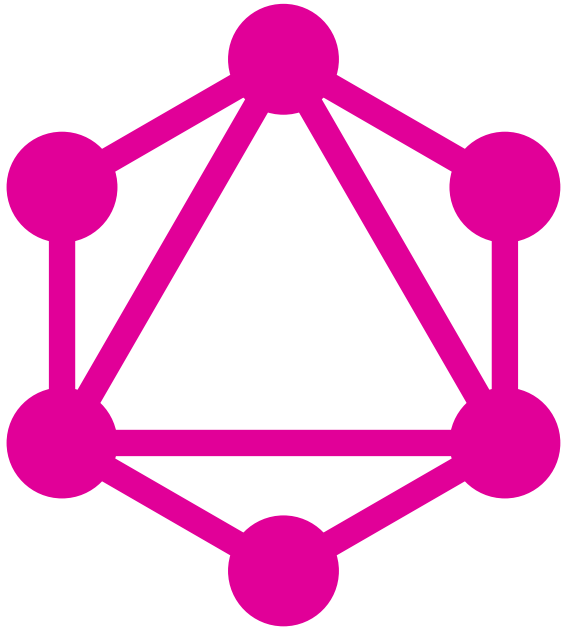
Đại học Quốc gia Thành phố Hồ Chí Minh
Trường Đại học Khoa học tự nhiên
Khoa Công nghệ thông tin
Môn: Phát triển ứng dụng cho thiết bị di động nâng cao



Research Project

GRAPHQL CLIENT IN FLUTTER

Nhóm: 17
Giảng viên hướng dẫn: ThS. Phạm Hoàng Hải

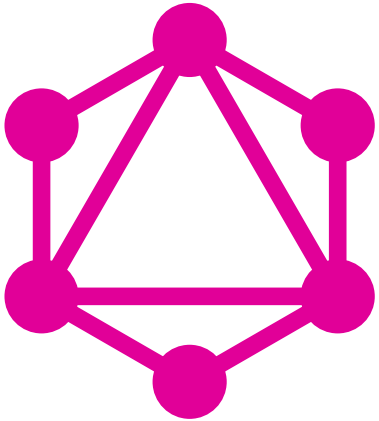
Group Members

Mã số sinh viên	Họ tên
20120392	Phạm Thụy Bích Truyền
20120454	Lê Công Đắt
20120470	Nguyễn Văn Hào

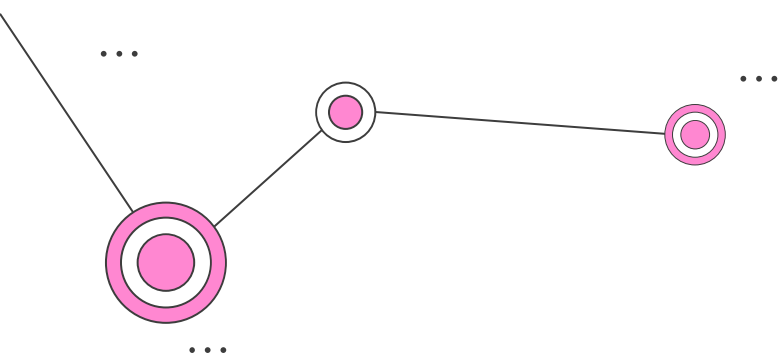
Table of Contents

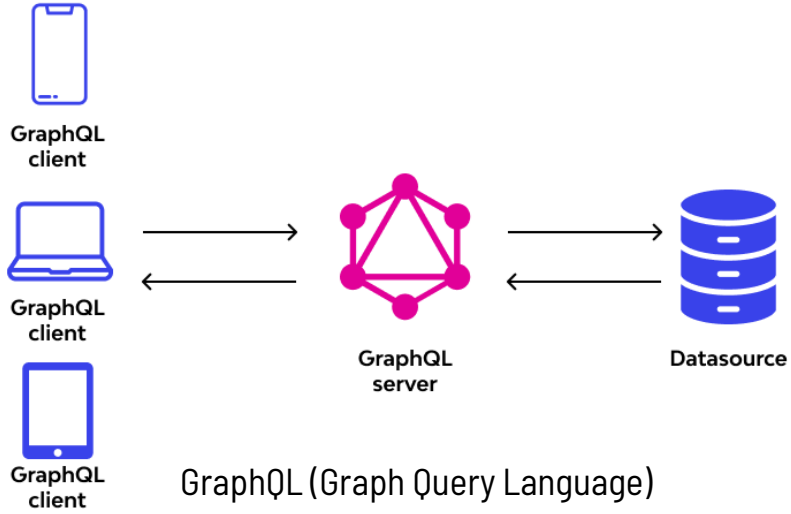
01

Introduction & Overview



What is GraphQL?

