GraphQL

A Query Language

GraphQL

Training Agenda

- NodeJS Overview
- GraphQL Introduction
- Schemas & Queries
- Mutations
- Subscriptions
- GraphQL Testing with Frontend

NodeJS : An Overview

Node.js is a platform built on Chrome's JavaScript runtime for easily building fast and scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.

NodeJS : Features

Extremely fast

I/O is Asynchronous and Event Driven

Single threaded

Highly Scalable

No buffering

Open source

GraphQL : Overview

GraphQL is a query language

Provides server-side runtime for executing queries

Can define your own type system for your specific app data

GraphQL is a new API standard that provides a more efficient, powerful and flexible alternative to REST

The Core Of GraphQL

GraphQL enables *declarative data fetching* where a client can specify exactly what data it needs from an API.

GraphQL server only exposes a single endpoint and responds with precisely the data a client asked for.

Data fetching with REST vs GraphQL

REST API

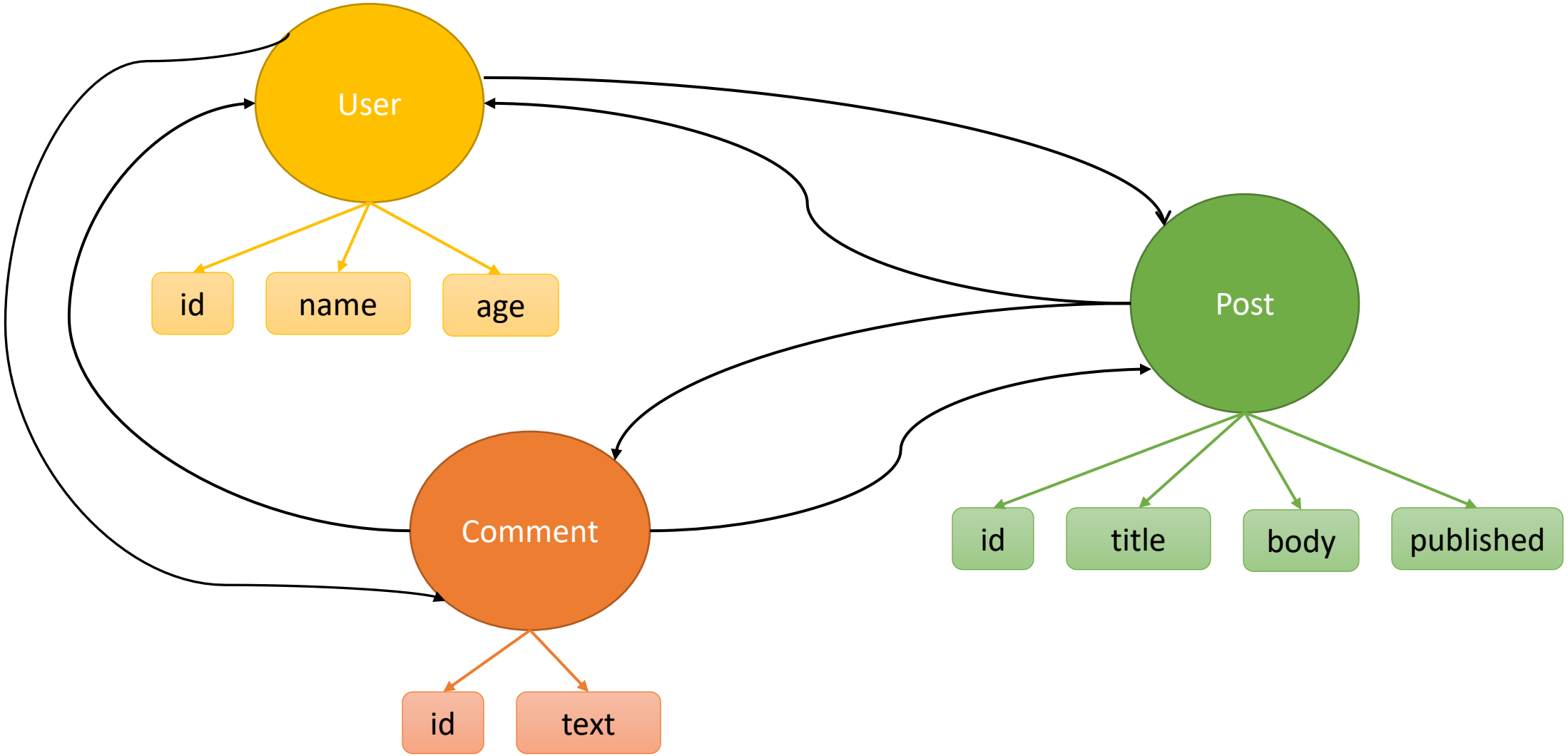
- With a REST API, you would typically gather the data by accessing multiple endpoints.

GRAPHQL API

- Simply send a single query to the GraphQL server that includes the concrete data requirements.
- The server then responds with a JSON object where these requirements are fulfilled.

No More Over or Under-fetching

Understanding Graph



GraphQL Scalar Types

A Scalar type stores a single value

ID	Used to store unique identifier
String	Used to store string data as UTF-8 characters
Boolean	Used to store true or false
Int	Used to store 32-bit integer numbers
Float	Used to store double precision floating-point number

Schema Definition Language (SDL)

GraphQL has its own type system that's used to define the *schema* of an API.

The syntax for writing schemas is called Schema Definition Language (SDL).

```
type User {
    name : String!
    age : Int!
}
```

GraphQL : Operations

Query	Mutation	Subscription
Fetching data in efficient and flexible manner	Creating and Updating data on Server	Running the Code when a certain event occurred

Fetching Data Using Queries

Unlike REST, GraphQL APIs only expose *a single endpoint*. This works because the structure of the data that's returned is not fixed.

GraphQL APIs allow to let the client decide what data is actually needed.

The client needs to send more *information* to the server to express its data needs - this information is called a *query*.

Writing Data With Mutations

The majority of applications also need some way of making changes to the data that's currently stored in the backend. With GraphQL, these changes are made using so-called *Mutations*.

There are three kinds of mutations:

- Creating new data
- Updating existing data
- Deleting existing data

Realtime Updates With Subscriptions

Unlike queries and mutations that follow a typical "*request-response*-cycle", *Subscriptions* represent a *stream* of data sent over to the client.

Whenever that particular event then actually happens, the server pushes the corresponding data to the client.

When a client *subscribes* to an event, it will initiate and hold a steady connection to the server.

Creating Schema

Schema is often seen as a *contract* between the server and client.

The *schema* is one of the most important concepts when working with a GraphQL API. It specifies the capabilities of the API and defines how clients can request the data.

A Schema is simply a collection of GraphQL types. However, when writing the schema for an API, there are some special *root* types: Query, Mutation and Subscription

Structure vs Behavior in a GraphQL server

GraphQL has a clear separation of *structure* and *behaviour*.

The structure of a GraphQL server is its *Schema*, an abstract description of the server's capabilities.

The structure comes to life with a concrete *implementation* that determines the server's *behaviour*. Key components for the implementation are so-called *resolver* functions.

Each field in a GraphQL schema is backed by a *resolver*.

References

Web

- https://graphql.org
- https://www.howtographql.com
- http://expressjs.com
- https://mongoosejs.com
- https://www.apollographql.com

Videos

- https://www.youtube.com/watch?v=Dr2dDWzThK8&t=2s
- https://www.youtube.com/watch?v=yqWzCV0kU_c