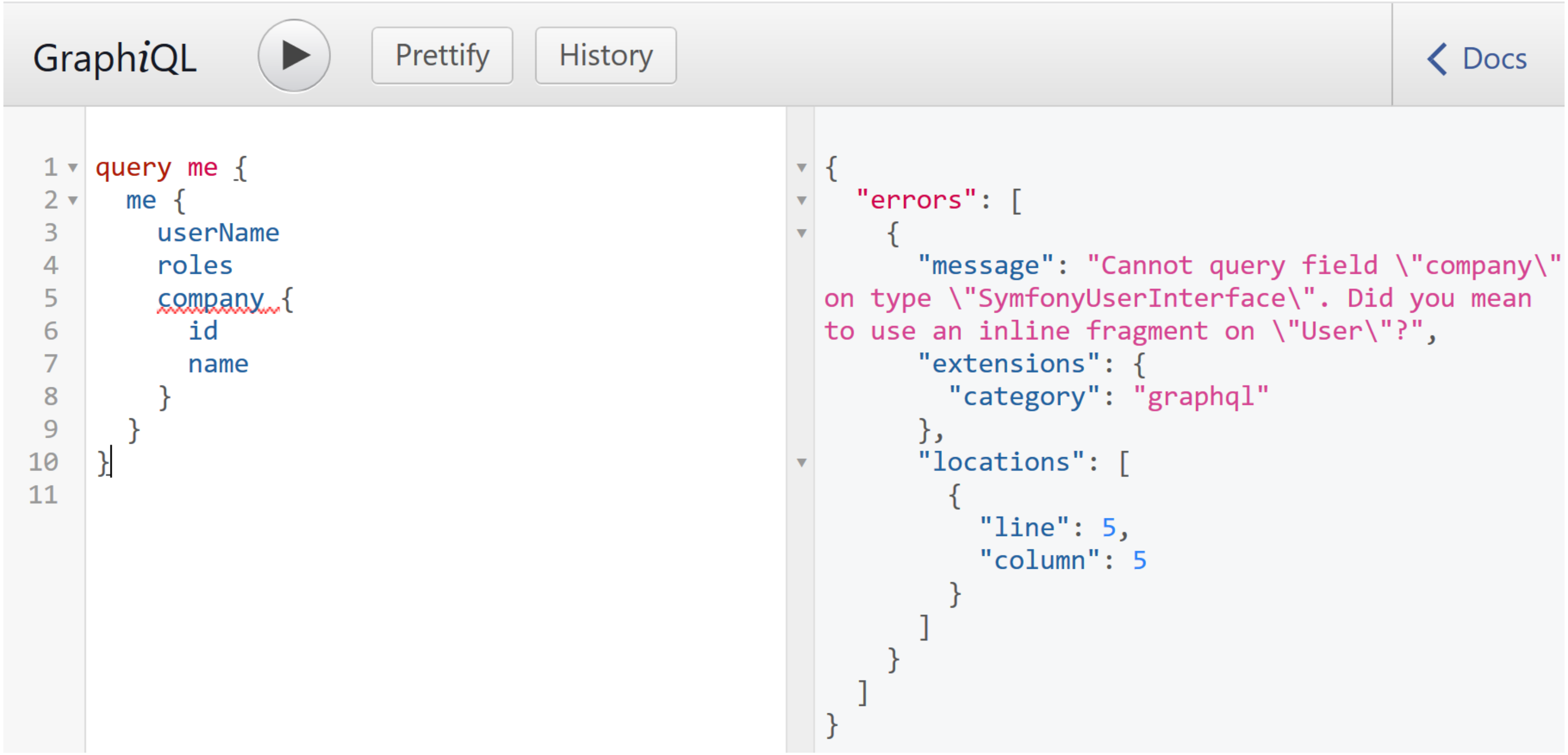
The “me” query returns a Symfony “UserInterface”

Therefore, you cannot directly access the “company” or the “email” field from the “me” query



http://bit.ly/phpuk-graphqlite *Download the workshop!*

Authentication



```
query me {
  me {
    userName
    roles
    company {
      id
      name
    }
  }
}
```

```
{
  "errors": [
    {
      "message": "Cannot query field \"company\" on type \"SymfonyUserInterface\". Did you mean to use an inline fragment on \"User\"?",
      "extensions": {
        "category": "graphql"
      },
      "locations": [
        {
          "line": 5,
          "column": 5
        }
      ]
    }
  ]
}
```

http://bit.ly/phpuk-graphqlite

Download the workshop!

Authentication

To access fields from the “User” class you need to use a GraphQL “**fragment**”.

A “fragment” allows you to access fields from implementations from an interface

```
query me {
  me {
    userName
    roles
    ... on User {
      id
      company {
        id
        name
      }
    }
  }
}
```

http://bit.ly/phpuk-graphqlite *Download the workshop!*

I WANT THE USER LIST PROTECTED!

Authorization

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

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

@Right annotations should be used with @Field too!

http://bit.ly/phpuk-graphqlite Download the workshop!

Fine grained authorization

Sometimes, you need to grant access to a resource based on complex rules.

For instance:

“I can view only emails from my users in my own company”

http://bit.ly/phpuk-graphqlite

Download the workshop!

Fine grained authorization

```
/**
 * @Type()
*/
class User implements UserInterface, Serializable
{
    //…
    /**
     * @Field()
     * @Security("this.getCompany() == user.getCompany()", failWith=null)
    */
    public function getEmail(): ?string
    {
        return $this->email;
    }
}
```

http://bit.ly/phpuk-graphqlite Download the workshop!

Fine grained authorization (using voters)

```
/**
 * @Type()
*/
class User implements UserInterface, Serializable
{
    //…
    /**
     * @Field()
     * @Security("is_granted('email', this)", failWith=null)
    */
    public function getEmail(): ?string
    {
        return $this->email;
    }
}
```

http://bit.ly/phpuk-graphqlite   Download the workshop!

Fine grained authorization

Notice how security annotations are directly added to the model rather than at the controller level.

✓

WE NEED TO COMPUTE THE VAT!
FOR EVERY PRODUCT! NOW!

Hands on time!

- ► Getting started
- ► Pagination
- ► Authentication / Authorisation
- ► Autowiring
- ► The front-end side
- ► Mutations
- ► Performance

PHP UK
CONFERENCE

Improving our sample

GraphQLite makes it trivial to compute a "dynamic" field.

In our application, let's add a field to the "Product" type that computes the VAT of a product:

```
class Product
{
    // …

    /**
     * @Field()
     */
    public function getVat(): float
    {
        return $this->price * 0.2;
    }
}
```

http://bit.ly/phpuk-graphqlite Download the workshop!

Improving our sample

But sometimes, you need extra logic.

Your logic depends on complex services and does not belong to an entity / a model.

You need to access **a service** from a @Field annotated method in an entity.

Improving our sample

```
class Product
{
    // …

    /**
     * @Field()
     * @Autowire(for="$vatService")
     */
    public function getVat(VatService $vatService): float
    {
        return $vatService->getVat($this);
    }
}
```

http://bit.ly/phpuk-graphqlite

Download the workshop!

