

GraphQL

Introduction

Неделя 15

Вступление

Чтобы лучше понять, что такое GraphQL и зачем он применяется, проследим проблемы, которые привели к созданию этой технологии.

Началось все в 2012 году в компании Facebook*, где он был разработан с целями упрощения своих API для клиентских приложений.

Об этой и других проблемах, которые решает GraphQL узнаем на следующих слайдах.

* Социальная сеть Facebook запрещена на территории РФ, Meta Platforms Inc. признана в РФ экстремистской

Проблема 1. Избыточность/недостаток данных в ответе

Также эти проблемы известны под названиями Over-fetching и Under-fetching.

Понять просто:

Over-fetching – клиент получает **данных больше**, чем ему действительно нужно. Сервер тратит дополнительные ресурсы на то, чтобы вернуть клиенту данные, которые он все равно использовать не будет.

Under-fetching – клиент получает **меньше данных**, чем ему нужно. В итоге ему нужно выполнять дополнительные запросы для получения всех данных, что сказывается на производительности клиента и нагрузке на сервер.

Пример избыточности данных

Spotify отображает пользователю плейлисты. Будет ли хорошим решением проектировать API вот так?

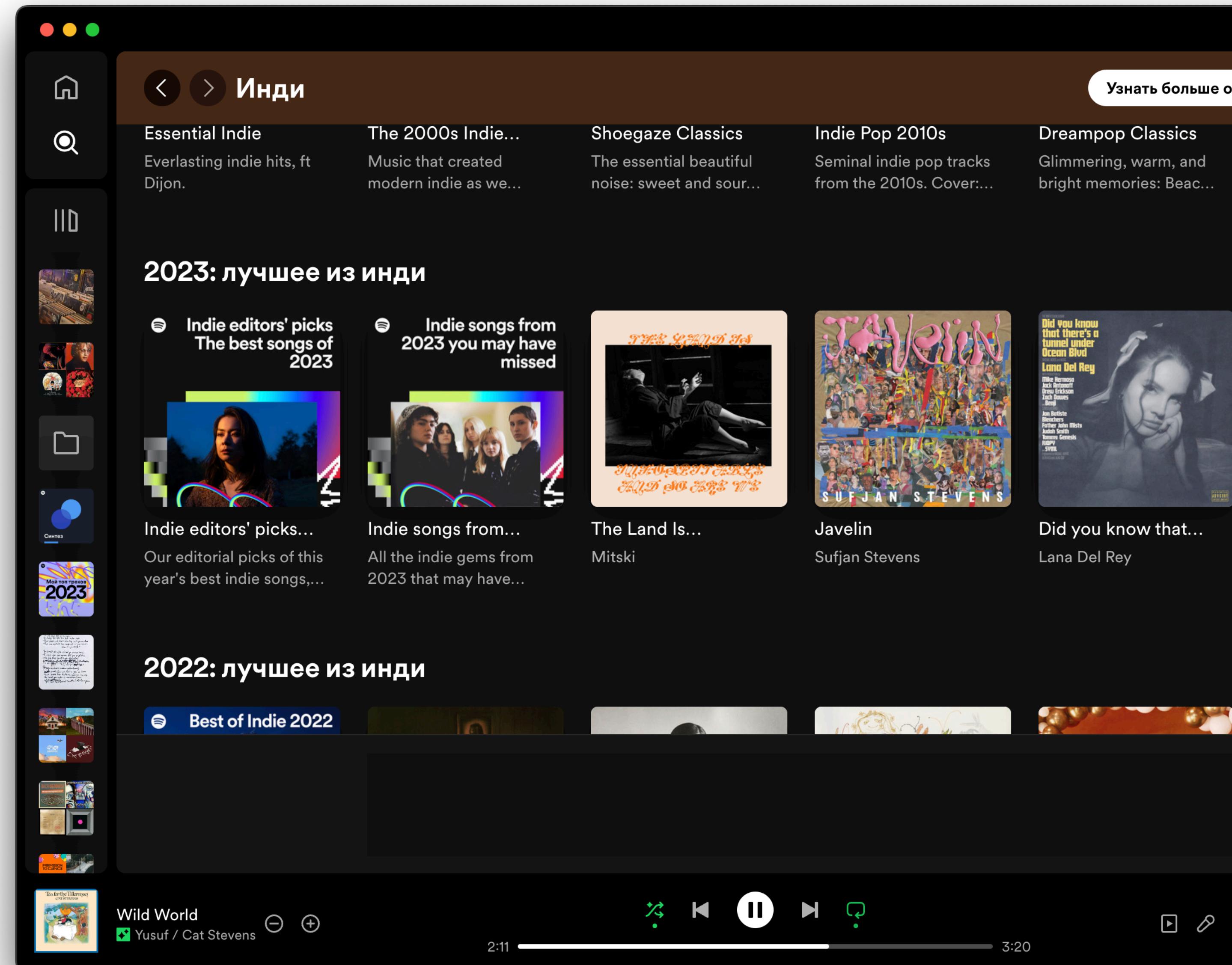
GET /playlists

{

title,
artist,
description,
image,
tracksNumber,
tracksInfo

...

}

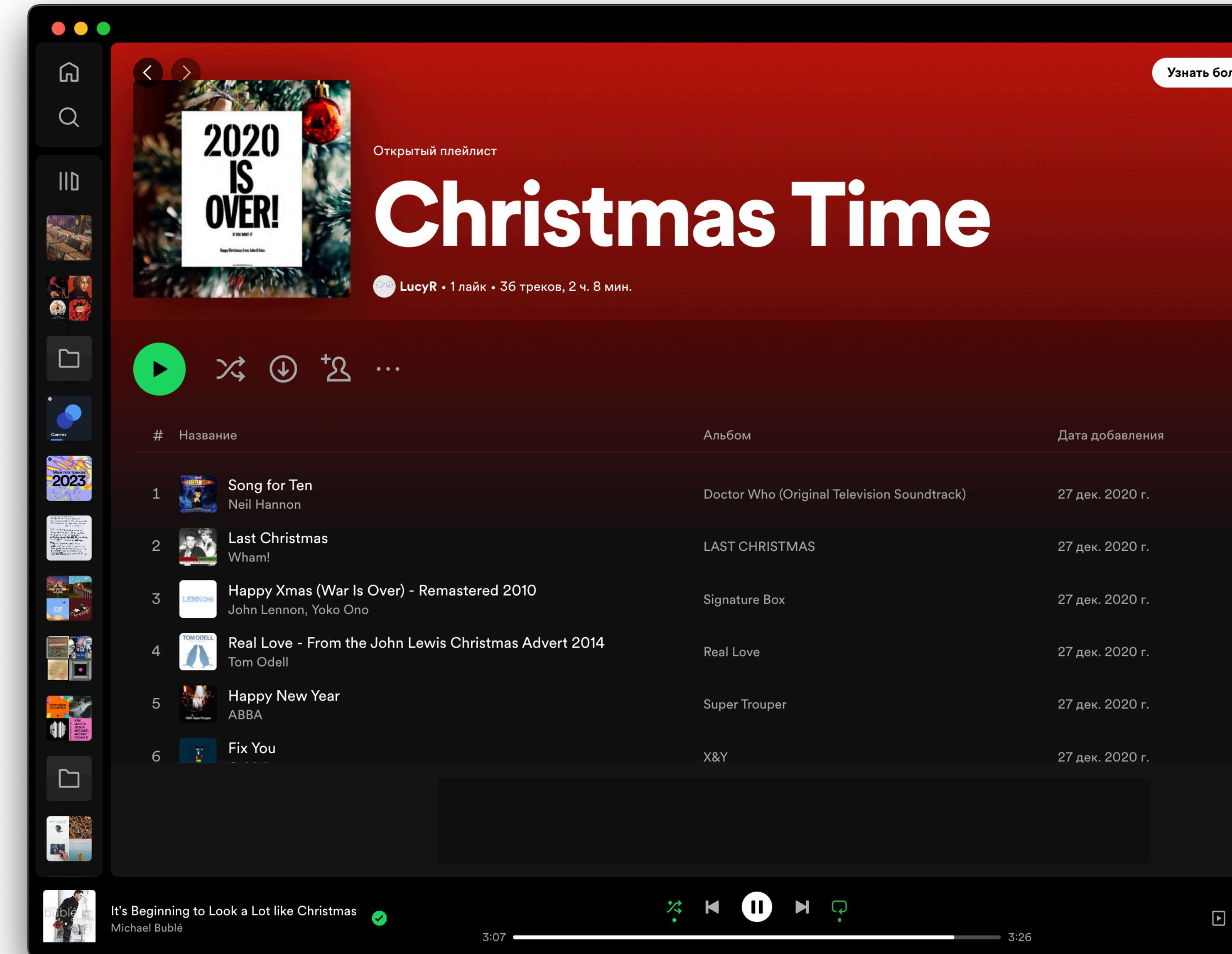


Пример недостатка данных

Spotify отображает пользователю содержимое плейлиста. Самый простой вариант реализации API может выглядеть вот так:

