АНДРЕЙ ЧИЖ

GraphQL API: Patterns

Стандартный слайд

В промышленной разработке уже **8+ лет**

Основная и любимая технология: **Node.js**

Сейчас **Backend Engineer / Team Lead @ Wix**
(160+ млн пользователей, 200+ млн сайтов)

Занимаюсь техническим консалтингом стартапов

Разрабатываю **high-frequency trading (HFT) системы для фондовых и крипто бирж**

Важные моменты о GraphQL

Крайне типичная ситуация

GraphQL = Business Mindset

Отображение данных на клиентах

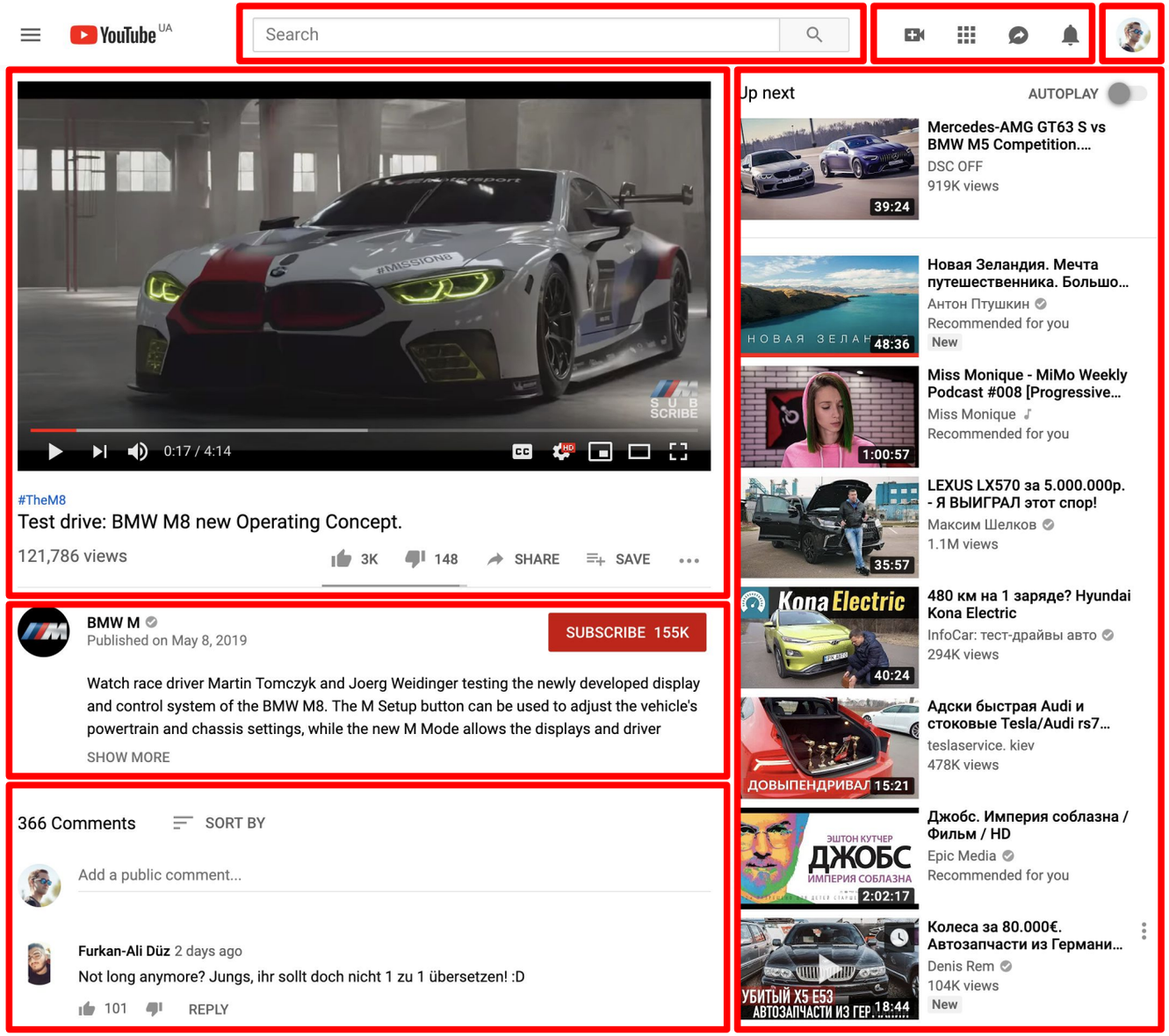
Хранение и получение данных

Доменная область бизнеса

Связи между бизнес доменами

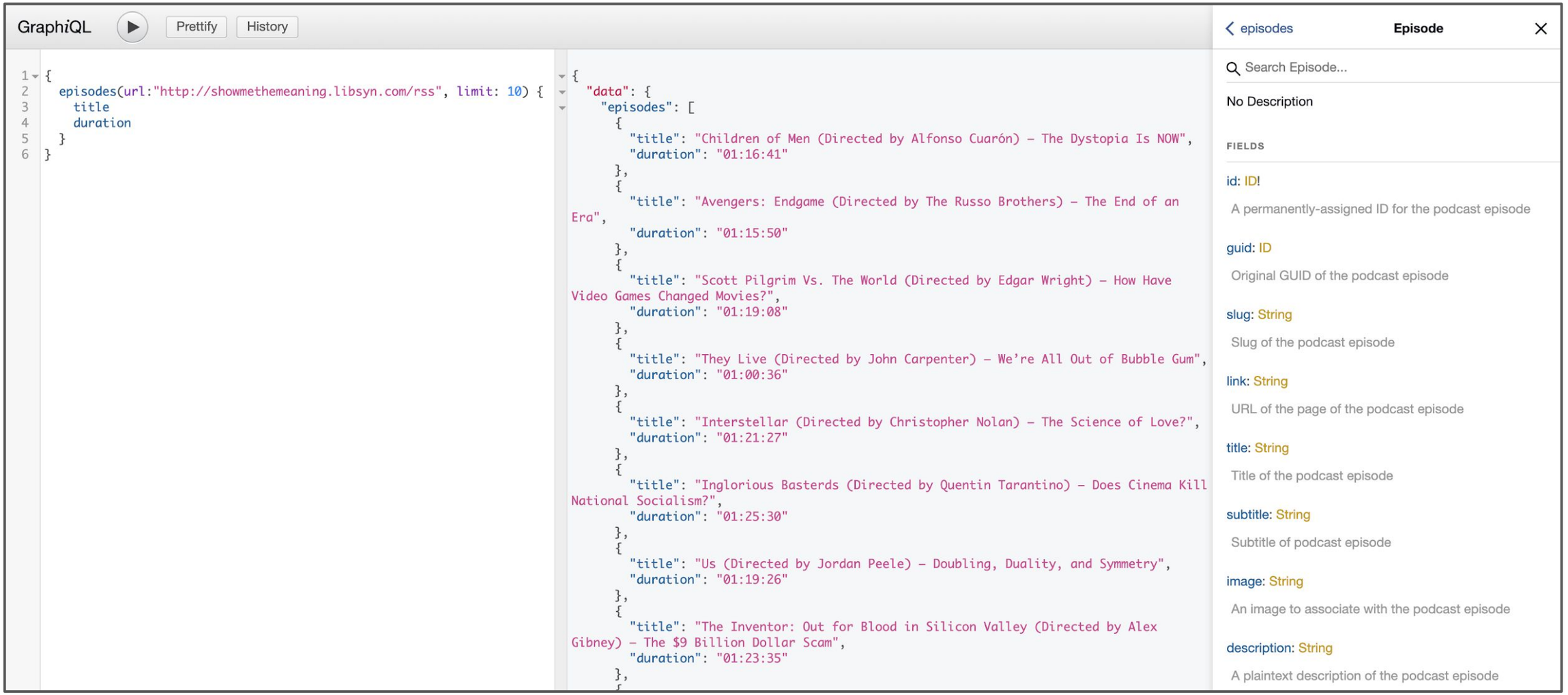
Удобный и понятный контракт внутри проекта

Все состоит из доменов



User	Setting
Channel	Playlist
Video	Media
Comment	Event

GraphiQL удобен и понятен



Нужен ли вам GraphQL?

Кто использует GraphQL?

Reference

Когда GraphQL это не лучший выбор?

