>_ abstract
IT Experience. Experience it.

Introduzione a GraphQL

Un’alternativa al REST

06/12/2024

Vincenzo Mattarella

>_ abstract
IT Experience. Experience it.

Agenda

01 Cos’è GraphQL

02 Perché GraphQL

03 Concetti fondamentali

04 Live Demo

05 Best Practices

06 Considerazioni finali

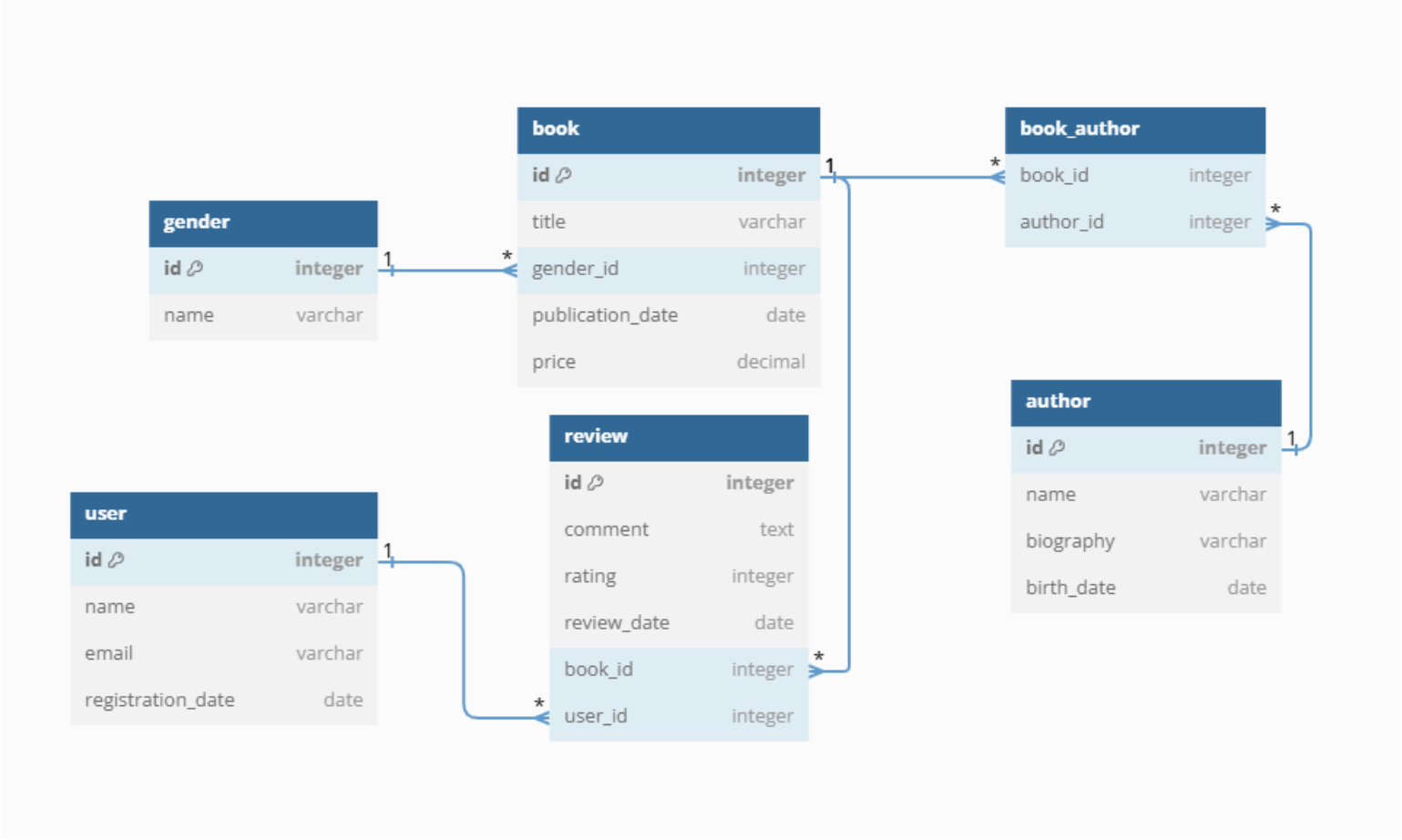
07 Q&A

>_ abstract
IT Experience. Experience it.

