Serverless + GraphQL
with Kotlin

나윤호

@soldier4443

About me

애정결핍덕후 백엔드 담당!

안드로이드 개발자

2017.02 - 2018.12 마이다스아이티

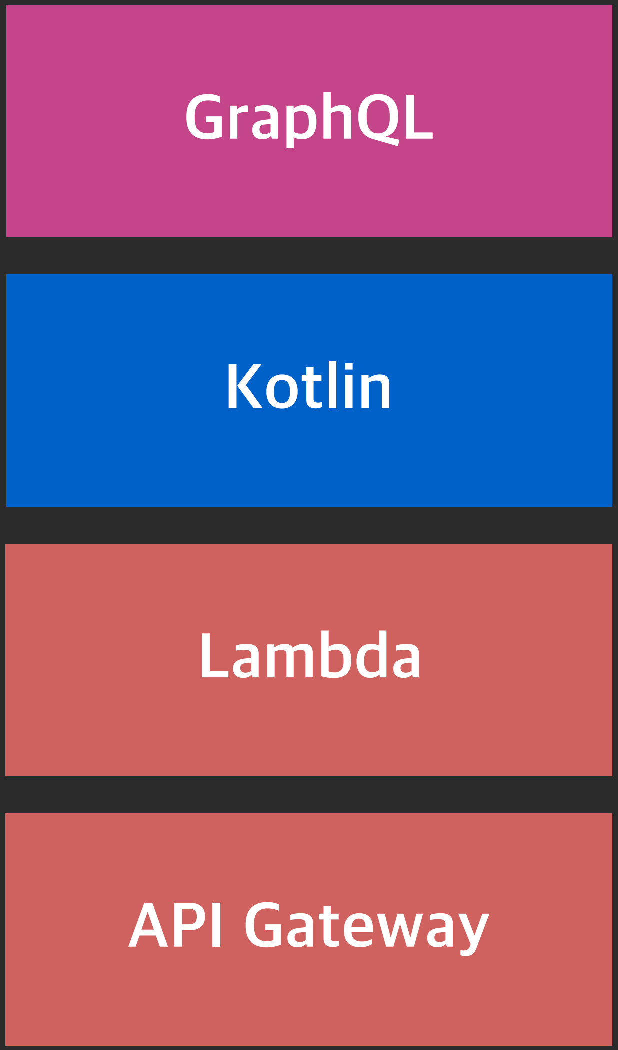
2019.04 - Riiid

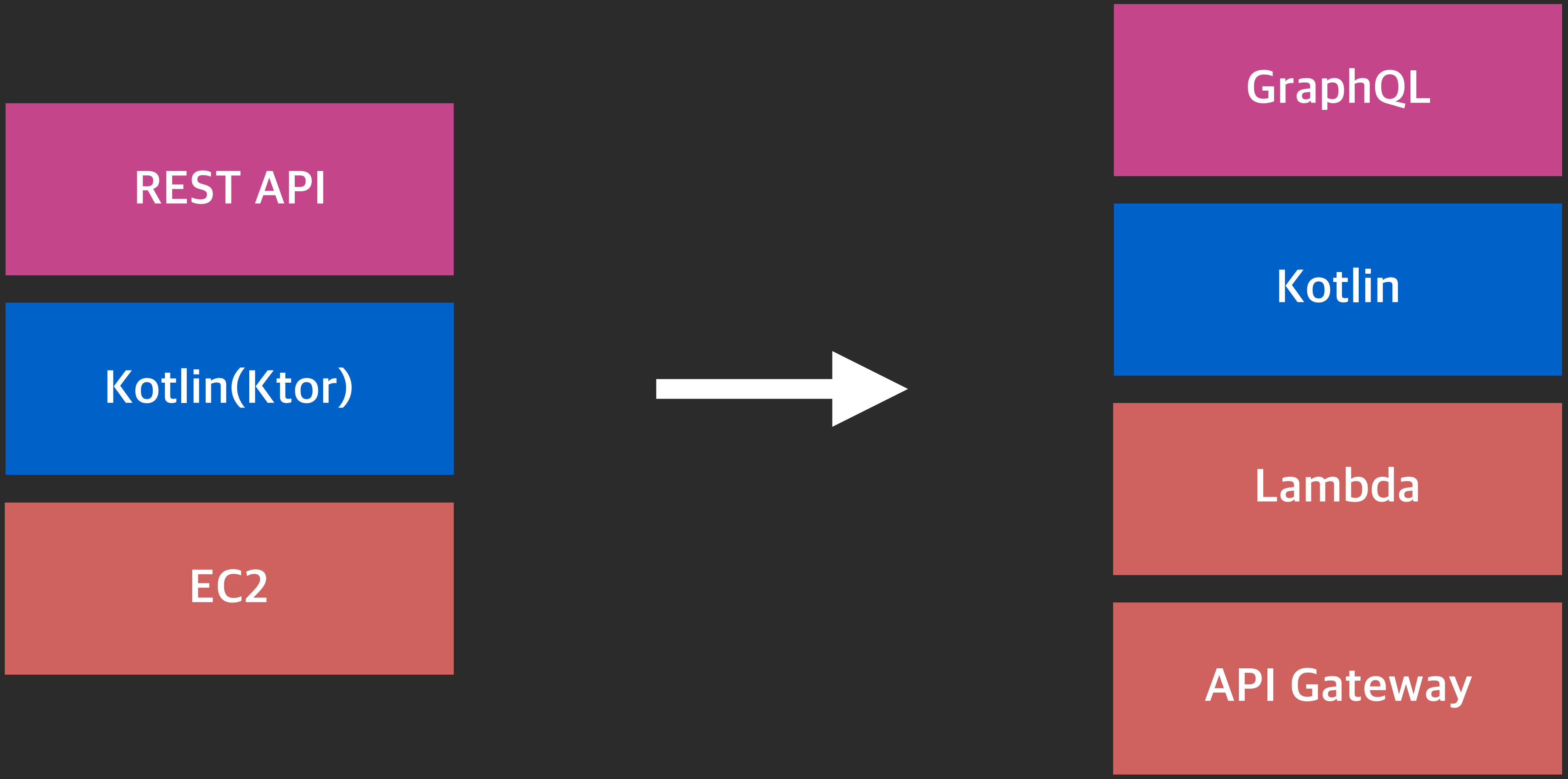
산타토익 앱 개발

신기술 덕후

비인간적
점수상승

Architecture


GraphQL
Kotlin
Lambda
API Gateway


REST API
Kotlin(Ktor)
EC2
GraphQL
Kotlin
Lambda
API Gateway

Why Kotlin

1. 익숙한 언어와 환경
2. 앱과 동일한 언어
3. 정적 타입 언어

Why Serverless

1. 비용 절감
2. 기능에만 집중
3. 개인 프로젝트와 싱크

Why GraphQL

1. API 문서 안 만들어도 될 것 같음
2. 재미

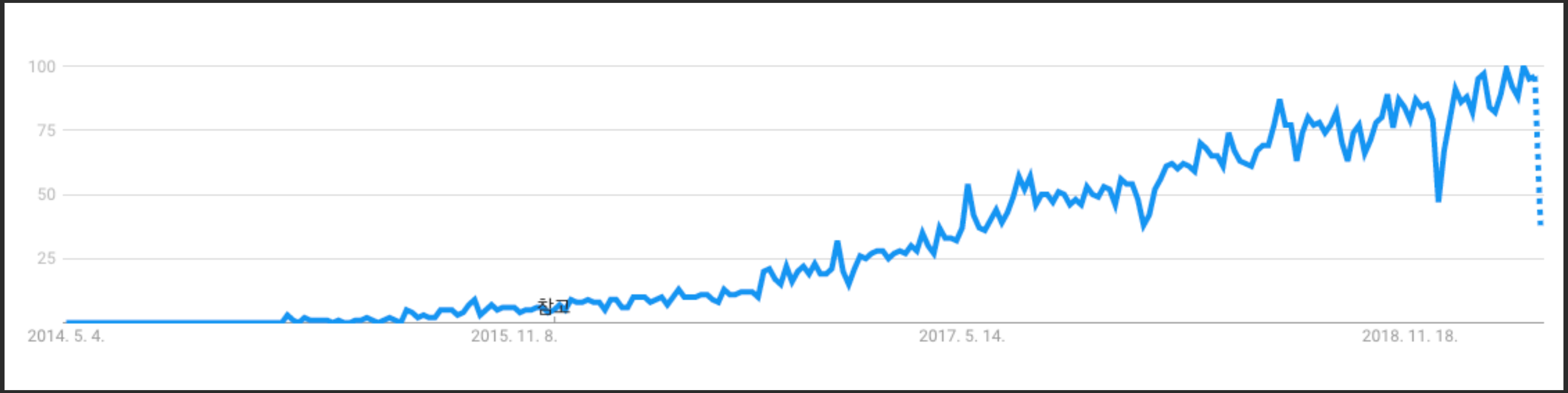
GraphQL

GraphQL

서버에 데이터를 요청하는 Query Language

Specification, not Implementation

꾸준히 성장하고 있습니다



REST API

각 use case에 대해 매 번 Endpoint를 정의

불필요한 정보까지 받아옴

서버가 클라이언트에 의존하게 될 가능성

GraphQL

클라이언트가 필요한 use case를 직접 명시

필요한 정보만 받아옴

서버는 Schema만 정의. 클라이언트와의 의존도 낮음

