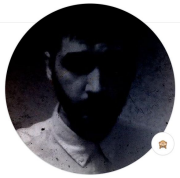
Особенности работы с GraphQL на реальном проекте

+ best practices



Александр Стрельцов
разработчик, опыт работ 9 лет
https://t.me/mark_tlen

Как работали с REST

Много эндпойнтов. Выбираются все данные, а не только нужные

GET: .../company/5

```
{
    "id": 3,
    "name": "My company",
    "inn": “123232123"
}
```

Как работали с REST

Много эндпойнтов. Выбираются все данные, а не только нужные

GET: .../company/5

```
{
    "id": 3,                        "workers": [],
    "name": "My company",           "tasks": [],
    "inn": "123232123"              "contractors": [],
}
```

Как работали с REST

Много эндпойнтов. Выбираются все данные, а не только нужные

```
{
    "id": 3,
    "name": "My company",
    "inn": "123232123",
    "workers": [],
    "tasks": [],
    "contractors": [],
}
```

.../company/5

.../company/5/workers

.../company/5/tasks

.../company/5/contractors

Немного о GraphQL

GraphQL - стандарт описания структуры данных и способов получения данных

```
{
  company (id: 5) {
    id
    name
    workers {
      name
    }
    tasks(filter: {read: UNREAD} first: 10) {
      name
    }
  }
}
```

Немного о GraphQL

Базовые типы Query и Mutation

```
query {
   company (id: 5) { id name }
}

mutation{
  createTask(input: {title: "1212"}){
    task { id }
  }
}
```

Немного о GraphQL

Resolvers вместо controller’s actions

```
resolve: '@=query("App\\GraphQL\\Resolver\\Greetings::sayHello", args["name"])'
```

```php
<?php
# src/GraphQL/Resolver/Greetings.php
namespace App\GraphQL\Resolver;

use Overblog\GraphQLBundle\Definition\Resolver\QueryInterface;

class Greetings implements QueryInterface
{
    public function sayHello($name)
    {
        return sprintf('hello %s!!!', $name);
    }
}
```

Немного о GraphQL

Resolvers вместо controller’s actions

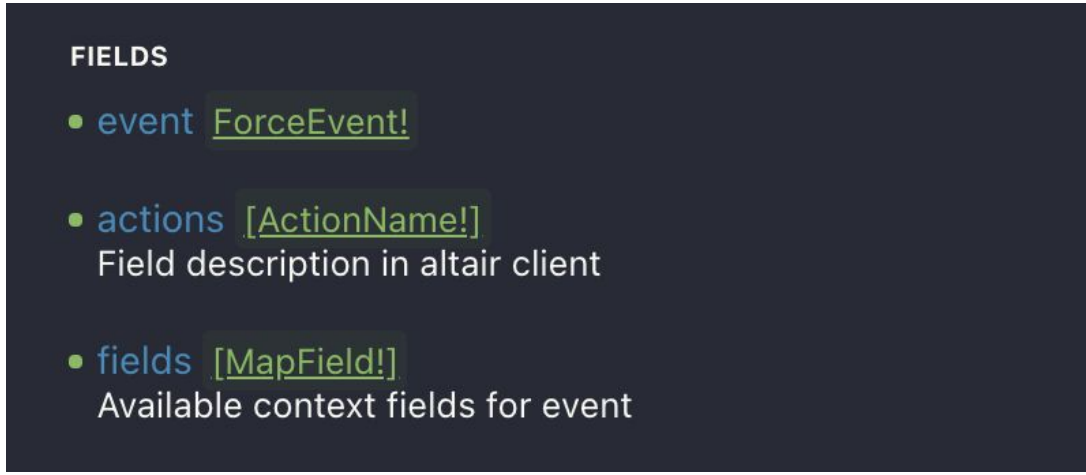
```
# config/graphql/types/MyType.types.yaml
MyType:
    type: object
    config:
        resolveField:
'@=query("App\\GraphQL\\Resolver\\Greetings", info,
args.name)'
        fields:
            hello:
                type: String
            goodbye:
                 type: String
```

```
class Greetings implements QueryInterface
{
    public function __invoke(ResolveInfo $info, $name)
    {
        if($info->fieldName === 'hello'){
            return sprintf('hello %s!!!', $name);
        }
        else if($info->fieldName === 'goodbye'){
            return sprintf('goodbye %s!!!', $name);
        }
        else{
            throw new \DomainException('Unknown greetings');
        }
    }
 }
```