☹ GraphQL: Кеширование

Представим, что нам нужно получить данные о исполнителе с ID = 123:

- **REST:** `GET /api/v1/artists/123`
- **GraphQL:** `GET /api/graphql?query={ artist(id:"123") { name, age }}`

А теперь давайте попробуем закешировать все в CDN:

- **REST: Все супер!**
- **GraphQL: есть нюанс - каждый запрос это новый ключ для CDN**
  - `GET /api/graphql?query={ artists(id:"123") { age, name }}`
  - `GET /api/graphql?query={ artists(id:"123") { age }}`
  - `GET /api/graphql?query={ artists(id:"123") { name }}`

Это решается с помощью крутых CDN (fastly) которые поддерживают суррогатные ключи и очистку кеша по АПИ

 GraphQL: Бизнес и деньги

- **Хотите переписать все на GraphQL когда текущая реализация АПИ (REST, Hypermedia API, OData, ORDS и тп) отлично быстро работает, есть документация, покрытие тестами и проект приносит деньги?**

GraphQL: Бизнес и люди

- Нет желания менять подходы в работе
- Нет желания изучать новые технологии / языки
- Нет критической массы для полноценного старта улучшения процессов / инструментов
- Вы завязаны на внешние команды / компании с посредственной экспертизой

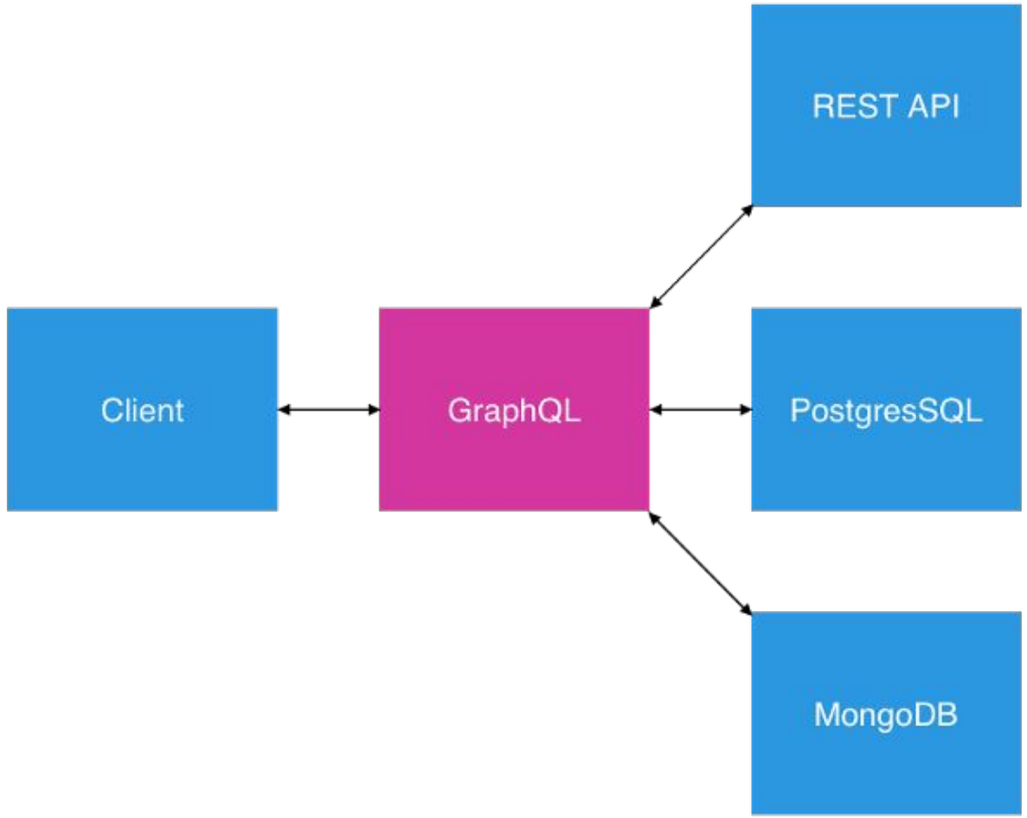
Когда GraphQL это отличный выбор?

