DEVIEW 2021

Domain Graph Service를 활용한 광고 서비스의 GraphQL API 구현 사례

이광운
쿠팡

CONTENTS

1. 광고 서비스

1.1 광고 유래

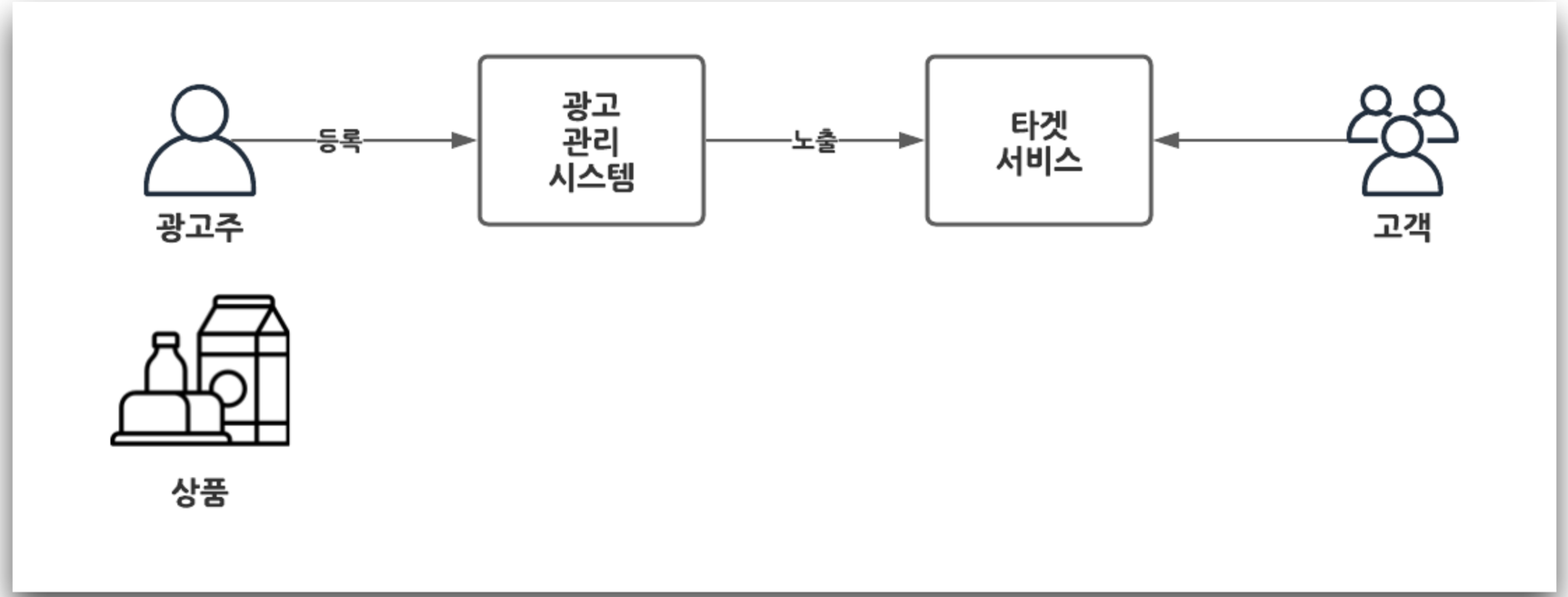
- 한성순보 회사설에 최초 <광고>라는 용어 등장
- 최초 근대 광고: 한성주보, 세창양행의 광고
- 최초 광고주: 세창양행

1886 - 독립신문 1904 - 대한매일신보 1908 - 소년 1914 - 담배광고

출처: kobaco 광고박물관

1.2 광고 시스템

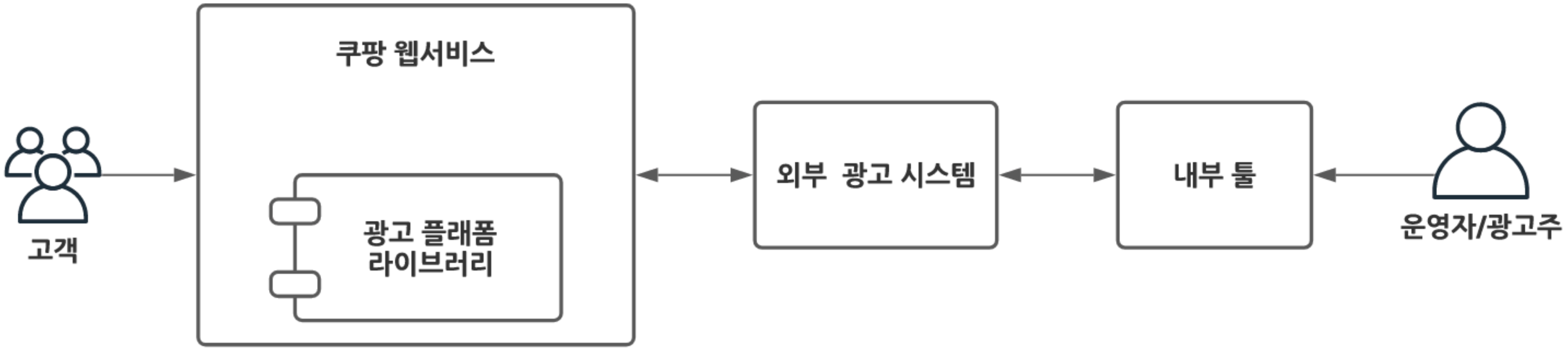
- 광고주가 자신의 상품을 여러 가지 매체를 통하여
소비자에게 널리 알리는 활동에 필요한 시스템



출처: 네이버 국어 사전

1.3 초기 광고 시스템 구조

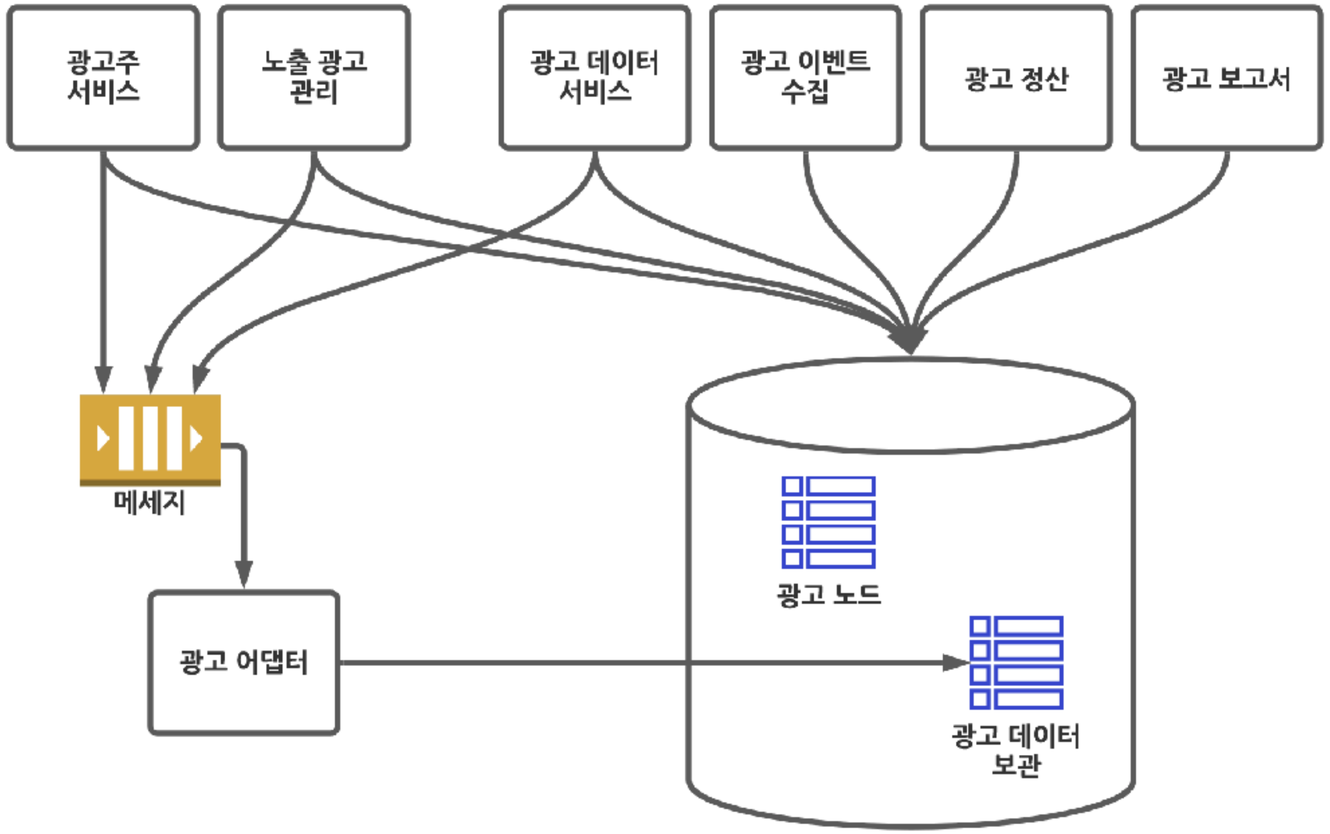
- 외부 광고 솔루션 도입
- API 연동 방식



1.4 추가 기능 구현을 위한 구조

DEVIEW 2021

- 추가 기능을 위한 별도 저장소 사용
- 다양한 팀에서 함께 사용



1.5 광고 서비스

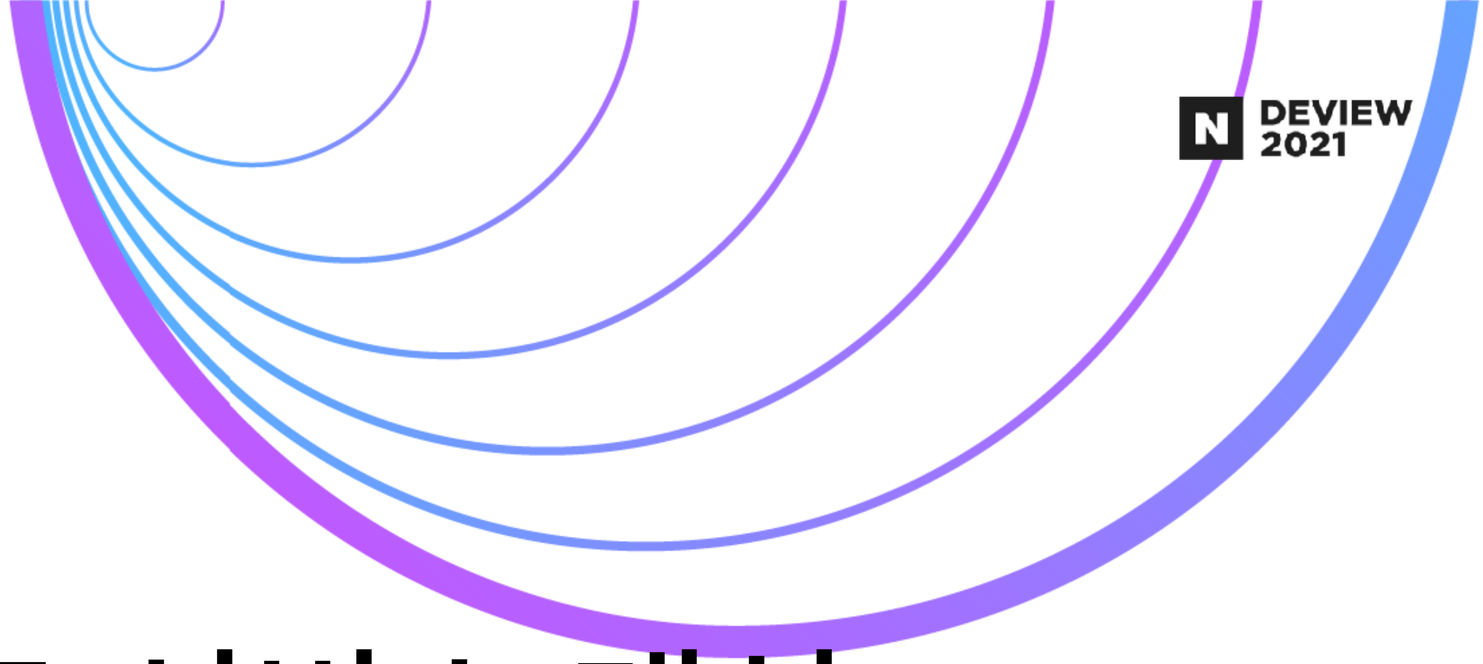
- 외부 광고 솔루션을 이용해 광고 서비스 시작
- 최초에는 상품 광고(Product Ad)