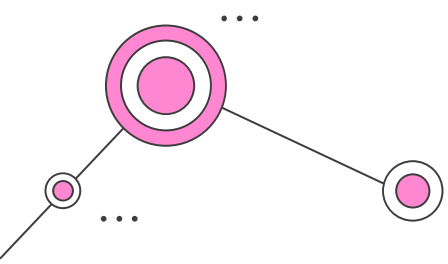




GraphQL (Graph Query Language)

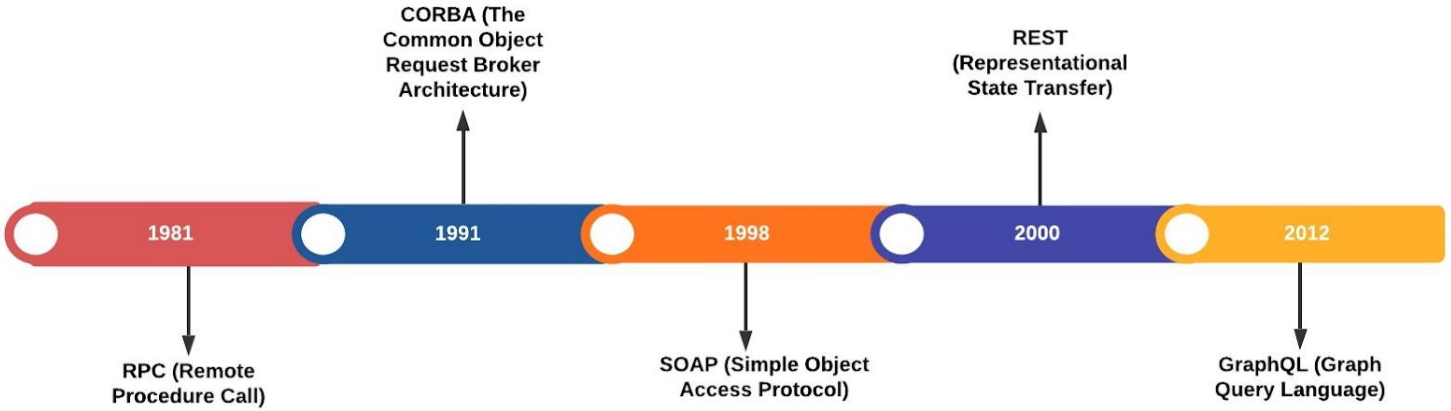
"GraphQL is a query language for APIs and a runtime for fulfilling those queries with your existing data"

https://graphql.org/



API là viết tắt của cụm từ "Application Programming Interface" (Giao diện Lập trình Ứng dụng). Đây là một tập hợp các quy tắc, giao thức và công cụ mà các phần mềm ứng dụng sử dụng để tương tác và giao tiếp với nhau.

Timeline



GraphQL được phát triển bởi Facebook vào năm 2012 (và phát hành vào năm 2015)

GraphQL Key Concepts

- Type system
- Schema
- Operation
- Query
- Mutation
- Resolver
- Fragment
- Variable
- Subscription
- Introspection