Il nostro case study

Gestione Libreria - ER

- Gestione libri e autori
- Recensioni utenti
- Categorizzazione per genere



>_ abstract
IT Experience. Experience it.

Cos’è GraphQL

Cos’è GraphQL

> Un linguaggio di query per le tue API

- Sviluppato da Facebook nel 2012
- Open source dal 2015
- Specification, non implementation
- Singolo endpoint intelligente
- Schema fortemente tipizzato

>_ abstract
IT Experience. Experience it.

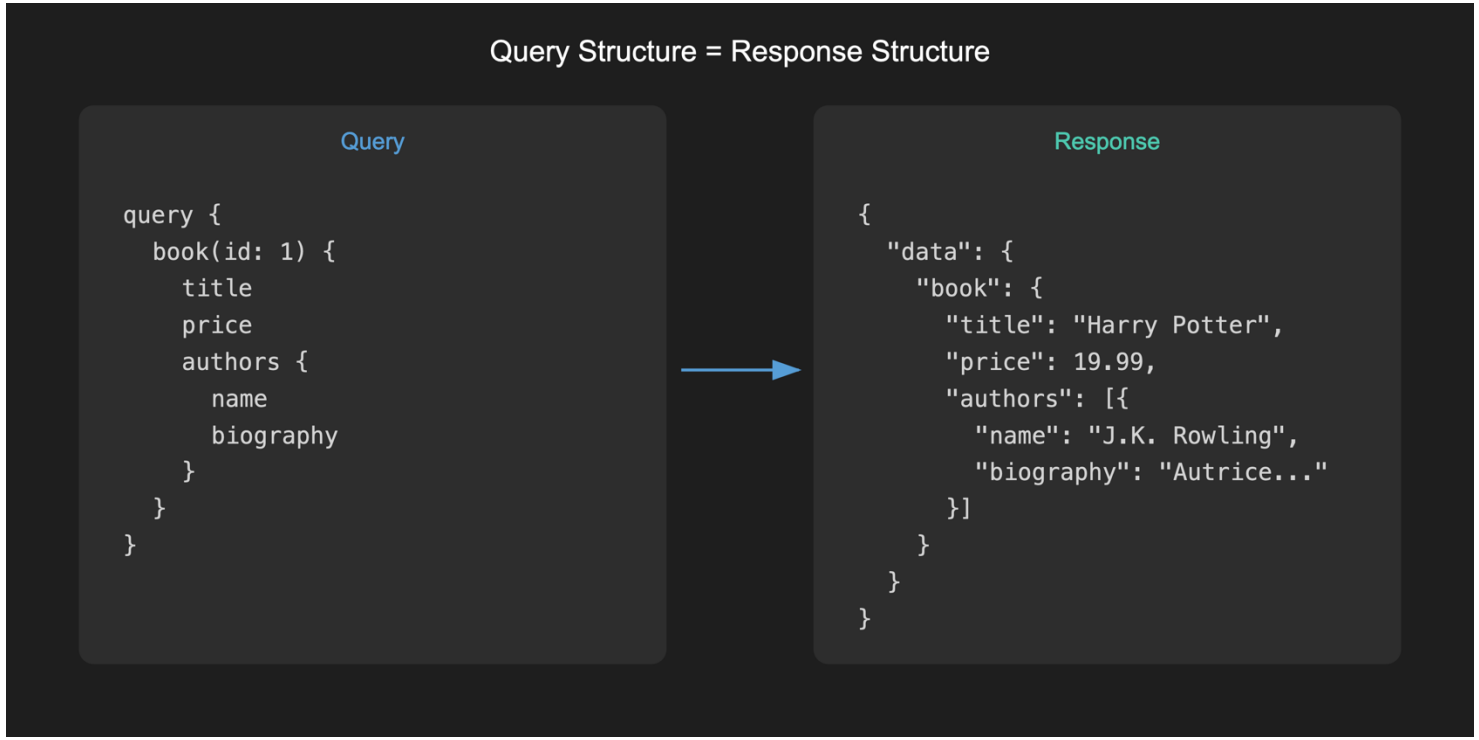
Vantaggi di GraphQL

- Over/under-fetching: elimina il problema della API che resituiscono troppi o troppo pochi dati
- Chiamate multiple: una singola query può recuperare dati da risorse differenti
- Contract-first: lo schema definisce un contratto chiaro tra frontend e backend
- Type Safety: Sistema di tipi forte che previene errori a runtime
- Developer experience: tooling eccellente e documentazione auto-generata

>_ abstract
IT Experience. Experience it.

