GraphQL

- امیرعلی سلیمی
- امیر نژاد ملایری
- علی قائم دوست

GraphQL چیست؟

GraphQL یک زبان پرس و جوی داده است که توسط Facebook توسعه داده شده است. این زبان، به عنوان یک جایگزین برای REST API ایجاد شده است و به کاربران امکانات و قابلیت‌های بیشتری را در اختیار می‌گذارد. در GraphQL، توسعه‌دهندگان می‌توانند داده‌های مورد نیاز خود را با استفاده از یک پرس و جوی ساده و قابل فهم، از سرور درخواست کنند. با استفاده ازGraphQL ، توسعه‌دهندگان می‌توانند داده‌هایی که احتیاج دارند را به صورت دقیق و تنها با یک درخواست از سرور دریافت کنند.

نحوه کار با GraphQL

GraphQLشامل چندین مفهوم اساسی است که باید با آن‌ها آشنا شد. در ادامه به بررسی این مفاهیم پرداخته می‌شود:

- **Schema**:

Schemaیک قرارداد برای ارتباط با API است. در این قرارداد، تعریف می‌شود که چه داده‌هایی در اختیار کاربران APIقرار می‌گیرند. همچنین، درGraphQL ، داده‌ها به صورت Type ها تعریف می‌شوند. هر Type شامل یک یا چند Field است که شامل داده‌هایی مانند اعداد، رشته‌ها، Booleanو غیره است.

- **Query**:

درGraphQL ، با استفاده از Query، می‌توانید داده‌های مورد نیاز خود را از سرور دریافت کنید Query .به صورت یک درخواست پرس و جوی ساده و قابل فهم است که با استفاده از آن، می‌توانید داده‌های مورد نیاز خود را از سرور دریافت کنید. هر Query شامل یک یا چند Field است که توضیحاتی درباره داده‌های مورد نیاز و نحوه بازگشت داده‌ها ارائه می‌دهند.

- **Mutation**:

Mutation به صورت یک درخواست پرس و جوی ساده و قابل فهم است که با استفاده از آن، می‌توانید داده‌های مورد نیاز خود را به سرور ارسال کنید. هر Mutation شامل یک یا چند Field است که توضیحاتی درباره داده‌های مورد نیاز و نحوه به روزرسانی داده‌ها ارائه می‌دهند.

- **Subscription**:

Subscription به صورت یک درخواست پرس و جوی ساده و قابل فهم است که با استفاده از آن، می‌توانید به صورت real-time، داده‌های مورد نیاز خود را از سرور دریافت کنید. این درخواست به کاربران امکان می‌دهد تا بدون نیاز به refresh صفحه، داده‌های جدید را دریافت کنند.

نمونه پیاده‌سازی GraphQL

در ادامه، نحوه پیاده‌سازی یک API ساده با استفاده از GraphQL را بررسی می‌کنیم. برای این منظور، یک Schemaساده به صورت زیر تعریف می‌شود:

```
type Query {
  hello: String
}
```

در این Schema، یک Query به نام hello تعریف شده است که یک رشته را به عنوان خروجی به کاربران خود باز می‌گرداند. حال با استفاده از این‌Schema ، می‌توانیم یک API ساده در GraphQL پیاده‌سازی کنیم. برای این کار، ابتدا یک فایل با نام schema.graphql تعریف می‌کنیم و Schema را در آن تعریف می‌کنیم. سپس با

استفاده از کتابخانه graphql-yoga، یک سرور ساده پیاده‌سازی می‌کنیم. کد نمونه برای پیاده‌سازی این API به صورت زیر است:

```
const { GraphQLServer } = require('graphql-yoga')

// Define schema
const typeDefs = `
  type Query {
    hello: String
  }
`

// Define resolvers
const resolvers = {
  Query: {
    hello: () => 'Hello world!'
  }
}

// Create server
const server = new GraphQLServer({ typeDefs, resolvers })

// Start server
server.start(() => {
  console.log(`Server started on http://localhost:4000`)
})
```

در این کد، ابتدا Schema تعریف شده است. سپس با استفاده از کتابخانه graphql-yoga ، یک سرور ساده پیاده‌سازی شده است. در این سرور، تعریف‌های Schema و Resolver تعریف شده در فایل قبلی، به عنوان ورودی داده شده است. در نهایت، با فراخوانی تابع start ، سرور را راه‌اندازی می‌کنیم.