1.6 초기 광고 서비스 돌아보기

- **얻은 이점들**
  - 외부 솔루션 도입 - 빠른 광고 시스템 구축
  - 고객에게 빠르게 상품 광고 시작
  - 회사의 수익에 기여

- **개선할 점**
  - 외부 솔루션 계약으로 API의 호출 제약
  - 다양한 광고 상품 지원 어려움
  - 공유 저장소 사용으로 인한 팀 간의 의존성
  - 대량의 데이터 처리(엑셀) 필요

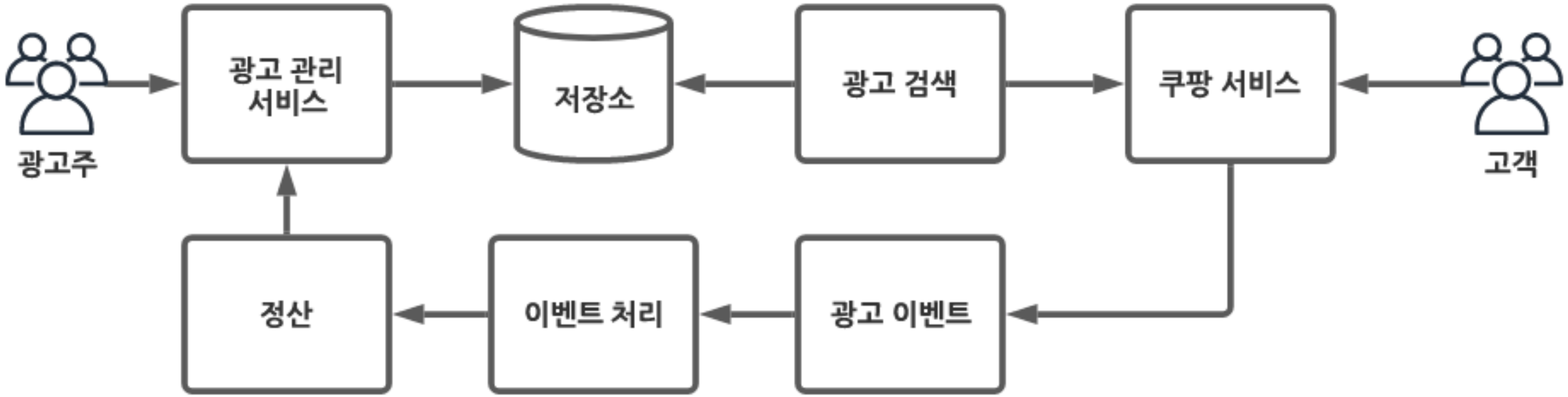
출처: https://pixabay.com/ko/



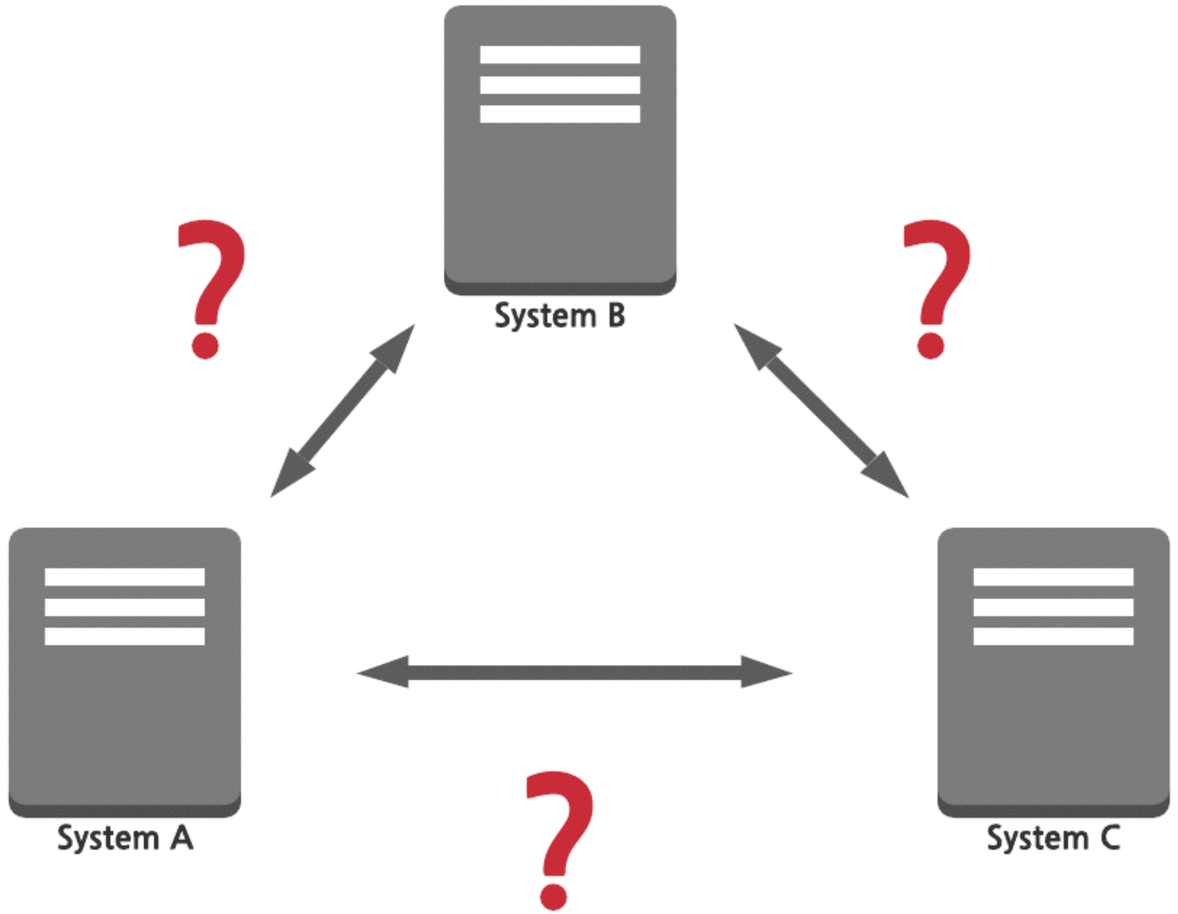
2. 광고 서비스 개선

2.1 도메인별 시스템 분리

- 공유 저장소에서 각각의 도메인 식별
- 도메인 기반으로 시스템 구축



2.2 분산된 시스템 간의 통신



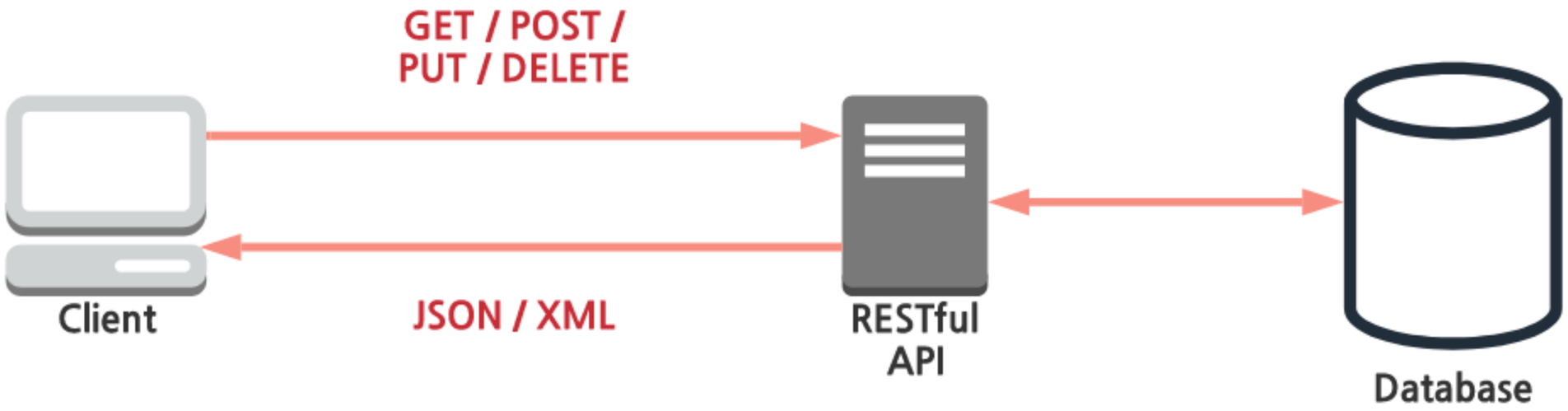
RESTful API

VS

GraphQL

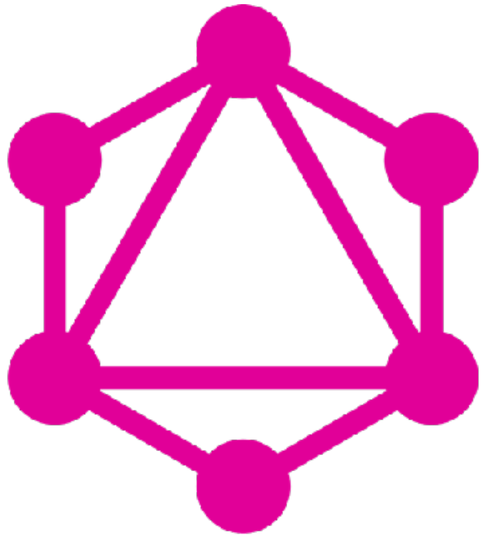
2.3 API, RESTful API

- 일반적이고 간단하고 쉽게 구현 가능
- REST이라는 설계 기준
- RESTful API



2.4 GraphQL

DEVIEW 2021



GraphQL

A query language for your API

2.5 GraphQL 특징

설계	GraphQL	REST
구성	스키마 / 타입 시스템	URL endpoints
동작	Query, Mutation, Subscription	CRUD
Endpoint	단일 접점(API 1개)	URL 집합
버전	API 버전 변경이 적음	다수의 API 버전 지원
데이터 포맷	단일 JSON 포맷	다수의 데이터 포맷
관점	클라이언트 주도 설계	서버 주도 설계
학습곡선	어려움	보통

3. GraphQL 도입과 얻은 점

3.1 GraphQL 장점

- 클라이언트에게 많은 제어권
- 단일접점(API가 1개)
- 타입 시스템, 자동화된 데이터 유형 검사
- 오버페칭, 언더페칭이 적다
- 상대적으로 빠르다

이미지 출처: 개비스콘 광고

3.1.1 클라이언트에게 많은 제어권

GraphQL Request	GraphQL Response

{
  here {
  name
  }
}
	{
	  "hero": {
	  "name": "Luke Skywalker"
	  }
	}

{
  here {
  name
  height
  hairColors
  }
}
	{
	  "hero": {
	  "name": "Luke Skywalker",
	  "height": 1.72,
	  "hairColors": [“black”, “brown“]
	  }
	}

3.1.2 단일 접점(API가 1개)

GraphQL	RESTful API
POST https://host/graphql query { books { name }} query { books(id:100) { name }}	GET https://host/books POST https://host/books/100 DELETE https://host/books/100
POST https://host/graphql query { authors { name }} query { authors(id:200) { name }}	GET https://host/authors POST https://host/authors/200 DELETE https://host/authors/200
POST https://host/graphql query { stores { name }} query { stores(id:300) { name }}	GET https://host/stores POST https://host/stores/300 DELETE https://host/stores/300