Type 시스템을 통해서 쿼리 검증 가능

```
onClick { demo() }
```

Basic queries

```
query {
  allUsers {
    name
    email
  }
}
```

Nested Fields

```
query {
  allUsers {
    id
    name
    posts {
      title
      text
    }
    likedPosts {
      title
      text
    }
  }
}
```

Query with arguments

```
query {
  User(id: "cixnekqnu2ify0134ekw4pox8") {
    name
    posts {
      title
      text
    }
    likedPosts {
      title
      text
    }
  }
}
```

Query with variables

```
query Posts($id: ID, $limit: Int) {
  User(id: $id) {
    name
    posts(first: $limit) {
      title
      text
    }
    likedPosts {
      title
      text
    }
  }
}
```

```
{
  "id": "cixnekqnu2ify0134ekw4pox8",
  "limit": 3
}
```

Type Defintion

```
type User {
  id: ID!
  name: String
  email: String
  posts: [Post!]!
  likedPosts: [Post!]
}

type Post {
  title: String
  text: String
}
```

Expose Queries

```
type Query {
  allUsers: [User!]!
  User(email: String, id: ID): User
}
```

```
query {
  allUsers {
    name
    email
  }
}
```

```
query {
  allUsers {
    id
    name
    posts {
      title
      text
    }
}
```

Introspection Queries

서버가 지원하는 스키마 정보를 물어볼 수 있는 Query

1. IDE에서 타입 추론 가능
2. 서버 문서나 코드를 보지 않고도 스키마 구조를 알 수 있음

Query

```
{
  __type(name: "User") {
    name
    kind
  }
}
```

Result

```
{
  "data": {
    "__type": {
      "name": "User",
      "kind": "OBJECT"
    }
  }
}
```

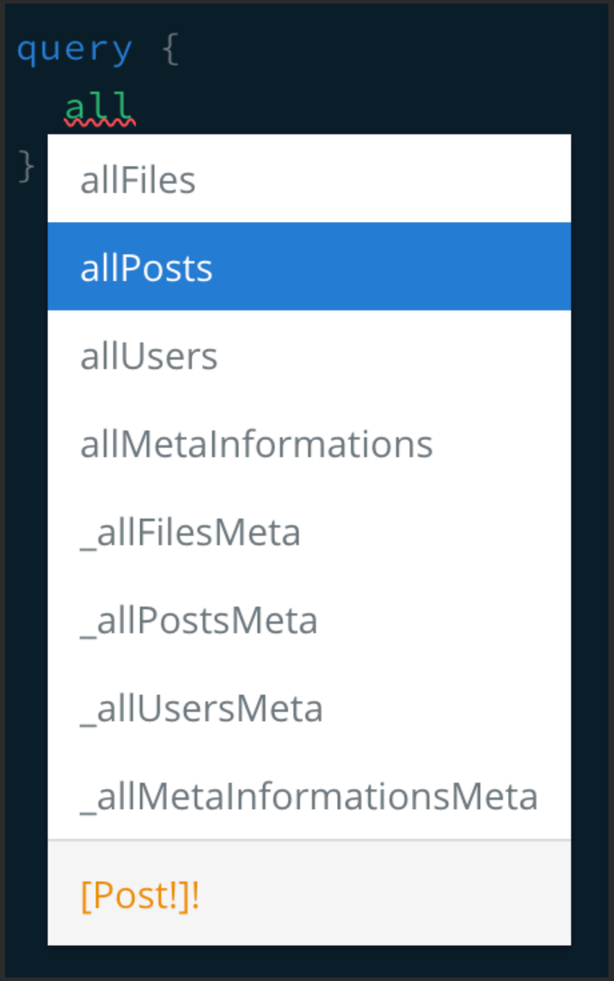
Query

```
{
  __type(name: "User") {
    name
    fields {
      name
      type {
        name
        kind
        ofType {
          name
          kind
        }
      }
    }
  }
}
```