GraphQL Type system

Object Types

Interfaces

Unions

Enumerations

Fields

List

Scalar Types

- String
- Float
- Boolean
- Int
- ID
- Custom scalars, e.g. Date

https://graphql.org/learn/schema/#type-system

GraphQL Type system

```
# Object Type
type Character {
name: String!
appearsIn: [Episode!]!
}
“””
Name and appearsIn are fields
“””
```

```
# Scalar Type
Int # A signed 32-bit integer.
Float ””” A signed double-precision floating-point value. “””
String # A UTF-8 character sequence
Boolean # true or false
ID “”” The ID scalar type represents a unique identifier, often used to refetch an object or as the key for a cache. The ID type is serialized in the same way as a String; however, defining it as an ID signifies that it is not intended to be human-readable “””
```

https://graphql.org/learn/schema/#type-system

Schema

```
interface Character {
id: ID!
name: String!
friends: [Character]
appearsIn: [Episode]!
}
```

```
type Human implements Character {
id: ID!
name: String!
friends: [Character]
appearsIn: [Episode]!
starships: [Starship]
totalCredits: Int
}
type Droid implements Character {
id: ID!
name: String!
friends: [Character]
appearsIn: [Episode]!
primaryFunction: String
}
```

https://graphql.org/learn/schema

Operation

Một operation trong GraphQL có thể là một Query, Mutation hoặc Subscription.
Đó là cấu trúc cấp cao xác định dữ liệu mà client muốn từ server.

Query

Truy vấn trong GraphQL là một request dữ liệu cụ thể từ máy chủ

```
query {
  hero {
    name
  }
}
```

```
{
  "data": {
    "hero": {
      "name": "R2-D2"
    }
  }
}
```

Query

```
query {
  hero {
    name
  }
  friends {
    name
  }
}
```

```
{
  "data": {
    "hero": {
      "name": "R2-D2"
    }
    "friends“: [
        {
          "name": "Luke Skywalker"
        },
        {
          "name": "Han Solo"
        },
      ]
  }
}
```

Mutation

Các mutation được sử dụng để thay đổi dữ liệu trên máy chủ. Chúng cho phép client CREATE, UPDATE hoặc DELETE dữ liệu.

```
mutation CreateReviewForEpisode($ep: Episode!,
$review: ReviewInput!) {
  createReview(episode: $ep, review: $review) {
    stars
    commentary
  }
}


# Variable
{
  "ep": "JEDI",
  "review": {
    "stars": 5,
    "commentary": "This is a great movie!"
  }
}
```

```
{
  "data": {
    "createReview": {
      "stars": 5,
      "commentary": "This is
a great movie!"
    }
  }
}
```

Subscription

Các subscription là cách nhận các cập nhật thời gian thực từ máy chủ. Chúng cho phép client đăng ký sự kiện cụ thể và nhận thông báo khi dữ liệu thay đổi.

Subscription

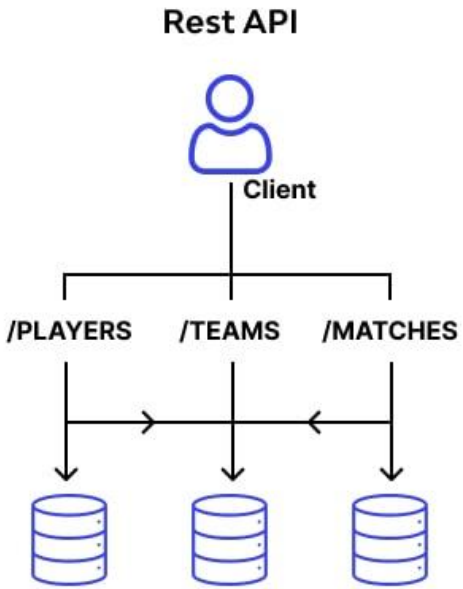
```
subscription OnCommentAdded($postID: ID!) {
     commentAdded(postID: $postID) {
          id
          content
     }
}
```

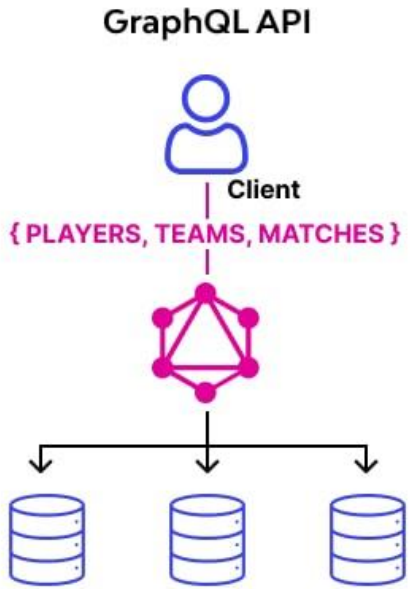
```
{
   "data": {
      "commentAdded": {
        "id": "123",
        "content": "What a
thoughtful and well written
post!"
        }
      }
}
```

