GraphQL как будущее клиентских API

Андрей Лось. Revolut

План

1. Что такое REST?
2. История создания GraphQL
3. GraphQL, каким его увидели мы
4. Как это работает?
5. Инструменты

Что такое REST?

Из чего состоит REST?

- **Ресурс** Например, *“**/users**”*.
- **Метод** GET, POST, DELETE, PUT.
- **Заголовки**
- **Тело запроса**
- **Тело ответа**
- **Статус ответа**

Примеры запросов

Получение:

/users GET

/users/:id GET

/users/:id/posts/ GET

/users/:id/posts/:id GET

Изменение:

/users/:id PUT {...}

/users/:id DELETE

/users/:id/change-password PUT {...}

/users/:id/ban PUT

Создание:

/users POST {...}

Основные фичи

- Простота
- Расширяемость
- Кэширование
- Обозримость
- Знакомость
- **Якобы**

История GraphQL

Проблемы, с которыми столкнулся Facebook

- Рост рынка смартфонов;
- Веб, вместо приложения;
- Недостаточная гибкость API;
- Сложные структуры данных;
- Множество запросов – дорого;
- Лишние данные.

Интернет по кабелю

/feed/news

50ms

/posts/1/comments

50ms

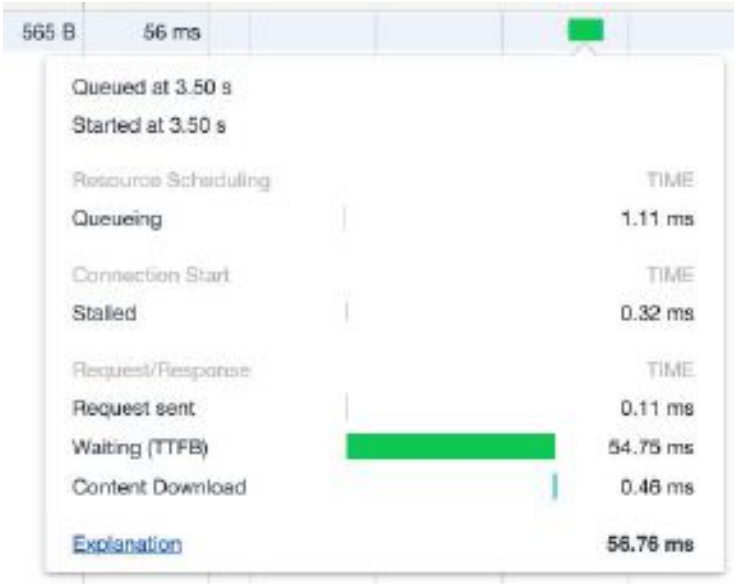
/comments/1/user

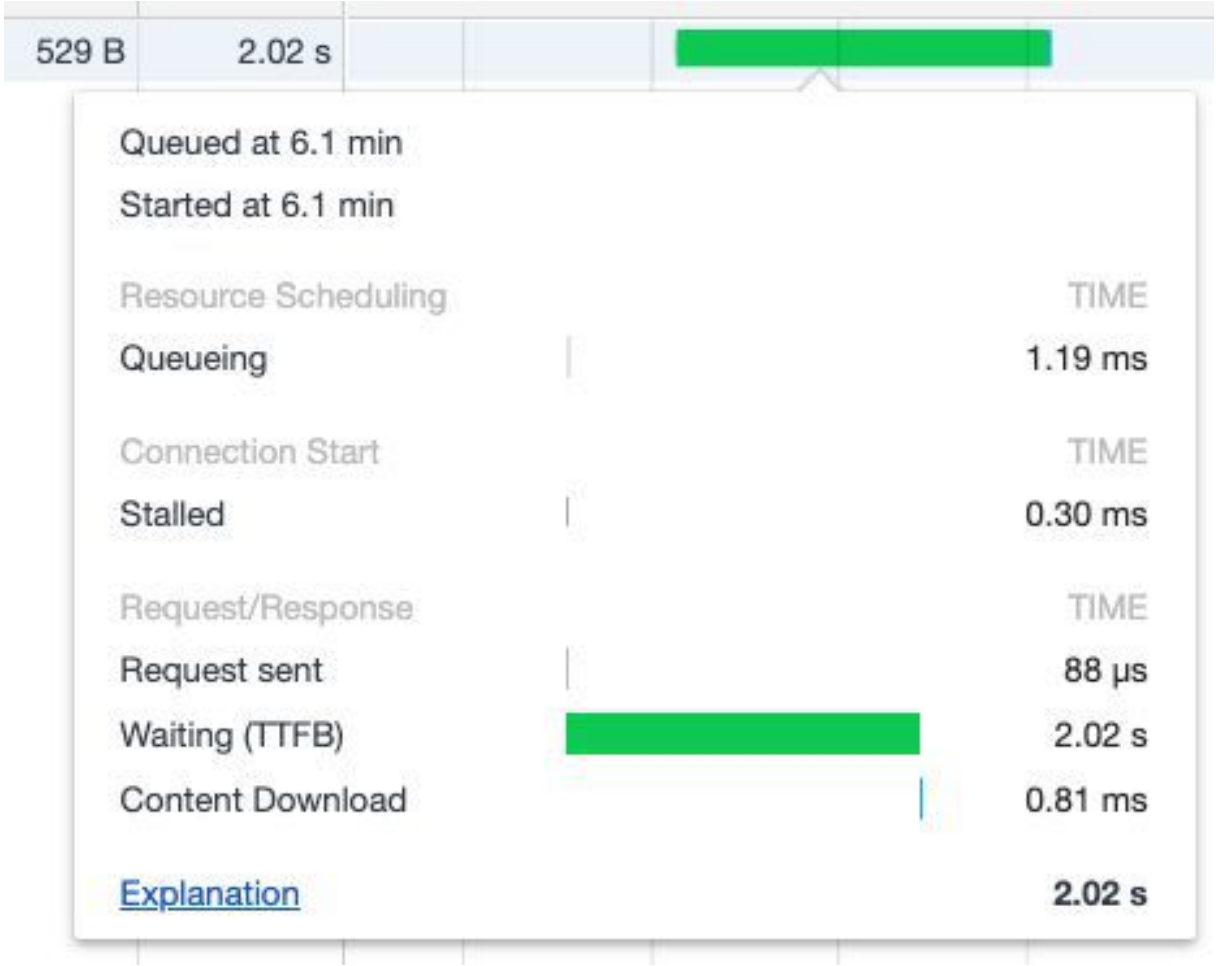
/comments/2/user

/comments/3/user

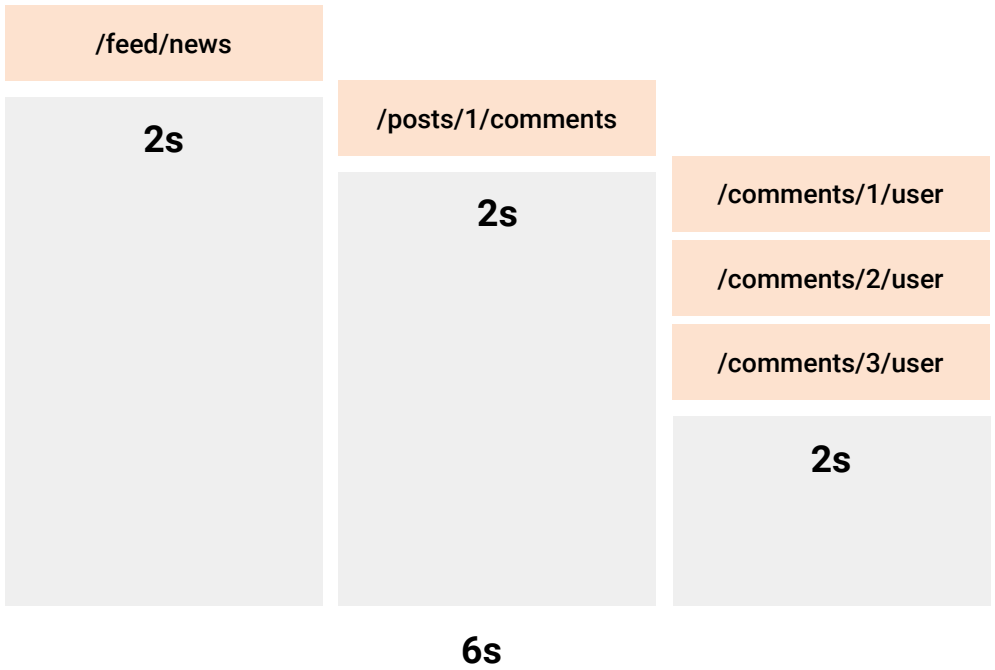
50ms

150ms




529 B
2.02 s
Queued at 6.1 min
Started at 6.1 min
Resource Scheduling
TIME
Queueing
1.19 ms
Connection Start
TIME
Stalled
0.30 ms
Request/Response
TIME
Request sent
88 µs
Waiting (TTFB)
2.02 s
Content Download
0.81 ms
Explanation
2.02 s