GraphQL: разнообразие клиентов

- Web (включая различные шаблоны отображения)
- Мобильные приложения
- Смарт-часы
- Смарт-ТВ
- IoT (холодильники, пылесосы и тп)
- Публичный АПИ

GraphQL: Composite Pattern

Супер удобен, когда вы хотите объединить данные из нескольких мест в один удобный АПИ



GraphQL: Proxy Pattern

Удобен, когда вы хотите добавить новый функционал в существующий АПИ

GraphQL: Facade Pattern

Удобен, когда вы хотите упростить сложный ответ существующего АПИ

GraphQL: как шанс исправить косяки

- У вас фронтенд нормализует данные АПИ из snake_case в camelCase и наоборот?
- Ответы АПИ не поддаются описанию в документации?
- Бизнес готов выделить время на исправление ошибок прошлого?
- Можете сейчас сделать хорошо???

Шаблоны и рекомендации

Наш план

- Стили именования
- Описание типов
- Фильтрация
- Сортировка
- Пагинация
- Мутации
- Лимиты
- Примеры

Стили именования

Общие правила из спецификации

- Регулярка для имен: **/[_A-Za-z][_0-9A-Za-z]*/**
- Имена в GraphQL регистрозависимые. Это значит, что **name**, **Name** и **NAME** это абсолютно разные имена.
- Подчеркивание так же важны и это значит, что **other_name** и **othername** это два разных имени.
- Все имена должны иметь одинаковую грамматическую форму.

Reference

Стиль именования типов

```
type video {
   ...
}

type Video_Category {
   ...
}
```

```
type Video {
   ...
}

type VideoCategory {
   ...
}
```

Стиль именования interfaces / inputs / unions

```
interface entity {
   ...
}

input Stream_Input {
   ...
}

union itemType = A | B
```

```
interface Entity {
   ...
}

input StreamInput {
   ...
}

union ItemType = A | B
```

Стиль именования полей

```
type MediaFile {
  used_in: [String]
  created_at: Date
  updated_at: Date
}
```

```
type MediaFile {
  usedIn: [String]
  createdAt: Date
  updatedAt: Date
}
```

Стиль именования ENUM-типов

```
enum VideoSource {
  Netflix
  Youtube
  Vimeo
}

enum SortOrder {
  titleAsc
  titleDesc
}
```

```
enum VideoSource {
  NETFLIX
  YOUTUBE
  VIMEO
}

enum SortOrder {
  TITLE_ASC
  TITLE_DESC
}
```

Стиль именования мутаций / подписок

```
type Mutation {
  userCreate(...)
  putUser(...)
}



type Subscription {
  user(...)
}
```

```
type Mutation {
  createUser(...)
  updateUser(...)
  blockUser(...)
  unblockUser(...)
}

type Subscription {
  userUpdated(...)
}
```

Стили именования

Тип	Стиль	Примеры

Interface
Type
Input
Union
Enum
	`PascalCase or UpperCamelCase`	`CreateImageInput`
		`UserStatusPayload`
		`VideoCard`

Enum (values)	**`CAPITALIZED_WITH_UNDERSCORES`**	`IN_PROGRESS` `DETELED`

Fields
Variables
Queries
Mutations
Subscriptions
	`camelCase`	`userId`
		`createdAt`
		`createImage`
		`commentAdded`

Описание типов

Список ненулевых значений

```
type ItemOne {
  tags: [String]!
}
type ItemTwo {
  tags: [String!]
}
```

```
type ItemThree {
  tags: [String!]!
}
```

- [String]! : [] or [‘a’] or [‘a’, null, ‘b’]
- [String!] : null or [‘a’] or [‘a’, ‘b’]
- [String!]! : [] or [‘a’] or [‘a’, ‘b’] ← The best for FED

Reference

Будьте внимательны с типом Int

- Это signed 32-bit integer.
- If the integer internal value represents a value less than **-2^{31}** or greater than or equal to **2^{31}**, a field **error should be raised**.
- В числах: **-2,147,483,648 / 2,147,483,648**

Эти числа в реальной жизни:

- Время (в миллисекундах): **~24.86 дней**
- Объем данных (в байтах): **~2.15 ГБ**

