DEVIEW 2020

GraphQL API 까짓거 운영해보지 뭐

박성현 NAVER Glace / CONOMI 개발

CONTENTS

1. GraphQL 이해하기
2. GraphQL API 운영하기
3. 겪어보니 알게된 중요한 점들

1. GraphQL 이해하기

1.1 GraphQL 이란?

DEVIEW 2020

GraphQL

- 서비스 도메인을 그래프 형태로 모델링
- API = 도메인 데이터 그래프
- 그래프 기반으로 쿼리를 요청

```
# 데이터 그래프 제공
type User {
  name: String
  age: Int
  posts: [Post]
}

type Post {
  content: String
}

type Query {
  user: User
}
```

```
# 쿼리 요청
query {
  user {
    name
    posts {
      content
    }
  }
}
```

1.2 주요 특징

1. API 엔드포인트가 하나다
2. 원하는 데이터만 요청할 수 있다
3. 스키마를 자유롭게 확장할 수 있다

2. GraphQL API 운영하기

GraphQL API 운영하기

DEVIEW 2020

네가지 공유

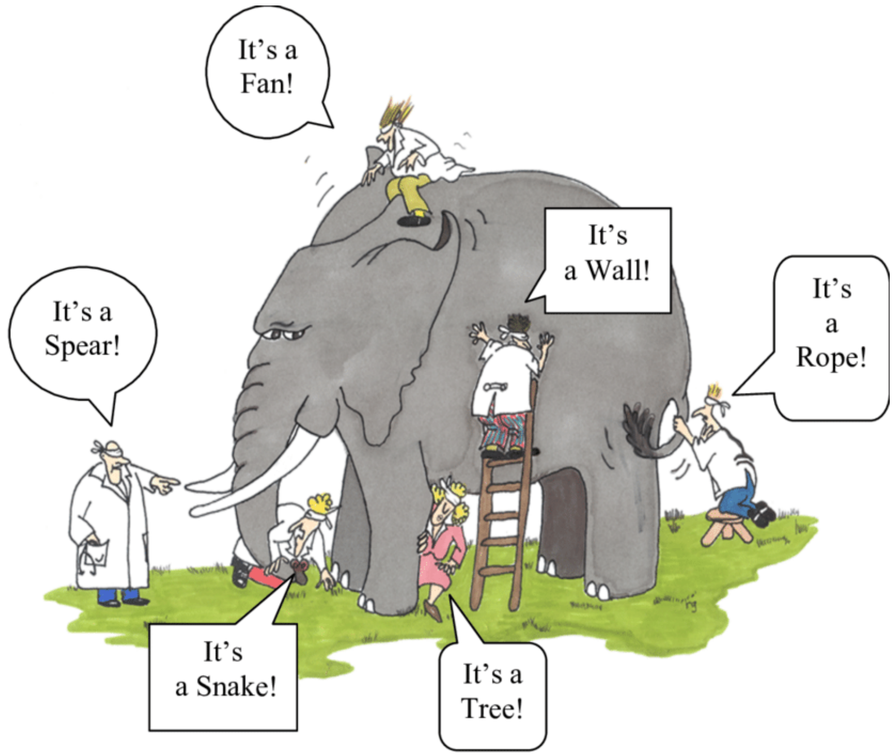
1. API 모니터링 환경 갖추기
2. Versioning 없는 API
3. API Gateway
4. API 문서 제공하기

2.1 API 모니터링 환경 갖추기

DEVIEW 2020

2.1.1 모니터링 도구

API 를 성공적으로 운영하려면 APM 이 필수



https://www.researchgate.net/figure/The-Blind-Men-and-the-Elephant_fig1_290127733

데이터 기반 의사결정

2.1 API 모니터링 환경 갖추기

DEVIEW 2020

2.1.2 API 관리를 위한 최소한의 정보

- 쿼리를 호출한 클라이언트가 누구인가? (OS, 앱 버전 등)
- 쿼리는 어떻게 구성이 되어 있나? (사용자 데이터 활용 / 레거시 제거 활용)
- 에러는 어디서 발생했나? (에러가 발생한 필드 / 파라미터 등)
- 쿼리 성능이 어떻게 되나? (API 병목 파악 / 원인 추적 / 개선 우선순위 결정 등)

2.1 API 모니터링 환경 갖추기

2.1.3 모니터링 요구사항 PoC

GraphQL 특징

1. API 엔드포인트가 하나다 → 새로운 APM 솔루션
2. FE에서 원하는 데이터만 요청한다 → 필드 단위 성능 / 에러 측정
3. 스키마를 자유롭게 수정할 수 있다 → API 서버 + 모니터링 서버 스키마 동기화 필요

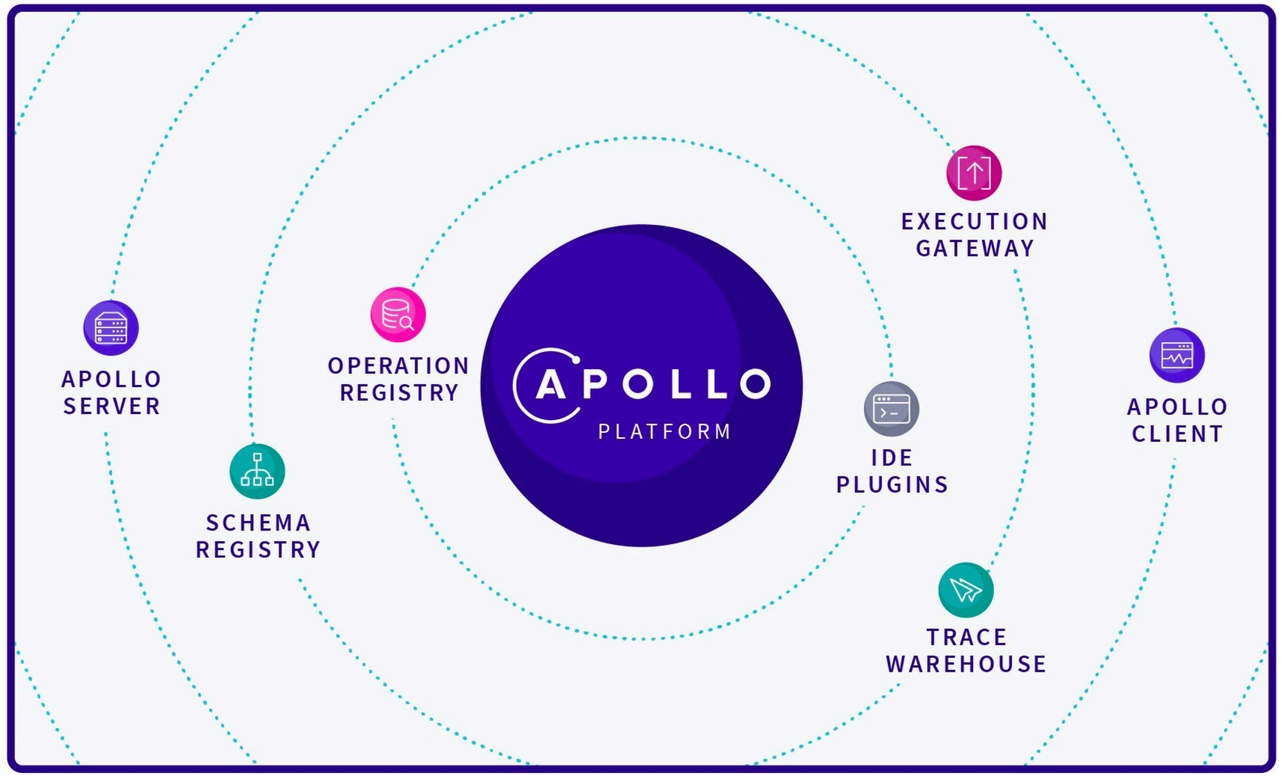
"쉽지 않아..."

