FullstackJS

Contents

Flow4b

Explain shortly about GraphQL, its purpose and some of its use cases

- GraphQL is query language for API's
- GraphQL gives a good and complete description of the data in your api, and gives the users the ability extract nested data and get what you exactly need.

Explain some of the Server Architectures that can be implemented with a GraphQL backend

- Retrieving data from a database and serve it to the user

- In more complex architectures it is possible to use GraphQl as a microservice to serve different backends

What is meant by the terms over- and under-fetching in relation to REST

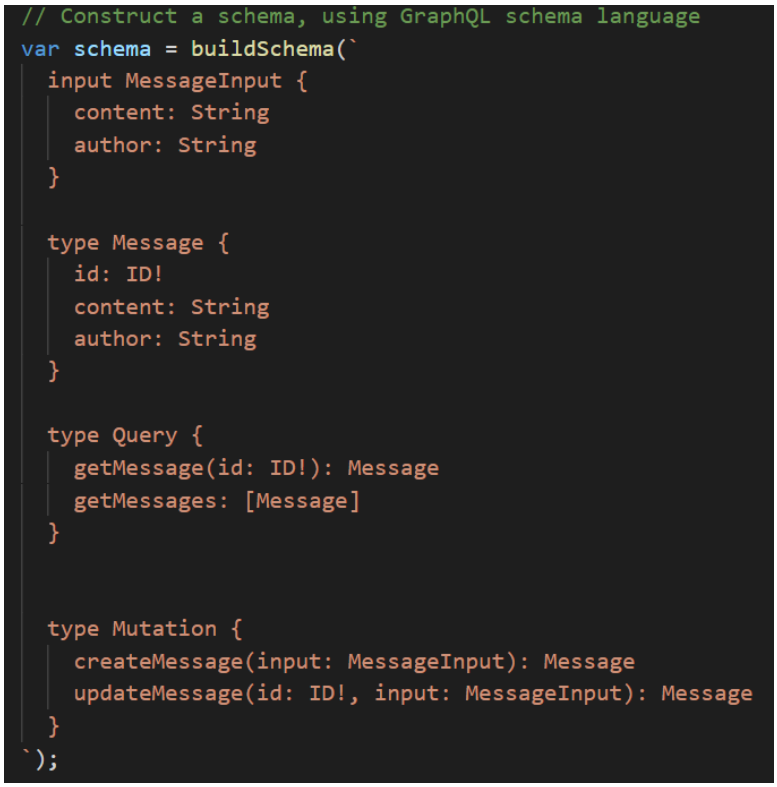
- Over- and under-fetching is a term that is used to describe the amount of the data that is retrieved in context to what is needed.
- Under-fetching is fetching not enough data, over-fetching is fetching more data than what is needed

Explain shortly about GraphQL's type system and some of the benefits we get from this

- One of the benefits is type-safety.
- When using GraphQl you can make queries that can guarantee return types

Explain shortly about GraphQL Schema Definition Language, and provide a number of examples of schemas you have defined.

- There exists different types of GraphQl schemas, there is scalar types, mutations, queries, enums and custom types
-



```
// Construct a schema, using GraphQL schema language
var schema = buildSchema(`
  input MessageInput {
    content: String
    author: String
  }

  type Message {
    id: ID!
    content: String
    author: String
  }

  type Query {
    getMessage(id: ID!): Message
    getMessages: [Message]
  }


  type Mutation {
    createMessage(input: MessageInput): Message
    updateMessage(id: ID!, input: MessageInput): Message
  }
`);
```

Provide a number of examples demonstrating data fetching with GraphQL. You should provide examples both running in a Sandbox/playground and examples executed in an Apollo Client

- 

```
const url = "https://yoururl.com/";
const opts = {
  method: "POST",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify({ query })
};
fetch(url, opts)
  .then(res => res.json())
  .then(console.log)
  .catch(console.error);
```

Provide a number of examples demonstrating creating, updating and deleting with Mutations. You should provide examples both running in a Sandbox/playground and examples executed in an Apollo Client.

- 

```
createMessage: function ({input}) {
  // Create a random id for our "database".
  var id = require('crypto').randomBytes(10).toString('hex');

  fakeDatabase[id] = input;
  return new Message(id, input);
},
updateMessage: function ({id, input}) {
  if (!fakeDatabase[id]) {
    throw new Error('no message exists with id ' + id);
  }
  // This replaces all old data, but some apps might want partial update.
  fakeDatabase[id] = input;
  return new Message(id, input);
},
```

Explain the Concept of a Resolver function, and provide a number of simple examples of resolvers you have implemented in a GraphQL Server.

- Each query in a GraphQl query can be seen as a function or a method of the previous type which returns the next type.
- Each field on each type is backed by a function called “resolve” which GraphQl server developer provides
- When a field is executed, the corresponding “resolver” is called to produce the next value
- GraphQl resolvers can return promises

Explain the benefits we get from using a library like Apollo-client, compared to using the plain fetch-API

- Apollo offers plenty of libraries for implementing an effective GraphQL tech stack for JavaScript applications, and their libraries are open-sourced to be more manageable. For instance, Apollo Link provides an API for chaining different features into a GraphQL control flow. This makes it possible for automatic network retries or RESTful API endpoints instead of a GraphQL endpoints