3.1.3 타입시스템 통한 엄격한 자동 검사

DEVIEW 2021

- GraphQL의 타입들, 자동 검사
  - The Query, Mutation, Subscription
    - operation types
  - Object types and fields
  - Scalar types
  - Enumeration types
  - Lists and Non-Null
  - Interfaces
  - Union types
  - Input types

```
query {
  allSpecies(first:10, ) {
    totalCount
    hero
  }
}
```

```
{
  "errors": [
    {
      "message": "Cannot query field \"hero\" on type
\"SpeciesConnection\".",
      "locations": [
        {
          "line": 35,
          "column": 5
        }
      ]
    }
  ]
}
```

3.1.4 오버페칭이 적다

DEVIEW 2021

- 클라이언트가 원하는 것보다 많은 데이터가 내려오는 상황
- 기존 API 방식에 비해 오버페칭이 적다

GraphQL Request

```
POST http://host/graphql
query {
  hero {
    name
    birthYear
  }
}
```

GraphQL Response

```
{
  "name": "Luke Skywalker",
  "birthYear": "19BBY",
}
```

RESTful API Request

```
GET http://host/book/100
```

RESTful API Response

```
{
  "name": "Luke Skywalker",
  "birthYear": "19BBY",
  "eyeColor": "blue",
  "gender": "male",
  "hairColor": "blond",
  "height": 172,
  "mass": 77,
  "skinColor": "fair",
  "homeworld": {
    “id": "cGxhbmV0czox"
  }
}
```

3.1.5 언더페칭이 적다

DEVIEW 2021

- 데이터를 요청하고나서 추가 데이터 요청을 하는 상황
- GraphQL에서는 기존 API 방식에 비해 적다

GraphQL	RESTful API

query {
  campaigns {
  id
  name
  adGroups {
  id
  name
  ads {
  id
  name
  }
  }
  }
}
	# 1차 요청 →
	https://host/campaigns
	
	# 2차 요청 →
	https://host/campaigns/<campaign-id>/ad-groups
	
	# 3차 요청 →
	https://host/campaigns/<campaign-id>/ad-groups/<ad-group-id>/ads

N+1

3.2 GraphQL 단점

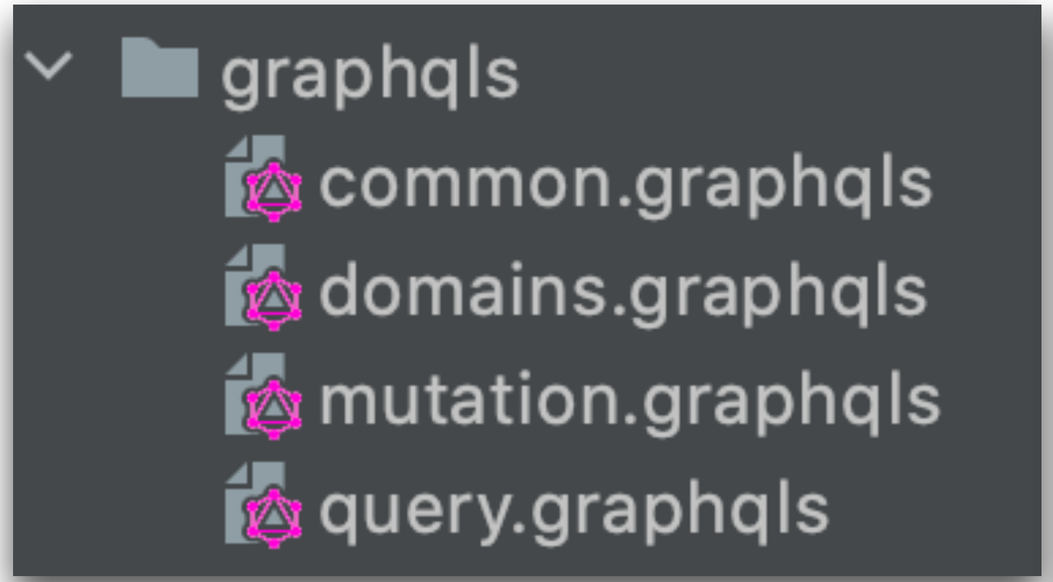
- GraphQL 스키마의 유지보수
  - 서버 개발자의 노력이 더 필요
- 캐싱 전략이 RESTful API에 비해 어렵다.
- 학습 곡선이 있다. (기존 RESTful API 보다)
- 클라이언트 요청에 제약이 있어야 한다.
  - 깊이
  - 개수

이미지 출처: 개비스콘 광고

3.2.1 GraphQL 스키마의 유지보수

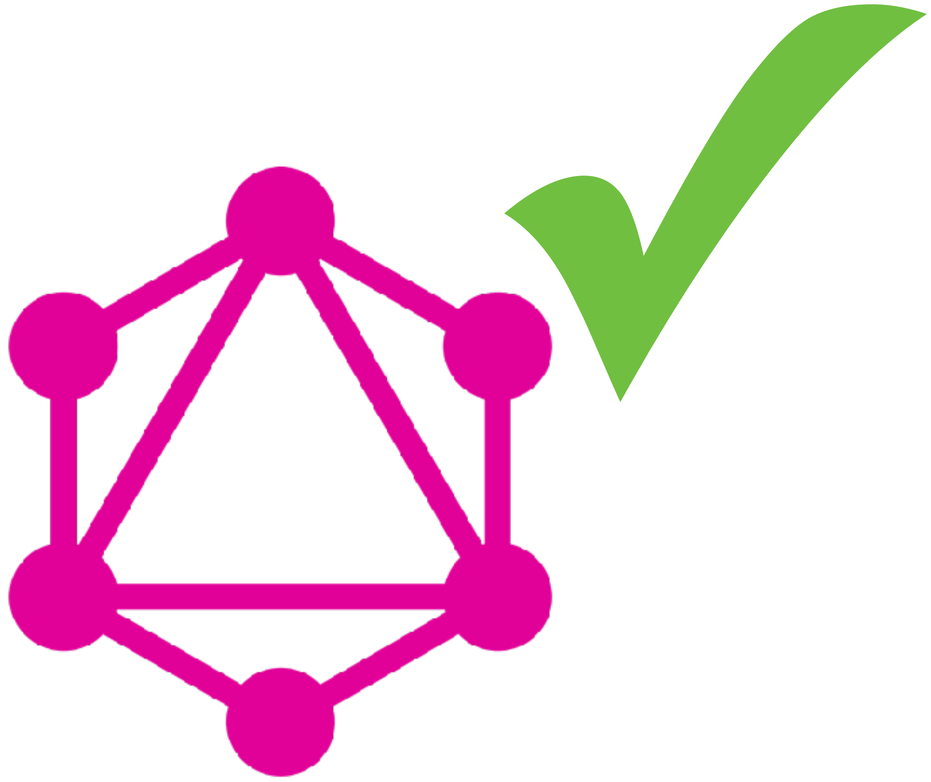
DEVIEW 2021

- GraphQL 스키마 파일 생성, 작성
- GraphQL 스키마 관련 코드 작성




```
"""캠페인 목록이나 단일 캠페인 조회 결과에 리턴되는 캠페인 정보이다."""
type PaCampaign {
    """캠페인 id"""
    id : ID!

    """광고주 id"""
    advertiserId : Long!

    """캠페인명"""
    name : String!

    """
    캠페인 시작일
    캠페인 시작일의 00:00:00 KST 시간에 대응되는 UTC 시간
    """
    startDate: DateTime!

    """
    캠페일 종료일
    캠페인 종료일의 23:59:59 KST 시간에 대응되는 UTC 시간
    """
    endDate: DateTime,
```

3.3 클라이언트(고객)을 위한 GraphQL 도입



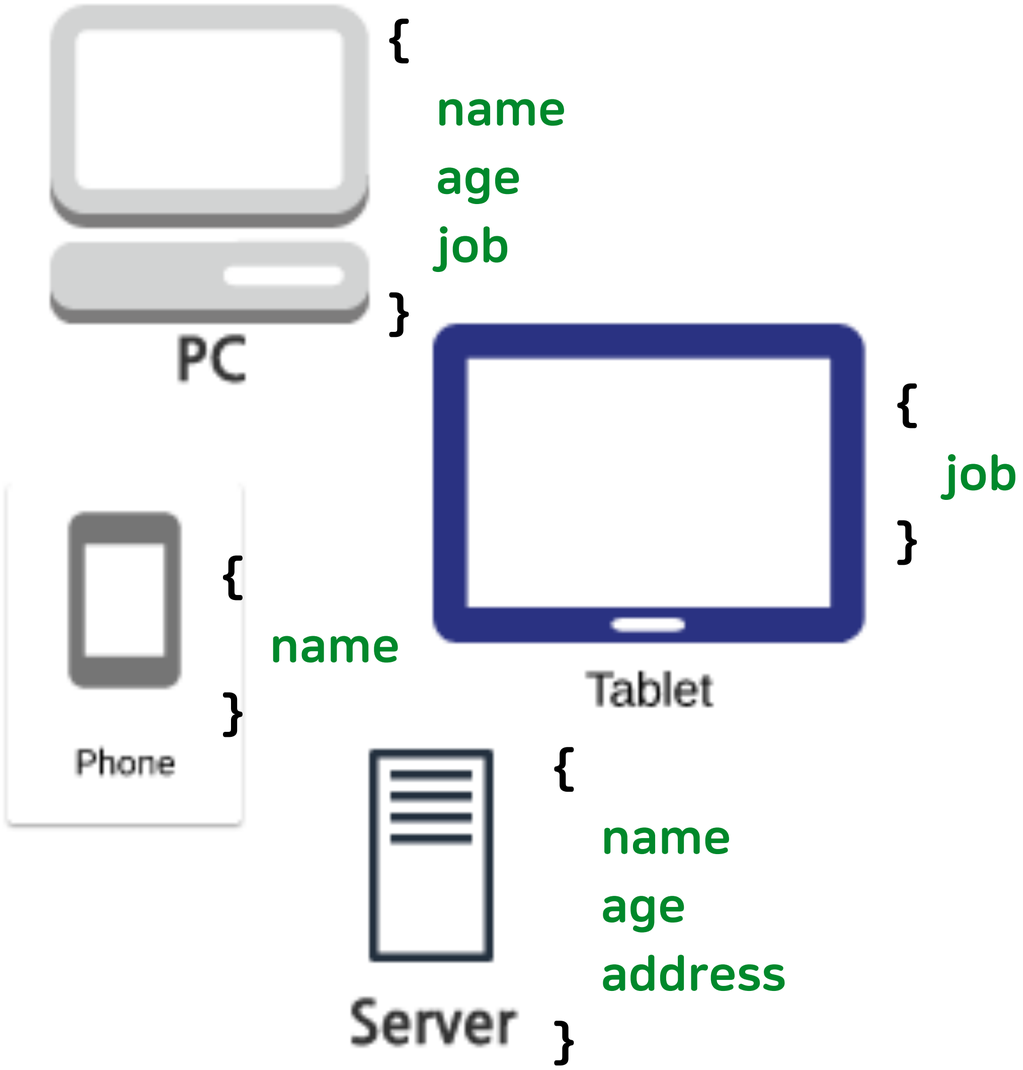
GraphQL

3.4 GraphQL 도입과 얻은 이점

- 클라이언트가 필요한 필드만 요청
- 명시적으로 조회와 변경이 분리되어 있음
- 타입시스템
- 단일접점
- 일관된 GraphQL 쿼리 방식
- 클라이언트와 서버의 명확한 의도
- API 버전
- 프레임워크의 커뮤니티와 성숙도