Come funziona

- Il client specifica la struttura dei dati che vuole ricevere
- Il server risponde con JSON che rispecchia esattamente quella struttura
- Un singolo endpoint gestisce tutte le operazioni dello schema (/graphql)
- Tre tipi di operazioni principali: Query, Mutation e Subscription



>_ abstract
IT Experience. Experience it.

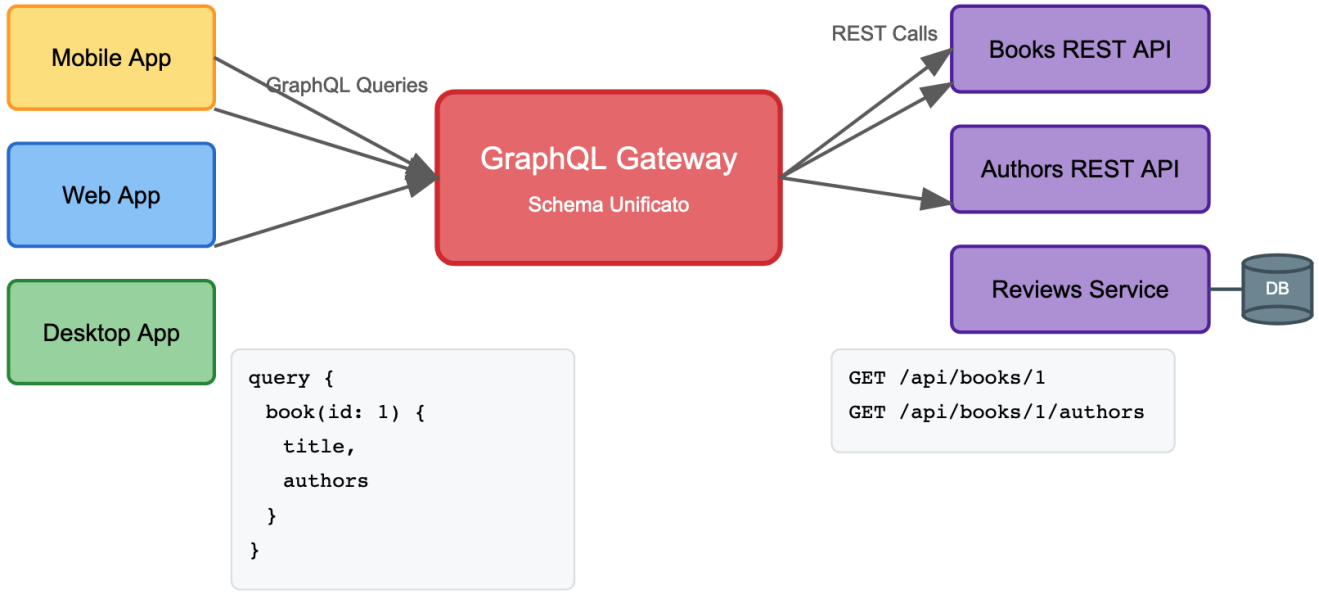
Perché GraphQL

02

>_ abstract
IT Experience. Experience it.

Gateway per servizi – BFF per diversi clients

- GraphQL può fare da tramite nell'aggregazione dei dati di più (micro)servizi
- Può essere utilizzato da client diversi (Web App, Mobile ecc) con necessità diverse e risposte a misura del chiamante
- Flessibilità per il client prioritaria. A volte limitare il contenuto dei payload può essere necessario per le prestazioni
- Dati fortemente relazionali



>_ abstract
IT Experience. Experience it.