GET /playlists/1

```
{  
    title,  
    artist,  
    description,  
    tracks: [  
        {  
            track: /tracks/et2f53s,  
        },  
        {  
            track: /tracks/er7bo2,  
        }  
        ...  
    ]  
}
```



Клиент: У меня тормозит API. Можете посмотреть почему?

Тем временем ответ API:

```
2   "message": "Operation performed successfully",
3   "timestamp": 1650534696544,
4   "data": {
5     "contractData": {
269     "assetDtoList": [
2061       "creditLineDtoList": [],
2062       "guaranteeDtoList": [],
2063       "basel2": null,
2064       "benefit": null,
2065       "cashFlow": [
70179         "companySigners": [],
70180         "costs": [
70536           "creditDecision": [
70579             "credits": [],
70580             "customerCosts": [],
70581             "customerSignerList": [
70627               "documents": [],
70628               "financialConditions": [
70872                 "flexiblePayment": [
70964                   "frameAgreement": null,
70965                   "ifrs9": [
70987                     "indexing": [
71073                     "invoicePayment": [
71363                       "italianAgency": null,
71364                       "messages": [],
71365                       "ordersAndSuppliers": [
71475                         "pledges": [],
71476                         "poolMembers": [],
71477                         "poolShare": [],
71478                         "privacy": [],
71479                         "provision": null,
71480                         "provisionDetailList": [],
71481                         "publicAdministration": [],
71482                         "revenues": [],
71483                         "sepa": [
71539                           "services": [
71933                             "statusHistory": [
72019                               "subRent": [],
72020                               "sundryList": [
72730                                 "variationHistory": [
72862                                 "tcCompany": [
73067                                   "tempCtrfin": null,
73068                                   "contractGuarantors": [
73069                                     "guarantorList": []
73070                                   ],
73071                                     "quotationPlans": []
73072                                   ],
73073                                   "count": 0,
73074                                   "totalCount": 0,
73075                                   "jwtToken": null
73076                                 ]
73076
@ithumor
```

Вариант решения

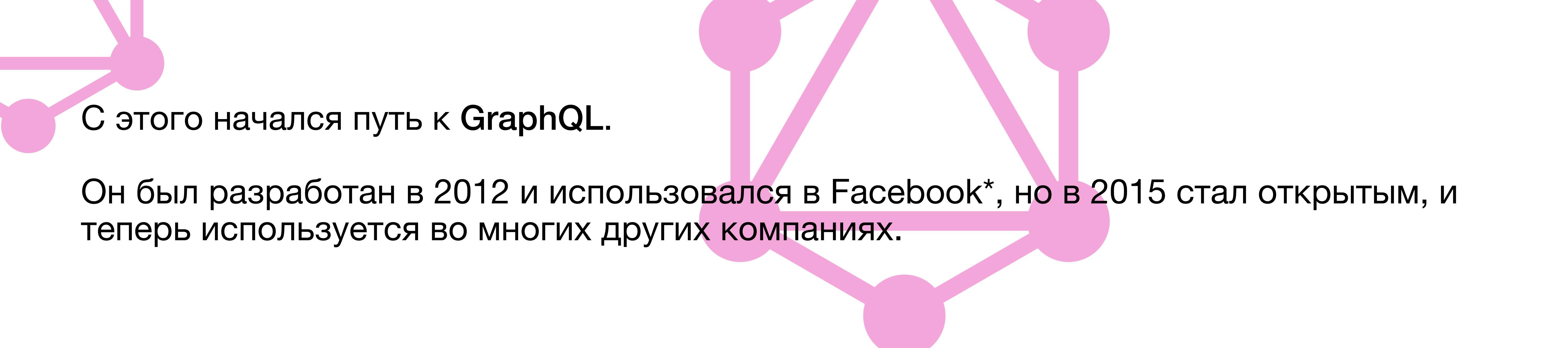
Давайте сделаем разные проекции для плейлиста.