Why GraphQL Performance Monitoring is Hard

2.1 API 모니터링 환경 갖추기

DEVIEW 2020

2.1.4 모니터링 도구 적용



Introducing the Apollo GraphQL Platform

Thank you "Apollo Studio" 👏

2.1 API 모니터링 환경 갖추기

DEVIEW 2020

2.1.5 Apollo Studio 설정

스키마 업로드

```
npx apollo service:push --variant $variant --endpoint schema.graphql
```

스키마 운영 (variant)

- develop: 개발 API 서버 모니터링
- production: 프로덕션 API 서버 모니터링

Apollo Studio 설정

```
// server (deprecated from 2.18)
new ApolloServer({
  engine: {
    apiKey: APOLLO_STUDIO_KEY,
    schemaTag: $variant,
  },
})

// client (@3)
new ApolloClient({
  name: clientName,
  version: clientVersion,
})
```

2.1 API 모니터링 환경 갖추기

2.1.5 Apollo Studio 설정

클라이언트 기준 정리

Source	Client Name	Client Version
React Native	iOS / Android	build.v + cp.v + build.n*
Native	Native iOS / Android	build.v + build.n
Web	conomi-web	-
Admin	admin	-
[ad-hoc] playground / cli / etc**	unknown client	unknown version

*cp: Code Push
**fake client: 호환성 관리 대상 제외

2.1 API 모니터링 환경 갖추기

DEVIEW 2020

2.1.5 Apollo Studio 설정

Operations Performance Errors Traces Operation — Last day · All clients (6) · All operation

Last day overview

Request Rate
581.6 rpm
24 hour median
Range: 0.009rpm - 0.685rpm
+8.91 % since yesterday

p95 Service Time
364.1 ms
24 hour median
Range: 199.1ms - 610ms
+4.46 % since yesterday

Error Percentage
0.47 %
24 hour median
Range: 0.03% - 1.85%
+62.48 % since yesterday

Highest Request Rate		Slowest p95 Service Time		Highest Error Percentage	
dc7d isRestaurantBookmar…	132.2 rpm	6e71 SearchHome	2.1 s	7211	100 %
04b8 userReceipts	51.6 rpm	d129 Businesses	1.9 s	a124	100 %
fb94 fetchMe	49.1 rpm	a04a getBusinesses	1.6 s	c936	100 %
6099 UserStatistics	20 rpm	3d76 SearchHome	1.5 s	8537	100 %
8b4f fetchUserStatistics	18.2 rpm	3846 FetchFollowGridFeed	1.4 s	6970	100 %

Request rate over time (RPM)

Requests

Oct 22nd 3:00am 6:00am 9:00am 12:00pm 3:00pm 6:00pm 9:00pm

Request latency over time

Requests

5.9s max

Activity in the last day

iOS

All versions < 0.1% errors
382 operations
53% requests

Versions	Requests	% Errors
1.3.1.v26 (2)		
1.3.2 (1)		
1.4.0.v28 (1)		
1.4.5 (1)		

Operation signature

```
query fetchViewerIsFollowingUser($id: ID!) {
  user(id: $id) {
    __typename
    id
    isViewer
    viewerIsFollowing
  }
}
```

107 operations failed outside the GraphQL context

Variable "$query" of required type "String!" was not provided.

10 instances of this error

Time	Operation
2020-10-10 7:27 am +09:00	e2a6 MenuFilter
2020-10-10 7:46 am +09:00	e2a6 MenuFilter
2020-10-10 9:23 am +09:00	e2a6 MenuFilter
2020-10-10 9:23 am +09:00	e2a6 MenuFilter
2020-10-10 9:56 am +09:00	e2a6 MenuFilter
2020-10-10 10:44 am +09:00	e2a6 MenuFilter
2020-10-10 10:45 am +09:00	e2a6 MenuFilter
2020-10-10 10:45 am +09:00	e2a6 MenuFilter
2020-10-10 10:46 am +09:00	e2a6 MenuFilter
2020-10-10 11:06 am +09:00	e2a6 MenuFilter

