GraphQL

A Query Language

GraphQL

Training Agenda

- NodeJS Overview
- GraphQL Introduction
- Schemas & Queries
- Mutations
- Subscriptions
- Data Persistence (MongoDB)
- Authentication
- 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

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 an API standard that provides 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

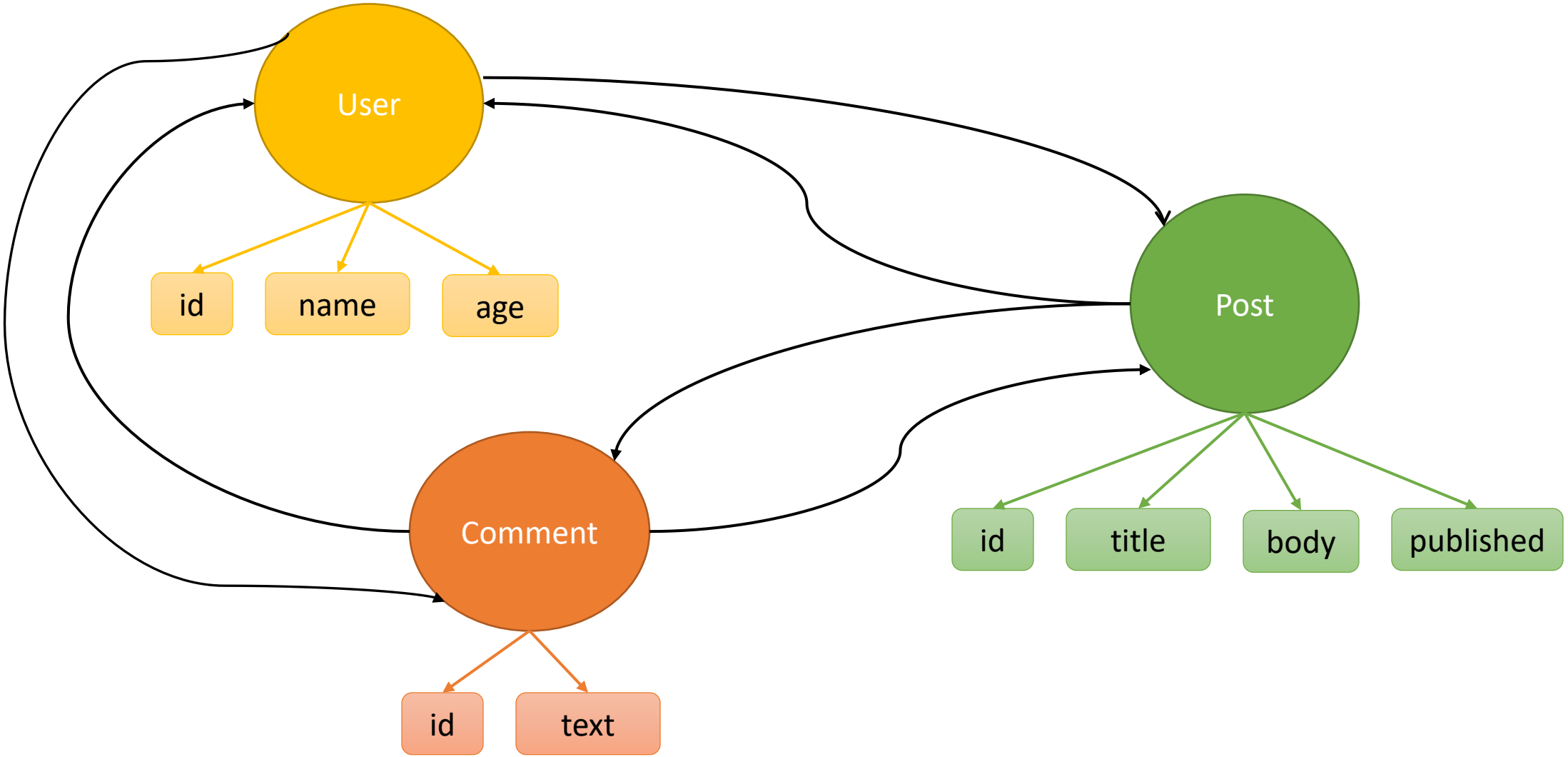
- With a REST API, you would typically gather the data by accessing multiple endpoints.