Reference

Кастомные скалярные типы

```
type UserInfo {
  email: String
  metadata: String
  createdAt: String
}
```

```
type UserInfo {
  email: Email
  metadata: JSON
  createdAt: DateTime
}
```

- Email - GraphQL email with validation
- DateTime - GraphQL ISO-8061 Date
- JSON - GraphQL unstructured data

Reference

Фильтрация

Используйте аргумент filter

```
input ImagesFilterInput {
   userId: ID!
   tag: [String!]
   hash: [String!]
}

images(filter: ImagesFilterInput!): …

images(filter: { userId: “12345” }) { … }
```

Сортировка

Вариант 1: комбинация 2 ENUM-ов

```
enum SortField {
  NAME
  CREATED_AT
}

enum SortOrder {
  ASC
  DESC
}

input SortInput {
  field: SortField!
  order: SortOrder!
}
```

```
images(sort: [SortInput!]): ...

images(sort: [
  {
    field: NAME,
    order: DESC
  },
  {
    field: CREATED_AT,
    order: ASC
  }
]) { … }
```

Reference

Вариант 2: ENUM с сортировками

```
enum SortImage {
  NAME_ASC
  NAME_DESC
  CREATED_ASC
  CREATED_DESC
  POPULAR
}
```

```
images(sort: [SortImage!]): ...

images(sort: [
  NAME_DESC,
  CREATED_ASC
]) { … }
```

Reference

Пагинация

Вариант с limit-offset

```
type Query {
  feeds(filter: {...}, limit: Int! = 10, skip: Int): [Feed!]!
}

type Query {
  feeds(filter: {...}, limit: Int! = 10, offset: Int): [Feed!]!
}
```

Reference

Вариант с постраничным форматом

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

type FeedPagination {
  items: [Feed!]!
  pageInfo: PaginationInfo!
}
```

```
type Query {
  feeds(
    filter: {...},
    page: Int! = 1,
    perPage: Int! = 10
  ): FeedPagination!
}
```

Reference

Relay Cursor Connections Specification

```
{
  user {
    id
    name
    friends(first: 10, after: "next_page_cursor") {
      edges {
        cursor
        node {
          id
          name
        }
      }
      pageInfo {
        hasNextPage
      }
    }
  }
}
```

Reference

Мутации

Используйте переменную input в мутациях

```
type Video implements Node {
  id: ID!
  title: String!
  description: String
  createdAt: DateTime!
  updatedAt: DateTime
}

input CreateVideoInput {
  title: String!
  description: String!
}
```

```
type Mutation {
  createVideo(input: CreateVideoInput!): ...
}

type Mutation {
  createVideo(input: { title: “Test” }): ...
}
```

Reference

Лимиты

Ограничение количества данных

```
query allPhotos {
  photos(limit: 99999) {
    title
    url
    postedBy {
      name
      avatar
    }
  }
}
```

Ограничение глубины запроса

```
query getPhoto($id: ID!) {
  photo(id: $id) {
    name
    url
    postedBy {
      name
      avatar
      postedPhotos {
        name
        url
        taggedUsers {
          name
          avatar
          postedPhotos {
            name
            url
          }
        }
      }
    }
  }
}
```

Ограничение сложности запроса

```
query everything($id: ID!) {
  photo(id: $id) {
    name                  # Complexity 5
    url                   # Complexity 5
  }
  users {
    id                    # Complexity 10
    name                  # Complexity 10
    avatar                # Complexity 10
    postedPhotos {
      name                # Complexity 100
      url                 # Complexity 100
    }
    inPhotos {
      name                # Complexity 200
      url                 # Complexity 200
      taggedUsers {
        id                # Complexity 500
      }
    }
  }
}
```

Что дальше?

Крутые ссылки

- GraphQL Spec
- Awesome GraphQL
- Shopify Designing a GraphQL API
- Lessons From 4 Years of GraphQL
- Designing GraphQL Mutations
- Дизайн и паттерны проектирования GraphQL-схем

Больше практики

- Берите проекты, разбирайте их на домены
- Пробуйте описывать типы этих доменов
- Пробуйте строить связи доменов между собой
- Ищите свои шаблоны в построении АПИ
- Делайте командное ревью нового АПИ
- Запускайте POC начиная с первых итераций ревью

Ссылка на презентацию

Спасибо!