Execution

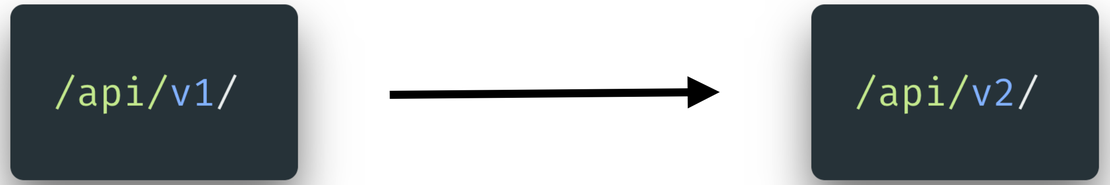
Resolvers	Timing
fetchViewerIsFollowingUser	11.0ms
service:tapas	
user: User	2.34ms
id: ID!	< 1ms
isViewer: Boolean!	< 1ms
viewerIsFollowing: Boolean!	< 1ms

waiting on server · server responding · child node

2.2 Versioning 없는 API

DEVIEW 2020

2.2.1 API versioning



- 새로운 기능 제공
- 리소스 재설계

→ 하나의 API로는 모든 요구사항을 만족할 수 없다

2.2 Versioning 없는 API

DEVIEW 2020

2.2.2 GraphQL 에서 해결한 방법

스키마는 실제 요구 사항을 기반으로 점진적으로 구축되어야하며
시간이 지남에 따라 원활하게 발전해야합니다

by Principled GraphQL

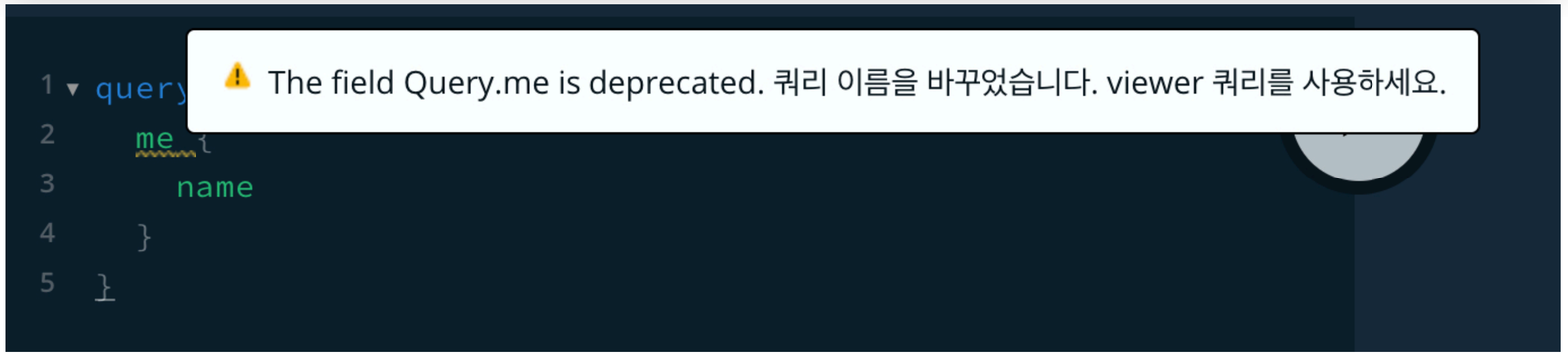
2.2 Versioning 없는 API

DEVIEW 2020

2.2.3 점진적인 스키마 발전의 필요 조건

1. FE 에서는 레거시 쿼리를 파악할 수 있어야 한다.
2. BE 에서는 레거시 쿼리를 안정적으로 제거할 수 있어야 한다.

```
type Query {
  me: User @deprecated(reason: "쿼리 이름을 바꾸었습니다. viewer 쿼리를 사용하세요.")
}
```



[GraphQL Spec] deprecated directive

2.2 Versioning 없는 API

DEVIEW 2020

2.2.4 안정적인 / 그렇지 않은 쿼리 제거

[PR] 코드 상에 있는 `hello` 쿼리를 제거했습니다
더이상 사용하지 않는 것 같아요

A1. "아 그런가요? 네 알겠습니다! 믿고 가겠습니다!"

A2. "해당 쿼리는 최근 한달 동안 RN iOS 1.5.0 버전에서 10번 정도 요청한 기록이 있네요
우리는 5번 이상 호출 기록이 있으면 `Breaking Change` 로 판단합니다
`@deprecated` 정도만 표시하고 상황을 지켜볼까요?"

→ 수집된 데이터 + 프로젝트 기준 = 데이터 기반의 신뢰할 수 있는 의사결정

2.2 Versioning 없는 API

DEVIEW 2020

2.2.5 Schema Validation

스키마를 안정적으로 변경할 수 있는가

1. 현재 API 가 서비스 하는 Live 스키마와의 변경점 파악
2. 변경된 내용과 수집한 데이터를 비교해 호환성이 깨지는지 파악

2.2 Versioning 없는 API

DEVIEW 2020

2.2.5 Schema Validation

[with Apollo Studio Schema Checks]

1. 스튜디오 내에 서비스 Live 스키마 관리 (Apollo Registry)

```
npx apollo service:push --variant $(API 환경) --endpoint $(API 스키마)
```

2. `apollo` cli 를 통해 Breaking Change 파악 (PASS | FAIL)

```
npx apollo service:check --variant $(API 환경) --endpoint $(API 스키마)
```

2.2 Versioning 없는 API

DEVIEW 2020

2.2.5 Schema Validation

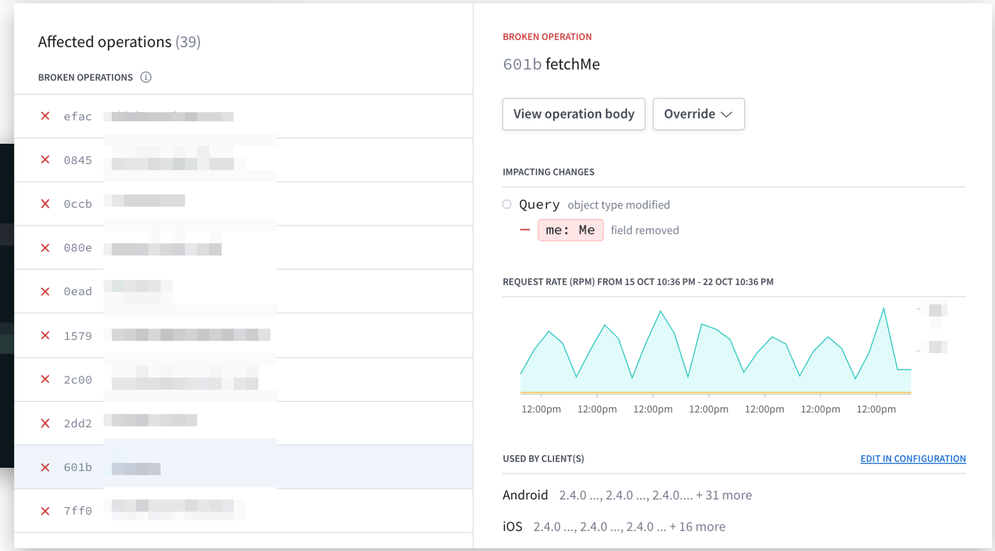
Validation PASS

```
yarn schema:production:validation
yarn run v1.22.4
$ npx apollo service:check --variant gateway-production --endpoint
 ›   Warning: apollo update available from 2.30.2 to 2.31.0.
  ✔ Loading Apollo Project
  ✔ Validated schema against metrics from variant gateway-production on graph
  ✔ Compared 0 schema changes against 0 operations
  ✔ Found 0 breaking changes and 0 compatible changes

No changes present between schemas
✨  Done in 2.95s.
```