Медленный 3G



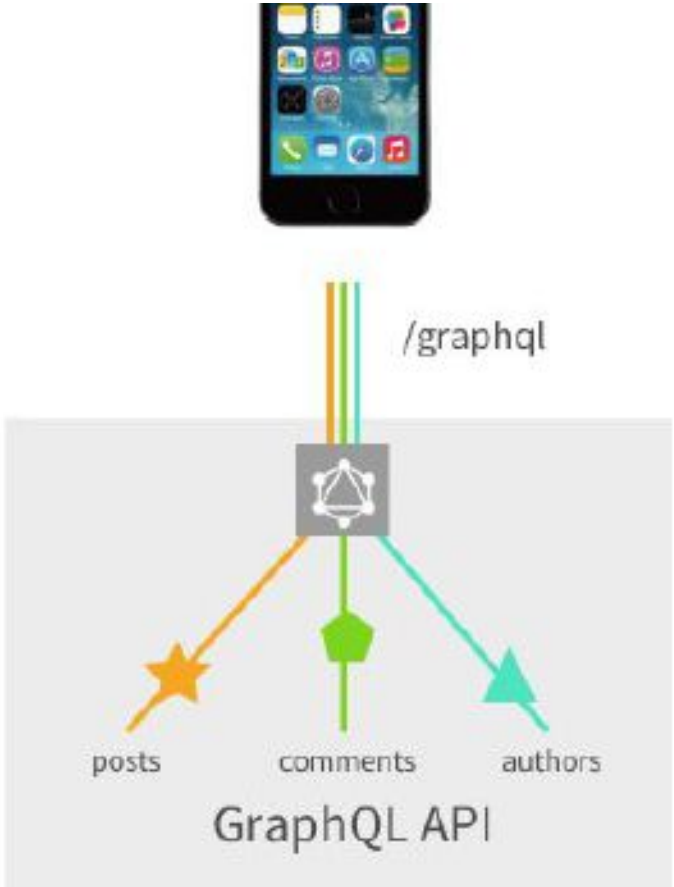
= 6 секунд

Время процессинга сервером:
15-30 мс

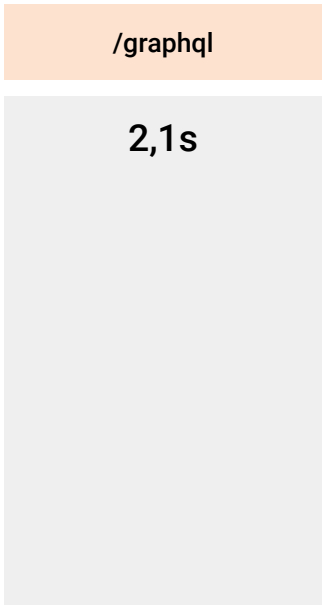
А что, если можно было бы?

- Получить все одним запросом;
- Получать только те поля, которые нужны клиенту;
- Типизировать, что принимает и отдает сервер;
- Получать документацию по умолчанию;
- И т.д.


/posts
/comments
/authors
REST API


/graphql
posts
comments
authors
GraphQL API

Медленный 3G



= 2,1 секунды

GraphQL, каким он дошел до нас

/graphql POST

Демо GraphiQL

Простой запрос

```
query {
  user(id: 1) {
    id
    firstName
    secondName
  }
}
```

/users/1

Запрос посложнее

```
query {
  feed(type: NEWS) {
    body
    title
    author {
      fullName
    }
    comments {
      author {
        fullName
      }
      body
    }
  }
}
```

Запрос посложнее

```
query {
  feed(type: NEWS) {
    body
    title
    author {
      fullName
    }
    comments {
      author {
        fullName
      }
      body
    }
  }
}
```

```
{
 "data": {
   "feed": [
     {
       "body": "...",
       "title": "...",
       "author": {
         "fullName": "..."
       },
       "comments": [
         {
           "body": "...",
           "author": {
             "fullName": "..."
           }
         }
       ]
     }
   ]
 }
}
```

Запрос посложнее

```
query {
  feed(type: NEWS) {
    body
    title
    author {
      fullName
    }
    comments {
      author {
        fullName
      }
      body
    }
  }
}
```

/feed/news

Запросить комментарии:
/posts/1/comments
…
/posts/n/comments

Запросить авторов:
/comments/1/author
…
/comments/n/author

Основы

Короче говоря

- Спецификация;
- Является частью Linux Foundation;
- Язык описания схемы;
- Язык запросов;
- Исполняющий движок.

Основные фичи

- Документация из коробки;
- Все типизировано;
- Мощнейшие инструменты разработчика;
- Гибкость. Получаешь только то, что нужно;
- Без версии, легко эволюционировать API.

Как это работает?