GraphQL vs REST

	GraphQL	REST
Mô tả	GraphQL là một ngôn ngữ truy vấn dành cho API và một môi trường chạy phía máy chủ	Là một kiến trúc phần mềm dùng để thiết kế các dịch vụ web
Truy xuất dữ liệu	Một điểm cuối HTTP duy nhất phản hồi các truy vấn xác định	Một tập hợp các điểm cuối HTTP thường trả về một tập dữ liệu đã định trước
Phiên bản	Thường chỉ có 1 phiên bản	Thường sử dụng phiên bản
Mã trạng thái HTTP	Không có đặc tả về phản hồi HTTP cho lỗi. Tất cả các phản hồi, bao gồm lỗi, thường trả về mã 200	Thực hiện mã trạng thái HTTP
Validation	Kiểm tra lỗi dựa trên metadata được tích hợp sẵn	Kiểm tra lỗi phải được thực hiện thủ công
Tài liệu	Tự động thông qua hệ thống kiểu dữ liệu và khả năng tự giám định	Không tự mô tả, có các công cụ như OpenAPI, Swagger, Postman
Lưu vào bộ nhớ đệm	Không	Có
Phương thức request	Queries, mutations, và subscriptions (qua POST cho HTTP)	Sử dụng tất cả các phương thức HTTP (GET, POST, PATCH, PUT, DELETE, ...)
Response Content-Type	JSON	Bất kỳ loại nào (JSON, XML, HTML, vv.)

GraphQL vs REST





GraphQL vs REST

GraphQL

REST REQUEST

```
GET https://sample.com/person/1
```

REST JSON

```
{
  “firstName”: “John”,
  “middleName”: “Andrew”
  “lastName”: “Smith”,
  “email”: “jas1992@gmail.com”,
  “relationshipStatus”: “single”
}
```

VS

GRAPHQL QUERY

```
{
person {
      firstName
      lastName
   }
}
```

GRAPHQL JSON

```
{
  “data”: {
    “person”: {
      “firstName”: “John”,
      “lastName”: “Smith”,
    }
  }
}
```

GraphQL vs REST

	GraphQL	REST
Mô tả	GraphQL là một ngôn ngữ truy vấn dành cho API và một môi trường chạy phía máy chủ	Là một kiến trúc phần mềm dùng để thiết kế các dịch vụ web
Truy xuất dữ liệu	Một điểm cuối HTTP duy nhất phản hồi các truy vấn xác định	Một tập hợp các điểm cuối HTTP thường trả về một tập dữ liệu đã định trước
Phiên bản	Thường chỉ có 1 phiên bản	Thường sử dụng phiên bản
Mã trạng thái HTTP	Không có đặc tả về phản hồi HTTP cho lỗi. Tất cả các phản hồi, bao gồm lỗi, thường trả về mã 200	Thực hiện mã trạng thái HTTP
Validation	Kiểm tra lỗi dựa trên metadata được tích hợp sẵn	Kiểm tra lỗi phải được thực hiện thủ công
Tài liệu	Tự động thông qua hệ thống kiểu dữ liệu và khả năng tự giám định	Không tự mô tả, có các công cụ như OpenAPI, Swagger, Postman
Lưu vào bộ nhớ đệm	Không	Có
Phương thức request	Queries, mutations, và subscriptions (qua POST cho HTTP)	Sử dụng tất cả các phương thức HTTP (GET, POST, PATCH, PUT, DELETE, ...)
Response Content-Type	JSON	Bất kỳ loại nào (JSON, XML, HTML, vv.)