Validation FAIL

```
yarn schema:development:validation
yarn run v1.22.4
$ npx apollo service:check --variant gateway-develop --endpoint
 ›   Warning: apollo update available from 2.30.2 to 2.31.0.
  ✔ Loading Apollo Project
  ✔ Validated schema against metrics from variant gateway-develop on graph
  ✔ Compared 21 schema changes against 226 operations over the last 7 days
  ✖ Found 19 breaking changes and 2 compatible changes
    → breaking changes found
```



2.2 Versioning 없는 API

DEVIEW 2020

2.2.6 Schema Validation Strategy

데이터 수집 단위: variant

- 수집한 데이터 = 실제 유저 사용 형태
- 개발에서 수집한 데이터는 실제 사용자를 대변할 수 없다

~~수집된 데이터 + 프로젝트 기준 = 데이터 기반의 신뢰할 수 있는 의사결정~~

Q. 항상 프로덕션 스키마를 기준으로 validation 해야 하나요?

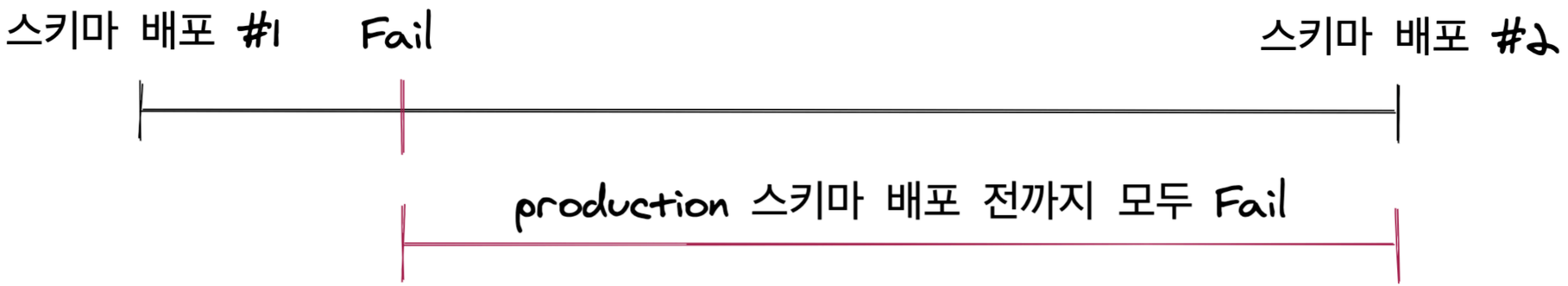
2.2 Versioning 없는 API

DEVIEW 2020

2.2.6 Schema Validation Strategy

실패 후유증

- production 스키마 배포에 따라 validation fail 의 영향력이 너무 크다!



- 이미 협의가 완료된 변경인 경우 PASS 면 좋겠어요 😞
- 제가 변경한 내용이 아닌 경우 PASS 면 좋겠어요 😕

2.2 Versioning 없는 API

DEVIEW 2020

2.2.6 Schema Validation Strategy

스키마 운영 기준 변경

- develop
  - git develop 브랜치와 개발 API 서버의 스키마 동기화
  - 스키마 변경 합의는 PR 내에서 마무리
  - 합의가 완료되어 PR 이 머지되면, develop variant 의 스키마를 업데이트
- production
  - 프로덕션 API 배포 시점에 스키마 배포

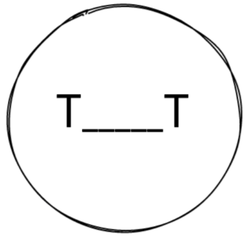
2.2 Versioning 없는 API

DEVIEW 2020

2.2.6 Schema Validation CI / CD

실패를 빠르게 알게 하자 → 수정 비용 감소

"PR 올릴 때 schema validation 확인 해주시고,
머지할 때는 베이스 브랜치 따라 Apollo Registry 에 스키마 푸시해주세요~ 😃"



개발자: 배울게 너무 많아요!

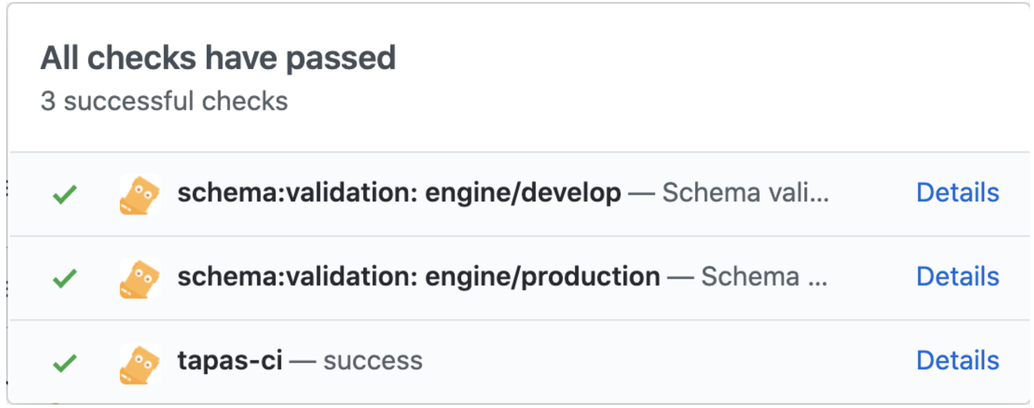
2.2 Versioning 없는 API

DEVIEW 2020

2.2.6 Schema Validation CI / CD

Schema Validation CI / CD 기준 (by Github Pull Request builder)