Никакой магии

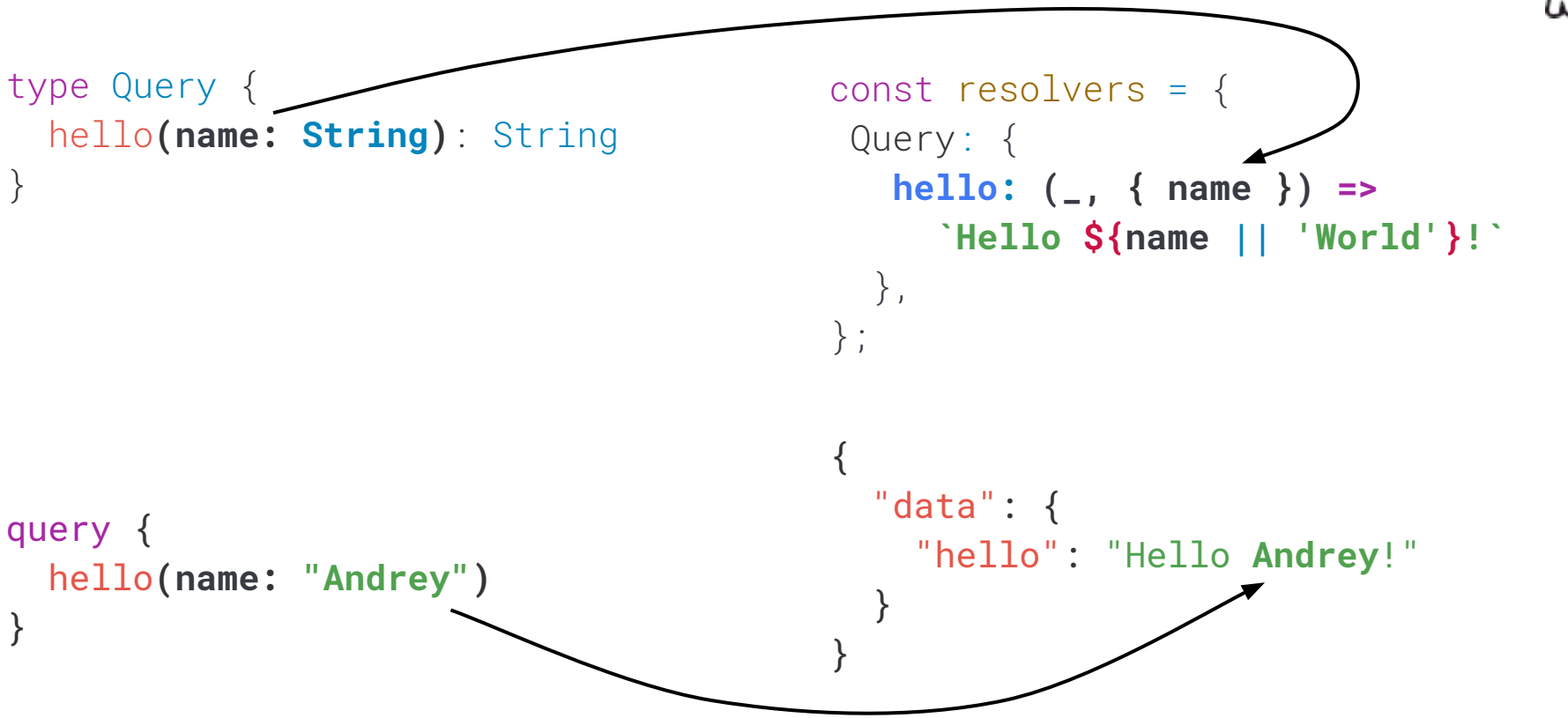
Демо

```
type Query {
  hello: String
}
```

```
query {
  hello
}
```

```
const resolvers = {
  Query: {
    hello: () => 'Hello world!'
  },
}
```

```
{
  "data": {
    "hello": "Hello World!"
  }
}
```


type Query {
hello(name: String): String
}
const resolvers = {
Query: {
hello: (_, { name }) =>
`Hello ${name || 'World'}!`
},
};
query {
hello(name: "Andrey")
}
{
"data": {
"hello": "Hello Andrey!"
}
}

```
query {
  user(id: "1") {
    id
    fullName
  }
}
```

```
{
  "data": {
    "user": {
      "id": "1",
      "fullName": "Andrey Los"
    }
  }
}
```

```
type Query {
  user(id: ID!): User
}

type User {
  id: ID!
  fullName: String!
}
```

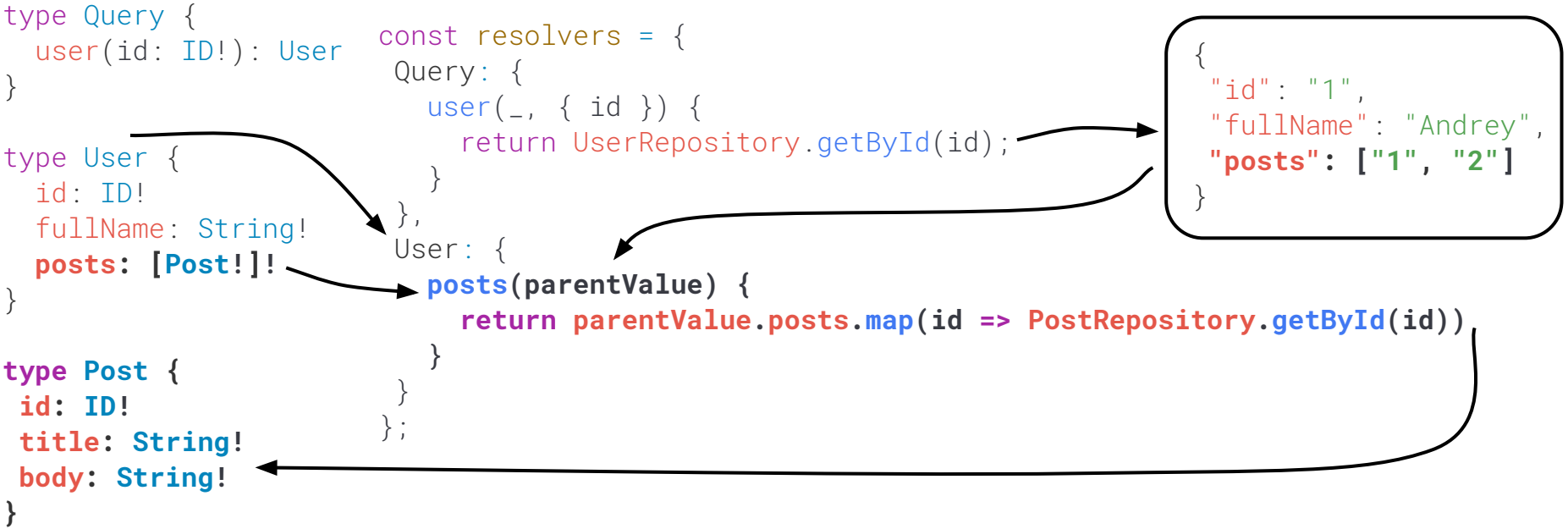
```
const resolvers = {
  Query: {
    user(_, { id }) {
      return UserRepository.getById(id)
    }
  },
  User: {
    id: parent => parent.id,
    fullName: parent => parent.fullName
  }
}
```

```
{
 "id": "1",
 "fullName": "Andrey"
}
```


