Thinking in Graphs

- https://graphql.org/learn/thinking-in-graphs/

It's Graphs All the Way Down

> GraphQL을 사용하면 비즈니스 도메인을 그래프로 모델링할 수 있습니다.[1)]

그래프는 우리의 자연스러운 정신 모델과 기본 프로세스에 대한 구두 설명과 유사하기 때문에 많은 실제 현상을 모델링하는 강력한 도구입니다. GraphQL을 사용하면 스키마를 정의하여 비즈니스 도메인을 그래프로 모델링할 수 있습니다. 스키마 내에서 다양한 유형의 노드와 노드가 서로 연결/관련되는 방식을 정의합니다. 클라이언트에서 이것은 객체 지향 프로그래밍과 유사한 패턴을 생성합니다: 다른 유형을 참조하는 유형. 서버에서 GraphQL은 인터페이스만 정의하므로 모든 백엔드(신규 또는 레거시!)에서 자유롭게 사용할 수 있습니다.[2)]

Shared Language

> 사물에 이름을 지정하는 것은 어렵지만 직관적인 API를 구축하는 데 중요한 부분입니다.[3)]

GraphQL 스키마를 팀과 사용자를 위한 표현적인 공유 언어로 생각하십시오. 좋은 스키마를 작성하려면 비즈니스를 설명하는 데 사용하는 일상적인 언어를 검토하십시오. 예를 들어 일반 영어로 이메일 앱을 설명하려고 합니다.[4)]

- 사용자는 여러 이메일 계정을 가질 수 있다.[5)]
- 각 이메일 계정에는 주소, 받은 편지함, 임시 보관함, 삭제된 항목 및 보낸 항목이 있다.[6)]
- 각 이메일에는 발신자, 수신 날짜, 제목 및 본문이 있다.[7)]
- 사용자는 수신자 주소 없이 이메일을 보낼 수 없다.[8)]

사물에 이름을 지정하는 것은 어렵지만 직관적인 API를 구축하는 데 중요한 부분이므로 시간을 내어 문제 도메인과 사용자에게 의미 있는 것이 무엇인지 신중하게 생각하십시오. GraphQL 스키마에서 노드 및 관계에 대해 직관적이고 내구성 있는 이름을 선택해야 하기 때문에 팀은 이러한 비즈니스 도메인 규칙에 대한 공유된 이해와 합의를 개발해야 합니다. 실행하려는 쿼리 중 일부를 상상해 보십시오.[9)]

내 모든 계정의 받은편지함에서 읽지 않은 이메일 수 가져오기[10)]

```
{
  accounts {
    inbox {
      unreadEmailCount
    }
  }
}
```

기본 계정의 처음 20개 초안에 대한 “ 미리보기 정보 ” 가져오기[11]