1. PR 생성 (CI 수행)
   - 모든 PR 에 대해 개발 스키마 검증
   - `master`, `hotfix/*`, `release/*` PR 은 추가로 프로덕션 스키마 검증
   - (Git Flow 브랜치 전략)
2. PR 머지 (CD 수행)
   - 개발 스키마에 변경된 내용 반영
   - (프로덕션은 PR 머지에 따른 CD 없이 API 배포와 연동)



2.2 Versioning 없는 API

DEVIEW 2020

2.2.7 Versioning 없는 API

이제 우리는

- Graph 스키마를 자유롭게 변경할 수 있다
- 스키마 변경의 영향 범위를 명확히 알 수 있다

→ Versioning 없이 API 를 안정적으로 운영할 수 있게 되었다 🎉

2.3 API Gateway

2.3.1 BE 서비스의 증가

검색 GraphQL API 를 다른 팀에서 개발

Q. 클라이언트에서 GraphQL API 2개를 호출해야 하나요?

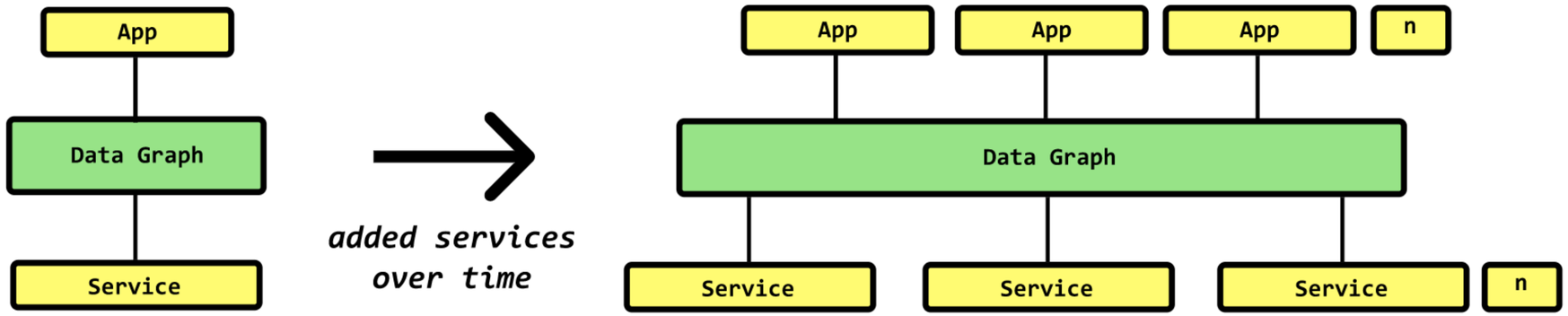
A. 회사에는 각 팀에서 만든 여러 그래프 대신
하나의 통합 그래프가 있어야 합니다

2.3 API Gateway

DEVIEW 2020

2.3.2 Apollo Federation

여러개의 BE 서비스를 하나의 API Gateway 를 통해 One Data Graph 제공



GraphQL Architectural Advantages

2.3 API Gateway

DEVIEW 2020

2.3.3 개발 프로세스에 녹여내기

이슈 #1 - 스키마 병합

- 두 팀의 스키마에 동일한 듯 다른 스키마가 존재
- ex) 위경도 입력을 받는 input 의 nullability 가 다름

```
# Team A Schema
input Point {
  latitude: Float
  longitude: Float
}
```

```
# Team B Schema
input Point {
  latitude: Float!
  longitude: Float!
}
```

→ Breaking Change 가 없는 방향으로 병합 (Nullable -> Not Null)

2.3 API Gateway

DEVIEW 2020

2.3.3 개발 프로세스에 녹여내기

이슈 #2 - 헤더 전달

- Gateway 가 직접 Apollo Studio 로 데이터 전송
- Gateway → 각 서비스 호출은 E2E

```
// Service Header
const headers = {
  'user-agent': 'node-fetch/1.0',
  //  ... ?!
}
```

- 각 서비스에서 요구하는 헤더 전달

```
const headers = {
  ...pick(ctx.req.headers, [
    AwarenessKey.ClientName, AwarenessKey.ClientVersion, 'authorization', 'geolocation', 'batch'
  ]),
  'X-Forwarded-For': ctx.ip,
}
```

2.3 API Gateway

DEVIEW 2020

2.3.3 개발 프로세스에 녹여내기

이슈 #3 - 스키마 설계

- 여러 팀이 하나의 스키마를 만들기 때문에, 스키마에도 통일성이 있어야 한다
  - 쿼리 / 타입 / 필드 이름, 페이지네이션 형태 등
- 현재는 코드 리뷰로 가이드 진행

우리는 모두 나무야~ 🤔

Next

- GraphQL 스키마 가이드 제공
- 스키마 설계 + 스키마 컨벤션 + 스키마 모니터링 가이드
  1. GraphQL at Enterprise Scale
  2. Scaling GraphQL at PayPal

2.4. API 문서 제공하기

DEVIEW 2020

2.4.1 API 문서가 필요한가요?

Q. GraphQL 은 API 문서 없어도 되는거 아닌가요?

A. 그런줄 알았는데요...

2.4. API 문서 제공하기

DEVIEW 2020

2.4.2 API 문서가 필요해요

서비스에서 제공하는 두가지 도구

- Playground: 쿼리 및 실제 응답 데이터 검증 (API 기능 테스트)
- GraphQL voyager: 그래프 시각화 (도메인 관계 파악)

사용자 요구사항 (다른 팀, 우리 팀, 그리고 미래의 나)

- Playground, voyager 는 depth 가 깊어서 도메인 개념 파악하는데 쉽지 않아요
- 실제 데이터를 볼 수 있는 예제 쿼리가 있으면 좋겠어요

→ 도메인 개념 파악을 위한 정적 문서 추가

2.4. API 문서 제공하기

DEVIEW 2020

2.4.2 API 문서가 필요해요

도메인 개념 파악을 위한 정적 문서 추가

- 별도 API 문서를 제공하는 서비스들도 많다
- Github, Shopify, Yelp, GitLab, etc (https://github.com/APIs-guru/graphql-apis)

개발자 요구사항 (다른 팀, 우리 팀, 그리고 미래의 나)

- Wiki 에 직접 작성은 부담입니당 (관리포인트 증가 / 동기화 이슈)
- 자동으로 생성되었으면 좋겠어요

2.4. API 문서 제공하기

DEVIEW 2020

2.4.4 Single Source of Truth

GraphQL Schema = Single Source Of Truth

👇

GraphQL Schema + 요구사항 = 😆😆😆😆

2.4. API 문서 제공하기

DEVIEW 2020

2.4.5 문서 자동 생성

1. 스키마에서 바로 단일 페이지 생성 (graphdoc)
2. 스키마 + gatsby 로 컨텐츠와 렌더링 분리 (like WordPress GraphQL)

OBJECT

Issue

An Issue is a place to discuss ideas, enhancements, tasks, and bugs for a project.