type Query {
user(id: ID!): User
}
const resolvers = {
Query: {
user(_, { id }) {
return UserRepository.getById(id)
}
}
}
type User {
id: ID!
fullName: String!
}
{
"id": "1",
"fullName": "Andrey"
}

```
query {
 user(id: "1") {
   id
   fullName
   posts {
     id
     title
     body
   }
 }
}
```

```
{
  "data": {
    "user": {
      "id": "1",
      "fullName": "Andrey",
      "posts": [
        {
          "id": "1",
          "title": "Hi",
          "body": "Hello World!"
        },
        {
          "id": "2",
          "title": "Cześć",
          "body": "Dzień dobry!"
        }
      ]
    }
  }
}
```


type Query {
user(id: ID!): User
}
type User {
id: ID!
fullName: String!
posts: [Post!]!
}
type Post {
id: ID!
title: String!
body: String!
}
const resolvers = {
Query: {
user(_, { id }) {
return UserRepository.getById(id);
}
},
User: {
posts(parentValue) {
return parentValue.posts.map(id => PostRepository.getById(id))
}
}
};
{
"id": "1",
"fullName": "Andrey",
"posts": ["1", "2"]
}


```
{
  "data": {
    "user": {
      "id": "1",
      "fullName": "Andrey",
      "posts": [
        {
          "id": "1",
          "title": "Hi",
          "body": "Hello World!"
        },
        {
          "id": "2",
          "title": "Cześć",
          "body": "Dzień dobry!"
        }
      ]
    }
  }
}
```