Преимущества GraphQL

- **Самодокументированный**
  Все типы и поля описываются в коде, отображаются клиентами

```
fields:
    event:
        type: ForceEvent!
    actions:
        type: "[ActionName!]"
        description: 'Field description in altair client'
    fields:
        type: "[MapField!]"
        description: 'Available context fields for event'
```



Преимущества GraphQL

- **Самодокументированный**
  Все типы и поля описываются в коде, отображаются клиентами
- **Выбираем, какие данные получить**
  Только нужные данные

```
query {
   company (id: 5) {
      id
      name
      workers { name }
      tasks { name }
   }
}
```

Преимущества GraphQL

- **Самодокументированный**
  Все типы и поля описываются в коде, отображаются клиентами
- **Выбираем, какие данные получить**
  Только нужные данные
- **Несколько query в одном запросе**
  Получаем несвязанные данные сразу

```
query {
   company (id: 5) {
      id
      name
   }
   tasks {
      text
      user { name }
   }
}
```

Проблемы GraphQL в PHP

- ❌ Глубина запроса
- ❌ Сложность запроса
- ❌ N + 1
- ❌ Группировка query и мутаций
- ❌ Синхронное получение полных данных
- ❌ Нет нативной реализации подписок

Глубина запроса

```
query {
   task (id: 5) {
      subtask {
         subtask {
            users { edges { node { id } } }
            subtask {
               #...
            }
         }
      }
   }
}
```

Сложность запроса

```
query { #Final Complexity = 50 * (50 + 1)
   tasks (first: 50) {# 50 * children complexity
      edges {
         node {
            id
            performer { id } # 1
            subtasks (first: 50) { #50
               edges {node {id}}
            }
         }
      }
   }
}
```

Проблема N+1

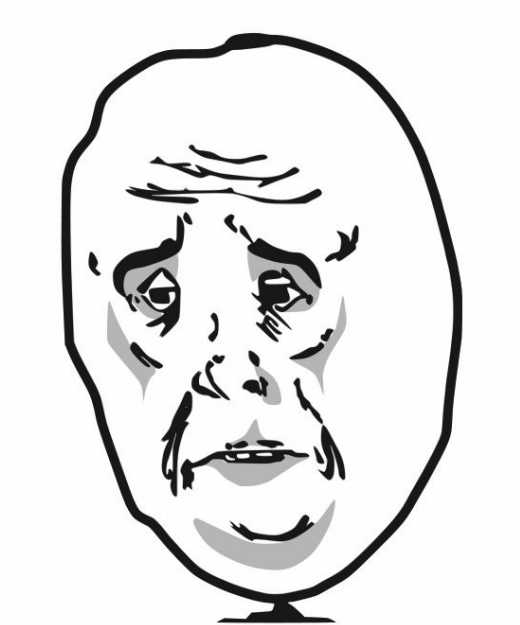
```
query {
   tasks { # 1 запрос на получение головных задач
      edges {
         node {
            id
   # +100 запросов для каждой задачи на получение подзадач
            subtasks (first: 100) {
               edges { node { id } }
            }
         }
      }
   }
}
```

Проблема N+1

```php
// Loader
public function loadTasks(Task $task)
{
    $batchLoad = function (array $ids) {
        $result = $this->taskRepository->getAllByIds($ids);

        return $this->distributeQueryResult($result, $ids);
    };

    return $this->getLoader(TaskLoader::class, $batchLoad)->load($task->getId());
}

// Resolver
public function subtasks(Task $task)
{
    return $this->taskLoader->loadTasks($task);
}
```