SERVICES IN YOUR MODEL?
ARE YOU KIDDING ME?

Improving our sample

Our `getVat` method requires directly the `VatService` class. This is not ideal because our entity is supposed to be independent of any service now depends on an external service.

Our models should be independent of any service. We can reach this independence by using interfaces (dependency inversion principle).

Download the workshop!

Improving our sample

```
class Product
{
    // …

    /**
     * @Field()
     * @Autowire(for="$vatService")
     */
    public function getVat(VatServiceInterface $vatService): float
    {
        return $vatService->getVat($this);
    }
}
```

http://bit.ly/phpuk-graphqlite Download the workshop!

Hands on time!

- ► Getting started
- ► Pagination
- ► Authentication / Authorisation
- ► Autowiring
- ► The front-end side
- ► Mutations
- ► Performance

PHP UK
CONFERENCE

GraphQL on the client side

There are 2 strategies:

Lightweight client

- Simple wrapper around “fetch”

Full featured client

- Cache
- Typescript types generation…

http://bit.ly/phpuk-graphqlite

Download the workshop!

GraphQL on the client side

There are 2 strategies:

Lightweight client

- Urql
- FetchQL
- GraphQL-Request

Full featured client

- Apollo (framework-agnostic)
- Relay (React only)

GraphQL on the client side
About Relay

Relay is the GraphQL client from Facebook.

It comes with a set of restrictions on the GraphQL schema.

To use Relay, you need a GraphQL server compatible with Relay (like *Lighthouse* or *Overblog/GraphQL-bundle*)

Zoom on Apollo

- Apollo has bindings with:
  - Angular
  - React
  - VueJS
  - Svelte

- You bundle a React/Angular/Vue component in a Apollo component and Apollo takes in charge the query to the server

http://bit.ly/phpuk-graphqlite

Download the workshop!

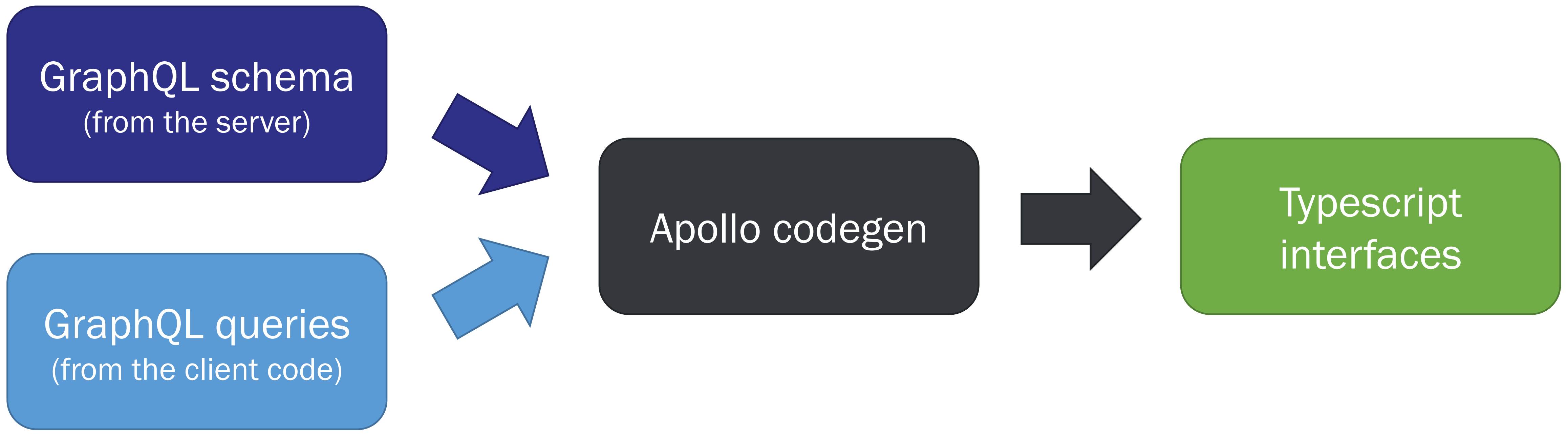
GraphQL on the client side – with types!

- GraphQL has types.
- Typescript has types.

- It makes a lot of sense to propagate types from GraphQL to typescript:

http://bit.ly/phpuk-graphqlite ← *Download the workshop!*

GraphQL on the client side – with types!



http://bit.ly/phpuk-graphqlite

Download the workshop!

Generating queries and types

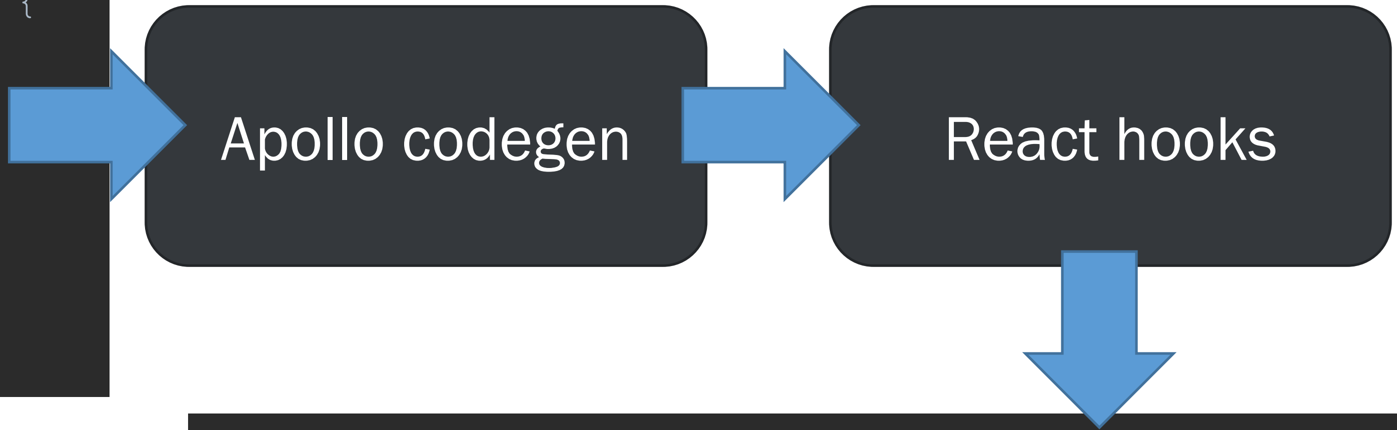
```
$ docker-compose exec front-react yarn run codegen
yarn run v1.16.0
$ graphql-codegen --config codegen.yml
  ✔ Parse configuration
  ✔ Generate outputs
Done in 7.30s.
```

http://bit.ly/phpuk-graphqlite *Download the workshop!*

Generating queries and types

query.graphql

```
query companies($search: String!) {
    companies(search: $search) {
        items(limit:10, offset: 0) {
            id
            name
            website
        }
        count
    }
}
```



```
const { data, error, loading } = useCompaniesQuery(
    { variables: { search: props.search } }
);
```

http://bit.ly/phpuk-graphqlite Download the workshop!