Инструменты

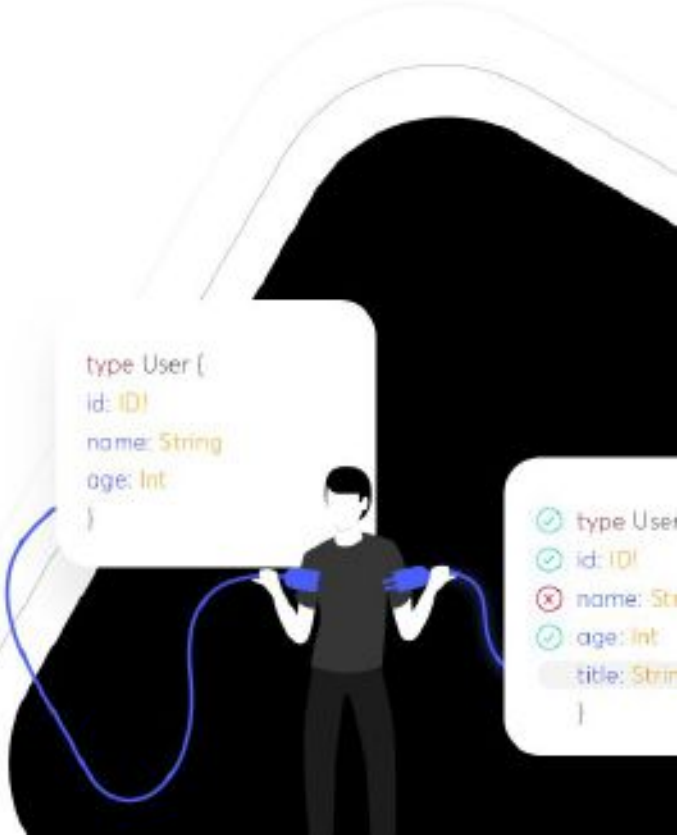
GraphiQL
GraphQL Playground

Демо: как он работает

GraphQL
inspector

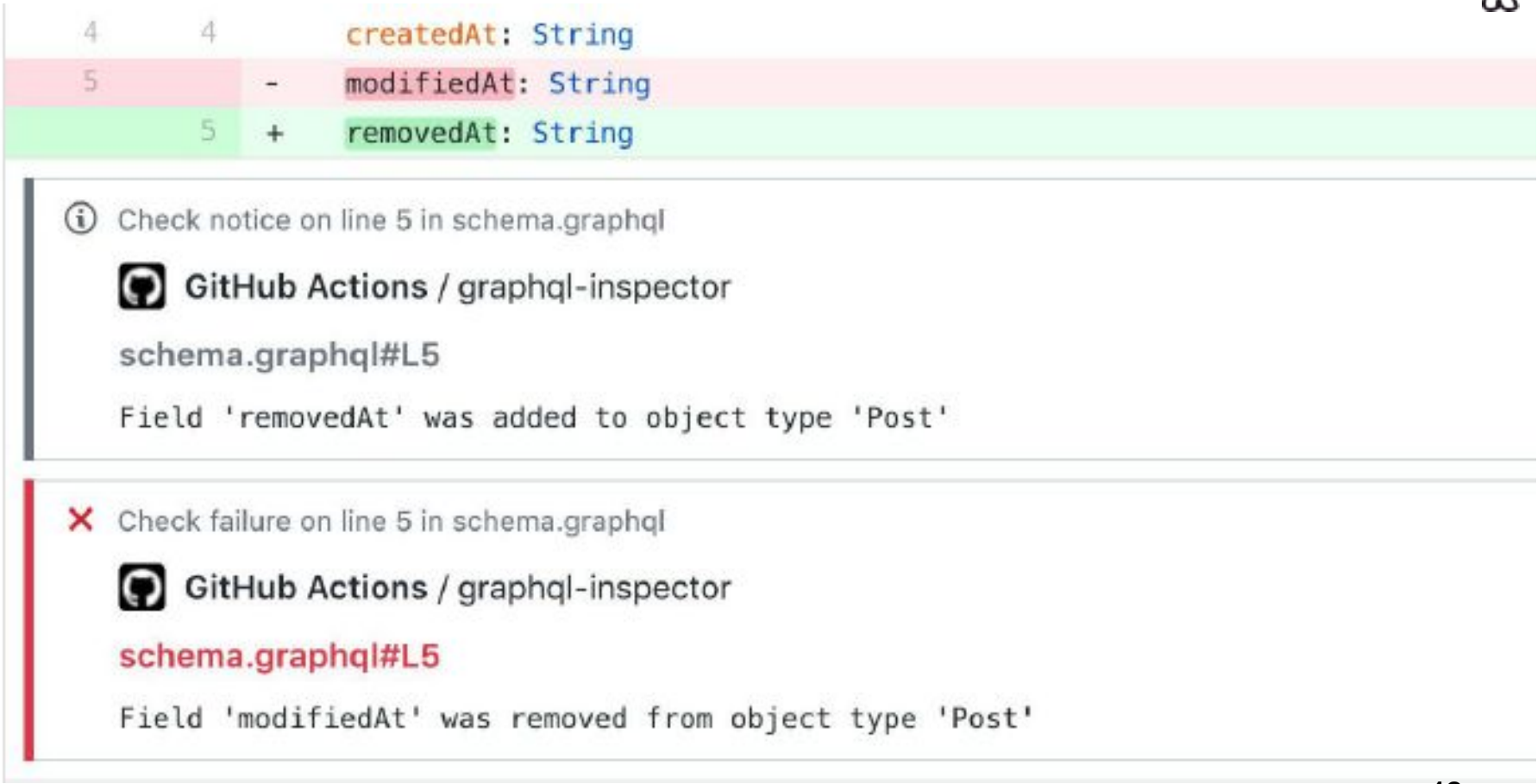
Bulletproof your GraphQL API

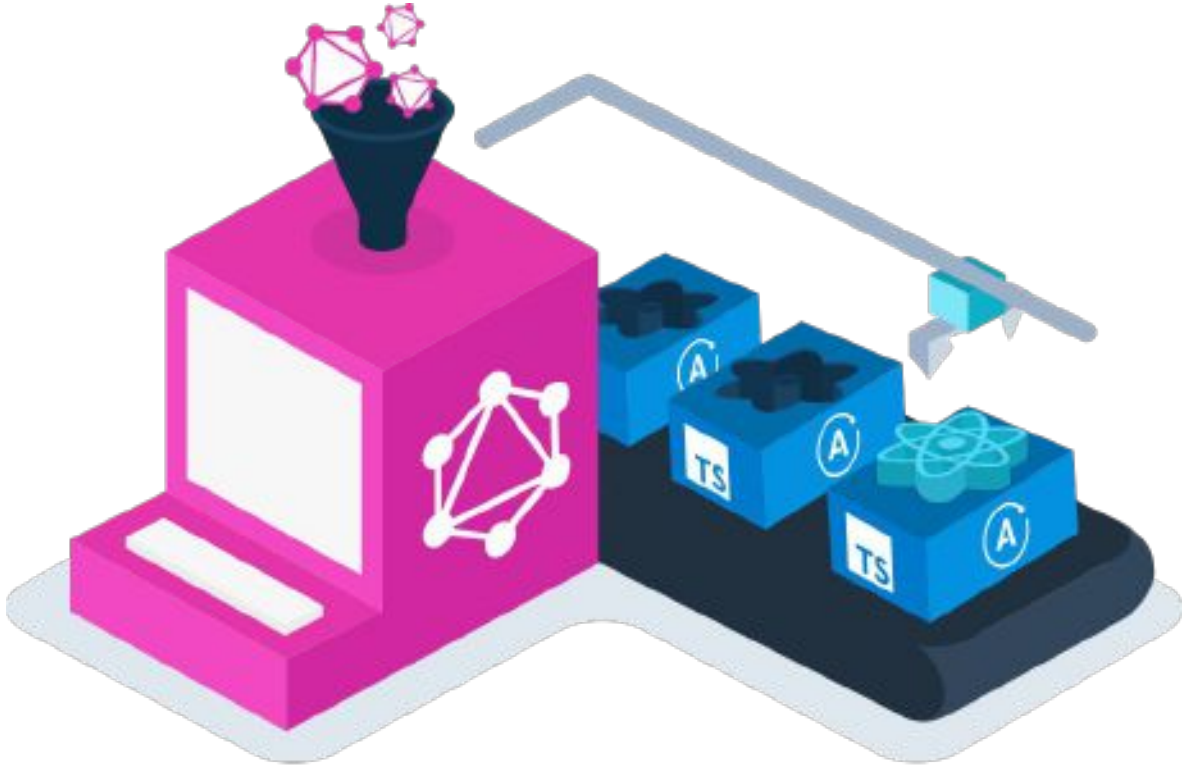
Detects every change, similar, duplicated types & Validates documents against a schema and looks for deprecated usage.