Группировка мутаций

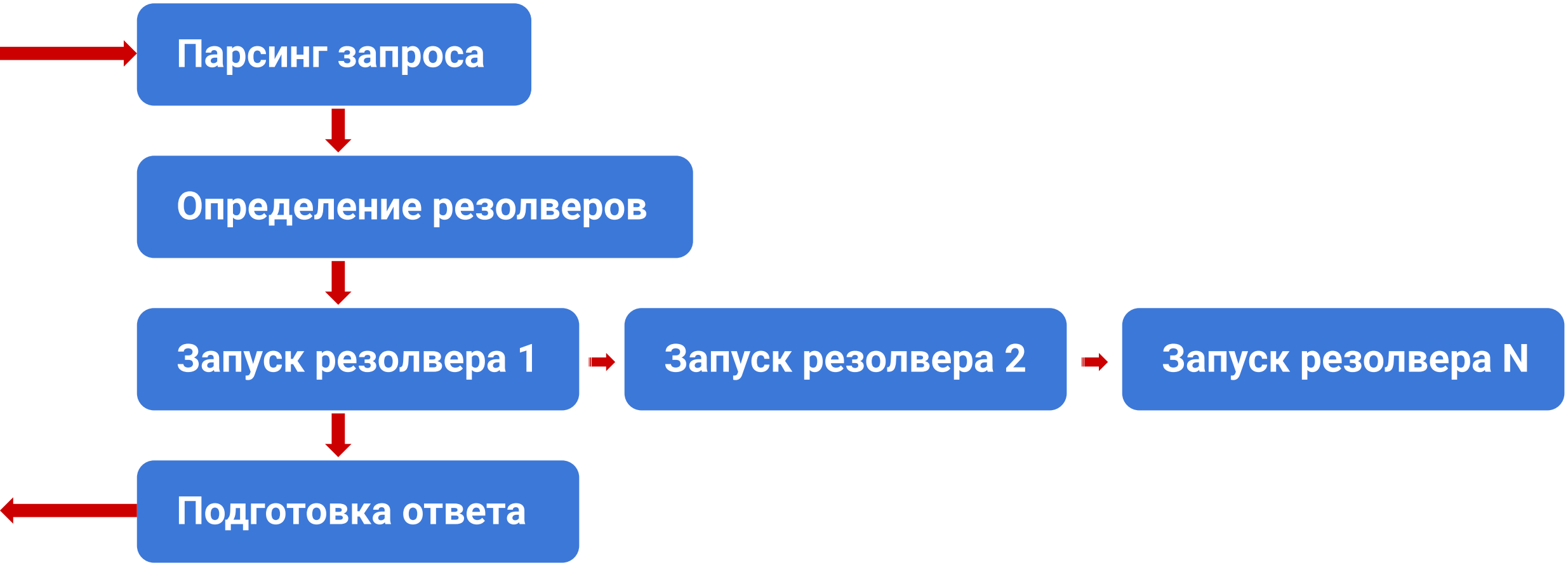
```
createTask (input CreateTaskInput!): CreateTaskPayload

editTask (input EditTaskInput!): EditTaskPayload

deleteTask (input DeleteTaskInput!): DeleteTaskPayload

...
```



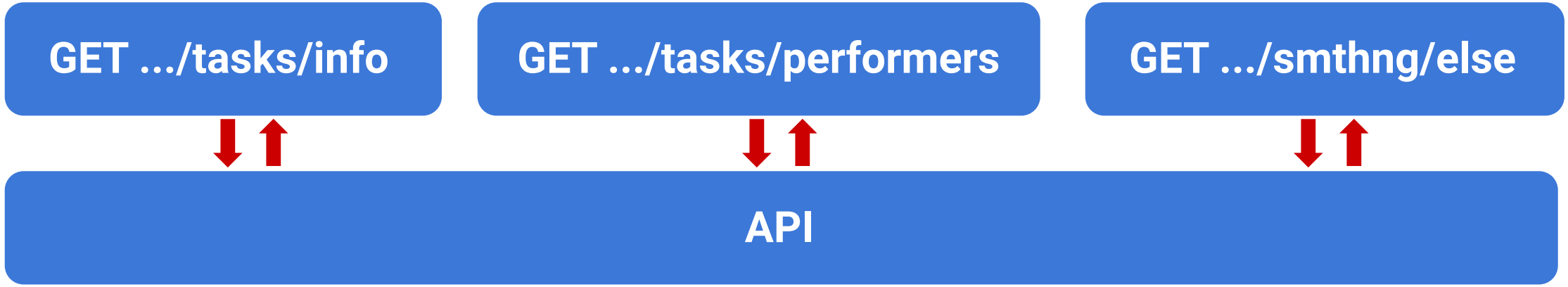
Группировка мутаций

```
mutation {
    task {
        create(input: { performer: 2 text: "create it"})
        edit(input: {id: 3 text: "edit it"})
        delete(input: {id: 3})
    }
}
```