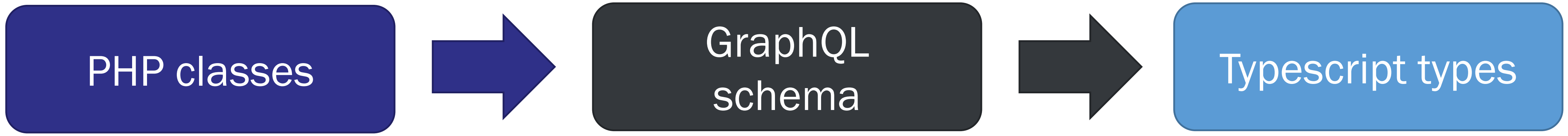
```
import * as React from 'react';
import {useCompaniesQuery} from '../../generated/graphql';
import CompanyList from './CompanyList';

interface Props {
    search: string;
}

const CompanyListContainer: React.FC<Props> = (props: Props) =>
{
    const { data, error, loading } = useCompaniesQuery(
        { variables: { search: props.search } }
        );

    if (loading) {
        return <div>Loading...</div>;
    }

    if (error || !data) {
        return <div>ERROR</div>;
    }

    return <CompanyList data={data} />;
};

export default CompanyListContainer;
```

http://bit.ly/phpuk-graphqlite

Download the workshop!

End-to-end typing

- With GraphQLite + Apollo + Typescript, we can propagate types from the server to the client side



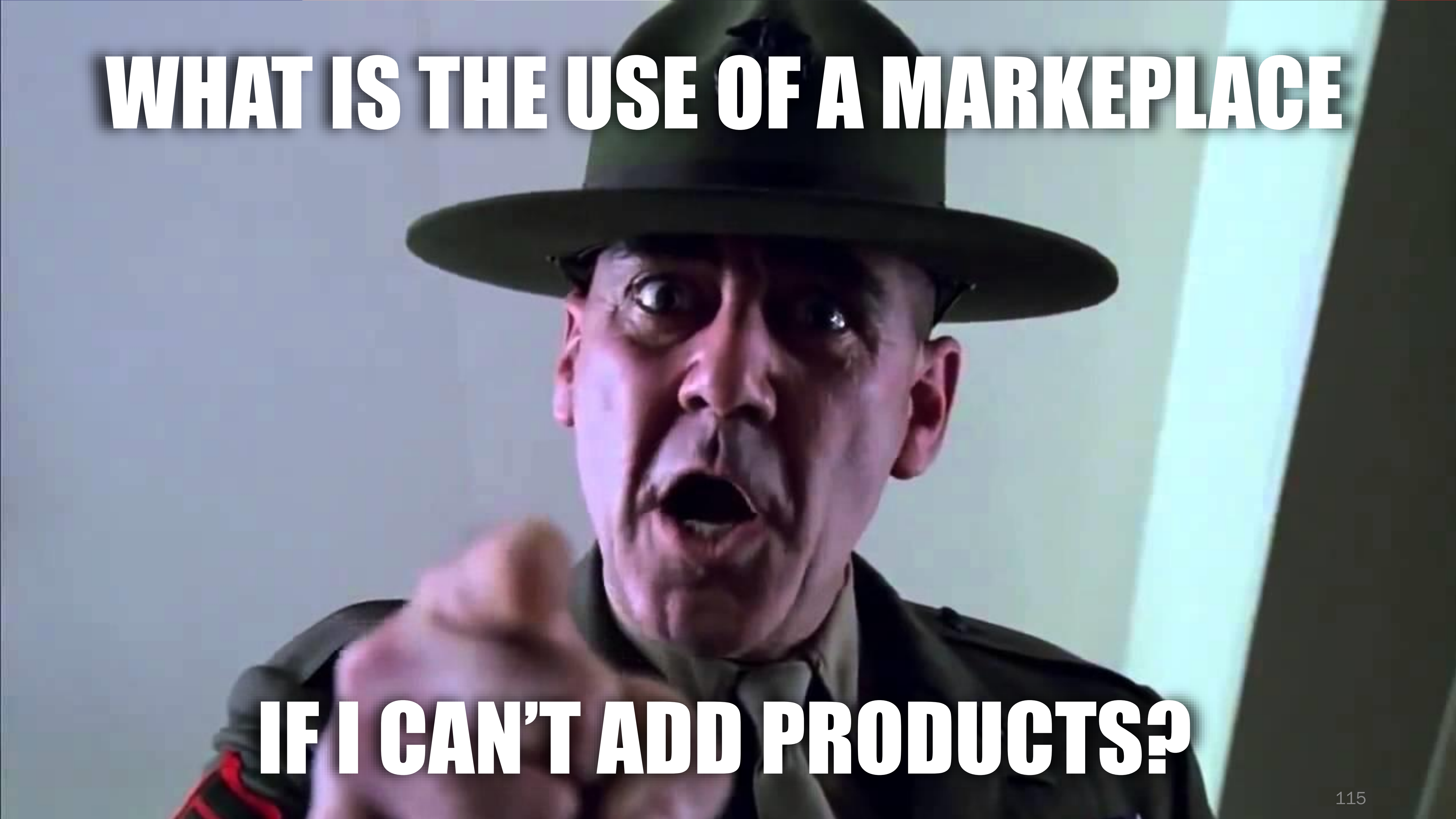
- This is **insanely cool**. You get:
  - Autocompletion in your IDE
  - Checking of your code at compilation time
  - Extremely easy refactoring

http://bit.ly/phpuk-graphqlite *Download the workshop!*

But where is Redux?

- Apollo comes internally with its own store.
- Redux is really less useful with Apollo and you can simply scrap ~90% of your reducers.
- Still useful for niche places (like managing the current logged user)

http://bit.ly/phpuk-graphqlite Download the workshop!


WHAT IS THE USE OF A MARKEPLACE
IF I CAN'T ADD PRODUCTS?

Hands on time!

- Getting started
- Pagination
- Authentication / Authorisation
- Autowiring
- The front-end side
- Mutations
- Performance

PHP UK CONFERENCE

Mutations

Use a mutation to change the state of your application.