3.4.1 클라이언트가 필요한 필드만 요청

DEVIEW 2021



 # 기존 API 운영에 어려웠던 점

- 다양한 광고 클라이언트
- 다양한 광고 매체
- 오버페칭 발생
- 언더페칭 발생

 # GraphQL로 얻게된 이점

- 필요한 필드만 요청/처리
- 다양한 클라이언트에 유연한 대응
- 늘어나는 광고 매체 유연한 대응

3.4.2 명시적인 조회와 변경의 분리

DEVIEW 2021

 # 기존 API 운영에 어려웠던 점

- 유사하거나 중복된 API
- HTTP Method(POST) 용도가 다름
  - 조회
  - 변경

 # GraphQL로 얻게된 이점

- 명시적으로 조회와 변경 구분
  - Query(조회)
  - Mutation(변경)
  - Subscription(구독)
- 유지보수성 향상

이미지 출처: https://pixabay.com/ko/

3.4.3 타입 시스템

DEVIEW 2021

 # 기존 API 운영에 어려웠던 점

- 데이터 유형에 따른 오류
- 다운스트림에 영향
- 추가적인 데이터 유형 검사 코드

 # GraphQL로 얻게된 이점

- 타입시스템
- 자동화된 데이터 유형 검사
- 하위 스트림의 영향도 감소
- 좀 더 중요한 비지니스에 집중

이미지 출처: https://pixabay.com/ko/

3.4.4 단일접점

DEVIEW 2021

 # 기존 API 운영에 어려웠던 점

- 시간이 갈수록 늘어나는 API 접점
- API간의 정체성 모호
- API 기능의 유사성과 중복
- API 버전관리

 # GraphQL로 얻게된 이점

- 유사하거나 중복된 API 감소
- 잦은 버전 상승 줄이기
- API 유지보수성 향상

3.4.5 일관된 쿼리 방식

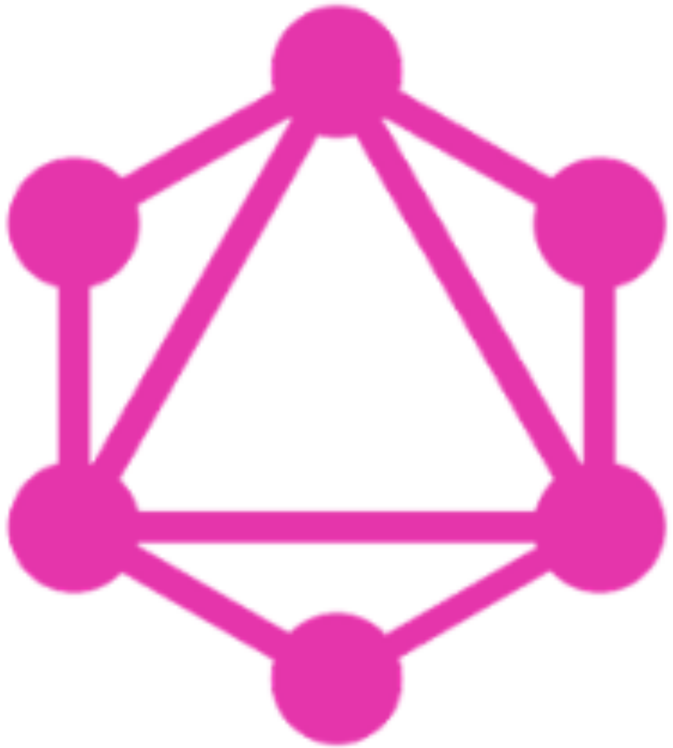
DEVIEW 2021

 # 기존 API 운영에 어려웠던 점

- RESTful한 API인가?
- 다양한 팀의 다양한 RESTful API
- 클라이언트 개발 복잡도 상승

 # GraphQL로 얻게된 이점

- 어떤 GraphQL 서비스든 단일의 GraphQL 쿼리
- 클라이언트 복잡성 저하



3.4.6 클라이언트나 서버의 명확한 의도

DEVIEW 2021

 # 기존 API 운영에 어려웠던 점

- 호출하는 데이터와 의도 모호
- 응답하는 데이터와 의도 모호

 # GraphQL로 얻게된 이점

- 명확한 요청과 응답
- 클라이언트 의도와 맥락
- API 개선, API 설계 향상

이미지 출처: https://pixabay.com/ko/

3.5 GraphQL 도입을 돌아보기

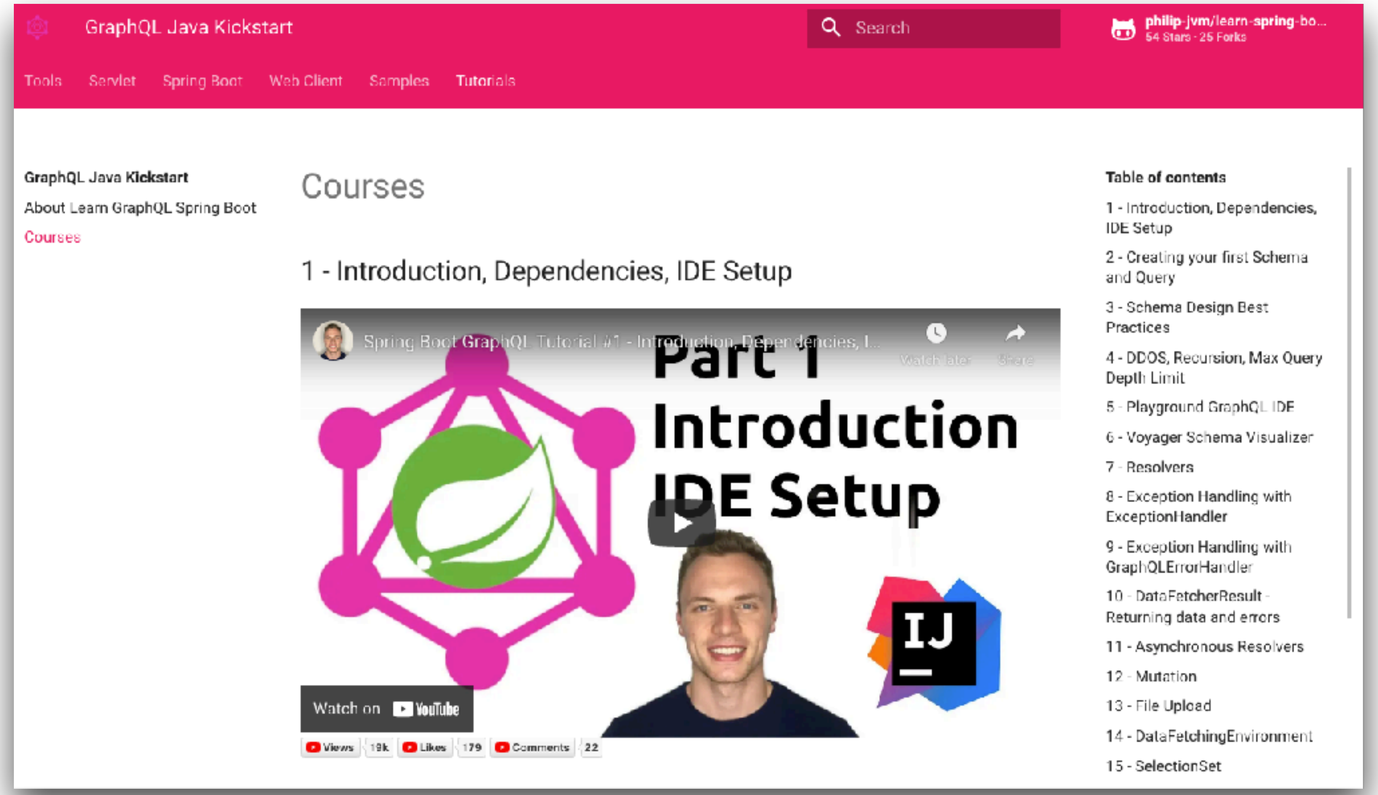
기대한 점	기대 결과
클라이언트 제어권	😀
명시적인 조회, 변경 분리	😀
타입 시스템	😀
단일 접점(API 1개)	😑
일관된 쿼리	😀
명확한 맥락(의도)	😀
버전 관리	😀
기존 인프라와의 통합	😰

4. Netflix Domain Graph Service 도입과 얻은 점

4.1 도입 초기: GraphQL Java Kickstart

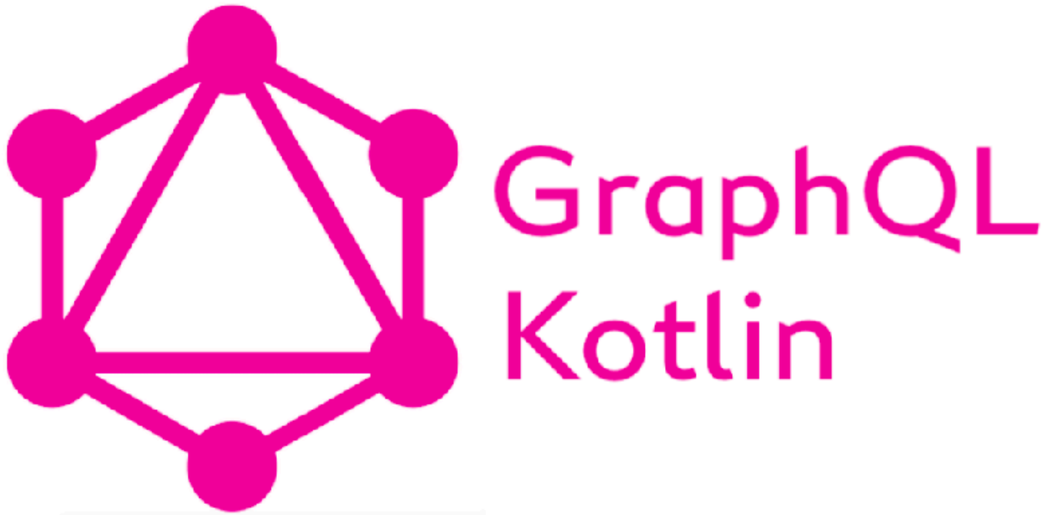
DEVIEW 2021

- 팀에서 사용하는 언어: 자바
- 팀에서 사용하는 Spring Boot 통합 이점
- 다양한 Spring Boot Starter
  - graphql
  - graphiql (도구)
  - playground (도구)
  - voyager (도구)
- 동영상 강좌 제공



4.2 도입 중반: GraphQL Kotlin

- 팀의 메인 언어 변경: 자바 -> 코틀린
- GraphQL-Java와 유사, 쉽게 적용 가능
- 신규 프로젝트에 GraphQL Kotlin
- Code First(Schema Generator) 사용



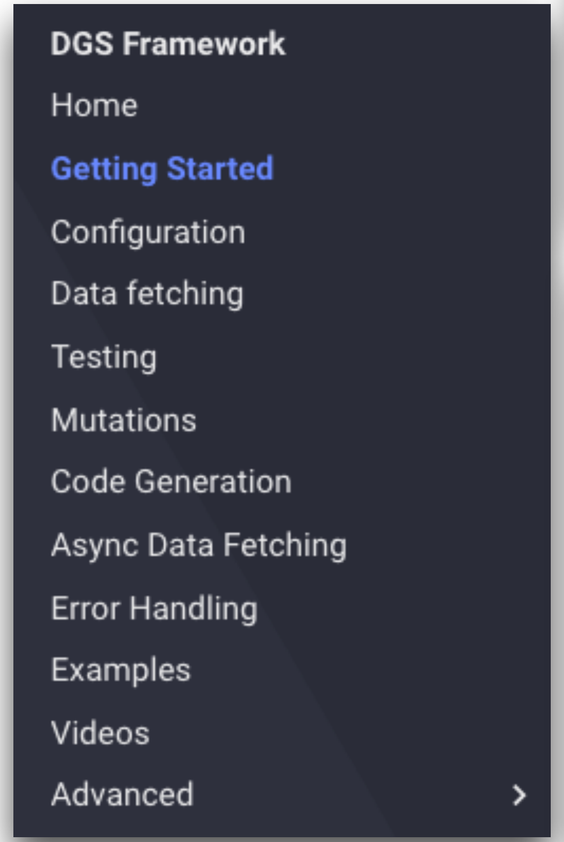
4.3 기존 GraphQL Framework 돌아보기

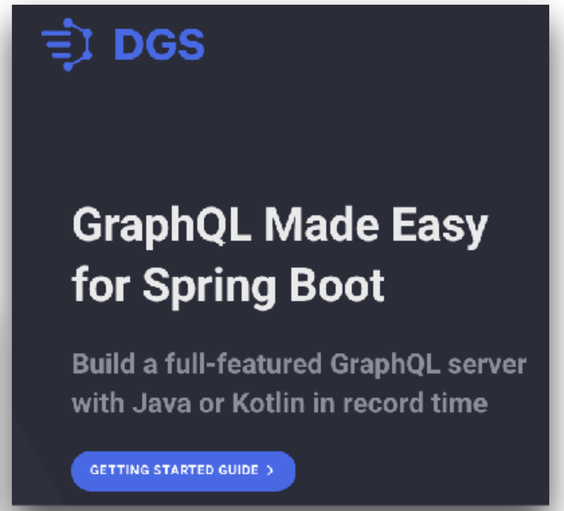
항목	회고
Framework 개발 문서	😕
Test Framework 지원	😕
Annotation Programming 지원	😕
오류 처리	😕
Best Practice 참고 사례	😕