Синхронное получение данных



Для REST



Тяжелые данные в GraphQL можно вынести в отдельный запрос

```
{
    easyData {}
    stillEasyData {}
    ~~heavyData {}~~
}
```

Auth GraphQL

Аутентификация:
- rest эндпойнты
- GraphQL query

Идентификация:
- JWT
- session

Авторизация:
- на уровне GraphQL-схемы
- на уровне полей (в resolve методах)
- на уровне связей между типами (в resolve методах)

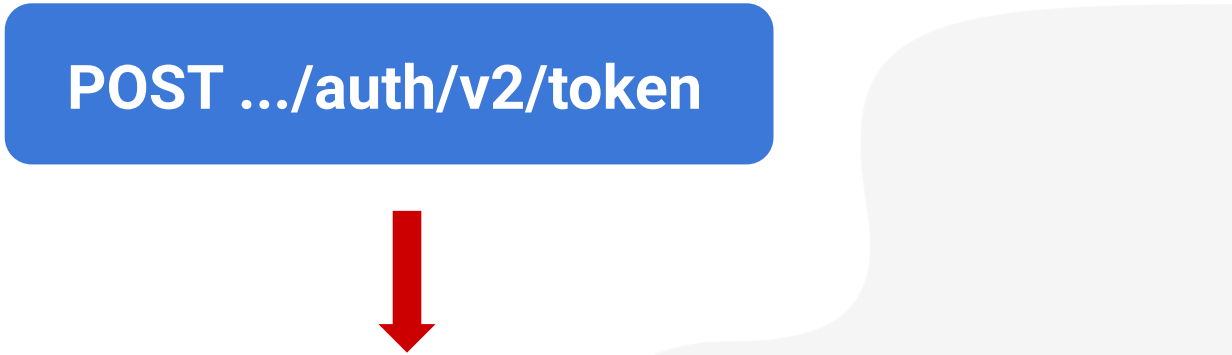
Auth GraphQL

Аутентификация:

- rest эндпойнты
- GraphQL query/mutation

Идентификация:

- JWT
- session



Авторизация:

- на уровне GraphQL-схемы
- на уровне полей (в resolve методах)
- на уровне связей между типами (в resolve методах)

Auth GraphQL

Аутентификация:

- rest эндпойнты
- GraphQL query/mutation

```
{
    auth (input AuthInput!): AuthPayload
}
```

Идентификация:

- JWT
- session

Авторизация:

- на уровне GraphQL-схемы
- на уровне полей (в resolve методах)
- на уровне связей между типами (в resolve методах)

Auth GraphQL

Аутентификация:

- rest эндпойнты
- GraphQL query/mutation

Идентификация:

- JWT
- session

Авторизация:

- на уровне GraphQL-схемы
- на уровне полей (в resolve методах)
- на уровне связей между типами (в resolve методах)

```
query {
    viewer { # открыто для всех
        getNews
        getAds
    }
    me { # данные для текущего юзера
        nickname
        photo
    }
    admin { # доступно только админам
        shutdown
        exposePersonalData
    }
}
```

Auth GraphQL

Аутентификация:

- rest эндпойнты
- GraphQL query/mutation

Идентификация:

- JWT
- session

Авторизация:

- на уровне GraphQL-схемы
- на уровне полей (в resolve методах)
- на уровне связей между типами (в resolve методах)

```
# yaml schema difinition
tasks:
    type: Boolean
    access: "@=hasRole('SUPER_ADMIN')"
```

Auth GraphQL

Аутентификация:
- rest эндпойнты
- GraphQL query/mutation

Идентификация:
- JWT
- session

Авторизация:
- на уровне GraphQL-схемы
- на уровне полей (в resolve методах)
- на уровне связей между типами (в resolve методах)

Auth GraphQL

Аутентификация:

- rest эндпойнты
- GraphQL query/mutation

Идентификация:

- JWT
- session

Авторизация:

- на уровне GraphQL-схемы
- на уровне полей (в resolve методах)
- на уровне связей между типами (в resolve методах)

```
public function createTask($data)
{
    if (!$security->isGranted('TASK_CREATE')) {
        throw new createAccessDeniedException();
    }
}
```

Auth GraphQL

Аутентификация:

- rest эндпойнты
- GraphQL query/mutation

Идентификация:

- JWT
- session

```
public function getTasks($data)
{
    if (!$security->isGranted('TASK_LIST')) {
        throw new AccessDeniedException();
    }
    $tasks = $repo->getTasks();
    $result = [];
    foreach($tasks as $task) {
        if ($security->isGranted('TASK_LIST', $task)) {
            $result[] = $task;
        }
    }
}
```

Авторизация:

- на уровне GraphQL-схемы
- на уровне полей (в resolve методах)
- на уровне связей между типами (в resolve методах)

Best practices

Группируйте аргументы в input-типы

```
query {
    articles(filter: ArticleFilter, limit: Int): [Article]
}

input ArticleFilter {
    lang: Stirng
    userId: Int
    rating: MinMaxInput
}

input MinMaxInput {
    min: Int
    max: Int
}
```

Best practices

Объединяйте аргументы в общий input-тип и создавайте уникальные input-типы для каждой мутации

```
# Good:
mutation ($input: UpdatePostInput!) {
    updatePost(input: $input) { ... }
}

# Not so good - гораздо сложнее писать запрос (дубль переменных)
mutation ($id: ID!, $newText: String, ...) {
    updatePost(id: $id, newText: $newText, ...) { ... }
}
```

Best practices

Мутация должна возвращать уникальный payload-тип.
Поля в payload необязательные

```
type Mutation {
-   createPerson(input: ...): Person                    # BAD
+   createPerson(input: ...): CreatePersonPayload       # GOOD
}

+ type CreatePersonPayload {
+    recordId: ID
+    record: Person
+    error: Error
+    # ... любые другие поля, которые пожелаете
+ }
```

Best practices

Создавайте собственные скалярные типы (String -> DateTime)

```
type Mutation {
-   setTime(date: String): SetTimePayload   # BAD
+   setTime(date: DateTime): SetTimePayload   # GOOD
}
```

Best practices

Выделяйте аргументы filter, sort, paginationInfo

```
type Query {
  articles(filter: ArticleFilter sort: [ArticleSort!]): [Article]
}

input ArticleFilter {
  authorId: Int
  tags: [String]
  lang: LangEnum
}

enum ArticleSort { ID_ASC, ID_DESC, TEXT_MATCH }
```

Best practices

Для постраничной разбивки:

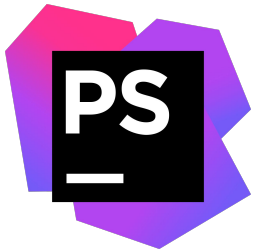
```
type PaginationInfo {
  totalPages: Int!
  totalItems: Int!
  page: Int!
  perPage: Int!
  hasNextPage: Boolean!
  hasPreviousPage: Boolean!
}
```

Для инфинити скролл
Relay Cursor Connections Specification:

```
{
  articles(first: 10, after: "cursorVal"){
    edges {
      cursor
      node { # на 3-уровне данные записи
        id
        name
      }
    }
    pageInfo {
      hasNextPage
    }
  }
}
```

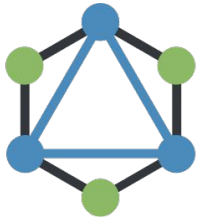
Инструменты

GraphiQL



JS-GraphQL

GraphQL
Playground



Altair

Полезные ссылки

Symfony bundle, который использовался в проекте:
https://github.com/overblog/GraphQLBundle

Пример вложенных мутаций на Lighthouse for Laravel:
https://lighthouse-php.com/master/eloquent/nested-mutations.html

GraphQL best practices:
https://graphql-rules.com

Auth with GraphQL:
https://nodkz.github.io/conf-talks/articles/graphql/auth/

Спецификация по GraphQL:
https://spec.graphql.org/

Schema diffing:
https://graphql-rules.com/rules/authorization-schema-diffing