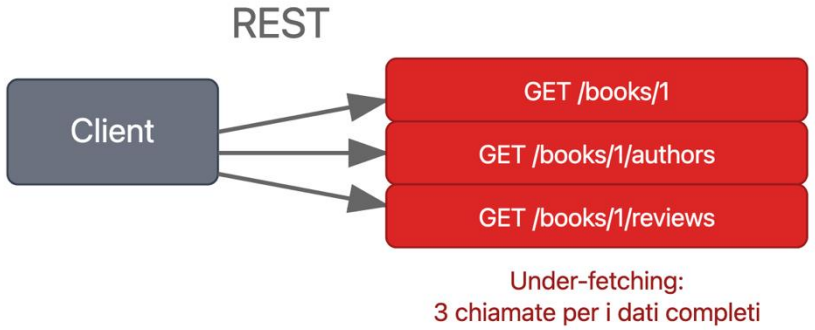
Problemi con REST

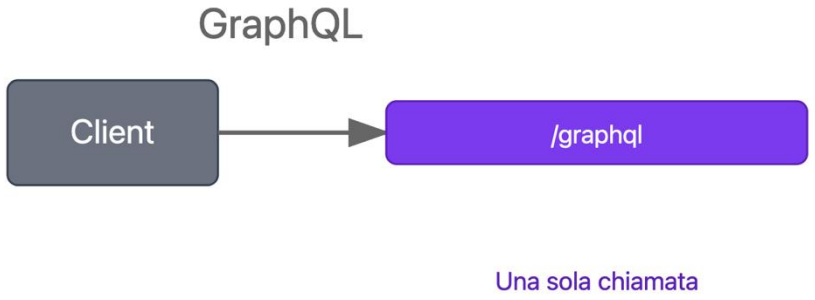
UNDER-FETCHING

Una chiamata in una pagina di preview può avere bisogno di campi di dettaglio, per cui devo fare più chiamate:
/books/{id}
/books/{id}/authors
/books/{id}/reviews
ecc

OVER-FETCHING

Si può decidere di popolare l'entità con tutte le relazioni che possiede, ma si ritornerebbero molti campi inutili che appesantiscono il trasporto





>_ abstract
IT Experience. Experience it.

Problemi con REST

```
query Book {
    book(id: "1") {
        title
        authors {
            name
        }
        reviews {
            rating
        }
    }
}
```

```
{
    "data": {
        "book": {
            "title": "Harry Potter e la Pietra Filosofale",
            "authors": [
                {
                    "name": "J.K. Rowling"
                }
            ],
            "reviews": [
                {
                    "rating": 5
                }
            ]
        }
    }
}
```

>_ abstract
IT Experience. Experience it.