Result

```
{
  "data": {
    "__type": {
      "name": "User",
      "fields": [
        {
          "name": "id",
          "type": {
            "name": null,
            "kind": "NON_NULL",
            "ofType": {
              "name": "ID",
              "kind": "SCALAR"
            }
          }
        },
        {
          "name": "friends",
          "type": {
            "name": null,
            "kind": "LIST",
            "ofType": {
              "name": "Character",
              "kind": "INTERFACE"
            }
          }
        }
      ]
    }
  }
}
```

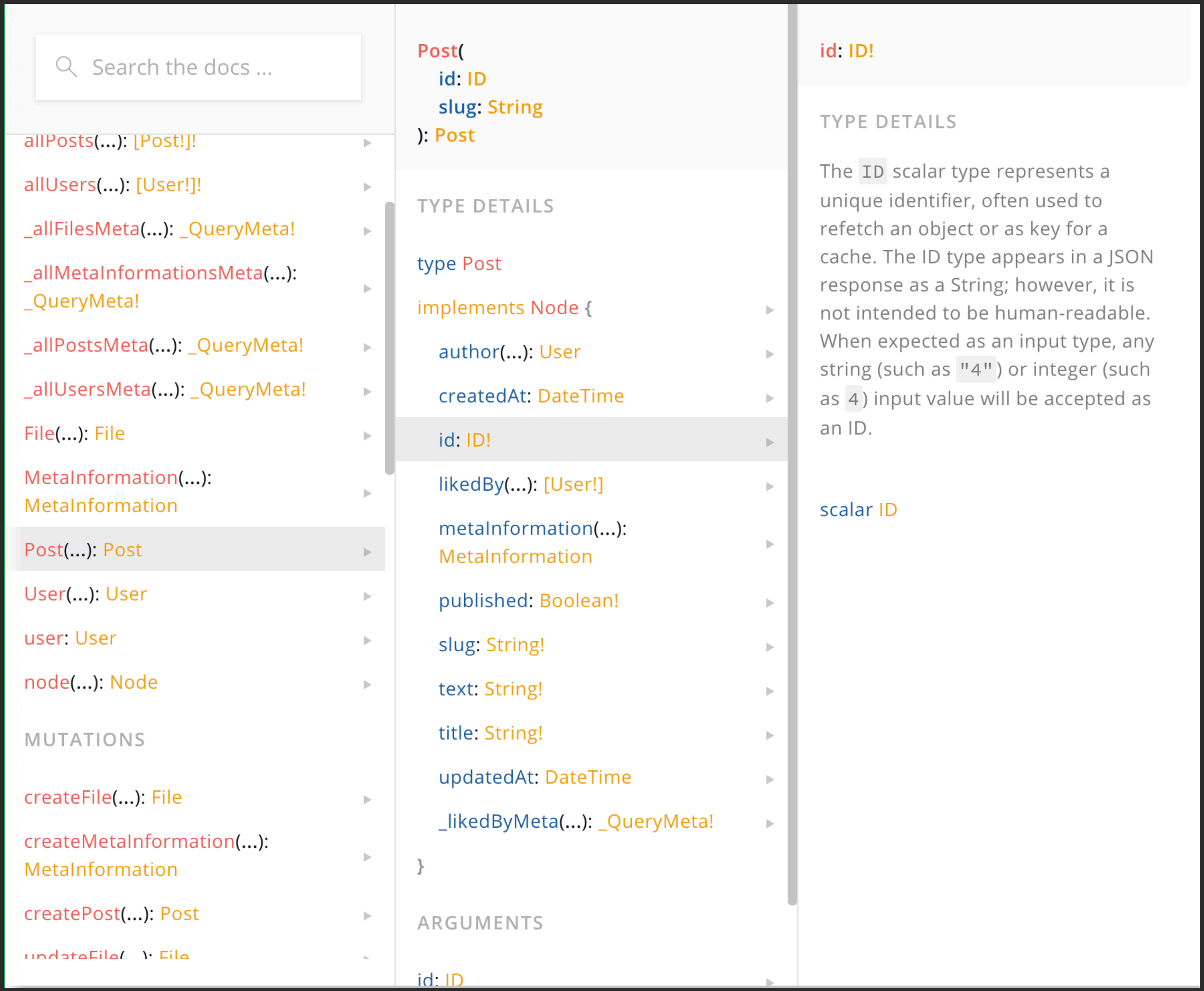
Auto completion



```
query {
  all
}
```