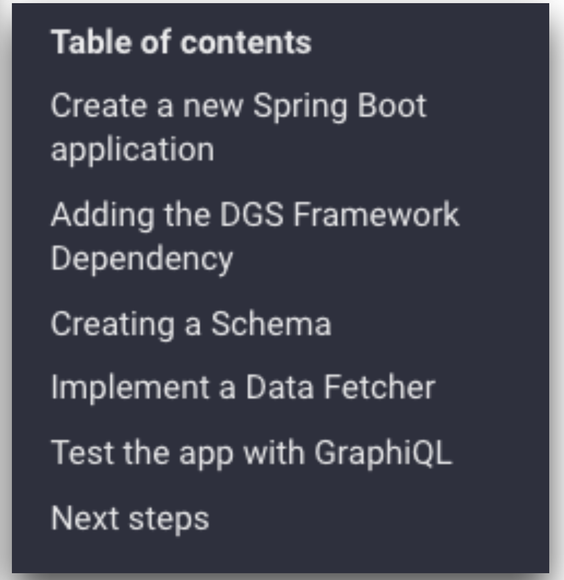
4.4 현재: Domain Graph Service (DGS)

DEVIEW 2021

- 갈증을 느낀 상황
- Netflix DGS 2021년 2월 Released
- GraphQL for Spring Boot
- Full-featured
- 너무 아름답고 감동적인 문서
- 빠르게 도입 고려







4.5 DGS 도입시 고려한 점과 얻은 점

- 기존 GraphQL Framework보다 더 나은 기능을 지원하는가?
- Annotation Programming을 지원하는가?
- Test Framework를 지원하는가?
- File Upload를 지원하는가?
- Logging/Metric Infra와 통합은?
- GraphQL Federation을 지원하는가?
- Java뿐만 아니라 Kotlin까지 지원하는가?

이미지 출처: https://pixabay.com/ko/

4.5.1 기존보다 더 나은 기능을 지원하는가?

DEVIEW 2021

Domain Graph Service의 지원 기능들

- Annotation 기반 프로그래밍 모델
- GraphQL Test 관련 프레임워크
- Java/Kotlin 코드 생성하는 플러그인
- GraphQL Federation과 쉬운 연동
- Spring Security 통합
- GraphQL 구독 기능 (WebSockets and SSE)
- 파일 업로드
- 에러 처리
- GraphQL interface/union types 자동 매핑
- GraphQL 자바 클라이언트 지원
- 로깅, 메트릭 등의 모듈 형태 통합

이미지 출처: https://pixabay.com/ko/

4.5.2 Annotation Programming 지원

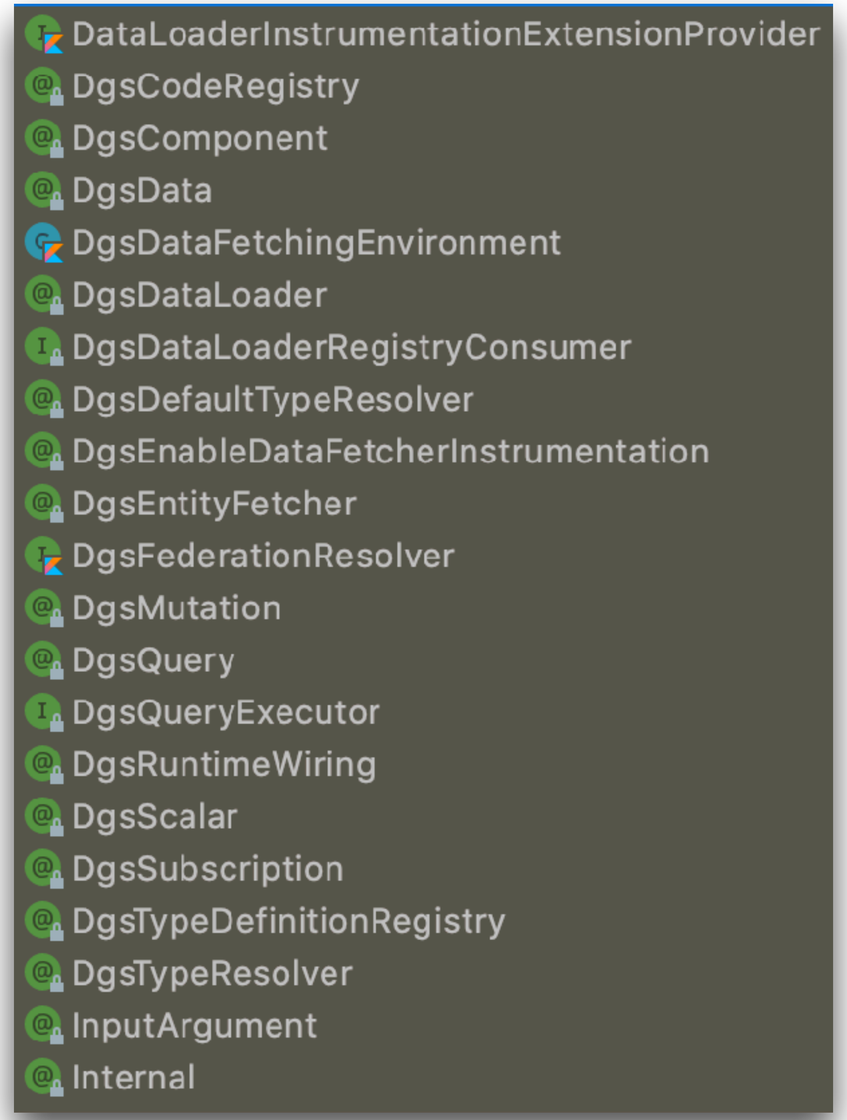
DEVIEW 2021

 # 기존 GraphQL Framework의 어려웠던 점

- Query, Mutation, Subscription 인터페이스
- 다양한 어노테이션 지원 부족

 # DGS를 통해 얻게된 이점

- @DgsComponent
- @DgsMutation
- @DgsQuery
- @DgsData
- @DgsDataLoader
- @InputArgument



4.5.2.1 개선된 코드

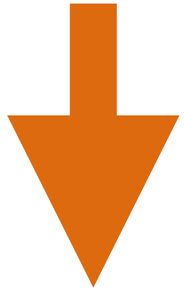
DEVIEW 2021

Before

```
interface Mutation
interface Query
```

After

```
@DgsQuery
@DgsMutation
```



```
@DgsQuery
fun adStore(@InputArgument storeId: Long): AdStore? {}
@DgsMutation
fun createCampaign(@InputArgument input: CreateCampaignInput) {}
```

4.5.3 Test Framework를 지원하는가?

DEVIEW 2021

기존 GraphQL Framework의 어려웠던 점

- HTTP 요청 클라이언트 만들기
- 테스트용 GraphQL Query 작성
- 응답 처리 어려움
- GraphQL Mocking

DGS를 통해 얻게된 이점

- GraphQL 테스트 편의 클래스 지원
- GraphQL Mocking 지원

```
val qb = StringBuilder()
val mapper = ObjectMapper()

val jsonInputNode = mapper.valueToTree<JsonNode>(input)
val jsonUserInfoNode = mapper.valueToTree<JsonNode>(userInfo)
val variables = ObjectMapper().createObjectNode()
variables.set<JsonNode>("var_input", jsonInputNode)
variables.set<JsonNode>("var_userInfo", jsonUserInfoNode)

// Variables
qb.append("mutation CreateGroupAds_v2_3b_Call(\n")
qb.append("    ${'$'}var_input: CreatePaGroupAdsInput_v2_3b!\n")
qb.append("    ${'$'}var_userInfo: UserInfo!\n")
qb.append(")\n")

// Query
qb.append("{ $methodName(input: ${'$'}var_input, userInfo: ${'$'}var_userInfo)")

// Output
qb.append("""
{
    requestId
    appErrors {
      reasonCode
```

4.5.3.1 간편해진 테스트와 BDD 스타일

DEVIEW 2021

```
@Autowired
lateinit var dqe: DgsQueryExecutor

// given
var …
val …

// when
dqe.executeAndExtractJsonPath()
dqe.executeAndExtractJsonPathAsObject()

// then
assertNotNull()
assertEquals()
```

```
@Test
fun findRoleBy() {
    // given
    val advertiserId = 4L
    val adType = AdType.PA
    val resourceType = ResourceType.ADVERTISER

    // when
    val response = dgsQueryExecutor.executeAndExtract
        """
            { resourceRoles(adType: $adType, resource
        """.trimIndent() ,
        jsonPath: "data.resourceRoles",
        ResourceRoleResponse::class.java
    )

    // then
    assertNotNull(response)
    assertEquals(advertiserId, response.resourceId)
    assertEquals(adType, response.adType)
}
```