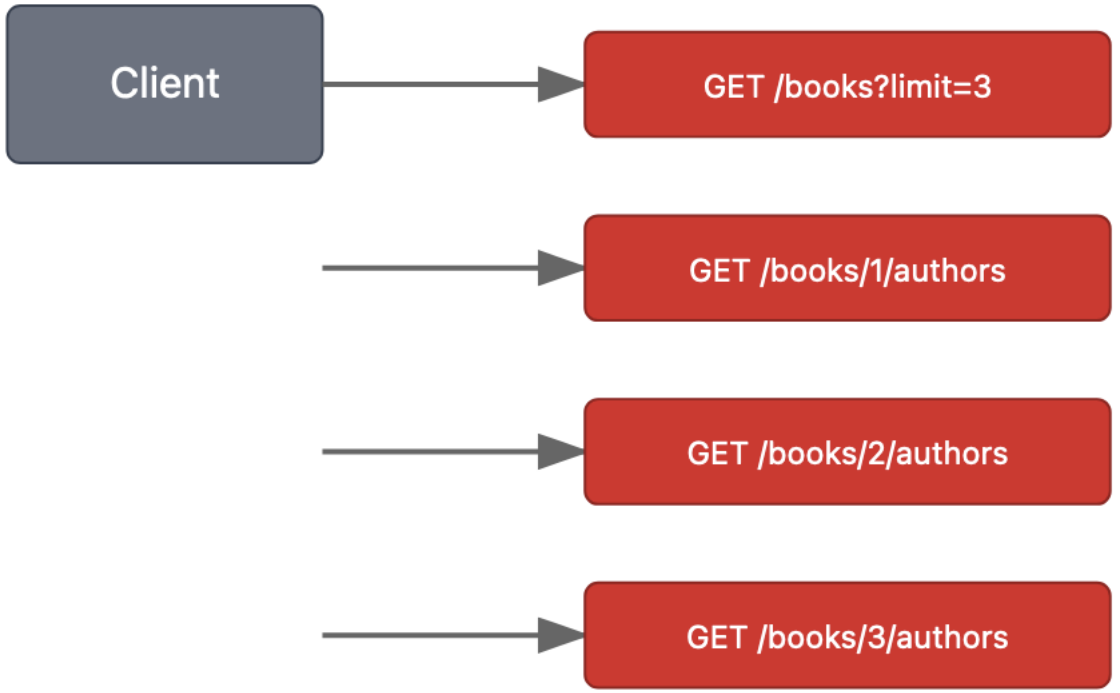
Problema di N+1

A meno che la risposta REST non sia già completamente popolata con tutti i campi di relazione (alto rischio di over-fetching), l'utilizzo di api REST crea alcune situazioni poco ottimizzate.

Immaginiamo di avere a disposizione gli endpoint:

/books?limit=3 e /books/{id}/authors

Allora avremo necessità di effettuare 4 chiamate per ottenere tutti i dati necessari



>_ abstract
IT Experience. Experience it.

Versionamento delle api

REST

- In REST gestire versione differenti porta alla creazione di endpoint multipli (*/v1/books*, */v2/books* ecc)
- Difficoltà nel capire chi utilizza endpoints deprecati
- Difficoltà nell'eliminare endpoint obsoleti e deprecati da tempo
- Necessità di manutenere più versioni quando ci sono dei cambiamenti importanti a basso livello

GraphQL

- Deprecazione graduale delle funzionalità
- Versionamento integrato a livello di campo
- Schema cresce ed evolve insieme alle necessità

```
type Book {
    id: ID!
    title: String!
    genderName: String @deprecated(reason: "Use `gender` property.")
    gender: Gender
    authors: [Author]
    reviews: [Review]
}
```

>_ abstract
IT Experience. Experience it.

Sintassi e fondamenti

03

>_ abstract
IT Experience. Experience it.

Intro – Schema e Operations

- Lo schema GraphQL è un insieme di types e fields che descrivono tutto quello che noi possiamo fare con I dati sul server GraphQL.
  E' scritto in un linguaggio che si chiama *SDL* (Schema Definition Language)
- Un client può interrogazione il server GraphQL, modificarne I dati o sottoscriversi a degli eventi
- Tutte e 3 le operazioni hanno un corrispettivo nello *SDL* che è appunto: *Query*, *Mutation* e *Subscription*

```
type Query {
    books: [Book]
    book(id: ID!): Book
}

type Mutation {
    addReviewAndGetBook(inputReview: AddReviewInput!): AddReviewResponse!
}

type Subscription {
    bookCreated: Book
}
```

>_ abstract
IT Experience. Experience it.

Tipi ed interfaces

- Con *type* dichiariamo le proprietà che di fatto compongono un oggetto con cui possiamo interagire e di cui possiamo chiedere informazioni
- Il punto esclamativo (!) indica campi non nullabili
- GraphQL supporta array tipizzati con controllo di nullabilità sia a livello di struttura array che a livello di contenuto
- Le *interface* definiscono un set di proprietà
- I *types* possono implementare un'*interface*, permettendo riuso e polimorfismo

```
# Object Type
type Libro {
  id: ID!
  titolo: String! #Not null
  autori: [Autore!]! #Not null array of not null value
}

# Interface
interface Prodotto {
  id: ID!
  nome: String!
  prezzo: Float!
}

# Type implements interface
type Libro implements Prodotto {
  id: ID!
  nome: String!
  prezzo: Float!
  autore: String
}
```

>_ abstract
IT Experience. Experience it.

Fields ed arguments

- Le query GraphQL sono dichiarative: specifichi esattamente quali dati vuoi ottenere
- Richiedi solo I *fields* di cui hai bisogno (titolo, prezzo)
- Gli *arguments* sono I parametri di input
- *Arguments singoli*: filtra I risultati con parametri specifici (es. Id del libro)
- *Arguments multipli*: combina più parametri per query avanzate (es. limite di risultati, filtro per genere)
- *Argument* singoli ma a loro volta tipizzati

```
query {
  libro {
    titolo # simple field
    prezzo # simple field
  }

  # Arguments
  libro(id: "123") {
    titolo
  }

  # Arguments multipli
  libri(limit: 10, genere: "FANTASY") {
    titolo
  }
}
```

>_ abstract
IT Experience. Experience it.