GraphQL vs REST

	GraphQL	REST
Hiệu suất	Nhanh	Gọi API nhiều lần gây tốn thời gian
Độ phức tạp của truy vấn	Có thể trở nên rất phức tạp bởi request của client khác nhau	Có các endpoints khác nhau cho từng truy vấn nên truy vấn đơn giản hơn
Mức độ phổ biến	Đang phát triển	Rất phổ biến
Tài nguyên và cộng đồng	Đang phát triển	Lớn
Learning curve	Dốc	Rất đơn giản
Use case	Microservices và mobile apps Băng thông giới hạn App có data lớn, phức tạp, cần kết hợp nhiều data	App có data đơn giản, tổ chức tốt, không yêu cầu truy vấn phức tạp

Benefits of using GraphQL in Flutter

Truy xuất dữ liệu hiệu quả

Linh hoạt

Request duy nhất

Dữ liệu thời gian thực

Giảm phiên bản

Ít phụ thuộc

Tài liệu thụ động

Strongly Typed

Tối ưu cho các framework front-end hiện đại

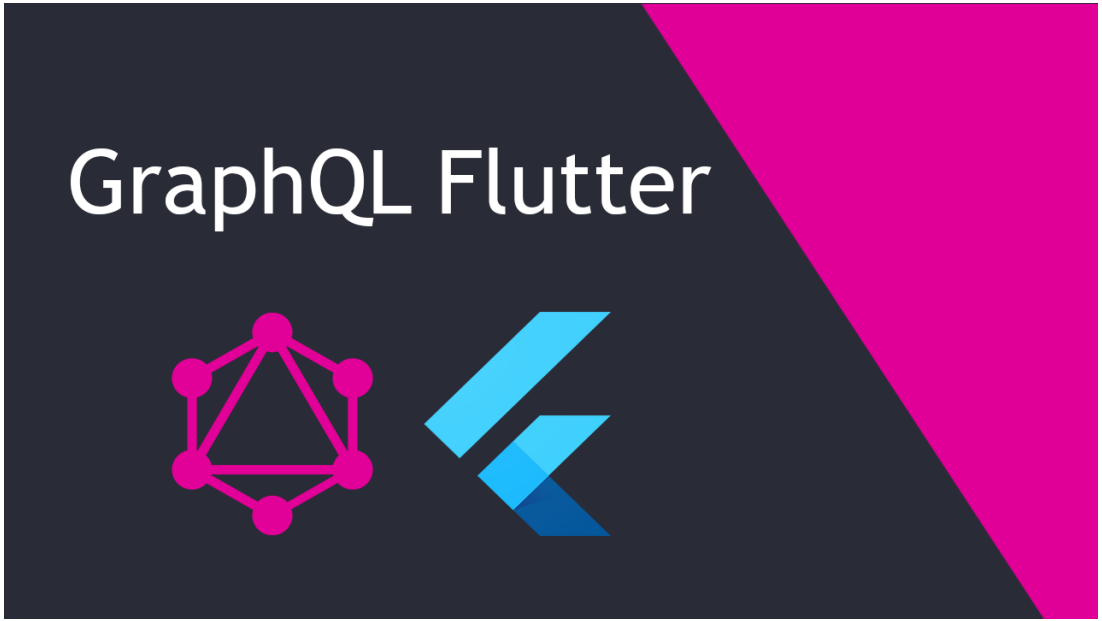
02

Setting Up GraphQL Client

Introduce The Package

Package link: https://pub.dev/packages/graphql_flutter

Với graphql_flutter, chúng ta có thể định nghĩa các truy vấn GraphQL bằng cú pháp đặc biệt trong mã Dart và nó cung cấp các widget để lấy và hiển thị dữ liệu một cách hiệu quả trong giao diện người dùng Flutter



Install package

Cách 1: Thêm tên package vào dependencies của pubspec.yaml:

```
dependencies:
  graphql_flutter: ^5.1.2
```

Cách 2: Chạy lệnh sau trong command line:

```
flutter pub add graphql_flutter
```

Sau đó import package để sử dụng:

```
import 'package:graphql_flutter/graphql_flutter.dart';
```

Set up the client #1

```
class GraphQLConfig {
  static HttpLink httpLink = HttpLink('GRAPHQL SERVER URL');
  GraphQLClient clientToQuery() {
    return GraphQLClient(
      cache: GraphQLCache(),
      link: httpLink,
    );
  }
}
static GraphQLConfig graphQLConfig = GraphQLConfig();

GraphQLClient client = graphQLConfig.clientToQuery();
```

