# GraphQL

## Thành viên nhóm:


- 22120011 - Lê Tuấn Anh
- 22120067 - Lâm Hồng Anh Đức
- 22120183 - Nguyễn Đặng Minh Lân

# GraphQL

# Nội dung

- GraphQL là cái chi rứa?
- Những gì cần biết về GraphQL
- Tại sao GraphQL hiệu quả?
- Khi nào sử dụng GraphQL?
- Thảo luận

# GraphQL là cái gì?

# GraphQL là cái gì?

## Trước khi biết:

- 1 loại truy vấn mới trên database?
- Có SQL là ngôn ngữ truy vấn có cấu trúc cho Database, vậy GraphQL chắc là ngôn ngữ truy vấn bằng đồ thị cho Database kiểu kéo thả trực quan hơn?

## Sau khi biết:

- GraphQL là ngôn ngữ truy vấn và runtime cho API

# GraphQL là cái gì?

Hiểu đơn giản là 1 cách khác để xây dựng API thay vì RESTAPI thông thường hay dùng.

## REST: Nhiều endpoints với response cố định

```
/users/123
```

## GraphQL: Một endpoint với response tùy ý

```
query{
    user(id: 123) {
        name
        posts{
            title
        }
    }
}
```

# Những gì cần biết về GraphQL

# Những gì cần biết về GraphQL

## Schema

- **Mô hình hóa** dữ liệu
- Cốt lõi là **hợp đồng** giữa Client và Server

# Những gì cần biết về GraphQL

## Schema

- Định nghĩa **User object**

```
type User {
    id: ID!
    name: String!
    posts: [Posts!]!
    createdAt: DateTime!
}
```

# Những gì cần biết về GraphQL

## Schema

- Định nghĩa **Post object**

```
type Post {
    id: ID!
    title: String!
    content: String
    createdAt: DateTime!
}
```

# Những gì cần biết về GraphQL

## Schema

- Định nghĩa input

```
input PostInput {
  title: String!
  content: String
  isPublished: Boolean
}
```

# Những gì cần biết về GraphQL

## Schema

- Định nghĩa **Query method** cho Post

```
type Query {
  posts: [Post!]!

  post(id: ID!): Post
}
```

# Những gì cần biết về GraphQL

## Schema

- Định nghĩa **Mutation method** cho Post

```
type Mutation {
  createPost(data: PostInput!): Post!

  updatePost(id: ID!, data: PostInput!): Post

  deletePost(id: ID!): Post
}
```

# Những gì cần biết về GraphQL

## Schema

- Định nghĩa **Subscription method** cho Post

```
type Subscription {
  postAdded: Post!

  postUpdated(id: ID!): Post
}
```

# Những gì cần biết về GraphQL

## Queries

- Phương thức **GET**
- Cốt lõi là để **lấy** dữ liệu

```
query {
    user(id: 123) {
        name
        posts{
            title
        }
    }
}
```

# Những gì cần biết về GraphQL

## Mutation

- Phương thức **POST, PUT, PATCH, DELETE**
- Cốt lõi là để **thay đổi** dữ liệu

```
mutation {
    createPost(data: {"title": "abc", "content": "xyz"}){
        id
    }
}
```

# Những gì cần biết về GraphQL

## Mutation

- Phương thức **POST, PUT, PATCH, DELETE**
- Cốt lõi là để **thay đổi** dữ liệu

```
mutation {
    createPost(data: {"title": "abc", "content": "xyz"}){
        id
    }
}
```

# Những gì cần biết về GraphQL

## Subscription

- Lắng nghe sự kiện **(Real-Time)**
- Ứng dụng trong app chat, thông báo, chứng khoáng, bảng tỉ số bóng đá, cá độ, tài xỉu ...
- Client mở một kết nối bền vững liên tục (thường là **WebSocket**) với Server.
- Cần cả HttpLink (HTTP) và WebSocketLink (WSS)

# Những gì cần biết về GraphQL

## Subscription

- Cú pháp **tương tự** query, khác ở chỗ thay query thành subscription

```
subscription OnPostAdded {
  postAdded {
    id
    title
    createdAt
  }
}
```

# Tại sao GraphQL hiệu quả?

# Tại sao GraphQL hiệu quả?

- Thay vì backend phải biết rõ frontend và ngược lại, 2 thằng **chỉ cần biết qua 1 thằng trung gian duy nhất** là GraphQL.
- **Lấy đủ** không thấy dư hoặc thiếu.
- **Tiết kiệm dữ liệu, app chạy nhanh hơn** vì payload JSON nhỏ hơn trong 1 số trường hợp.
- **Typing mạnh mẽ, Schema rõ ràng**, code an toàn, ít bug.
- Không cần đợi backend sửa API endpoint.