4.5.4 Code Generation

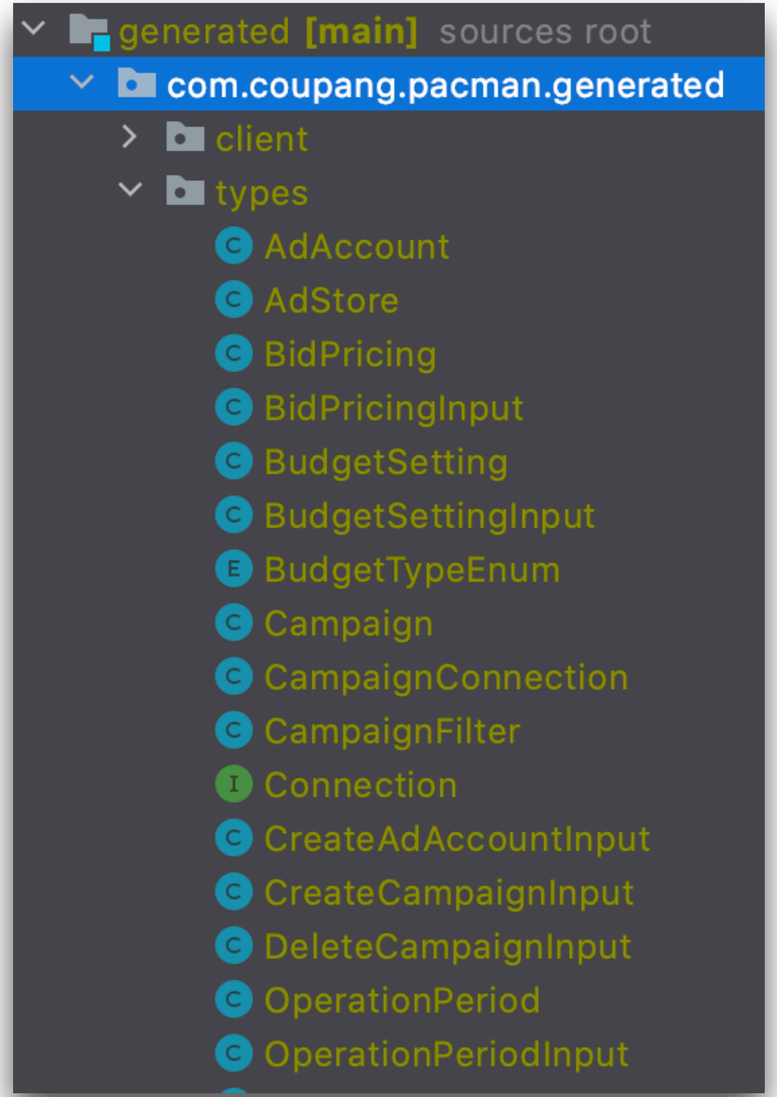
DEVIEW 2021

 # 기존 GraphQL Framework의 어려웠던 점

- 모든 코드를 직접 작성

 # DGS를 통해 얻게된 이점

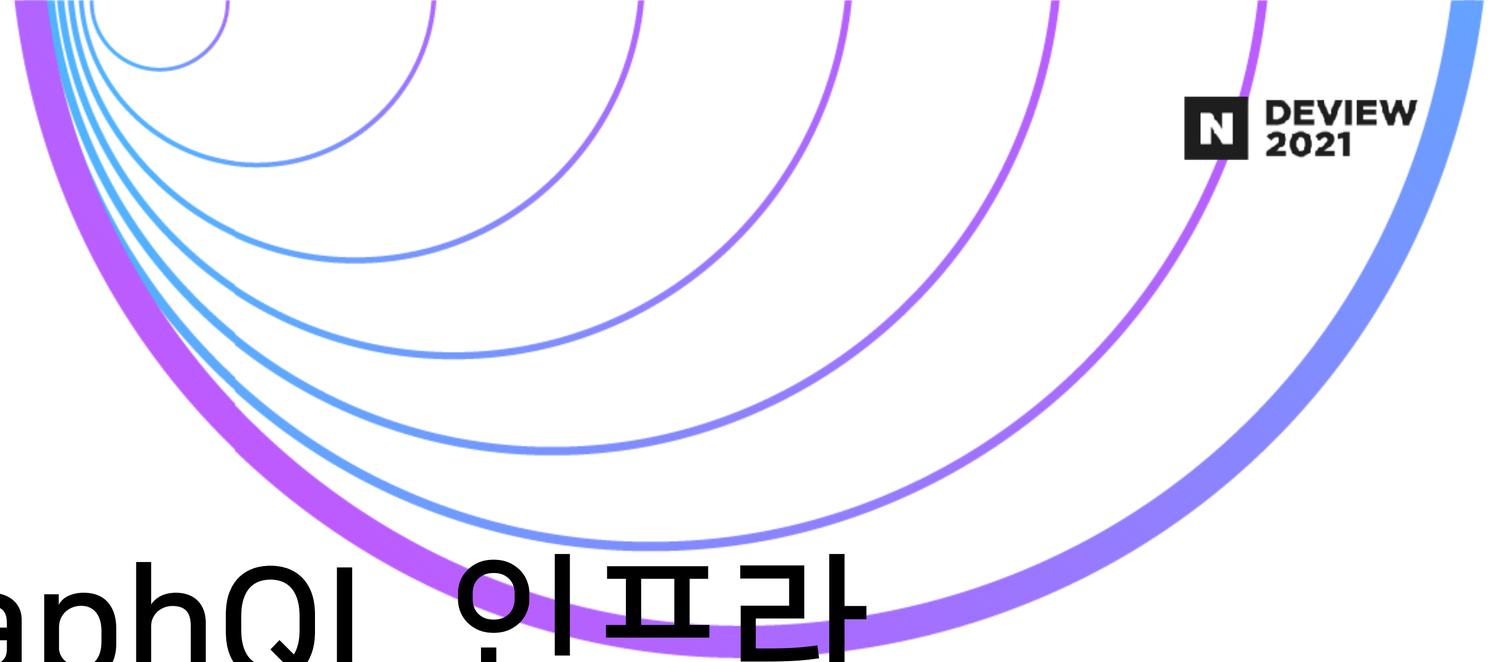
- 스키마를 이용한 코드 자동 생성
- Type, Input, Constants 등등
- 직접 코딩하지 않음, 개발 생산성 도움
- 간단한 설정만으로 쉽고, 부드럽게 제공



4.6 Domain Graph Service 돌아보기

DEVIEW 2021

항목	회고
Framework 개발 가이드	
Test Framework 지원	
Annotation Programming 지원	
오류 처리	
로깅과 메트릭 통합	
스키마 기반 코드 자동 생성	



DEVIEW 2021

5. GraphQL 인프라 운영 경험

5.1 GraphQL 로깅, 로깅 뷰어

DEVIEW 2021

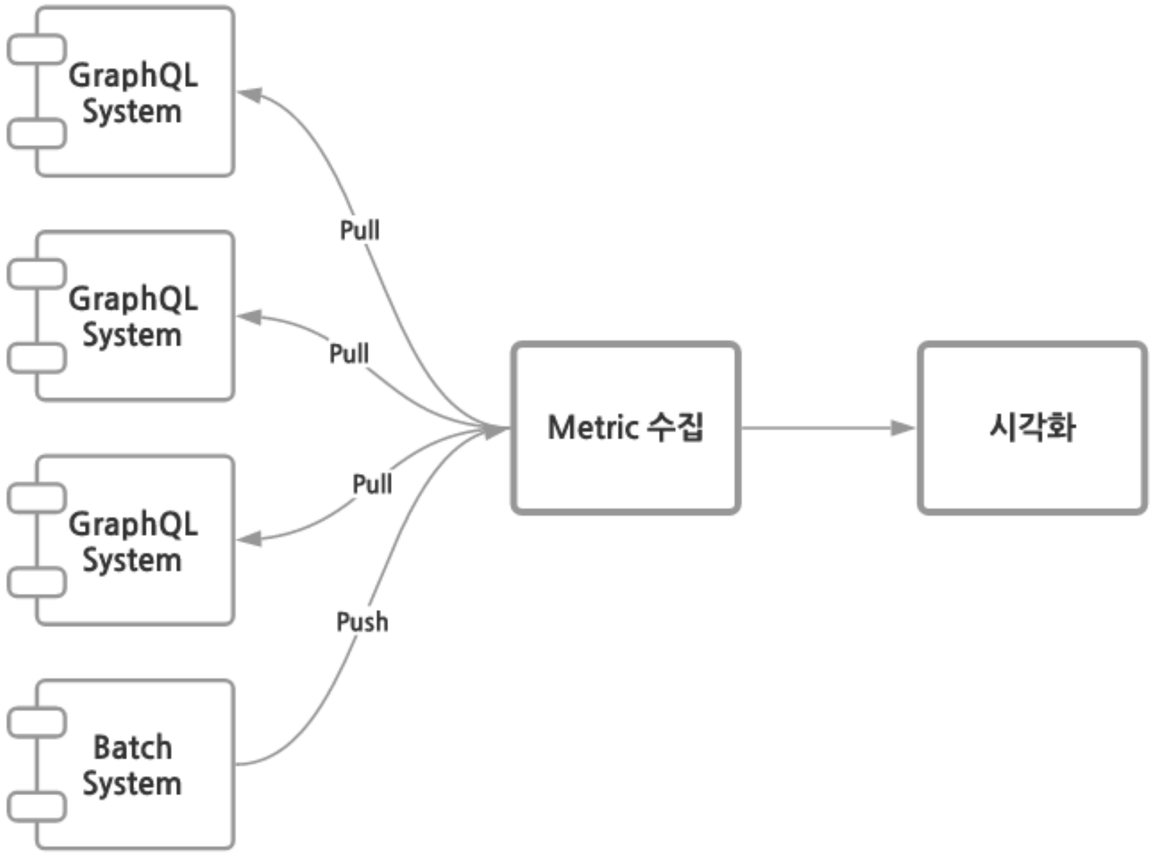
- 클라이언트 요청별 Execution ID
- 요청별 소요 시간 측정
  - executionId: d52c2d1
  - name: getAdvertisers
  - elapsedTime: 4ms

```
    Create an unique identifier from the given string
    Returns: a query execution identifier
public static ExecutionId generate() {
    return new ExecutionId(UUID.randomUUID().toString());
}
```

5.2 GraphQL 서비스 지표 수집/모니터링

DEVIEW 2021

- GraphQL 서비스 모니터링 2가지
  - 시스템 모니터링
  - GraphQL 모니터링



5.3 GraphQL 작업 단위 모니터링

DEVIEW 2021

GraphQL 단일접점

5.3.1 GraphQL 작업 구분 방법

DEVIEW 2021

- 클라이언 요청의 GraphQL 작업 구분
  - Query(조회)
  - Mutation(변경)
  - Subscription(구독)
- 작업별 로깅, 지표 수집, 시각화

class SimpleInstrumentation
class InstrumentationExecutionParameters
class ExecutionInput

executionInput.getQuery()

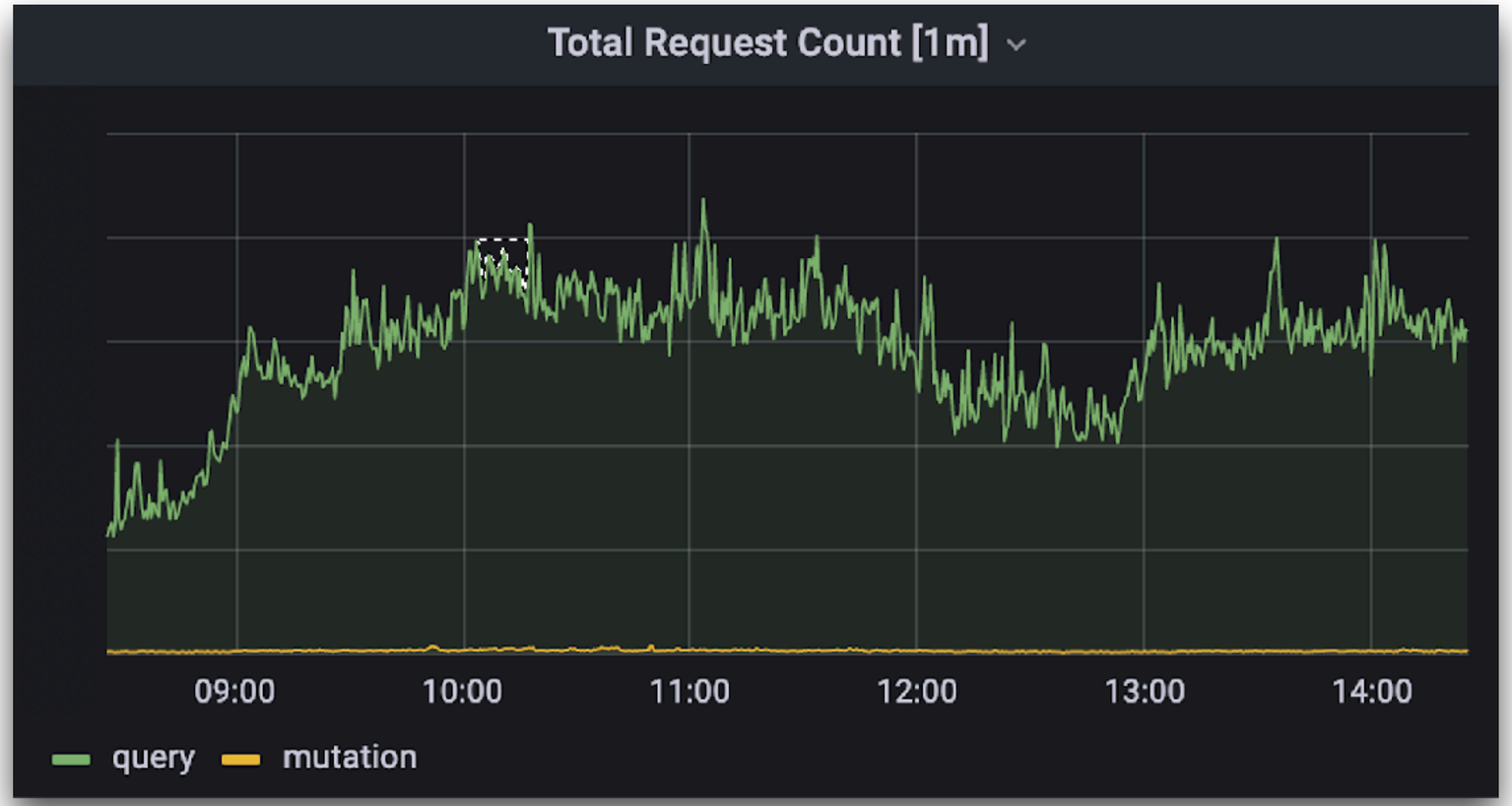
```
when (parameters.query) {
    "mutation" -> println("mutation logging/metric")
    "query" -> println("query logging/metric”)
    "subscription" -> println("subscription logging/metric")
    else -> {
        println("unknown graphql job")
    }
}
```