graphql-inspector diff OLD_SCHEMA NEW_SCHEMA

```
Detected the following changes (6) between schemas:

✖ Field posts was removed from object type Query
✖ Field modifiedAt was removed from object type Post
✔ Field Post.id changed type from ID to ID!
✔ Deprecation reason on field Post.title has changed from No more used to undefined
✔ Field Post.title changed type from String to String!
✔ Field Post.createdAt changed type from String to String!

error Detected 2 breaking changes
```


4 4 createdAt: String
5 - modifiedAt: String
5 + removedAt: String
Check notice on line 5 in schema.graphql
GitHub Actions / graphql-inspector
schema.graphql#L5
Field 'removedAt' was added to object type 'Post'
Check failure on line 5 in schema.graphql
GitHub Actions / graphql-inspector
schema.graphql#L5
Field 'modifiedAt' was removed from object type 'Post'



{ GraphQL }

code generator

TypeScript

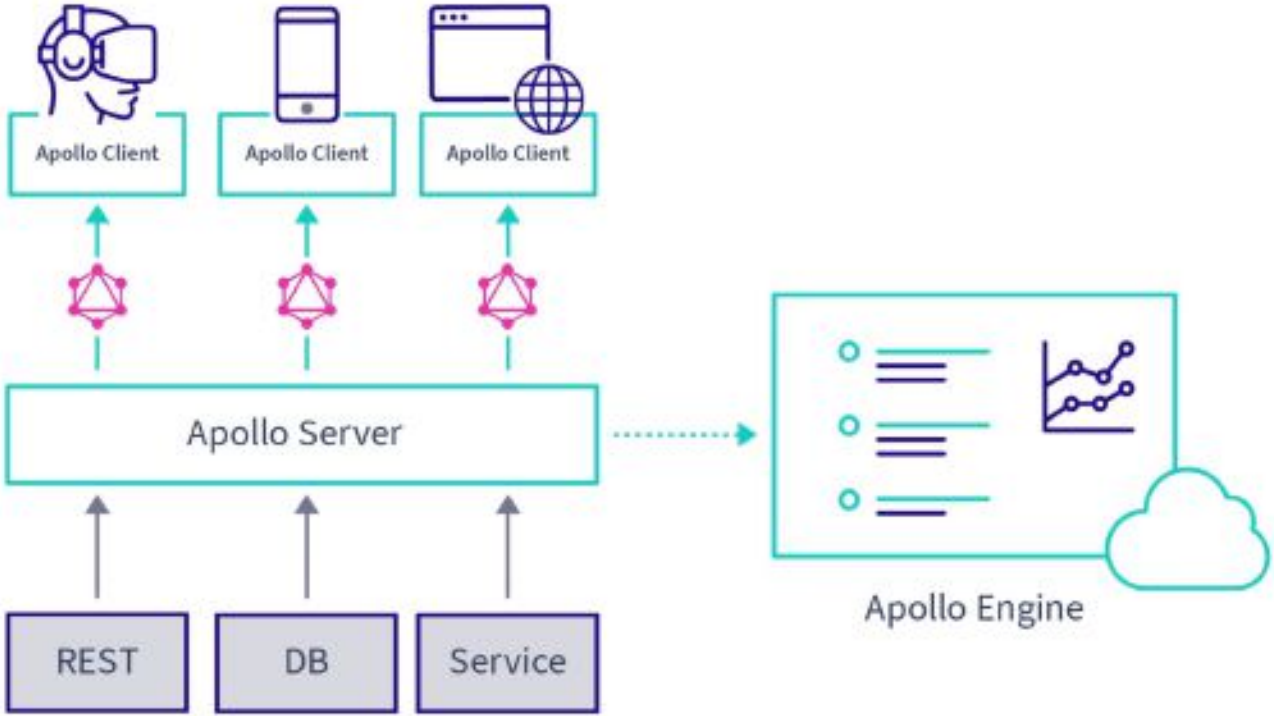
```
schema {
  query: Query
}

type Query {
  me: User!
  user(id: ID!): User
}

enum Role {
  USER,
  ADMIN,
}

type User {
  id: ID!
  username: String!
  email: String!
  role: Role!
}
```

