🧬 Introduction to GraphQL

Facilitator: Alberto Camarena

Presented at: Global Hack Week 🌐

Date: 15/04/25 📅

📚 Workshop Overview

Sessions

1. **GraphQL Fundamentals & Querying**
2. **Building a GraphQL Backend**
3. **Creating a Frontend with GraphQL**

Objectives

- Understand what GraphQL is and how it compares to REST
- Learn the core components of GraphQL
- Build a GraphQL server
- Develop a frontend that queries the GraphQL API

🧠 What is GraphQL?

> A query language for APIs and a runtime for fulfilling those queries with your existing data.

- Developed by Facebook in 2012, open-sourced in 2015
- Single endpoint API
- Schema-based, strongly typed
- Reduces over-fetching and under-fetching

🔗 https://graphql.org

VS REST vs. GraphQL

Feature	REST	GraphQL
Endpoints	Multiple	Single
Data Fetching	Fixed structure	Flexible queries
Over/Under-fetching	Common	Avoided
Versioning	URI-based	Schema evolution
Introspection	Not supported	Built-in

🏗️ GraphQL Core Concepts

Schema

Defines the API structure using types and fields.

Operation Types

- `Query` : Read operations
- `Mutation` : Write operations
- `Subscription` : Real-time updates (optional)

Resolver

Functions that fetch the data for a field.

✍️ Deep Dive: GraphQL Schemas & Types

- **Schema as a Contract:**
  - Defines the available data and operations for clients.
  - Serves as a single source of truth between client and server.

```
type Query {
  me: User
}

type User {
  name: String
}
```

- **Types Explained:**
  - **Scalar Types:**
    - Fundamental data types like `String`, `Int`, `Float`, `Boolean`, `ID`.
  - **Object Types:**
    - Custom types that define objects with specific fields.
  - **Advanced Types:**
    - **Enums:** A set of predefined values.
    - **Interfaces & Unions:** For shared fields and flexible type relations.
    - **Input Types:** For complex arguments in mutations.
- **Useful Link:**
  - GraphQL Schema Documentation

🔍 Operation Type: Query

- **Purpose:**
  - Read and fetch data from the server.
- **How It Works:**
  - Defined in your schema as a `Query` type.
  - Each query field connects to a resolver that retrieves the corresponding data.
- **Analogy:**
  - Equivalent to HTTP GET in REST.

🔗 GraphQL Queries Documentation

✨ Example Query

```
query GetCountry {
  country(code: "BR") {
    name
    capital
    emoji
    currency
    languages {
      name
    }
  }
}
```

✅ Ask only for the data you need

🔗 Try it in the browser: https://countries.trevorblades.com

✍️ Operation Type: Mutation

- **Purpose:**
  - Create, update, or delete data.
- **How It Works:**
  - Defined in the schema as a `Mutation` type.
  - Each mutation field is tied to a resolver that performs the corresponding write or data-changing action.
- **Analogy:**
  - Similar to HTTP POST/PUT/PATCH/DELETE in REST.

🔗 GraphQL Mutations Documentation

⚡ Operation Type: Subscription

- **Purpose:**
  - Provide real-time updates to clients.
- **How It Works:**
  - Defined in your schema as a `Subscription` type.
  - Uses resolvers to setup a live connection (often via WebSocket) to push data as events occur.
- **Analogy:**
  - Comparable to a live feed or real-time socket connection.

🔗 GraphQL Subscriptions Documentation

👆 Understanding Resolvers: The Heart of GraphQL

- **What Are Resolvers?**
  - Functions that actually fetch the data for each field in your schema.
- **How Resolvers Work:**
  - Each field in the GraphQL query is connected to a resolver.
  - **Resolver Function Arguments:**
    - `parent` : The result from the parent field.
    - `args` : Input arguments provided in the query.
    - `context` : Shared context (e.g., authentication info, data loaders).
    - `info` : Information about the execution state (e.g., field AST).

- **Example Resolver in Action:**

```
const resolvers = {
  Query: {
    post: (parent, args, context, info) => {
      // 'args' contains the parameters such as `id`
      return postsData.find(post => post.id === args.id);
    }
  },
  // Other resolvers for mutations and nested types...
};
```

- **Useful Link:**
  - Apollo Server Resolvers Guide

Bringing It All Together

- **Integration of Components:**
    i. **Schemas & Types:**
        - Define the *what*: the data structure and operations available (Queries, Mutations, Subscriptions).
    ii. **Resolvers:**
        - Define the *how*: the functions that fetch or modify the data for each schema field.
    iii. **Operation Types:**
        - **Query:** Retrieves data (read-only).
        - **Mutation:** Changes data (create, update, delete).
        - **Subscription:** Delivers real-time data updates.

- **Workflow Overview:**
    i. **Design the Schema:** Outline your types, queries mutations, and (optionally) subscriptions.
    ii. **Implement Resolvers:** Write functions to fulfill the schema's requirements.
    iii. **Test and Iterate:** Use GraphQL playgrounds to run queries and adjust your implementation.
- **Best Practices:**
    - Keep resolvers focused and simple.
    - Modularize your schema for better maintainability.
    - Leverage context for shared data operations.
- **Useful Link:**
    - GraphQL Best Practices

🛠️ Hands-On: Querying a Public API

Use: https://swapi-graphql.netlify.app

Example:

```
{
  allPeople {
    people {
      name
      gender
    }
  }
}
```

🧪 Try it in GraphiQL or Apollo Sandbox!