allFiles
allPosts
allUsers
allMetaInformations
_allFilesMeta
_allPostsMeta
_allUsersMeta
_allMetaInformationsMeta
[Post!]!

Generating documents



Search the docs ...

allPosts(...): [Post!]!
allUsers(...): [User!]!
_allFilesMeta(...): _QueryMeta!
_allMetaInformationsMeta(...): _QueryMeta!
_allPostsMeta(...): _QueryMeta!
_allUsersMeta(...): _QueryMeta!
File(...): File
MetaInformation(...): MetaInformation
Post(...): Post
User(...): User
user: User
node(...): Node

MUTATIONS

createFile(...): File
createMetaInformation(...): MetaInformation
createPost(...): Post
updateFile(...): File

```
Post(
  id: ID
  slug: String
): Post
```

TYPE DETAILS

```
type Post
implements Node {
  author(...): User
  createdAt: DateTime
  id: ID!
  likedBy(...): [User!]
  metaInformation(...): MetaInformation
  published: Boolean!
  slug: String!
  text: String!
  title: String!
  updatedAt: DateTime
  _likedByMeta(...): _QueryMeta!
}
```

ARGUMENTS

id: ID

id: ID!

TYPE DETAILS

The `ID` scalar type represents a unique identifier, often used to refetch an object or as key for a cache. The ID type appears in a JSON response as a String; however, it is not intended to be human-readable. When expected as an input type, any string (such as `"4"`) or integer (such as `4`) input value will be accepted as an ID.

scalar ID

Resources

Official Documentation

Apollo GraphQL

GraphQL Playground

Implementation

Module Hierarchy

:server:data

DB와 상호작용하는 모듈.