Aliases e fragments

- Gli *aliases* permettono di rinominare I campi nella response e interrogare lo stesso tipo più volte nella stessa query (altrimenti otterremmo errore!)
- I *fragments* sono blocchi riutilizzabili di campi che riducono la duplicazione del codice
- Un *fragment* è sempre definito per uno specifico type (“on Libro”)
- I *fragments* possono includere relazioni annidate (autori, recensioni)
- La sintassi ...NomeFragment applica tutti I campi definiti nel *fragment*

```
query {
  # Aliases "need libro twice"
  bestSeller: libro(id: "123") {
    titolo
  }
  novita: libro(id: "456") {
    titolo
  }
  ...LibroDetails # Fragment use
}

# Fragment definition
fragment LibroDetails on Libro {
  titolo
  autori {
    nome
  }
  recensioni {
    rating
  }
}
```

>_ abstract
IT Experience. Experience it.

Variabili e direttive

- Le variabili GraphQL rendono le query riutilizzabili e parametrizzabili
- Le direttive (*@include*, *@skip*) permettono di controllare dinamicamente quali campi includere
- Le variabili sono fortemente tipizzate (ID, Boolean) per garantire type safety
- Le variabili possono essere passate separatamente in un oggetto JSON, rendendo le query più pulite
- Perfetto per costruire query dinamiche lato client mantenendo la sicurezza sui tipi

```
# Query con variabili
query GetLibro($id: ID!, $includiRecensioni: Boolean!) {
  libro(id: $id) {
    titolo
    # Directive @include
    recensioni @include(if: $includiRecensioni) {
      rating
    }
    # Directive @skip
    prezzo @skip(if: $inOfferta)
  }
}

# Variables in JSON
{
  "id": "123",
  "includiRecensioni": true
}
```

>_ abstract
IT Experience. Experience it.

Scalars, scalars custom ed enums

- GraphQL fornisce tipi scalar built-in (Int, Float, String, Boolean e ID) per I dati primitivi
- E' possible definire Custom Scalars (Date, Email) per validazione o serializzazione personalizzata
- Gli Enum garantiscono che I valori siano limitati a un insieme predefinito di opzioni
- La type safety si estende fino all'uso degli enum sulle query, prevenendo errori a runtime

```
# Scalar types
Int
Float
String
# Custom Scalar
scalar Date
scalar Email

# Enum definition
enum GenereLibro {
  FANTASY
  GIALLO
  FANTASCIENZA
}

# Enum usage
query {
  libri(genere: FANTASY) {
    titolo
  }
}
```

>_ abstract
IT Experience. Experience it.

Union ed Input types

- Union types permettono di combinare più tipi in un unico risultato
- I fragments “in linea” consentono di gestire specificamente ogni tipo nell’union
- Gli input types (già nominati relativamente agli argomenti) definiscono la struttura dei dati in ingresso per mutations e queries
- Gli input types sono object-like ma possono contenere solo scalar, enum o altri input types

```
# Union type
union SearchResult = Libro | Autore | Recensione

# Query con union
query {
  search(term: "fantasy") {
    ... on Libro {
      titolo
    }
    ... on Autore {
      nome
    }
  }
}

# Input type
input LibroInput {
  titolo: String!
  autoreId: ID!
  genere: GenereLibro
}
```

>_ abstract
IT Experience. Experience it.

Live Demo

04

Demo Setup

> Il codice utilizzato per la demo sarà visionabile su GitHub
>
> https://github.com/it-Abstract/graphql-intro

- Java, Spring Boot, JPA e TestContainers
- Schema libreria completo
- Database
- Collection postman

>_ abstract
IT Experience. Experience it.

Demo 1

Dettaglio Libro – Overfetch vs Underfetch

01 Di cosa abbiamo bisogno?

Abbiamo bisogno di disegnare un'interfaccia grafica che mostri il dettaglio di un libro mostrando anche nomi degli autori e una lista di commenti ricevuti delle reviews

02 Problemi con REST

- Se decidiamo di popolare di ufficio tutti i campi delle relazioni allora incorriamo nell'overfetching
- Se decidiamo di non aggiungerle di ufficio allora abbiamo bisogno di più chiamate per recuperare tutti I dati

03 Soluzione GraphQL

Otteniamo in risposta esattamente ciò di cui abbiamo bisogno ottimizzando numero di chiamate e dimensioni del payload della risposta

>_ abstract
IT Experience. Experience it.