GraphQL Schema definition

```
type Issue implements Node, Comment, Issueish, Reactable, RepositoryNode, Timeline, UniformResourceLocatable {

    # A list of Users assigned to the Issue or Pull Request.
    #
    # Arguments
    # first: Returns the first _n_ elements from the list.
    # after: Returns the elements in the list that come after the
    # specified global ID.
    # last: Returns the last _n_ elements from the list.
    # before: Returns the elements in the list that come before the
    # specified global ID.
    assignees(first: Int, after: String, last: Int, before: String): UserConnection!

    # The author of the issue or pull request.
    author: Author

    # Identifies the body of the issue.
    body: String!
```

The GraphQL Query Language

Features of the GraphQL Query Language

On this page, we'll explore various various features of the GraphQL Query Language

This page is setup as a progression, showing basic usage of GraphQL queries, to more advanced queries and features of the GraphQL query language

Hello World

This is essentially the "hello world" of GraphQL Queries. Here, we are just asking the site for it's Title and URL. Basic settings almost any application built on top of WordPress would need.

```
{
  generalSettings {
    title
    url
  }
}
```

2.4. API 문서 제공하기

DEVIEW 2020

2.4.5 문서 자동 생성

	장점	단점
graphdoc	명령어 하나면 완성	오픈소스 유지보수 안됨 (2017~) 낮은 확장성 사용자 요구사항 만족 못함
gatsby	자유로운 템플릿 활성화된 개발 생태계 높은 확장성 (예시 쿼리, 별도 문서 등)	직접 개발

2.4. API 문서 제공하기

DEVIEW 2020

2.4.5 문서 자동 생성

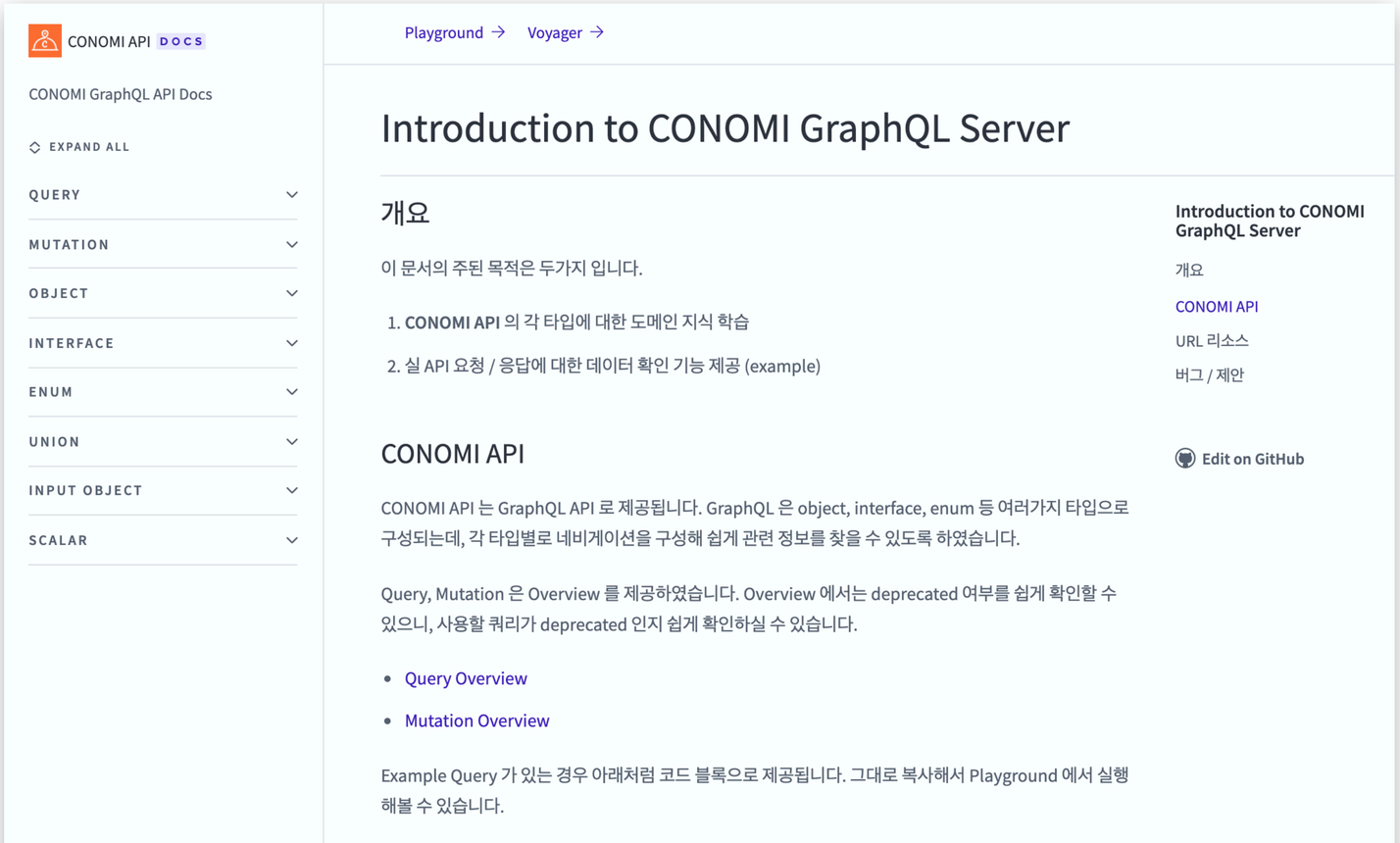
지금 좀 고생하지 뭐!