:server:graphql

GraphQL 구현. data 모듈의 데이터로 Schema를 생성하는 역할

:server:api

AWS Lambda와 상호작용하는 모듈

```kotlin
return KGraphQL.schema {
    // Configuration for this getSchema
    configure {
        useDefaultPrettyPrinter = true
    }

    // List of supported types
    type<User>()
    type<InstantTask>()
    enum<TaskType>()

    // Custom scalar types
    longScalar<LocalDateTime> {
        serialize = LocalDateTimeConverter.converter
        deserialize = LocalDateTimeConverter.inverter
    }

    // Available queries
    query("user") {
        suspendResolver { id: String ->
            repository.getUserById(id)
        }
    }

    query("users") {
        suspendResolver { -> repository.getUsers() }
    }

    query("task") {
        suspendResolver { id: String ->
            repository.getTaskById(id)
```

```kotlin
return KGraphQL.schema {
    // Configuration for this getSchema
    configure {
        useDefaultPrettyPrinter = true
    }

    // List of supported types
    type<User>()
    type<InstantTask>()
    enum<TaskType>()

    // Custom scalar types
    longScalar<LocalDateTime> {
        serialize = LocalDateTimeConverter.converter
        deserialize = LocalDateTimeConverter.inverter
    }

    // Available queries
    query("user") {
        suspendResolver { id: String →
            repository.getUserById(id)
        }
    }

    query("users") {
        suspendResolver { → repository.getUsers() }
    }

    query("task") {
        suspendResolver { id: String →
            repository.getTaskById(id)
```

```kotlin
data class User(
    val id: String,
    val username: String,
    val tasks: List<Task> = listOf()
)

abstract class Task(
    open val type: TaskType,
    open val id: String,
    open val name: String
)

enum class TaskType {
    INSTANT
}

data class InstantTask(
    override val type: TaskType = INSTANT,
    override val id: String,
    override val name: String,
    val time: LocalDateTime
) : Task(type, id, name)
```

```kotlin
return KGraphQL.schema {
    // Configuration for this getSchema
    configure {
        useDefaultPrettyPrinter = true
    }

    // List of supported types
    type<User>()
    type<InstantTask>()
    enum<TaskType>()

    // Custom scalar types
    longScalar<LocalDateTime> {
        serialize = LocalDateTimeConverter.converter
        deserialize = LocalDateTimeConverter.inverter
    }

    // Available queries
    query("user") {
        suspendResolver { id: String →
            repository.getUserById(id)
        }
    }

    query("users") {
        suspendResolver { → repository.getUsers() }
    }

    query("task") {
        suspendResolver { id: String →
            repository.getTaskById(id)
```

```kotlin
interface Converter<T, U> {
    val converter: (T) → U
    val inverter: (U) → T
}
```

```
    // Custom scalar types
    longScalar<LocalDateTime> {
        serialize = LocalDateTimeConverter.converter
        deserialize = LocalDateTimeConverter.inverter
    }

    // Available queries
    query("user") {
        suspendResolver { id: String →
            repository.getUserById(id)
        }
    }

    query("users") {
        suspendResolver { → repository.getUsers() }
    }

    query("task") {
        suspendResolver { id: String →
            repository.getTaskById(id)
        }
    }

    query("tasks") {
        suspendResolver { → repository.getTasks() }
    }
}
```

```
interface Repository {
    suspend fun getUsers(): List<User>

    suspend fun getUserById(id: String): User?

    suspend fun getTasks(): List<Task>

    suspend fun getTaskById(id: String): Task?
}
```

```
type Query {
  users: [User!]!
  user(id: ID): User
  tasks: [Task!]!
  task(id: ID): Task
}
```

```kotlin
class PostRequestHandler :
    RequestHandler<APIGatewayProxyRequestEvent, APIGatewayProxyResponseEvent> {

    override fun handleRequest(
        input: APIGatewayProxyRequestEvent?,
        context: Context?
    ): APIGatewayProxyResponseEvent {
        val body = input?.body?.fromJson<PostRequestParams>()
            ?: return error(422, "body is missing in POST request!")

        val query = body.query
        val variables = body.variables.toJson()

        return body(schema.execute(query, variables))
    }
}
```

```kotlin
class PostRequestHandler :
    RequestHandler<APIGatewayProxyRequestEvent, APIGatewayProxyResponseEvent> {

    override fun handleRequest(
        input: APIGatewayProxyRequestEvent?,
        context: Context?
    ): APIGatewayProxyResponseEvent {
        val body = input?.body?.fromJson<PostRequestParams>()
            ?: return error(422, "body is missing in POST request!")

        val query = body.query
        val variables = body.variables.toJson()

        return body(schema.execute(query, variables))
    }
}
```

```kotlin
class PostRequestHandler :
    RequestHandler<APIGatewayProxyRequestEvent, APIGatewayProxyResponseEvent> {

    override fun handleRequest(
        input: APIGatewayProxyRequestEvent?,
        context: Context?
    ): APIGatewayProxyResponseEvent {
        val body = input?.body?.fromJson<PostRequestParams>()
            ?: return error(422, "body is missing in POST request!")

        val query = body.query
        val variables = body.variables.toJson()

        return body(schema.execute(query, variables))
    }
}
```

```yaml
GraphQLPost:
  Type: "AWS::Serverless::Function"
  Properties:
    Handler: "com.lovelessgeek.housemanager.api.handler.PostRequestHandler::handleRequest"
    CodeUri: "./build/libs/api-all.jar"
    Events:
      IndexApi:
        Type: "Api"
        Properties:
          Path: "/v1/graphql"
          Method: "post"
    Runtime: "java8"
    Timeout: 40
    MemorySize: 256
```

Test / Deployment

Run local

```
./gradlew :server:api:runLocalStartApi
```

Deploy

```
./gradlew :server:api:deploySamApp
```

https://github.com/importre/aws-sam-gradle-plugin

Thank You