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!
}
```

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*.

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

Most applications 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.

MongoDB

NoSQL Database

MongoDB : Introduction

MongoDB is a document-oriented NoSQL database used for high volume data storage.

MongoDB makes use of collections and documents

Documents consist of key-value pairs which are the basic unit of data in MongoDB

Collections contain sets of documents and function which is the equivalent of relational database tables

MongoDB : Components

_id	MongoDB is a document-oriented NoSQL database used for high volume data storage.
Collections	A grouping of MongoDB documents. A collection is the equivalent of a table which is created in any other RDMS such as Oracle or MS SQL.
Cursor	A pointer to the result set of a query. Clients can iterate through a cursor to retrieve results.
Database	Container for collections like in RDMS wherein it is a container for tables. Each database gets its own set of files on the file system. A MongoDB server can store multiple databases.
Document	A record in a MongoDB collection is basically called a document. The document, in turn, will consist of field name and values

MongoDB : Components

Field	A name-value pair in a document. A document has zero or more fields. Fields are analogous to columns in relational databases
JSON	This is known as JavaScript Object Notation. This is a human-readable, plain text format for expressing structured data

Why MongoDB ?

Document-oriented	MongoDB is a NoSQL type database, instead of having data in a relational type format, it stores the data in documents.
Ad hoc queries	MongoDB supports searching by field, range queries, and regular expression searches. Queries can be made to return specific fields within documents.
Indexing	Indexes can be created to improve the performance of searches within MongoDB. Any field in a MongoDB document can be indexed.
Replication	MongoDB can provide high availability with replica sets.
Load balancing	MongoDB uses the concept of sharding to scale horizontally by splitting data across multiple MongoDB instances.

Difference between MongoDB & RDBMS

RDBMS	MongoDB	Difference
Table	Collection	In RDBMS, the table contains the columns and rows which are used to store the data whereas, in MongoDB, this same structure is known as a collection.
Row	Document	In RDBMS, the row represents a single, implicitly structured data item in a table. In MongoDB, the data is stored in documents.
Column	Field	In RDBMS, the column denotes a set of data values. These in MongoDB are known as Fields
Joins	Embedded Documents	In RDBMS, data is sometimes spread across various tables and in order to show a complete view of all data, a join is formed across tables to get the data. In MongoDB, the data is normally stored in a single collection, but separated by using Embedded documents. Hence there is no concept of joins in MongoDB.

DB, Shell and Tools

MongoDB Database Installation

- https://www.mongodb.com/try/download/community

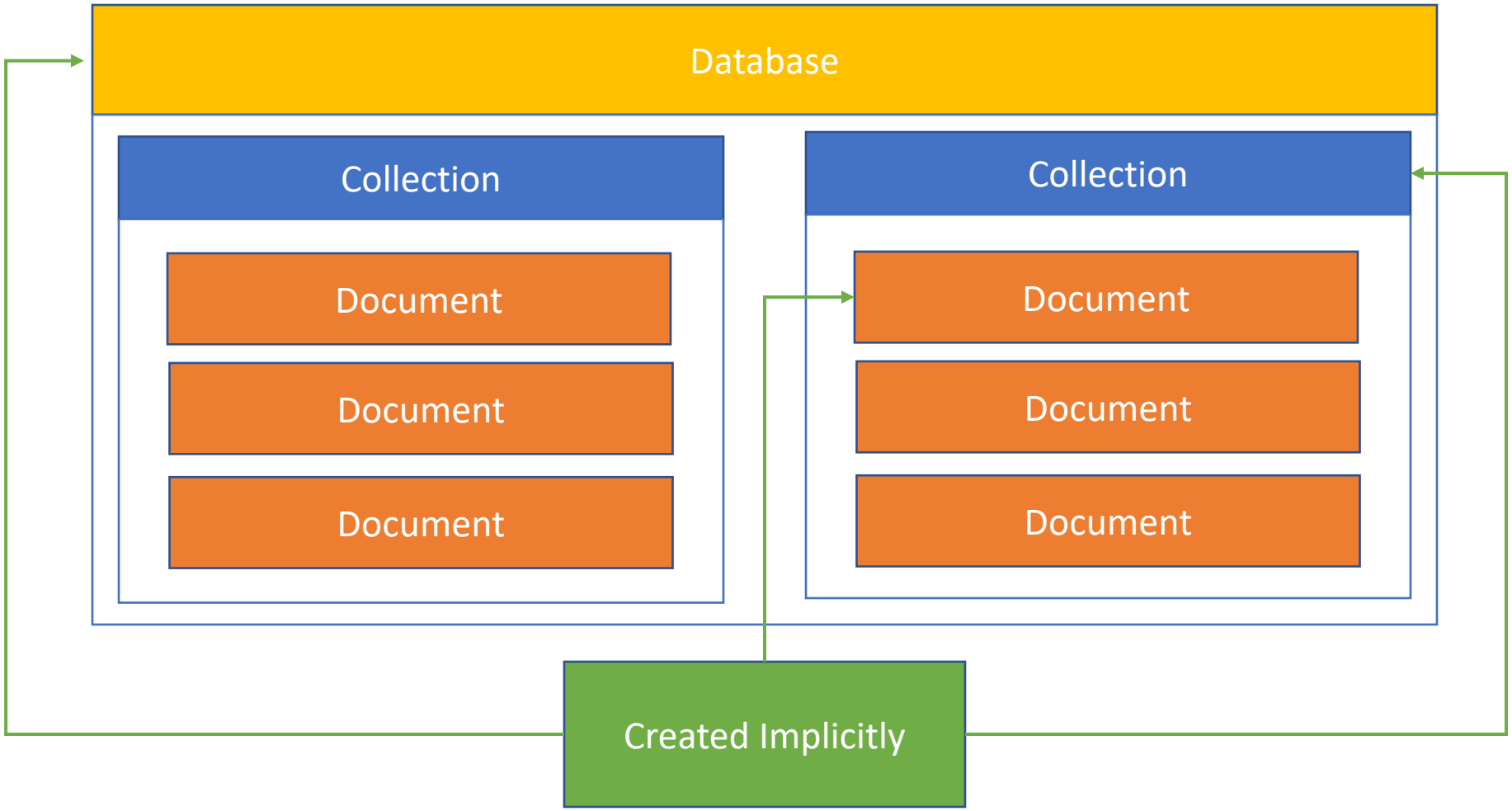
Mongo Shell Installation

- https://www.mongodb.com/try/download/shell

Mongo Database Tools

- https://www.mongodb.com/try/download/database-tools

Databases, Collections, Documents



References

Web

- https://graphql.org
- https://www.howtographql.com
- https://www.apollographql.com

Videos

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