Demo 2

Aggiunta di una review – Dato aggiornato

01 Di cosa abbiamo bisogno?

Vogliamo aggiungere la recensione di un libro ma allo stesso tempo Vedere aggiornarsi le informazioni che riguardano recensioni ma anche il libro stesso o le statistiche

02 Problemi con REST

- Bisognerebbe effettuare chiamate diverse sia nell'ottenimento delle informazioni che dopo il salvataggio (posso ottenere il dato aggiornato dopo la POST, ma non avrò tutte le informazioni sulle statistiche
- Ogni chiamata può fallire e quindi N gestioni diverse di errore
- Latenza
- Test di integrazione più complesso

03 Soluzione GraphQL

- Chiediamo immediatamente I dati aggiornati del libro (contestualmente all'interrogazione)
- Otteniamo le informazioni con un'unica risposta
- Scegliamo quali dati ricevere dalla risposta

>_ abstract
IT Experience. Experience it.

Best Practices

05

>_ abstract
IT Experience. Experience it.

Schema

- Naming conventions: PascalCase per I tipi, camelCase per proprietà e argomenti, SCREAM_SNAKE_CASE per enums.
- Non aggiungere verbi ai nomi delle query, aggiungerli alle mutations
- Nullability: A meno che non siano fortemente necessari per gli input o per le logiche favorire la nullability alla non-nullability
- Pagination patterns
- Error handling: enrichment del payload

>_ abstract
IT Experience. Experience it.

Performance

- DataLoader per il problema N+1 – prevenire query ridondanti nelle relazioni annidate
- Strategie di caching – ottimizzazione attraverso caching strategico a diversi livelli dell'API
- Query complexity – contenere la complessità delle query tenendo sempre conto che sia interamente consumabile da un client
- Rate limiting – limitare la frequenza con cui le risorse vengono utilizzate su più livelli
- Operazioni batch – raggruppare operazioni multiple ed eseguirle in blocco

>_ abstract
IT Experience. Experience it.

Sicurezza

- Query depth limiting – limitare la profondità delle query annidate
- Query complexity analysis – analizzare e limitare la complessità delle query
- Trusted documents – utilizzare dei trusted documents che risiedono sul server comunicando tramite id che riportano al document stesso
- Input validation – validazione rigorosa degli input
- Rate limiting - limitare la frequenza con cui le risorse vengono utilizzate su più livelli

>_ abstract
IT Experience. Experience it.

Considerazioni finali

06

>_ abstract
IT Experience. Experience it.

Quando usare GraphQL

API pubbliche o usate da client diversi (mobile vs web)

Dati fortemente relazionali

Aggregazione di microservizi

Flessibilità per il client prioritaria

Operazioni di upload/download

Forte utilizzo di caching

Dati non relazionali

Real time (meglio WebSocket)

>_ abstract
IT Experience. Experience it.

Extra - Considerazioni tecniche

Trade-offs e sfide

- Complessità iniziale di startup
- Curva di apprendimento
- Necessità di ottimizzare le query (N+1, caching ecc)
- Monitoring e debugging un pò più complesso
- Dimensione del payload delle risposte

>_ abstract
IT Experience. Experience it.

Extra - Strategie di migrazione

- Iniziare con feature isolate o con schema ridotto
- Approccio incrementale con pattern BFF
- Coesistenza iniziale con REST APIs esistenti
- Strategia su schema design
- Utilizzo di tools come: GraphQL Code Generator, DataLoader, Persisted Queries, APQ e monitoring (Apollo Studio, GraphQL Inspector)

>_ abstract
IT Experience. Experience it.

Q&A

07

>_ abstract
IT Experience. Experience it.

Riferimenti e link utili

- https://graphql.org/learn
- https://www.howtographql.com
- https://www.apollographql.com/tutorials
- https://docs.spring.io/spring-graphql/reference/federation.html
- https://github.com/Netflix/dgs-framework
- https://graphql.org/learn/best-practices

>_ abstract
IT Experience. Experience it.

Thank you