Sau đó có thể dung client để trực tiếp thực hiện các request hoặc request thông qua các widgets mà graphql_flutter cung cấp bằng cách sử dụng GraphQLProvider bọc bên ngoài các widgets đó

Set up the client #2

```
ValueNotifier<GraphQLClient> client = ValueNotifier(
    GraphQLConfig().clientToQuery(),
  );

// Bọc toàn bộ ứng dụng bằng GraphQLProvider để có thể sử dụng
được client

return GraphQLProvider(
      client: client,
      child: MaterialApp(
              ...

        ),
    );
```

03

Define GraphQL Query and Mutation

Define Query and Mutation

```
const getGames = """
  query Game ($gameId: ID!){
    game(id: $gameId) {
      id
      title
      platform
      reviews {
        rating
        content
        author {
          name
        }
      }
    }
  }
""";
```

QUERY

- Bắt đầu bằng từ khóa query và tên.

- Theo sau là tên thao tác tùy chọn và biến.

- Chỉ định các trường bạn muốn truy xuất.

Define Query and Mutation

```
const createGame = r"""
  mutation AddGame($game: AddGameInput!) {
    addGame(game: $game) {
    id
    title
    platform
   }
 }
""";
```

MUTATION

- Bắt đầu bằng từ khóa query và tên.

- Theo sau là tên thao tác tùy chọn và biến.

- Chỉ định các trường bạn muốn truy xuất làm kết quả của mutation.

Variable

```
const getGames = """
  query Game ($gameId: ID!){
    game(id: $gameId) {
      id
      title
      platform
      reviews {
        rating
        content
        author {
          name
        }
      }
    }
  }
""";
```

- Khai báo: thêm dấu $ vào trước tên biến.

- Mỗi biến phải có một kiểu dữ liệu.

- Truyền giá trị: thực hiện như một JSON riêng biệt cùng với query và mutation.

```
{
  "game": {
    "platform": [“Mobile"],
    "title": “Gunny"
  }
}
```

Fragment

```
const getGames = """
  query Game ($gameId: ID!){
    game(id: $gameId) {
      id
      title
      platform
      reviews {
        ...ReviewFields
      }
    }
  }
""" + fragment;
```

```
const fragment = r"""
    fragment ReviewFields on Review {
      rating
      content
      author {
        name
      }
    }
  """;
```

- Khai báo: bằng từ khóa ‘fragment’, theo sau là tên fragment và kiểu mà nó áp dụng.

- Chọn trường dữ liệu muốn truy vấn.

- Sử dụng: toán tử spread(…) + tên fragment

04

Making GraphQL Requests

Request using Client

- Đối với Query:

```
QueryResult queryResult = await client.query(QueryOptions)
```

- Đối với mutation:

```
QueryResult queryResult = await client.mutate(MutationOptions)
```

- Đối với subscription:

```
Stream<QueryResult> result = client.subscribe(SubscriptionOptions)
```

QueryOptions

```
QueryOptions(
    document: gql(GrapqhQuery.getGame),
    fetchPolicy: FetchPolicy.noCache,
    variables: {
      'gameId': id,
    },
    pollInterval: const Duration(seconds: 10),
)
```

MutationOptions

```
MutationOptions(
    document: gql(GrapqhQuery.createGame),
    fetchPolicy: FetchPolicy.noCache,
    variables: {
      "game": {
        "title": title,
        "platform": platforms,
      }
    },
)
```

SubscriptionOptions

```
SubscriptionOptions(
    document: gql(GrapqhQuery.subscriptionQuery),
    fetchPolicy: FetchPolicy.noCache,
    variables: {
      “variableName": variableName
    },
)
```

Handle QueryResult

```
if (result.hasException) {
      print('GraphQL Error: ${result.exception.toString()}');
      // Xử lý lỗi dựa trên result.exception.graphqlErrors
      // result.exception.graphqlErrors.forEach((error) {
      //   print('GraphQL Error Message: ${error.message}');
      // });
    } else {
      // Xử lý kết quả thành công
      print('Query result: ${result.data}');
    }
}
```