2.4. API 문서 제공하기

DEVIEW 2020

2.4.5 문서 자동 생성

- 컨텐츠 생성: graphql-schema-to-markdown
- 문서 생성: gatsby-theme-apollo-docs



2.4. API 문서 제공하기

DEVIEW 2020

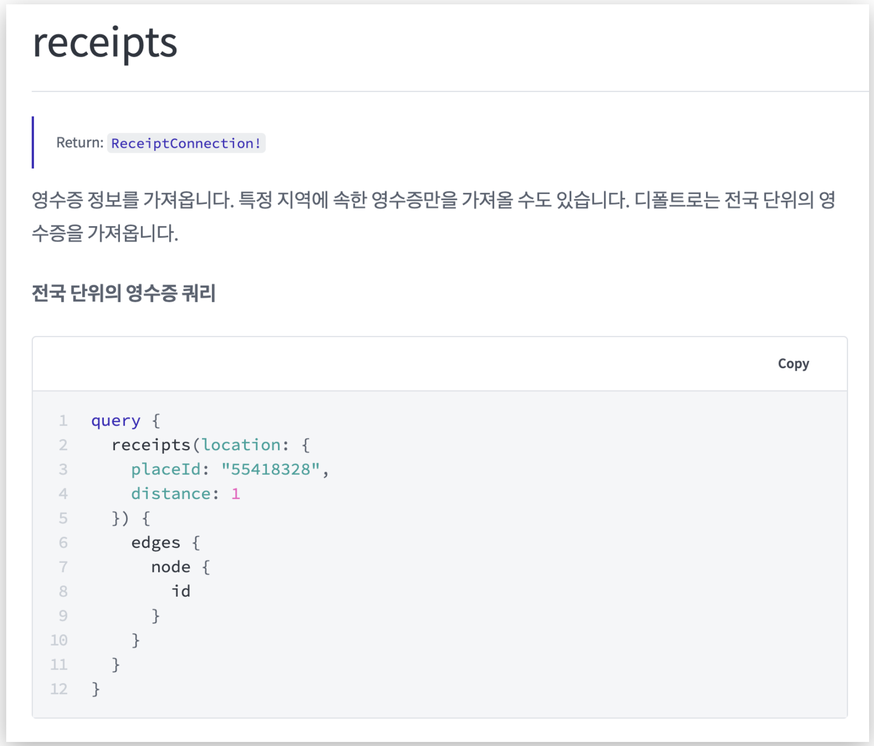
2.4.5 문서 자동 생성

개발자: 예제 쿼리 제공

```
type Query {
  """
  영수증 정보를 가져옵니다.
  특정 지역에 속한 영수증만을 가져올 수도 있습니다.
  디폴트로는 전국 단위의 영수증을 가져옵니다.

  @example 전국 단위의 영수증 쿼리
  query {
    receipts(location: {
      placeId: "55418328",
      distance: 1
    }) {
      edges {
        node {
          id
        }
      }
    }
  }
  """
  receipts( ... ): ReceiptConnection!
}
```

소비자: 도메인을 더 직관적으로 이해



→ 사용자 / 개발자 요구사항을 모두 만족했습니다~ 🎉

GraphQL API 까짓거 운영해보지 뭐

현실

1. 아직은 Apollo Studio 만한 모니터링 도구가 없다 (돈 나가요)
2. One Graph 를 위해 많은 노력이 요구된다 (Data Graph Maintainer)
3. 자료가 많이 없다 (해외도요)