Например, пусть одна возвращает информацию для отображения на главной странице, а другая – более подробную.

```
GET /playlists/1?projection=main
{
    title,
    artist,
    description,
    image
}
```

```
GET /playlists/1?projection=descriptive
{
    title,
    artist,
    description,
    image,
    tracksNumber,
    tracks: {...}
}
```

Это все, конечно, круто. Но если планируется расширение вашего приложения, придется создавать новые проекции.



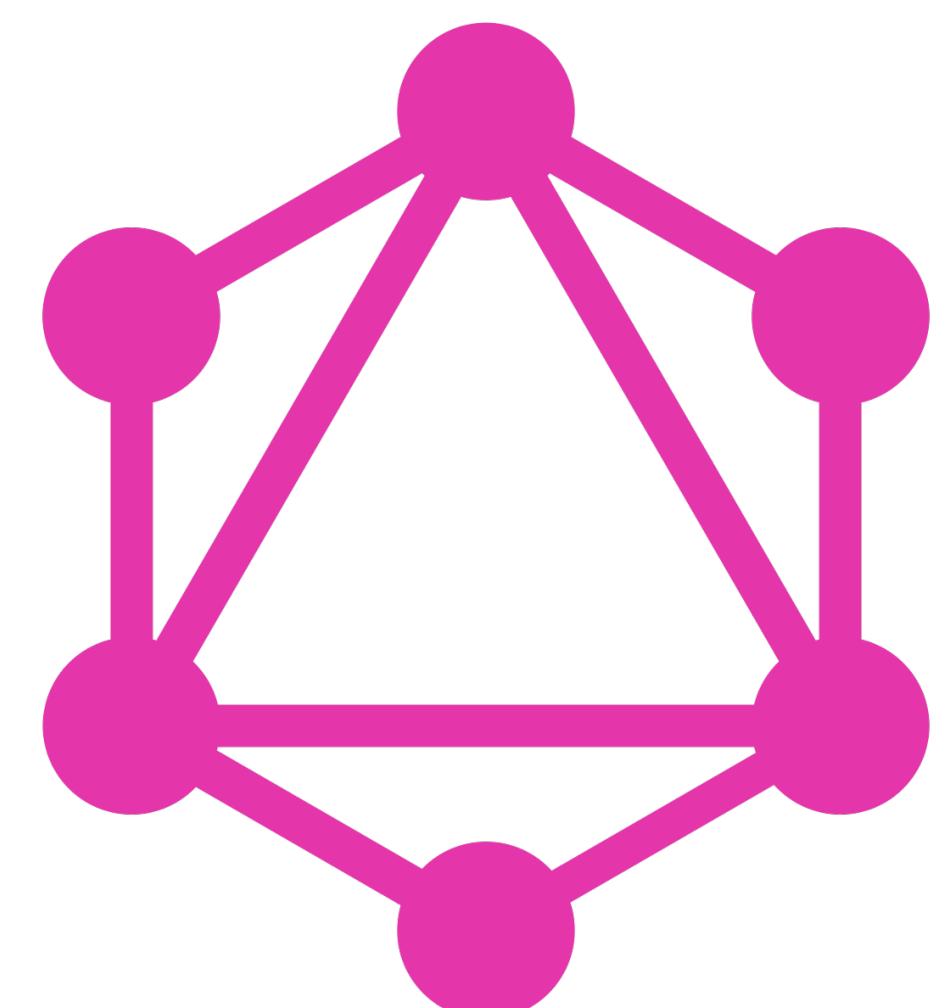
С этого начался путь к GraphQL.

Он был разработан в 2012 и использовался в Facebook*, но в 2015 стал открытым, и теперь используется во многих других компаниях.



Из названия следует, что это **язык запросов** (Graph Query Language), можно также сказать, что это технология для дизайна API (тут имеется ввиду design как проектирование).

О том, почему он Graph станет понятнее дальше.



* Социальная сеть Facebook запрещена на территории РФ, Meta Platforms Inc. признана в РФ экстремистской

Почему он Graph?