Query Widget

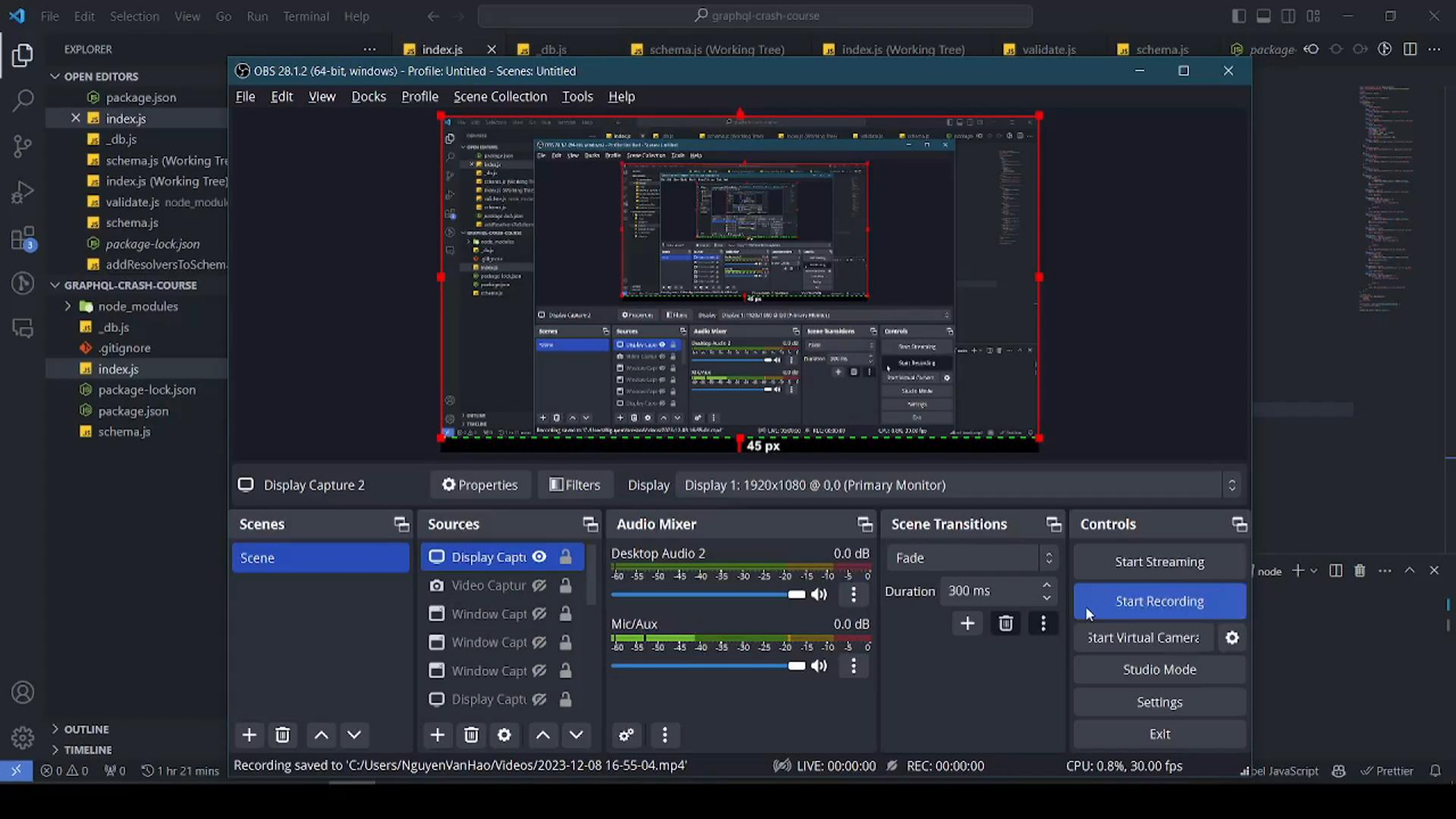
```
Query(
      options: queryOptions,
      builder: (QueryResult result,
          {VoidCallback? refetch, FetchMore? fetchMore}) {
        if (result.hasException) {
          return Text(result.exception.toString(),
          );
        }
        if (result.isLoading) {
          return const CircularProgressIndicator();
        }

        final Game game = Game.fromJson(result.data!['game']);
        return Text(game.title);
```