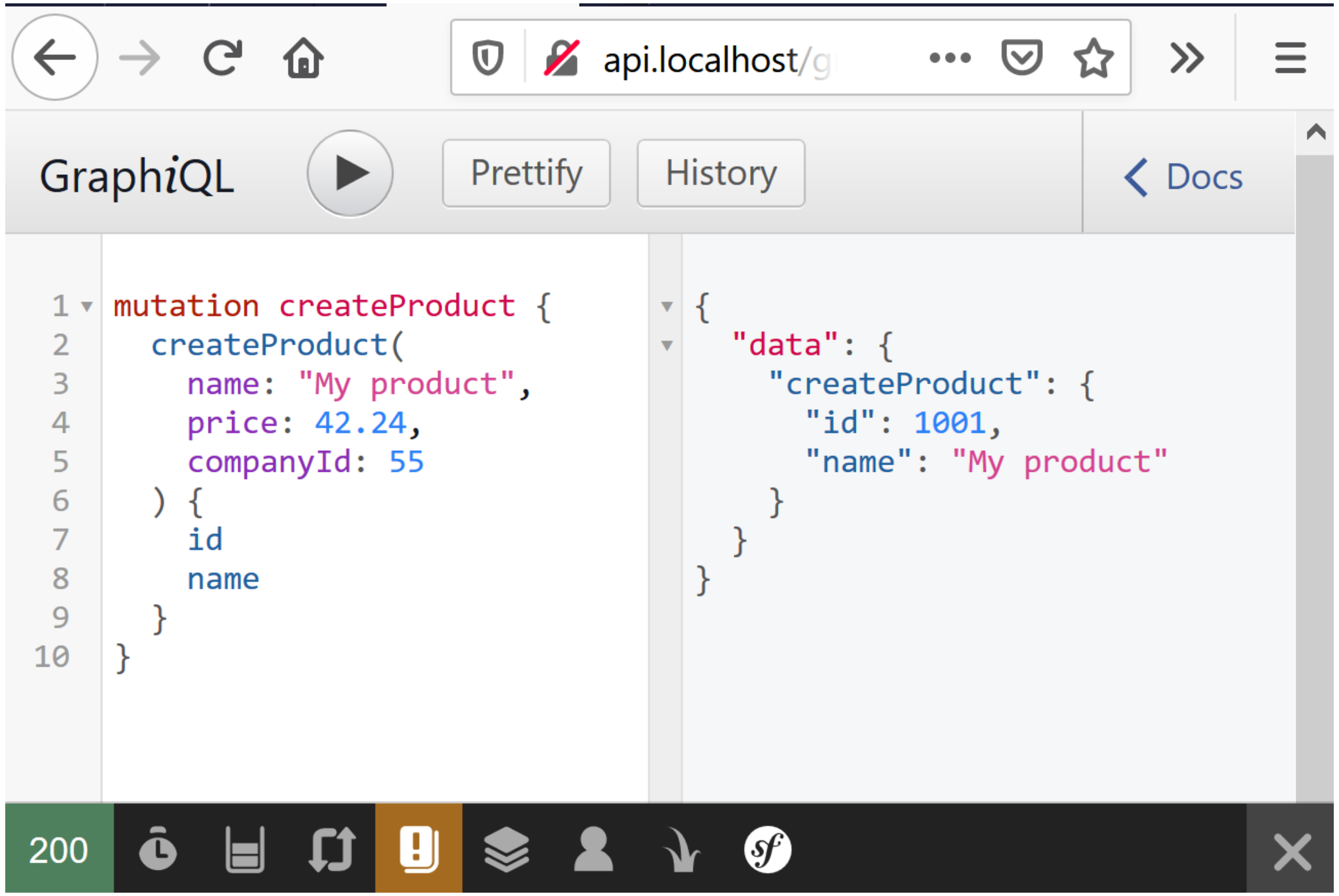
Mutations are similar to queries, only the annotation changes.

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;
    }
}
```

http://bit.ly/phpuk-graphqlite Download the workshop!

Mutations



```
mutation createProduct {
  createProduct(
    name: "My product",
    price: 42.24,
    companyId: 55
  ) {
    id
    name
  }
}
```

```
{
  "data": {
    "createProduct": {
      "id": 1001,
      "name": "My product"
    }
  }
}
```

http://bit.ly/phpuk-graphqlite

Download the workshop!

Mutations

GraphQLite maps function arguments to the GraphQL model. Therefore, you don't need any serializer / deserializer!

GraphQLite maps automatically scalar types (int, string, float…)

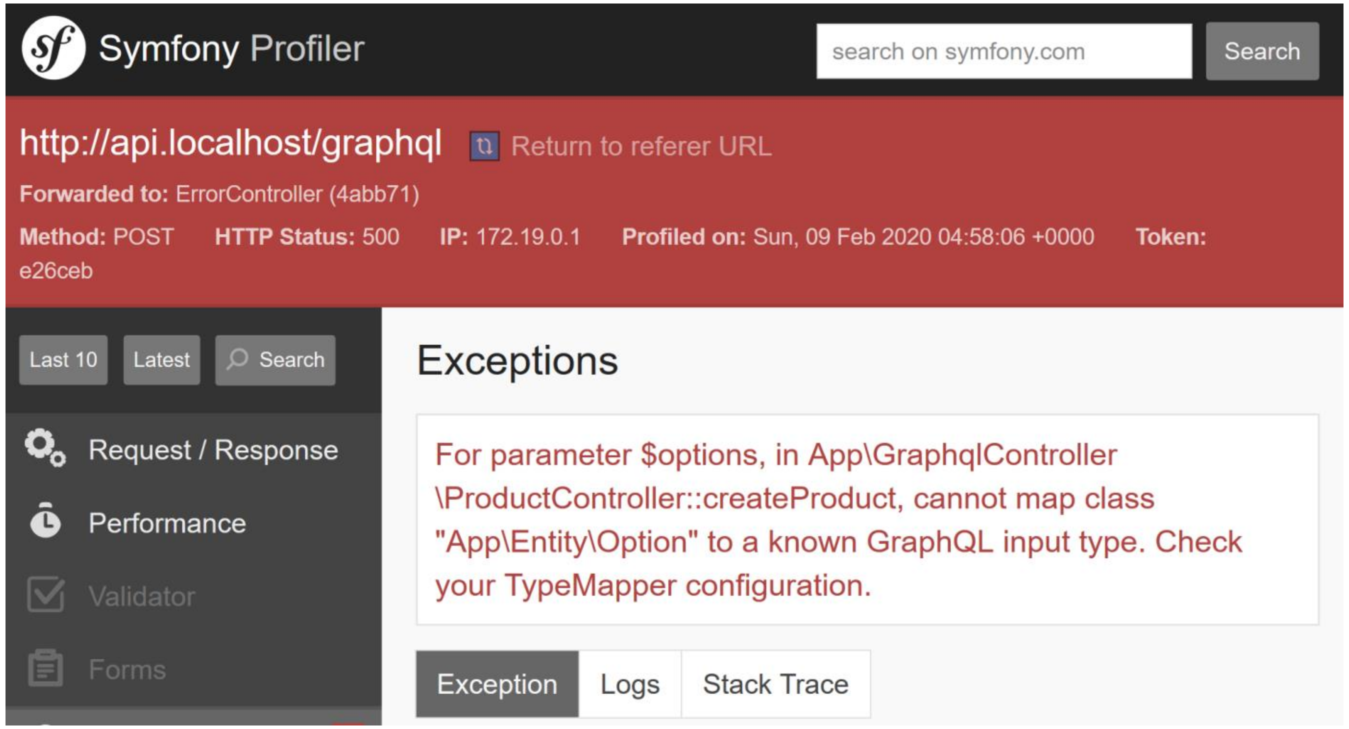
But it needs some help to map objects passed as arguments.

http://bit.ly/phpuk-graphqlite

Download the workshop!

Mutations