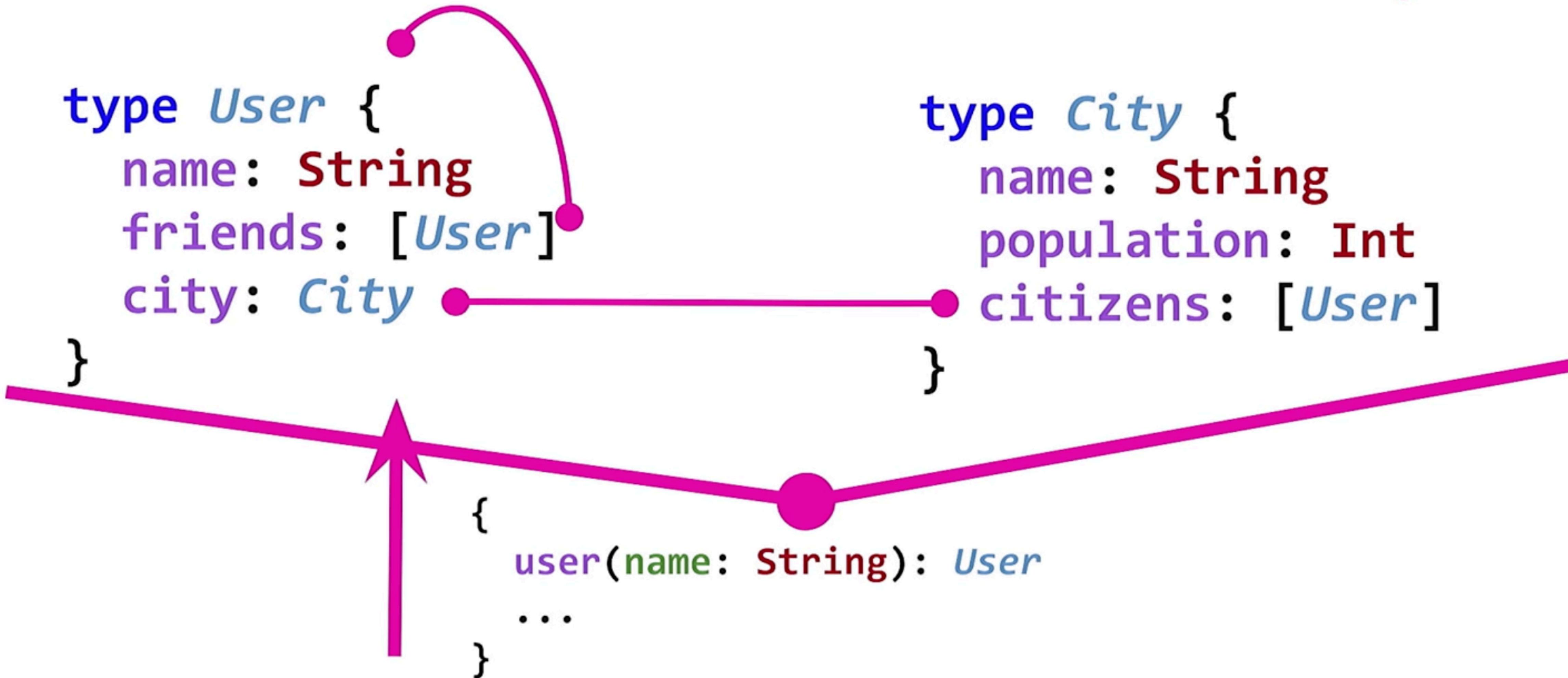


Схема данных

Это то, что было на предыдущем слайде – описание того, какие объекты данных являются узлами графа.

```
type Character {  
    name: String!  
    appearsIn: [Episode!]!  
}
```

В GraphQL из коробки поддерживаются различные типы данных (листья в графе).

- `Int` : A signed 32-bit integer.
- `Float` : A signed double-precision floating-point value.
- `String` : A UTF-8 character sequence.
- `Boolean` : `true` or `false`.
- `ID` : The ID scalar type represents a unique identifier, often used to refetch an object or as the key for a cache. The ID type is serialized in the same way as a String; however, defining it as an `ID` signifies that it is not intended to be human-readable.