Mutation Widget

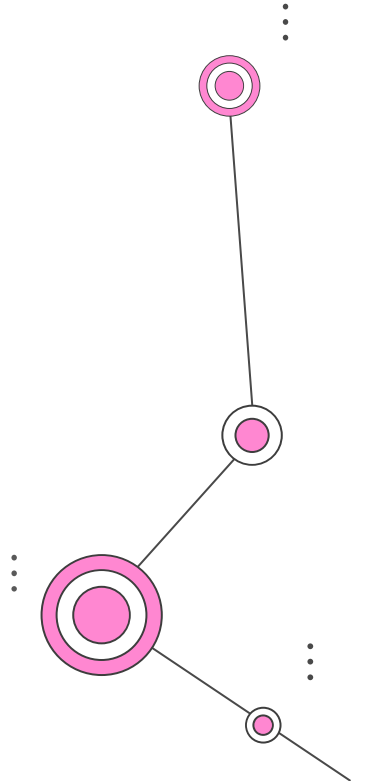
```
Mutation(
        options: MutationOptions(
          document: gql(GrapqhQuery.addReview),
          onCompleted: (data) { Navigator.of(context).pop(); },
         ),
         builder: (RunMutation runMutation, QueryResult? result) {
            return TextButton(
                      onPressed: () {
                          runMutation({
                              'review': {
                                'rating': int.parse(ratingCtrl.text),
                                    'content': contentCtrl.text,
                                    'game_id': game.id,
                                  }
                                });
                       },
                       child: const Text('Create'),
                    );
          },
)
```

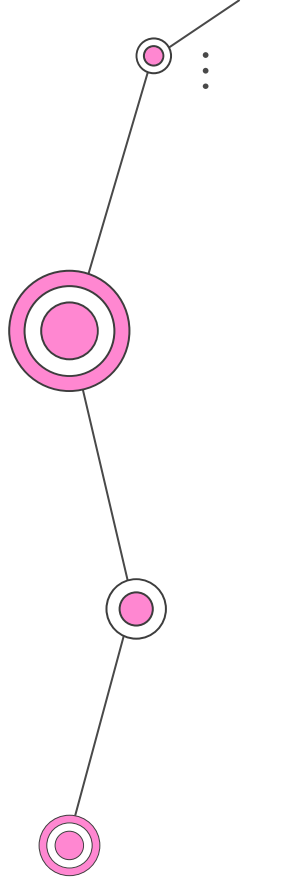
05

Demo application


File Edit Selection View Go Run Terminal Help
graphql-crash-course
EXPLORER
index.js
_db.js
schema.js (Working Tree)
index.js (Working Tree)
validate.js
schema.js
package
OPEN EDITORS
package.json
index.js
_db.js
schema.js (Working Tre
index.js (Working Tree)
validate.js node_modul
schema.js
package-lock.json
addResolversToSchem
GRAPHQL-CRASH-COURSE
node_modules
_db.js
.gitignore
index.js
package-lock.json
package.json
schema.js
OUTLINE
TIMELINE
1 hr 21 mins
OBS 28.1.2 (64-bit, windows) - Profile: Untitled - Scenes: Untitled
File Edit View Docks Profile Scene Collection Tools Help
45 px
Display Capture 2
Properties
Filters
Display
Display 1: 1920x1080 @ 0,0 (Primary Monitor)
Scenes
Scene
Sources
Display Capti
Video Captur
Window Capt
Window Capt
Window Capt
Display Capti
Audio Mixer
Desktop Audio 2
0.0 dB
Mic/Aux
0.0 dB
Scene Transitions
Fade
Duration
300 ms
Controls
Start Streaming
Start Recording
Start Virtual Camera
Studio Mode
Settings
Exit
Recording saved to 'C:/Users/NguyenVanHao/Videos/2023-12-08 16-55-04.mp4'
LIVE: 00:00:00
REC: 00:00:00
CPU: 0.8%, 30.00 fps
Prettier

Thanks For Listening!





Q & A