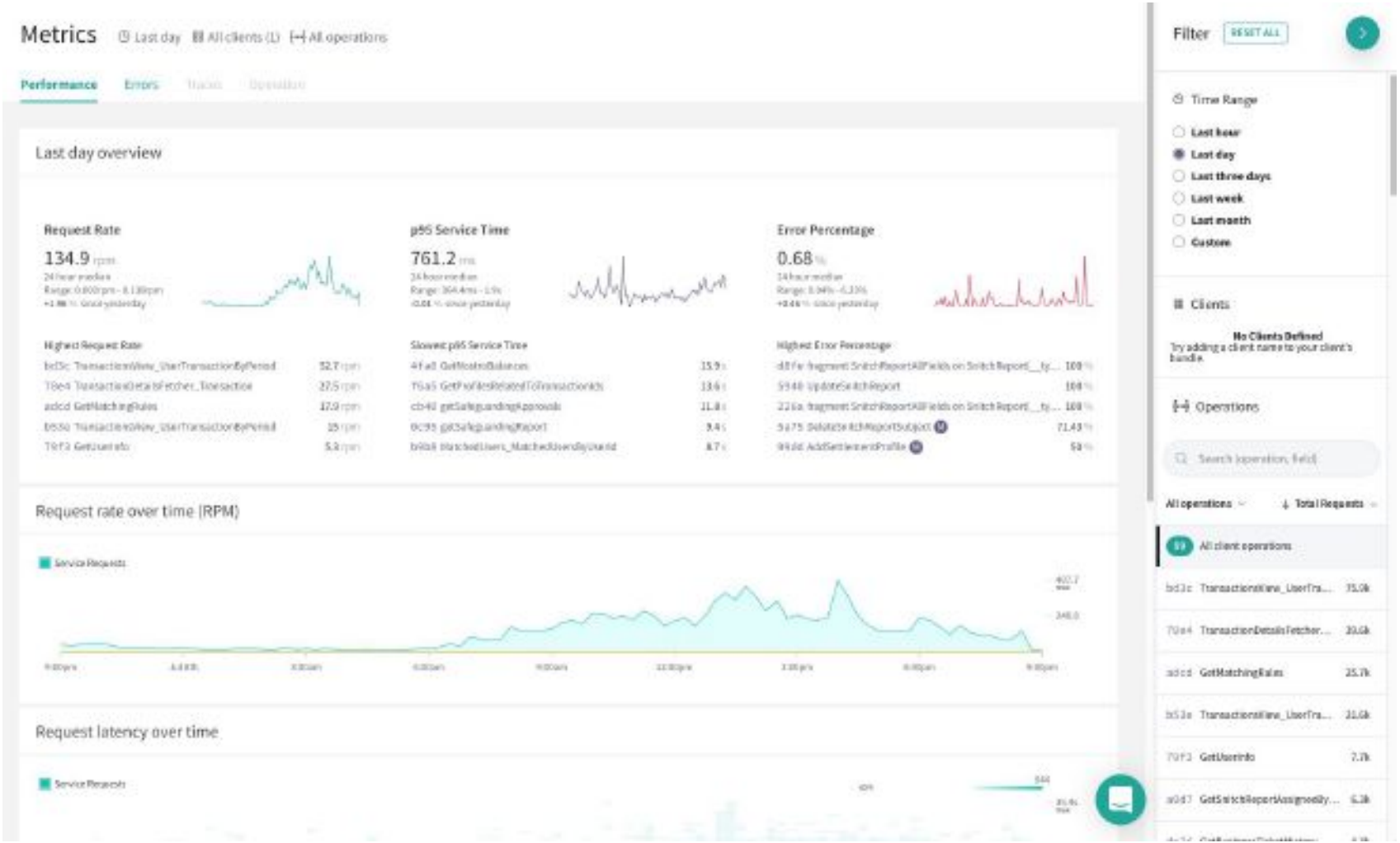
```
export type Maybe<T> = T | null;
/** All built-in and custom scalars, mapped to their actual values
export type Scalars = {
  ID: string;
  String: string;
  Boolean: boolean;
  Int: number;
  Float: number;
};

export type Query = {
  __typename?: "Query";
  me: User;
  user?: Maybe<User>;
};

export type QueryUserArgs = {
  id: Scalars["ID"];
};

export enum Role {
  User = "USER",
  Admin = "ADMIN"
}

export type User = {
  __typename?: "User";
  id: Scalars["ID"];
  username: Scalars["String"];
  email: Scalars["String"];
  role: Role;
};
```

GraphQL Voyager

Демо: как он работает

Apollo Engine




Metrics
Last day
All clients (1)
All operations
Performance
Errors
Last day overview
Request Rate
134.9
p95 Service Time
761.2
Error Percentage
0.68
Request rate over time (RPM)
Request latency over time
Filter
RESET ALL
Time Range
Last hour
Last day
Last three days
Last week
Last month
Custom
Clients
No Clients Defined
Operations
All client operations

Андрей Лось

Twitter: @RIP212

GitHub: @RIP21

Ссылки

GraphQL Codegen

GraphQL Modules

Apollo Engine

Apollo Server

Код примера из доклада

GraphQL Voyager

Onegraph