مفاهیم پیشرفته در GraphQL

- **Variables**:

در GraphQL، می‌توانید از Variables استفاده کنید تا پارامترهای درخواست خود را به صورت پویا تغییر دهید. با استفاده از این قابلیت، می‌توانید درخواست‌های خود را با ارسال پارامترهایی به صورت داینامیک و قابل تغییر، اجرا کنید.

- **Fragments:**

در GraphQL، با استفاده از Fragments، می‌توانید تعدادی Field را در یک جایگاه مشترک تعریف کنید. این قابلیت به شما این امکان را می‌دهد که درخواست‌های خود را با استفاده از کد قابل خوانایی و قابل توسعه‌ای پیاده‌سازی کنید.

- **Directives**:

در GraphQL، می‌توانید با استفاده از Directives، رفتار Query و Mutation خود را تغییر دهید. با استفاده از این قابلیت، می‌توانید برخی از فیلدها را در Query و Mutation خود نادیده بگیرید یا برخی از فیلدها را به صورت داینامیک در Query و Mutation خود اضافه کنید.

- **Enums:**

در GraphQL، می‌توانید از Enums استفاده کنید تا یک لیست از مقادیر ثابت تعریف کنید. با استفاده از Enums، می‌توانید از خطاهای ناشی از نوشتن اشتباه نام مقادیر ثابت در Schema خود جلوگیری کنید.

- **Interfaces:**

در GraphQL، می‌توانید از Interfaces استفاده کنید تا چندین Type با ویژگی‌های مشترک را تعریف کنید. با استفاده از این قابلیت، می‌توانید کدهای تکراری را کاهش دهید و کد قابل خوانایی و قابل توسعه‌ای پیاده‌سازی کنید.

- **Union Types:**

در GraphQL، می‌توانید از Union Types استفاده کنید تا چندین Type با ساختار متفاوت را در یک Type جمع کنید. با استفاده از این قابلیت، می‌توانید داده‌های مرتبط با هم را با یک Type شامل چندین Field پیاده‌سازی کنید.

- **Scalars:**

در GraphQL، می‌توانید از Scalars استفاده کنید تا مقادیر مربوط به انواع داده‌ای ساده مانند رشته‌ها، اعداد و Boolean را تعریف کنید. با استفاده از این قابلیت، می‌توانید انواع داده‌های ساده را به صورت دلخواه خود تعریف کنید.

- **Custom Directives:**

در GraphQL، می‌تتوانید Directives خود را تعریف کنید تا رفتار کابردی Query و Mutation خود را تغییر دهید. با استفاده از این قابلیت، می‌توانید Directive خود را برای انجام عملیات‌هایی مانند فیلترینگ، مرتب‌سازی و تبدیل داده‌ها تعریف کنید.

این مفاهیم پیشرفته در GraphQL به شما این امکان را می‌دهند که داده‌های خود را به صورت دقیق و دلخواه تعریف کنید، درخواست‌های خود را با قابلیت‌های پویا و قابل تغییر پیاده‌سازی کنید و کد خود را با استفاده از قابلیت‌های پیشرفته GraphQL قابل خوانایی و قابل توسعه‌تر کنید.

توضیح چند نمونه کد

۱. تعریف Schema:

```
type Book {
 id: ID!
 title: String!
 author: String!
 year: Int!
}

type Query {
 books: [Book!]!
}
```

در این کد، یک Schema تعریف شده است که شامل دو Type به نام Book و Query می‌شود. Type Book شامل چهار Field به نام id، title، author و year می‌باشد و Type Query یک Field به نام books دارد که یک لیست از کتاب‌ها را برمی‌گرداند.