```
/**
 * @Mutation()
 * @param Option[] $options
 */
public function createProduct(
         string $name,
         float $price,
         int $companyId,
         array $options = []): Product
{
    // ...
}
```



http://api.localhost/graphql Return to referer URL

Forwarded to: ErrorController (4abb71)

Method: POST **HTTP Status:** 500 **IP:** 172.19.0.1 **Profiled on:** Sun, 09 Feb 2020 04:58:06 +0000 **Token:** e26ceb

Exceptions

For parameter $options, in App\GraphqlController\ProductController::createProduct, cannot map class "App\Entity\Option" to a known GraphQL input type. Check your TypeMapper configuration.

http://bit.ly/phpuk-graphqlite *Download the workshop!*

Mutations

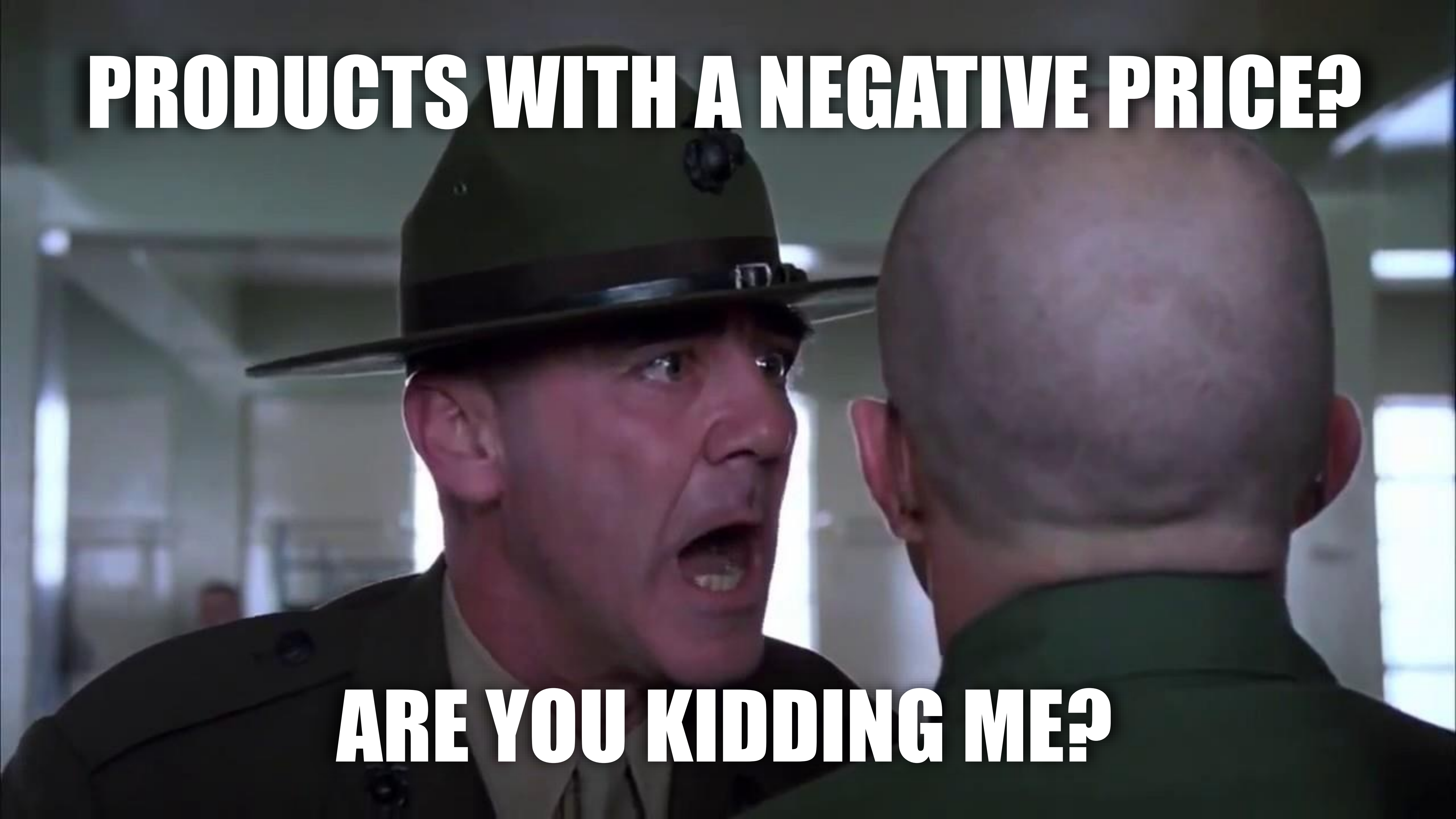
In GraphQL, objects that are passed in argument to fields/functions are called "**input type**".

To define an input type in GraphQLite, we use the **@Factory** annotation.

Mutations

```
use TheCodingMachine\GraphQLite\Annotations\Factory;

class ProductController
{
    // …

    /**
     * @Factory()
     */
    public function optionFactory(string $name, float $price): Option
    {
        return new Option($name, $price);
    }
}
```

http://bit.ly/phpuk-graphqlite Download the workshop!


PRODUCTS WITH A NEGATIVE PRICE?
ARE YOU KIDDING ME?

Validation

PHP UK CONFERENCE

Validation

The GraphQL type system validates automatically the structure of the data passed by our client.

However, the client could still pass garbage data.

Let’s add validation!

Validation

The easiest way to add validation is to throw exceptions if data is not valid.

Validation

```
/**
 * @Mutation()
 * @param Option[] $options
 */
public function createProduct(
    string $name,
    float $price,
    int $companyId,
    array $options): Product
{
    if ($name === '') {
        throw new GraphQLException('Empty product names are not allowed', 400);
    }
    if ($price < 0) {
        throw new GraphQLException('The price must be positive', 400);
    }
    // ...
}
```

http://bit.ly/phpuk-graphqlite

Download The workshop!

Validation

```
mutation createProduct {
  createProduct(
    name: "",
    price: -12,
    companyId: 55
  ) {
    id
  }
}
```

```
{
  "errors": [
    {
      "message": "Empty product names
                  are not allowed",
      "extensions": {
        "category": "Exception"
      },
      "locations": [
        {
          "line": 59,
          "column": 3
        }
      ],
      "path": [
        "createProduct"
      ]
    }
  ]
}
```

http://bit.ly/phpuk-graphqlite

Download the workshop!

Validation

⚠ GraphQLite integrates with validation system from Symfony and Laravel.

So validation works differently based on your framework.

In Symfony, GraphQLite relies on the symfony/validator component.

Validation

```php
use Symfony\Component\Validator\Constraints as Assert;

class Product
{
    /**
     * @ORM\Column(type="string", length=255)
     * @Assert\NotBlank()
     */
    private string $name;