Запрос данных. Query

```
type Query {  
    hero(episode: Episode): Character  
    droid(id: ID!): Droid  
}
```

```
query {  
    hero {  
        name  
    }  
    droid(id: "2000") {  
        name  
    }  
}
```

```
{  
    "data": {  
        "hero": {  
            "name": "R2-D2"  
        },  
        "droid": {  
            "name": "C-3PO"  
        }  
    }  
}
```

Типы запросов: Query, Mutation, Subscription

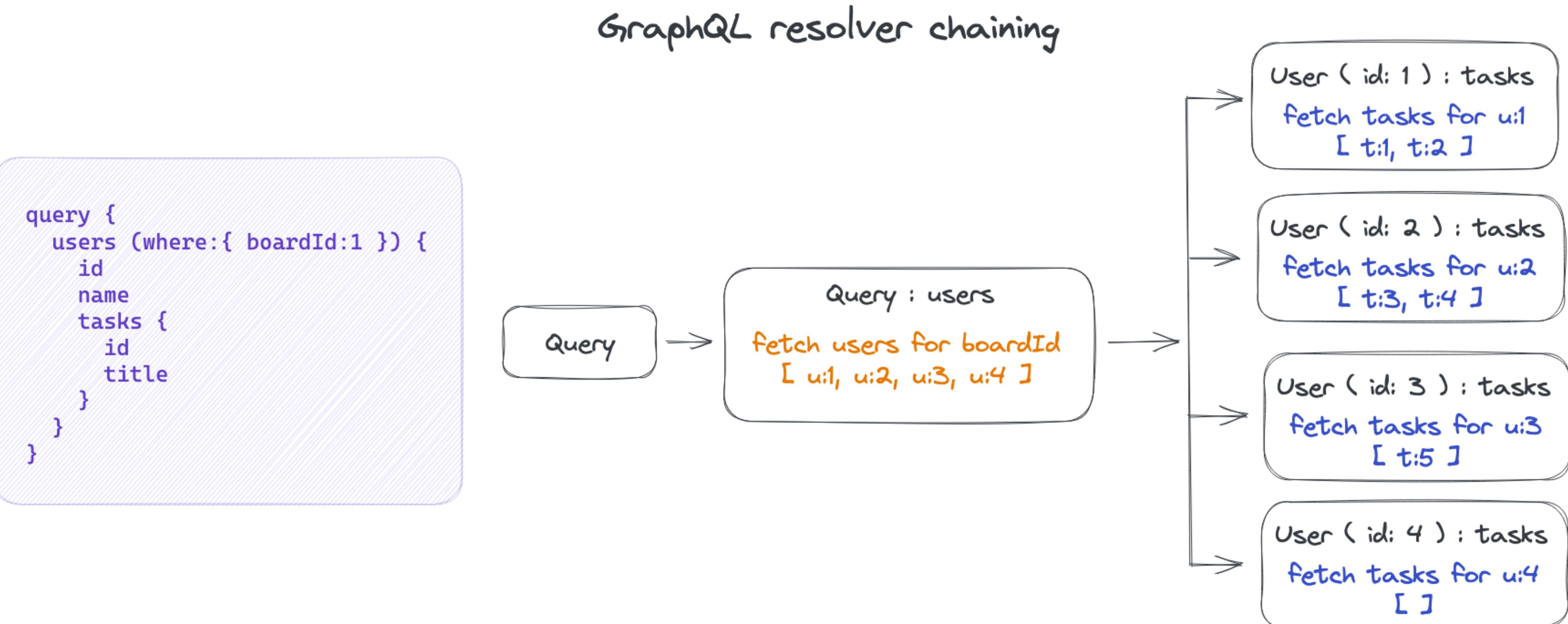
Operation type

Operation name

Variable definitions

```
query HeroNameAndFriends ($episode: Episode) {  
  hero(episode: $episode) {  
    name  
  }  
}
```

Проблема n+1 запросов



Проблема безопасности

```
query {  
    me {  
        friends {  
            friends {  
                friends {  
                    friends {  
                        friends {  
                            name  
                        }  
                    }  
                }  
            }  
        }  
    }  
}
```

Также, имеет смысл ограничивать сложность запроса.

Задание

Проект: <https://github.com/LucyRez/graphql-music>

1. Реализовать запрос для получения песни по id
2. Реализовать мутацию для добавления песни в плейлист
3. Реализовать мутацию для создания пачки (batch) песен