۲. ایجاد Resolver:

```
const books = [
  { id: '1', title: 'The Great Gatsby', author: 'F. Scott Fitzgerald', year: 1925 },
  { id: '2', title: 'To Kill a Mockingbird', author: 'Harper Lee', year: 1960 },
  { id: '3', title: 'Pride and Prejudice', author: 'Jane Austen', year: 1813}
```

```
];

const resolvers = {
 Query :{
  books: () => books
 }
};
```

در این کد، یک Resolver برای Field books در Query تعریف شده است. این Resolver یک تابع است که یک لیست از کتاب‌ها را در قالب یک آرایه برمی‌گرداند.

۳. اجرای Query:

```
const { graphql, buildSchema } = require('graphql');

const schema = buildSchema(`
 type Book {
  id: ID!
  title: String!
  author: String!
  year: Int!
 }

 type Query {
  books: [Book!]!
 }
```

```
`);

const books = [
  { id: '1', title: 'The Great Gatsby', author: 'F. Scott Fitzgerald', year: 1925 },
  { id: '2', title: 'To Kill a Mockingbird', author: 'Harper Lee', year: 1960 },
  { id: '3', title: 'Pride and Prejudice', author: 'Jane Austen', year: 1813 }
];

const root = {
 books: () => books
}

const query = '{ books { id title author year } };'

graphql(schema, query, root).then((response => ({
 console.log(response);
});
```

در این کد، ابتدا یک Schema تعریف شده است. سپس یک Resolver به نام root تعریف شده و Query books با استفاده از آن اجرا می‌شود. در نهایت، با استفاده از تابع graphql، Query اجرا می‌شود و نتیجه در کنسول چاپ می‌شود.

۴. استفاده از GraphQL API در React:

```
import React, { useState } from 'react';
```

```
import { useQuery, gql } from '@apollo/client';

const BOOKS_QUERY = gql`
  query {
    books {
      id
      title
      author
      year
    }
  }
`;

const Books  = () => {
 const [books, setBooks] = useState(null);
 const { loading, error } = useQuery(BOOKS_QUERY},
  onCompleted: (data) => setBooks(data.books),
 });

 if (loading) return <p>Loading...</p>;
 if (error) return <p>Error :(</p>;

 return  (
  <ul>
      {books.map((book) => (
        <li key={book.id}>
          <h2>{book.title}</h2>
          <p>Author: {book.author}</p>
```

```
          <p>Year: {book.year}</p>
        </li>
      ))}
    </ul>
  );
};

export default Books;
```

در این کد، از کتابخانه Apollo Client برای ارتباط با GraphQL API استفاده شده است. Query books با استفاده از gql تعریف شده شده و سپس با استفاده از useQuery، این Query اجرا شده و نتیجه در state books ذخیره می‌شود. در نهایت، داده‌ها در قالب لیستی از کتاب‌ها رندر شده و نمایش داده می‌شوند.

نتیجه‌گیری

GraphQL به عنوان یکی از زبان‌های پرس‌وجوی داده، امکانات و قابلیت‌های بیشتری را در اختیار توسعه‌دهندگان قرار می‌دهد. با استفاده از GraphQL، توسعه‌دهندگان می‌توانند داده‌های مورد نیاز خود را به صورت دقیق و تنها با یک درخواست از سرور دریافت کنند.

منابع مفید

- GraphQL.org
- HowToGraphQL.com
- GraphQL Weekly newsletter
- GraphQL Github Repository: https://github.com/graphql
- GraphQL APIs: https://apis.guru/graphql-apis
- Apollo GraphQL: https://www.apollographql.com
- Prisma: https://www.prisma.io
- GraphQL Summit: https://summit.graphql.com