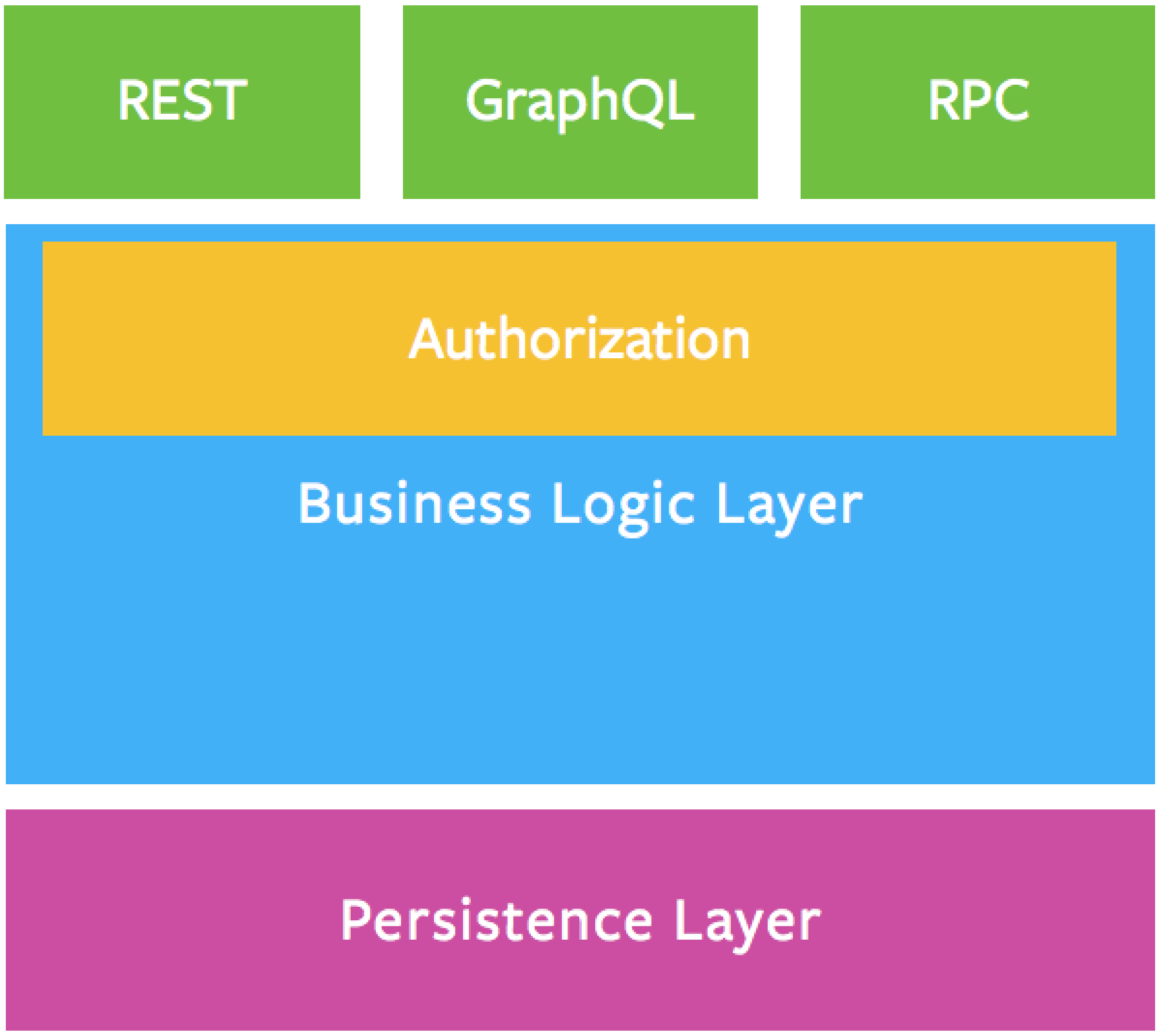
```
{
  mainAccount {
    drafts(first: 20) {
      ...previewInfo
    }
  }
}

fragment previewInfo on Email {
  subject
  bodyPreviewSentence
}
```

Business Logic Layer

> 비즈니스 논리 계층은 비즈니스 도메인 규칙을 적용하기 위한 단일 정보 소스 역할을 해야 합니다.[12]

실제 비즈니스 로직을 어디에서 정의해야 합니까? 어디에서 유효성 검사 및 권한 부여 확인을 수행해야 합니까? 답: 전용 비즈니스 로직 계층 내부. 비즈니스 논리 계층은 비즈니스 도메인 규칙을 적용하기 위한 단일 정보 소스 역할을 해야 합니다.[13]



위의 다이어그램에서 시스템의 모든 진입점(REST, GraphQL 및 RPC)은 동일한 유효성 검사, 권한 부여 및 오류 처리 규칙으로 처리됩니다.[14)]

Working with Legacy Data

> 레거시 데이터베이스 스키마를 미러링하는 것보다 클라이언트가 데이터를 사용하는 방식을 설명하는 GraphQL 스키마를 구축하는 것을 선호합니다.[15)]