5.3.2 GraphQL 작업 단위 모니터링

DEVIEW 2021

 # 개선한 부분

- 클라이언트의 요청을 GraphQL 작업 단위
- Query
- Mutation
- Subscription

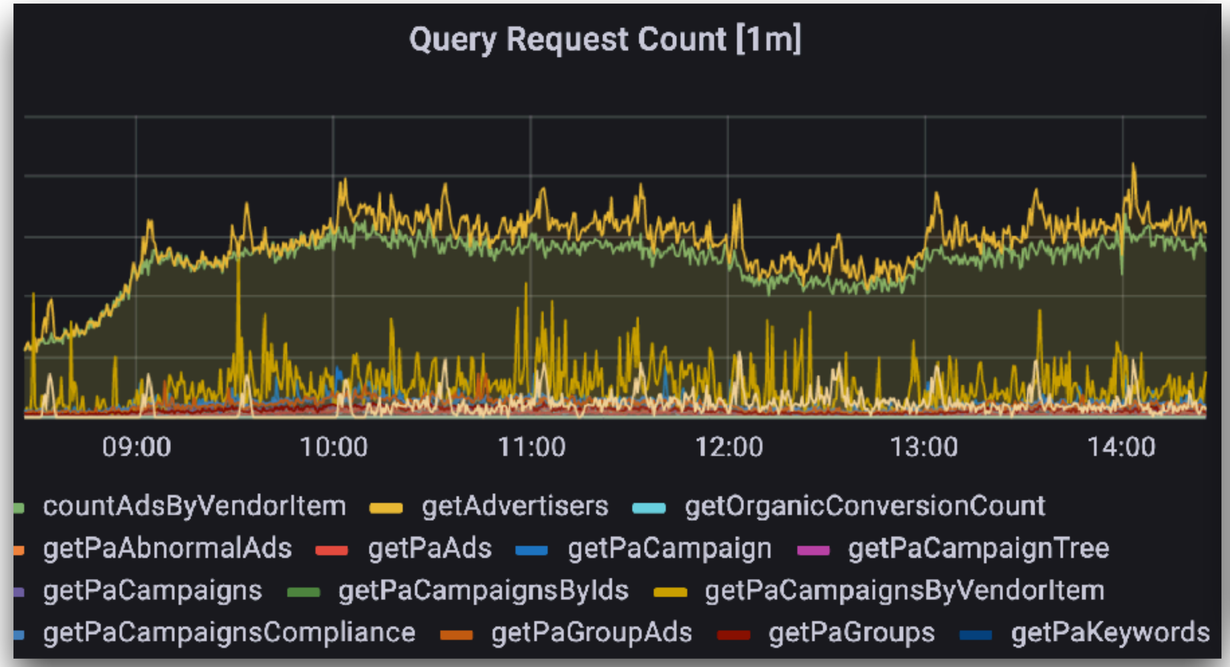


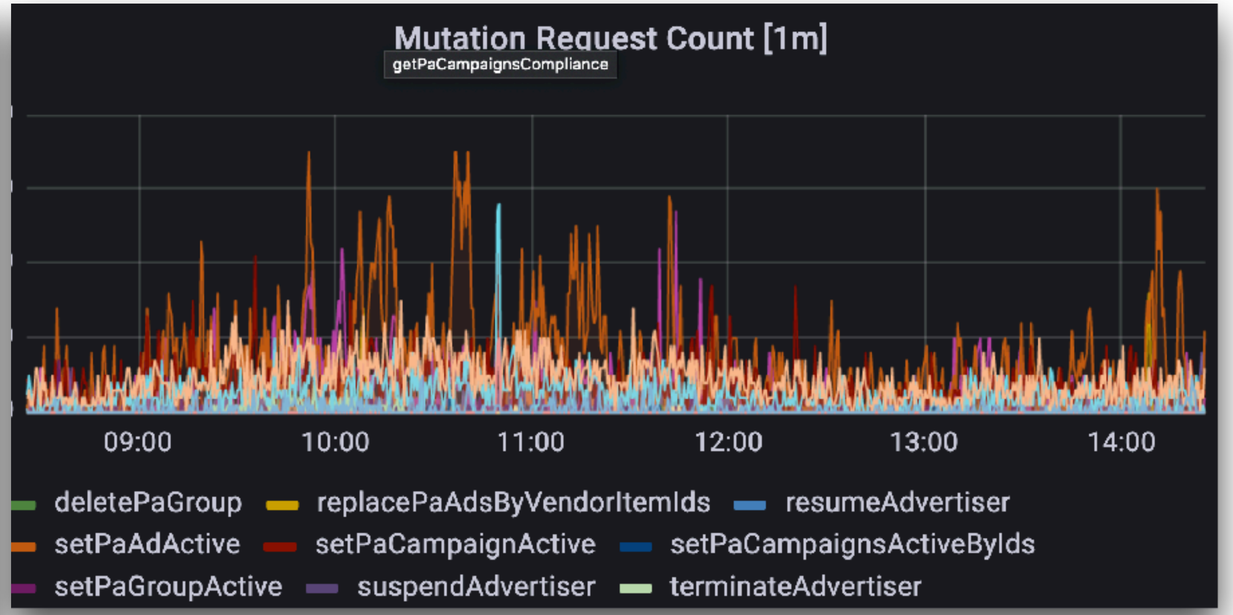
5.3.3 GraphQL 세부 작업 모니터링

DEVIEW 2021

 # 개선한 부분

- GraphQL Query(조회)/Mutation(변경)의 상세 작업을 구분
- Request Count, Latency, Error Count
- 세부 작업별 상태 확인, 개선, 확장성과 안정성 향상



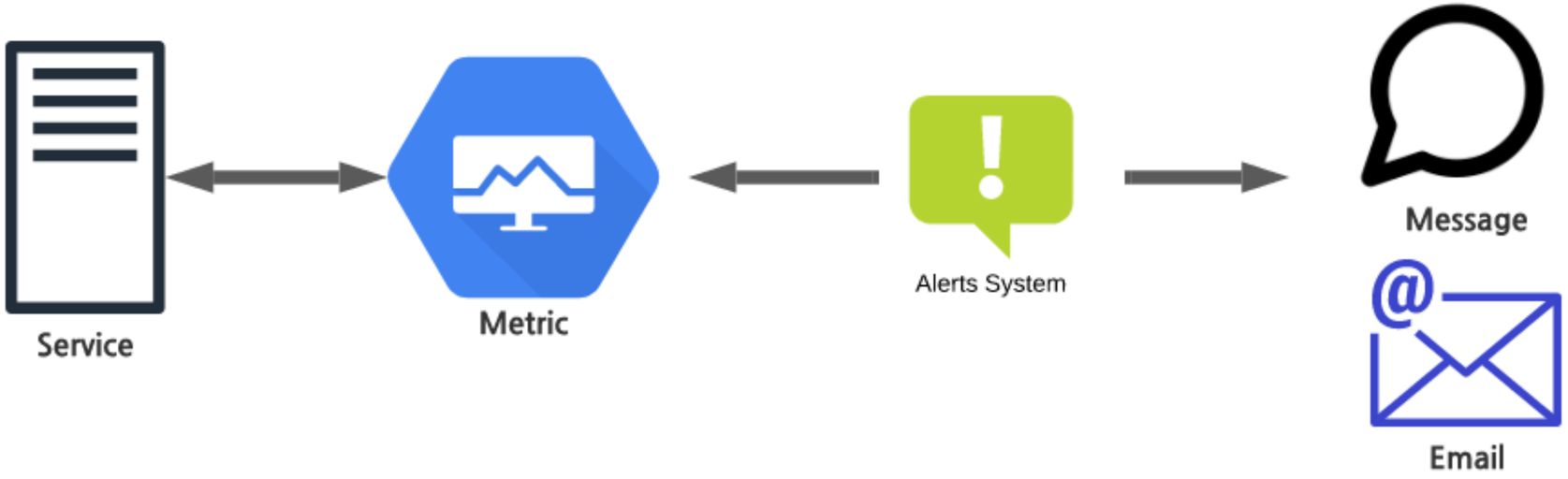


5.4 GraphQL API 서비스 알림 구조

DEVIEW 2021

 # 개선된 부분

- 메트릭 기반으로 알림 설정
- GraphQL Query와 Mutation 등의 구분된 알림
- 이하 세부 작업별 구분된 알림



6. GraphQL 개발시 배운 점 (Feat. DGS)

6.1 Schema First vs Code First

DEVIEW 2021

- GraphQL 개발의 2가지 방향
  - Schema First
  - Code First

- 팀의 GraphQL 개발 방향
  - 초기: Schema First
  - 중반: Code First
  - 현재: Schema First

이미지 출처: https://pixabay.com/ko/

6.1.1 Schema First를 선택한 이유

DEVIEW 2021

 # Schema First 선택 이점

- 스키마 정의 후 이를 중심으로 개발: 협업 안정적
- 스키마 변경시: 호환성 체크 쉬움
- 더 나은 코드 가독성
- DGS에서는 둘 다 지원

이미지 출처: https://pixabay.com/ko/

6.2 클라이언트 요청 필드 처리

DEVIEW 2021

 # 어려웠던 점

- 필요한 필드만 요청, 그러나…
- 불필요한 필드 처리

 # 개선된 점

- 요청된 필드를 파악하고 처리
- 불필요한 필드 처리 감소
- 서버의 처리량 향상

이미지 출처: https://pixabay.com/ko/

6.2.1 Domain Graph Service 클래스

DEVIEW 2021

 # 개선된 내용

- DgsDataFetchingEnvironment
- DataFetchingFieldSelectionSet
    - contains(java.lang.String fieldGlobPattern)
    - containsAllOf(java.lang.String fieldGlobPattern, java.lang.String... fieldGlobPatterns)
    - containsAnyOf(java.lang.String fieldGlobPattern, java.lang.String... fieldGlobPatterns)