    /**
     * @ORM\Column(type="float", nullable=true)
     * @Assert\GreaterThanOrEqual(0)
     */
    private ?float $price;
}
```

http://bit.ly/phpuk-graphqlite Download the workshop!

Validation

```
/**
 * @Mutation()
 * @param Option[] $options
 */
public function createProduct(
        string $name,
        float $price,
        int $companyId,
        array $options = []): Product
{
    // …
    // Let's validate the product
    $errors = $this->validator->validate($product);
    // Throw an appropriate GraphQL exception if validation errors are encountered
    ValidationFailedException::throwException($errors);
    // …
}
```

http://bit.ly/phpuk-graphqlite Download the workshop!

Validation

```
mutation createProduct {
  createProduct(
    name: "",
    price: -12,
    companyId: 55
  ) {
    id
  }
}
```

```
{
  "errors": [
    {
      "message": "This value
                  should not be blank.",
      "extensions": {
        "field": "name",
        "category": "Validate"
      },
      // ...
    },
    {
      "message": "This value should be
            greater than or equal to 0.",
      "extensions": {
        "field": "price",
        "category": "Validate"
      },
      // ...
    }
  ]
}
```

http://bit.ly/phpuk-graphqlite Download the workshop!

Validation

Once again, we managed to push the validation rules on the model layer (instead of keeping them in the controller).

This is great, as these validation rules (like the fact that a price cannot be negative) clearly belongs to the model.

Hands on time!

- ► Getting started
- ► Pagination
- ► Authentication / Authorisation
- ► Autowiring
- ► The front-end side
- ► Mutations
- ► Performance

PHP UK
CONFERENCE

Performance

Try running this query:

```
query companies {
  companies(search: "") {
    items(limit:100, offset: 0) {
      id
      name
      products {
        name
        price
        vat
      }
    }
    count
  }
}
```

```
api_1 | doctrine.DEBUG: SELECT c0_.id AS id_0, c0
["%%","%%",[2701,2702,2703,2704,2705,2706,2707,27
api_1 | doctrine.DEBUG: SELECT t0.id AS id_1, t0.
api_1 | doctrine.DEBUG: SELECT t0.id AS id_1, t0.
api_1 | doctrine.DEBUG: SELECT t0.id AS id_1, t0.
api_1 | doctrine.DEBUG: SELECT t0.id AS id_1, t0.
api_1 | doctrine.DEBUG: SELECT t0.id AS id_1, t0.
api_1 | doctrine.DEBUG: SELECT t0.id AS id_1, t0.
…
…
…
```

You will see 1 query to fetch companies and 100 queries to fetch the products!

http://bit.ly/phpuk-graphqlite

Download the workshop!

Performance

Resolver (methods annotated with @Field) are called once per object.

It is very easy to create GraphQL queries that generate a huge number of SQL queries.

http://bit.ly/phpuk-graphqlite Download the workshop!

Performance

A common pattern used by GraphQL servers to avoid the “N+1” queries problem is to use the “data-loader” pattern.

With the "data-loader" pattern, we run only one giant query to fetch the “products”, passing all the IDs in one giant "WHERE … IN (…)" request.

```
class Company
{
    /**
     * @Field(prefetchMethod="prefetchProducts")
     * @return Product[]
     */
    public function getProducts($sortedProducts)
    {
        return $sortedProducts[$this->getId()] ?? [];
    }

    /**
     * @param Company[] $companies
     * @Autowire(for="$productRepository")
     * @return array<int, array<Product>>
     */
    public function prefetchProducts(iterable $companies, ProductRepository $productRepository)
    {
        $products = $productRepository->findByCompanies($companies);

        $sortedProducts = [];
        foreach ($products as $product) {
            $sortedProducts[$product->getCompany()->getId()][] = $product;
        }

        return $sortedProducts;
    }
}
```

```php
class Company
{
    /**
     * @Field(prefetchMethod="prefetchProducts")
     * @return Product[]
     */
    public function getProducts($sortedProducts)
    {
        return $sortedProducts[$this->getId()] ?? [];
    }

    /**
     * @param Company[] $companies
     * @Autowire(for="$productRepository")
     * @return array<int, array<Product>>
     */
    public function prefetchProducts(iterable $companies, ProductRepository $productRepository)
    {
        $products = $productRepository->findByCompanies($companies);

        $sortedProducts = [];
        foreach ($products as $product) {
            $sortedProducts[$product->getCompany()->getId()][] = $product;
        }

        return $sortedProducts;
    }
}
```

```php
class ProductRepository extends ServiceEntityRepository
{

    /**
     * @param Company[] $companies
     */
    public function findByCompanies(array $companies)
    {
        $ids = array_map(function(Company $company) {
            return $company->getId();
        }, $companies);

        return $this->createQueryBuilder('p')
            ->join(Company::class, 'c')
            ->andWhere('c.id IN (:values)')
            ->setParameter('values', $ids)
            ->getQuery()
            ->getResult()
            ;
    }
}
```

http://bit.ly/phpuk-graphqlite

Download the workshop!

```php
class Company
{
    /**
     * @Field(prefetchMethod="prefetchProducts")
     * @return Product[]
     */
    public function getProducts($sortedProducts)
    {
        return $sortedProducts[$this->getId()] ?? [];
    }

    /**
     * @param Company[] $companies
     * @Autowire(for="$productRepository")
     * @return array<int, array<Product>>
     */
    public function prefetchProducts(iterable $companies, ProductRepository $productRepository)
    {
        $products = $productRepository->findByCompanies($companies);

        $sortedProducts = [];
        foreach ($products as $product) {
            $sortedProducts[$product->getCompany()->getId()][] = $product;
        }

        return $sortedProducts;
    }
}
```

Performance

Try running this query again:

```
query companies {
  companies(search: "") {
    items(limit:100, offset: 0) {
      id
      name
      products {
        name
        price
        vat
      }
    }
    count
  }
}
```