때때로 클라이언트가 데이터를 사용하는 방식을 완벽하게 반영하지 않는 레거시 데이터 소스로 작업하는 자신을 발견하게 될 것입니다. 이러한 경우 레거시 데이터베이스 스키마를 미러링하는 것보다 클라이언트가 데이터를 사용하는 방식을 설명하는 GraphQL 스키마를 구축하는 것을 선호합니다.[16)]

" 무엇 " 이 아닌 " 어떻게 " 를 표현하도록 GraphQL 스키마를 구축하십시오. 그런 다음 이전 클라이언트와의 인터페이스를 중단하지 않고 구현 세부 정보를 개선할 수 있습니다.[17)]

One Step at a time

> 더 자주 확인 및 피드백 받기[18]

한 번에 전체 비즈니스 도메인을 모델링하지 마십시오. 대신 한 번에 하나의 시나리오에 필요한 스키마 부분만 작성하십시오. 스키마를 점진적으로 확장하면 올바른 솔루션을 구축할 수 있도록 더 자주 검증 및 피드백을 얻을 수 있습니다.[19]

Continue Reading

- Serving over HTTP

관련 문서

Plugin Backlinks: 아무 것도 없습니다.

1)

With GraphQL, you model your business domain as a graph

2)

Graphs are powerful tools for modeling many real-world phenomena because they resemble our natural mental models and verbal descriptions of the underlying process. With GraphQL, you model your business domain as a graph by defining a schema; within your schema, you define different types of nodes and how they connect/relate to one another. On the client, this creates a pattern similar to Object-Oriented Programming: types that reference other types. On the server, since GraphQL only defines the interface, you have the freedom to use it with any backend (new or legacy!).

3)

Naming things is a hard but important part of building intuitive APIs

4)

GraphQL 스키마를 팀과 사용자를 위한 표현적인 공유 언어로 생각하십시오. 좋은 스키마를 작성하려면 비즈니스를 설명하는 데 사용하는 일상적인 언어를 검토하십시오. 예를 들어 일반 영어로 이메일 앱을 설명하려고 합니다.

5)

A user can have multiple email accounts

6)

Each email account has an address, inbox, drafts, deleted items, and sent items

7)

Each email has a sender, receive date, subject, and body

8)

Users cannot send an email without a recipient address

9)

Naming things is a hard but important part of building intuitive APIs, so take time to carefully think about what makes sense for your problem domain and users. Your team should develop a shared understanding and consensus of these business domain rules because you will need to choose intuitive, durable names for nodes and relationships in the GraphQL schema. Try to imagine some of the queries you will want to execute:

10)

Fetch the number of unread emails in my inbox for all my accounts

11) Fetch the “preview info” for the first 20 drafts in the main account

12) Your business logic layer should act as the single source of truth for enforcing business domain rules

13) Where should you define the actual business logic? Where should you perform validation and authorization checks? The answer: inside a dedicated business logic layer. Your business logic layer should act as the single source of truth for enforcing business domain rules.

14) In the diagram above, all entry points (REST, GraphQL, and RPC) into the system will be processed with the same validation, authorization, and error handling rules.

15) Prefer building a GraphQL schema that describes how clients use the data, rather than mirroring the legacy database schema.

16) Sometimes, you will find yourself working with legacy data sources that do not perfectly reflect how clients consume the data. In these cases, prefer building a GraphQL schema that describes how clients use the data, rather than mirroring the legacy database schema.

17) Build your GraphQL schema to express “how” rather than “what”. Then you can improve your implementation details without breaking the interface with older clients.

18) Get validation and feedback more frequently

19) Don't try to model your entire business domain in one sitting. Rather, build only the part of the schema that you need for one scenario at a time. By gradually expanding the schema, you will get validation and feedback more frequently to steer you toward building the right solution.

From:
http://jace.link/ - **Various Ways**

Permanent link:
http://jace.link/open/thinking-in-graphs

Last update: **2022/09/02 06:56**