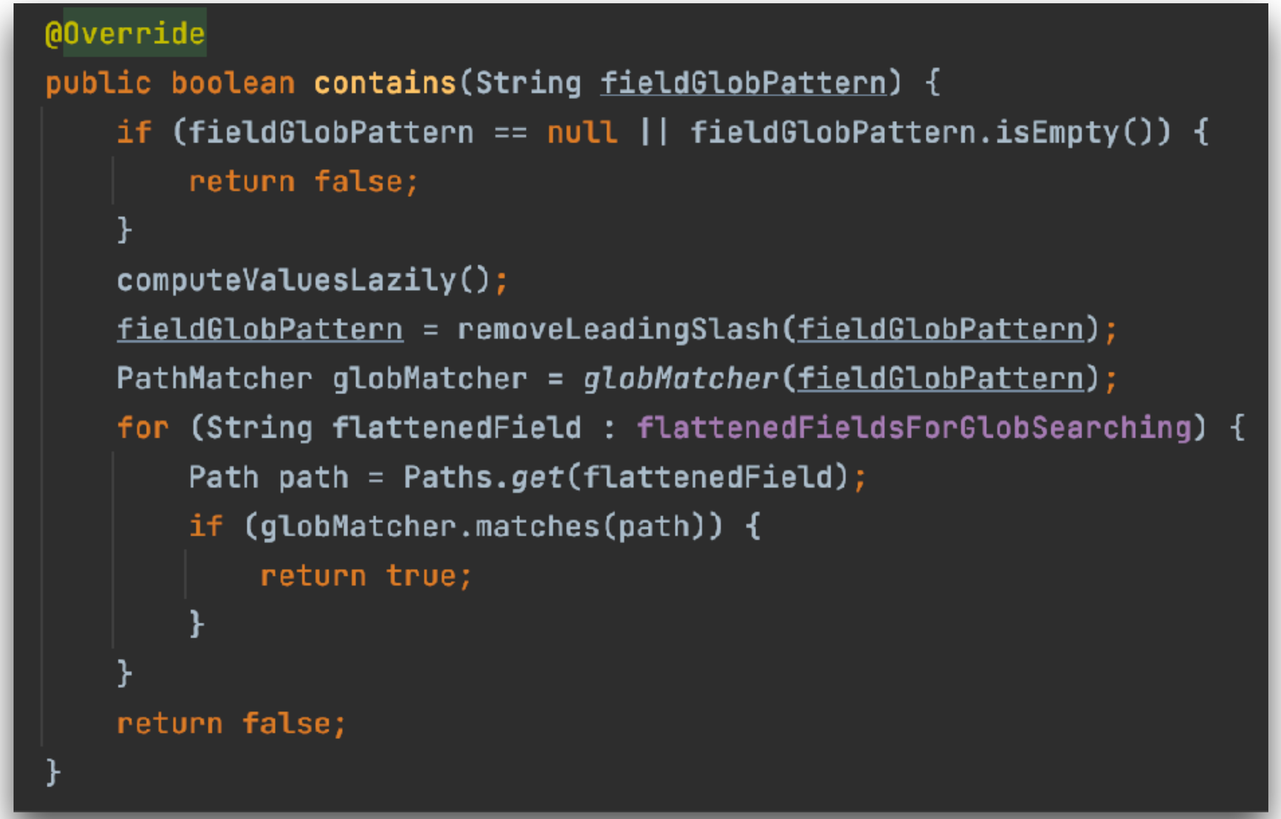
```
class DgsDataFetchingEnvironment(private val dfe: DataFetchingEnvironment) : DataFetchingEnvironment {
```

6.2.2 Domain Graph Service 필드 식별

DEVIEW 2021

 # 개선된 내용

```
val selectionSet = dataFetchingEnvironment.selectionSet
if (selectionSet.contains("필드 패턴")) {
    // 요청한 필드만 처리
}
```



```java
@Override
public boolean contains(String fieldGlobPattern) {
    if (fieldGlobPattern == null || fieldGlobPattern.isEmpty()) {
        return false;
    }
    computeValuesLazily();
    fieldGlobPattern = removeLeadingSlash(fieldGlobPattern);
    PathMatcher globMatcher = globMatcher(fieldGlobPattern);
    for (String flattenedField : flattenedFieldsForGlobSearching) {
        Path path = Paths.get(flattenedField);
        if (globMatcher.matches(path)) {
            return true;
        }
    }
    return false;
}
```

6.3 다양한 GraphQL 에러 응답 방식 존재

DEVIEW 2021

- 각 프레임워크별 다양한 에러 응답 방식
  - GraphQL Java/Kotlin
  - Domain Graph Service
  - Apollo
  - Spring GraphQL


```
{
  "errors": [
    {
      "message": "Name for character with ID 1002 could not be fetched.",
      "locations": [ { "line": 6, "column": 7 } ],
      "path": [ "hero", "heroFriends", 1, "name" ],
      "extensions": {
        "code": "CAN_NOT_FETCH_BY_ID",
        "timestamp": "Fri Feb 9 14:33:09 UTC 2018"
      }
    }
  ]
}
```

6.3.1 다양한 GraphQL 에러 응답 방식 비교

GraphQL Framework/Library	extensions
GraphQL Java	classification
Apollo	code exception
DGS	errorType, errorDetail, origin, debugInfo
Spring GraphQL	errorType

https://spec.graphql.org/June2018/#sec-Response-Format

6.3.2 우리가 추가한 에러 응답

DEVIEW 2021

 # 어려웠던 점

- Framework가 정의한 필드가 다양
- 현재 여러 GraphQL Framework 운영
- 클라이언트에서 응답처리 어려움

 # 해결 방법

- Framework와는 독립적인 에러 응답 필드 추가
  - errors > extensions > appReasonCode
  - errors > extensions > violations
- 클라이언트의 일관된 에러 응답 처리 가능

```
{
  "errors": [
    {
      "message": "...",
      "locations": [ ... ],
      "path": [ ... ],
      "extensions": {
        "appReasonCode": "...",
        "violations": "..."
      }
    }
  ]
}
```

6.4 클라이언트 관점의 디버깅/테스트

DEVIEW 2021

GraphQL 디버깅 및 테스트하는 방법들

- cURL
- GraphiQL
- Playground
- Postman (QA)

	GUI	Schema Doc 제공	Query	테스트 스크립트 생성
cURL	N	N	어려움	-
GraphiQL	Y	Y	쉬움	-
Playground	Y	Y	쉬움	-
Postman	Y	N	어려움	Y

6.5 GraphQL의 부분적 오류 대응

DEVIEW 2021

어려웠던 점
- 클라이언트가 필요한 필드만 요청
- GraphQL 요청 필드 중에 하나가 에러나는 경우

현재 방향
- 필수 필드 오류: All or Nothing
- 외부 의존성엔 CircuitBreaker 적용
  - 실패시 빈 기본값을 제공

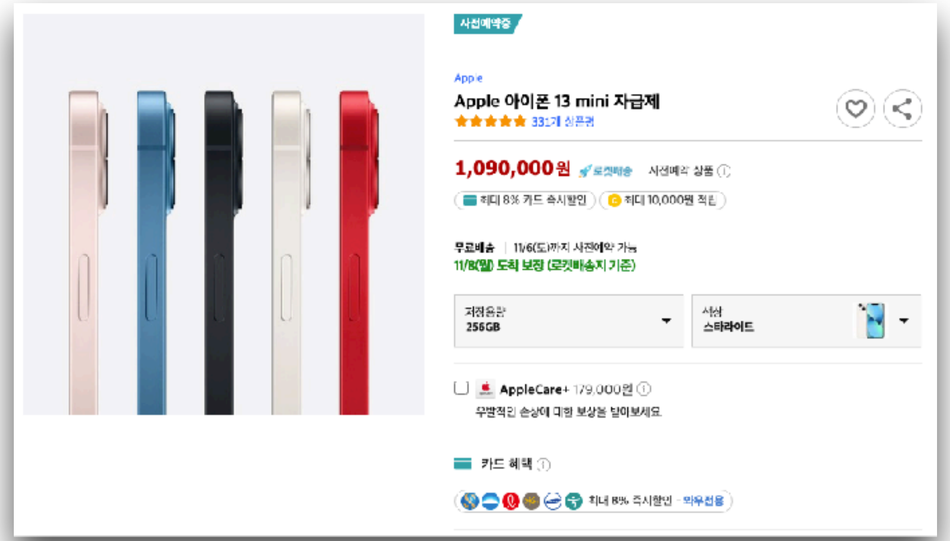
resilience4j

6.6 GraphQL 스키마 설계

DEVIEW 2021

어려웠던 점

- 클라이언트의 UI에 맞춘 설계
- UI에 따른 잦은 스키마 변경
- RESTful API와 차이가 없음



현재 방향

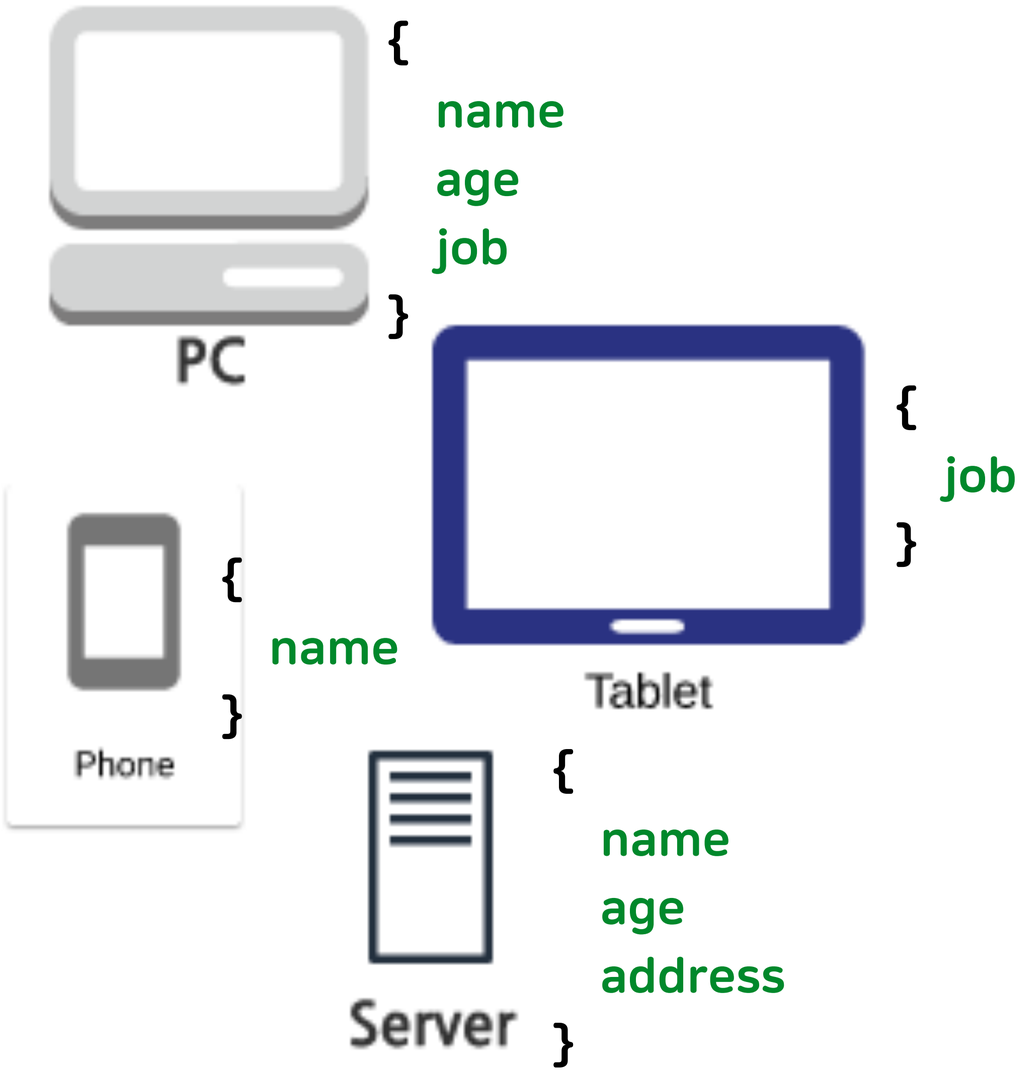
- 도메인 분석, 일반화된 스키마 설계
- 변경이 잦은 요청은 스키마 반영 최소
  - UI, UX
  - 그 외 특별한 케이스

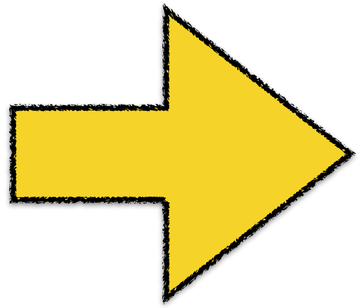
Campaign
- AdGroup
  - Ad
    - Keyword

Advertiser

발표 내용 정리

DEVIEW 2021





감사합니다.