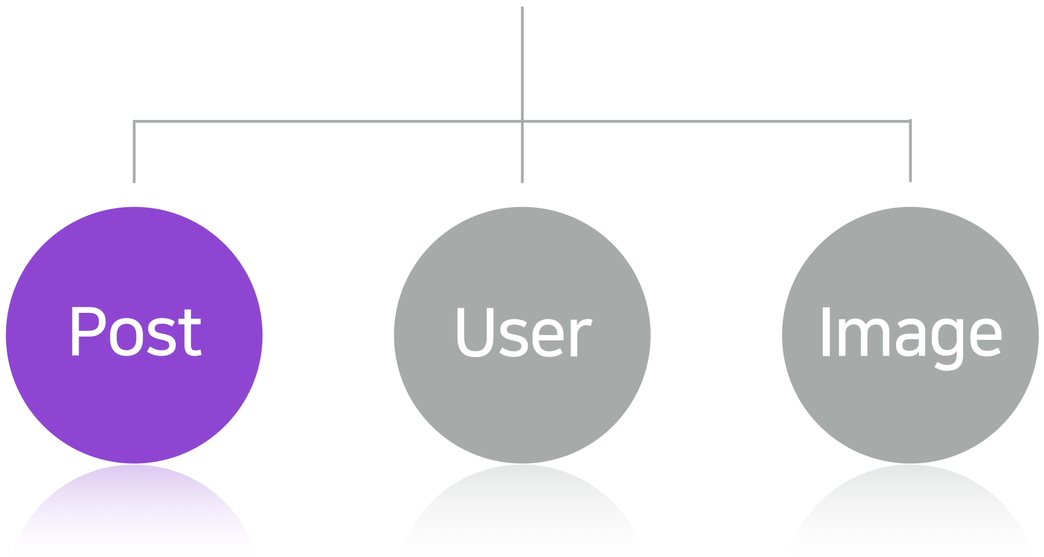
3. 겪어보니 알게된 중요한 점들

3.1 schema, schema, schema

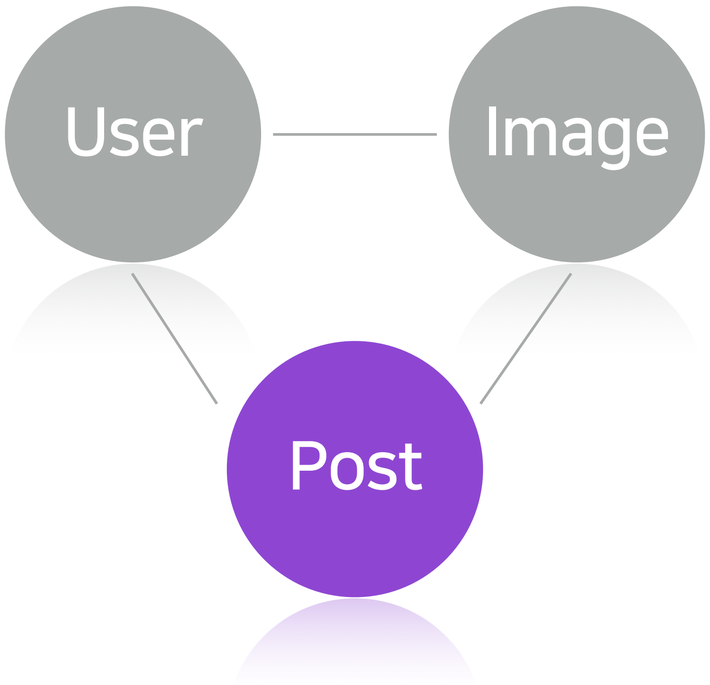
DEVIEW 2020

3.1.1 Graph 처럼 생각하기

REST (리소스 중심)



GraphQL (관계 중심)



3.1 schema, schema, schema

DEVIEW 2020

3.1.1 Graph 처럼 생각하기

스키마 핵심 특징

1. 비즈니스 도메인을 그래프로 설계
2. 구현과 분리된 비즈니스 도메인 설계
3. 타입 기반 커뮤니케이션 (Single Source of Truth)

패러다임 시프트 (Paradigm Shift)

[참고 #1], [참고 #2]

3.2 "원하는 정보만 알게 한다는 것"의 의미

DEVIEW 2020

3.2.1 Data Fatigue

2019년 FE 프레임워크를 배우는 기분 | Deview 2019 (NAVER 박재성님)

3.2 "원하는 정보만 알게 한다는 것"의 의미

DEVIEW 2020

3.2.1 Data Fatigue

2020년 API 응답을 분석하는 기분은,

피로감의 심화 그리고

"주는대로 받기"

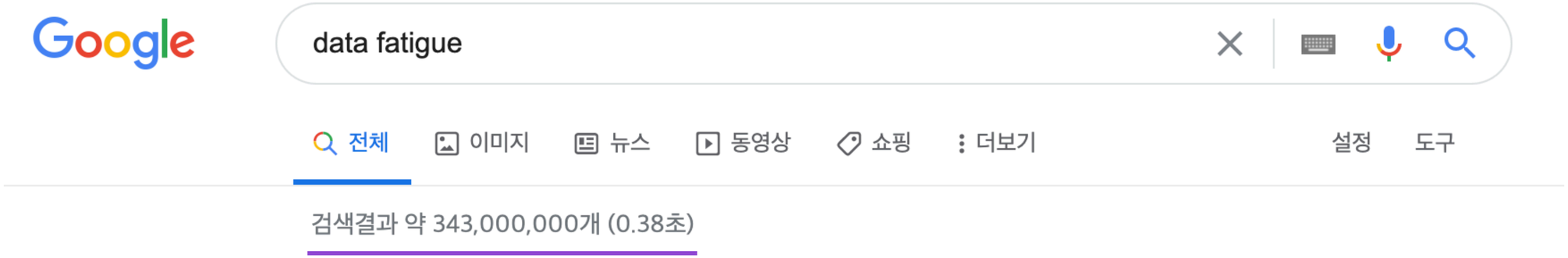
전략이 최고의 전략이 되어버림

다 어디서는 쓰고 있겠지...? 🤔

3.2 "원하는 정보만 알게 한다는 것"의 의미

DEVIEW 2020

3.2.1 Data Fatigue



넘쳐나는 데이터, 한정된 나의 에너지 🤣

3.2 "원하는 정보만 알게 한다는 것"의 의미

DEVIEW 2020

3.2.2 Data Fatigue가 없다면

[FE] 우리가 원하는 정보만 알게 된다면 말이지...

- 데이터 피로감 적어!
- 사용할 데이터를 더 명확히 드러낼 수 있어!
- 팀 모두가 시간이 갈수록 컨텍스트를 더 잘 이해할 수 있어!

```
query isViewerBookmarked($id: ID!) {
  restaurant(id: $id) {
    id
    viewerHasBookmarked
  }
}
```

3.2 "원하는 정보만 알게 한다는 것"의 의미

DEVIEW 2020

3.2.3 선택과 집중

무엇을 안 할지 정하는 것이

무엇을 할지 정하는 것 만큼 중요하다

Steve Jobs Insane Productivity Secrets

3.3 함께 하기

DEVIEW 2020

3.3.1 Graph Schema

활용 사례를 지원하도록 API를 설계했는가? (Red Hat)

1. API 가 클라이언트 요구사항을 충분히 지원하는가?
   - 클라이언트가 요구사항 만족 여부 판별 가능
   - 구현 중심의 API 응답 설계는 구현하다보면 달라진다
2. 스키마는 데이터 요구사항을 그대로 표현
   - 요구사항 만족 여부를 스키마로 확인 가능

→ 클라이언트가 요구사항 만족 여부를 스키마로 판별 가능

3.3 함께 하기

DEVIEW 2020

3.3.2 함께 하는 도메인 중심 설계

[BE] "사진있는 리뷰만 보기 버튼 노출" 조건을 아래 필드로 확인할 수 있습니다

```
# 구현 중심
query Restaurant($id: ID!) {
  restaurant(id: $id) {
    # 사진 리뷰 여부 판별
    hasPhotoReview
  }
}
```

3.3 함께 하기

DEVIEW 2020

3.3.2 함께 하는 도메인 중심 설계

[FE] `hasPhotoReview` 는 Photo 에 강하게 결합이 되어 있는 것 같은데요.
확장 가능하게 데이터를 제공하면 어떨까요?

```
# 데이터(도메인) 중심
query Restaurant($id: ID!) {
  restaurant(id: $id) {
    # 방법 1
    reviews(filterBy: { photo: EXIST })
    # 방법 2
    hasReview(type: PHOTO)
  }
}
```

3.3 함께 하기

3.3.2 함께 하는 도메인 중심 설계

DEVIEW 2020

1. 도메인을 더 잘 이해하기
2. 구현 중심이 아닌 데이터 중심으로 생각하기

FE 🤝 BE, 함께 해요 👋

Thank You

Q & A