```
api_1| doctrine.DEBUG: SELECT DISTINCT id_0 FROM
api_1| doctrine.DEBUG: SELECT c0_.id AS id_0, c0_
["%%","%%",[2701,2702,2703,2704,2705,2706,2707,27
api_1| doctrine.DEBUG: SELECT p0_.id AS id_0, p0_
[[2701,2702,2703,2704,2705,2706,2707,2708,2709,27
```

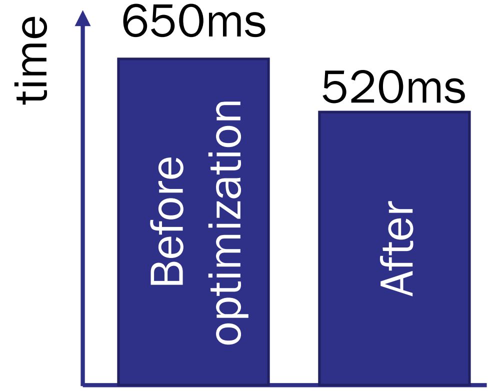
Success! The 100 queries have been turned in a unique query!

http://bit.ly/phpuk-graphqlite

Download the workshop!

Performance

Beware! You should always time your queries after applying an optimization!



http://bit.ly/phpuk-graphqlite

Download the workshop!

Data-loader pattern: some problems

The “data-loader” pattern is far from perfect:

- Unlike all other features, it puts some GraphQL peculiarities right into your models (not good)
- It is not extremely easy to understand

http://bit.ly/phpuk-graphqlite

Download the workshop!

Performance: solutions?

The “N+1” problem is not specific to GraphQL.

It is a recurring problem of all ORMs.

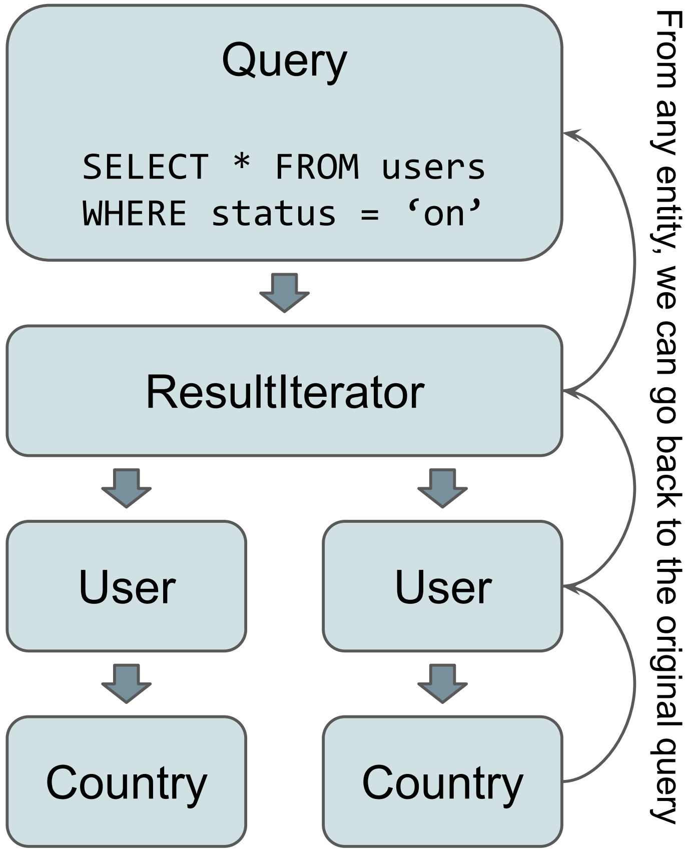
The code below triggers “N+1” calls.

```
$users = $userRepository->findAll();
foreach ($users as $user) {
    echo $user->getName().' lives in '.$user->getCountry()->getLabel()."\n";
}
```

Performance: solutions?

Hot take:
This problem should be solved at the ORM level and not at the GraphQL level!

If from an entity, we could go back to the query that generated it, it would be possible to do "eager lazy loading".



http://bit.ly/phpuk-graphqlite ← *Download the workshop!*

Performance: solutions?

Doctrine ORM does not support this notion of “eager lazy loading”, but a few ORMs can:

- NotORM
- Nette Database Explorer
- Laravel JIT loader
- TDBM 5.3 (work in progress)

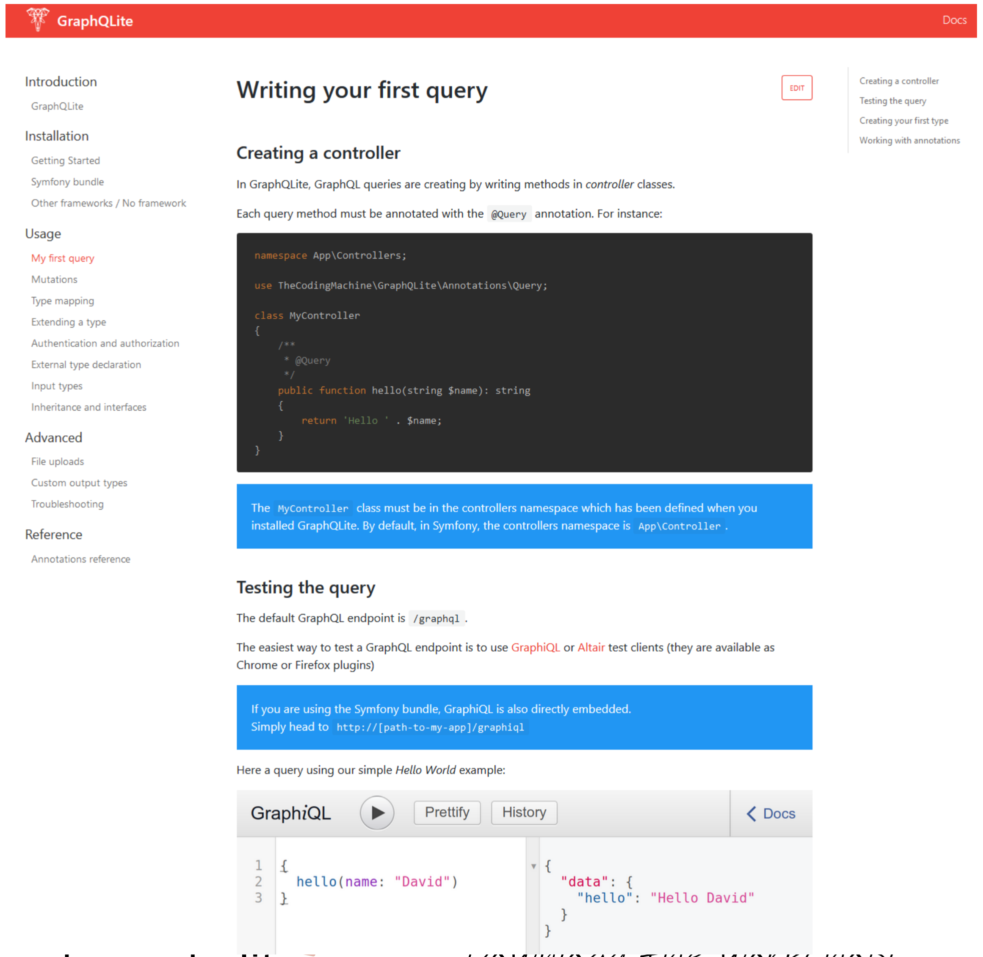
http://bit.ly/phpuk-graphqlite

Download the workshop!

More features?

PHP UK
CONFERENCE

More features!



GraphQLite Docs

Introduction
- GraphQLite

Installation
- Getting Started
- Symfony bundle
- Other frameworks / No framework

Usage
- My first query
- Mutations
- Type mapping
- Extending a type
- Authentication and authorization
- External type declaration
- Input types
- Inheritance and interfaces

Advanced
- File uploads
- Custom output types
- Troubleshooting

Reference
- Annotations reference

Writing your first query

EDIT

Creating a controller
Testing the query
Creating your first type
Working with annotations

Creating a controller

In GraphQLite, GraphQL queries are creating by writing methods in *controller* classes.

Each query method must be annotated with the `@Query` annotation. For instance:

```php
namespace App\Controllers;

use TheCodingMachine\GraphQLite\Annotations\Query;

class MyController
{
    /**
     * @Query
     */
    public function hello(string $name): string
    {
        return 'Hello ' . $name;
    }
}
```

The `MyController` class must be in the controllers namespace which has been defined when you installed GraphQLite. By default, in Symfony, the controllers namespace is `App\Controller`.

Testing the query

The default GraphQL endpoint is `/graphql`.

The easiest way to test a GraphQL endpoint is to use GraphiQL or Altair test clients (they are available as Chrome or Firefox plugins)

If you are using the Symfony bundle, GraphiQL is also directly embedded.
Simply head to `http://[path-to-my-app]/graphiql`

Here a query using our simple *Hello World* example:

```graphql
{
  hello(name: "David")
}
```

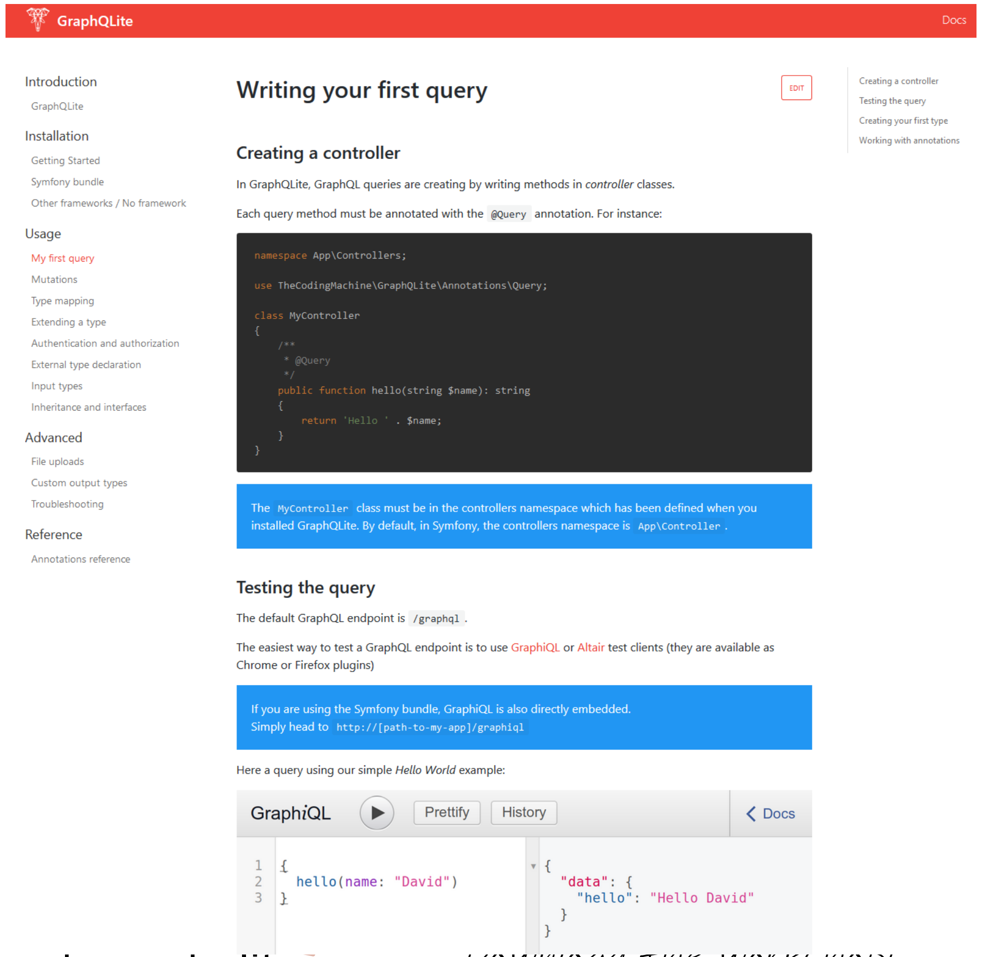
```json
{
  "data": {
    "hello": "Hello David"
  }
}
```

- 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

What’s next?



GraphQLite

Docs

Introduction
- GraphQLite

Installation
- Getting Started
- Symfony bundle
- Other frameworks / No framework

Usage
- My first query
- Mutations
- Type mapping
- Extending a type
- Authentication and authorization
- External type declaration
- Input types
- Inheritance and interfaces

Advanced
- File uploads
- Custom output types
- Troubleshooting

Reference
- Annotations reference

Writing your first query

EDIT

- Creating a controller
- Testing the query
- Creating your first type
- Working with annotations

Creating a controller

In GraphQLite, GraphQL queries are creating by writing methods in *controller* classes.

Each query method must be annotated with the `@Query` annotation. For instance:

```
namespace App\Controllers;

use TheCodingMachine\GraphQLite\Annotations\Query;

class MyController
{
    /**
     * @Query
     */
    public function hello(string $name): string
    {
        return 'Hello ' . $name;
    }
}
```

The `MyController` class must be in the controllers namespace which has been defined when you installed GraphQLite. By default, in Symfony, the controllers namespace is `App\Controller`.

Testing the query

The default GraphQL endpoint is `/graphql`.

The easiest way to test a GraphQL endpoint is to use GraphiQL or Altair test clients (they are available as Chrome or Firefox plugins)

If you are using the Symfony bundle, GraphiQL is also directly embedded.
Simply head to `http://[path-to-my-app]/graphiql`

Here a query using our simple *Hello World* example:

GraphiQL | Prettify | History | Docs

```
{
  hello(name: "David")
}
```

```
{
  "data": {
    "hello": "Hello David"
  }
}
```

- Support for subscriptions (real-time GraphQL)
- A simple to use JS client
- An ORM that solves natively the N+1 problem

http://bit.ly/phpuk-graphqlite

Download the workshop!

So… GraphQL everywhere?

PHP UK CONFERENCE

GraphQL everywhere?

- GraphQLite makes it trivial to write a GraphQL API. It is now easier to start a GraphQL API than a REST API! \o/
- GraphQL makes a lot of sense for most of our projects because it eases the separation between front-end and back-end
- And the tooling in JS/TS is awesome

http://bit.ly/phpuk-graphqlite

Download the workshop!

GraphQL everywhere?

- Performance warning! GraphQL itself is fast but…
- N+1 problem
- It is easy to write slow queries ➔ Warning with front facing websites.

http://bit.ly/phpuk-graphqlite *Download the workshop!*

GraphQL everywhere?

- Two strategies available to avoid the “N+1” problem:
  - Analyzing the GraphQL query and “joining” accordingly
  - Or the “data-loader” pattern
- But the real answer will come from ORMs
- + a need to set limits on the queries complexity to avoid “rogue” queries

http://bit.ly/phpuk-graphqlite *Download the workshop!*

David Négrier

@david_negrier

@moufmouf

graphqlite.thecodingmachine.io

Questions?

More cool stuff:

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

http://bit.ly/phpuk-graphqlite

Download the workshop!