# Khi nào sử dụng GraphQL?

- Nhiều Clients (mobile/web) cần dữ liệu khác nhau
- Dữ liệu có cấu trúc sâu, như một chuỗi các đối tượng lồng nhau, cần gọi nhiều API lại.
- Xuất hiện **over-fetching / under-fetching** quá nhiều khi xài REST.

# Thảo luận

# Thảo luận

## So sánh giữa (Query + Mutation) và (Subscription)

| | Query + Mutation | Subscription |
|---|---|---|
| Cơ chế | Pull | Push |
| Công nghệ | Http (request - reponse) | WebSocket |
| Chi phí | Chỉ tốn khi gọi | Server phải duy trì két nối với clien |
| Ứng dụng | Giỏ hàng, profile, sản phẩm | Chat, Grab/Uber |

# Thảo luận

> Không có cái nào trên đời là hoàn hảo, cái nào cũng có mặt tốt và xấu của nó.

- **GraphQL không tốt hơn REST. Nó chỉ đơn giản giải quyết các vấn đề khác nhau.**
- GraphQL được tạo ra để giải quyết vấn đề scaling của Facebook.
- Cân nhắc sử dụng GraphQL khi gọi quá nhiều API mỗi page.

# GraphQL cho Flutter GraphQL cho Flutter bằng Hygraph serverless

# Nội dung

- Dependencies
- GraphQLClient
- Query
- Mutation
- Publish

# Dependencies

# Dependencies

https://pub.dev/packages/graphql_flutter

- Thêm bằng terminal

```
flutter pub add graphql_flutter
```

- Thêm bằng file pubspect.yaml

```
dependencies:
  flutter:
    sdk: flutter
  ...
  graphql_flutter: ^5.2.1
```

# GraphQLClient

# GraphQLClient

## Yêu cầu:

- endpoint link
- token (nếu có)

## Code mẫu:

```
final HttpLink httpLink = ... (endpoint-link)
final String token = ... (token)

final AuthLink authLink = AuthLink(getToken: ()=> "Bearer $token");

final Link link = authLink.concat(httpLink);

return GraphQLClient(link: link, cache: GraphQLCache());
```

# Query

# Query

## Sử dụng

- raw query
- GraphQLClient

## Đầu vào

- raw query string
- tham số (nếu có)

## Đầu ra

- Map<String,dynamic>

# Query

## 1. Kiểu query không tham số

```
final result = await client.query(
    QueryOptions(document: gql(getPostsQuery)),
);
```

## 2. Kiểu query có tham số

```
final result = await client.query(
      QueryOptions(document: gql(getPostByIdQuery), variables: {"id": id}),
);
```

# Mutation

# Mutation

## Sử dụng

- raw mutation
- GraphQLClient

## Đầu vào

- raw mutation string
- tham số: **unique, id ...**

## Đầu ra

- Map<String,dynamic>

# Mutation

## 1. Kiểu mutation create

```
final result = await client.mutate(
      MutationOptions(
        document: gql(createPostMutation),
        variables: {
          "input": {"title": title, "content": content},
        },
      ),
);
```

# Mutation

## 2. Kiểu mutation update

```
final result = await client.mutate(
    MutationOptions(
      document: gql(updatePostMutation),
      variables: {
        "id": id,
        "input": {
          "title": title,
          "content": content,
        },
      },
    ),
);
```

# Mutation

## 3. Kiểu mutation delete

```
final result = await client.mutate(
    MutationOptions(
      document: gql(deletePostMutation),
      variables: {
        "id": id,
      },
    ),
);
```

# Mutation

## Lưu ý mutation khi sử dụng hygraph:

- Dữ liệu khi thay đổi đều sẽ ở bản draft, nếu muốn nó chính thức sử dụng được cần phải publish.
- Publish là **cơ chế riêng** của Hygraph (CMS), không phải chuẩn chung của mọi GraphQL server.

# Publish

# Publish

## Sử dụng

- raw mutation
- GraphQLClient

## Đầu vào

- raw mutation string
- tham số: id

## Đầu ra

- Map<String,dynamic>

# Publish

```
final result = await client.mutate(
      MutationOptions(
        document: gql(publishWishMutation),
        variables: {"ID": id},
      ),
);
```