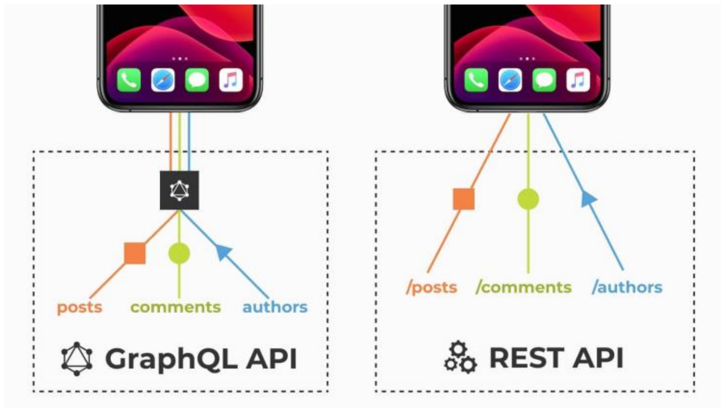
GraphQL Overview and Imlpementation

GraphQL benefits over Rest.

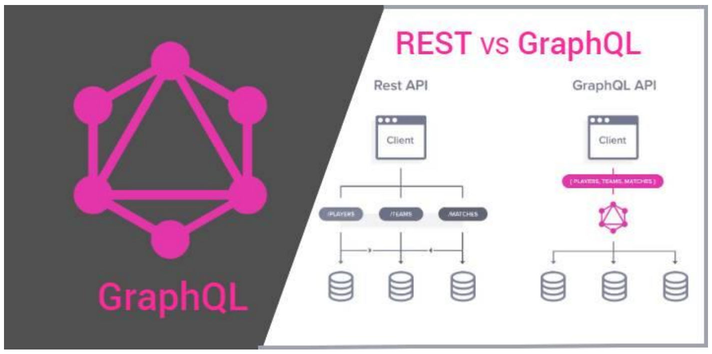
- The main advantage of GraphQl over REST is that REST responses contain too much data or sometimes not enough data, which creates the need for another request. GraphQL solves this problem by fetching only the exact and specific data in a single request.



- GraphQL is faster.
- Best for complex systems and microservices. We can integrate multiple systems behind GraphQL's API. It unifies them and hides their complexity.

GraphQL drowbacks over Rest.

- It's a simple query language. When we pass on a request with too many nested fields data at a single time, the complexity of the query increases and makes the request inefficient and heavy.



- Caching in GraphQL is more complicated to implement than Rest.
- Another problem with GraphQL is rate-limiting. In REST API, you can simply specify that we allow only this amount of requests in one day", but in GraphQL, it is difficult to specify this type of statement.

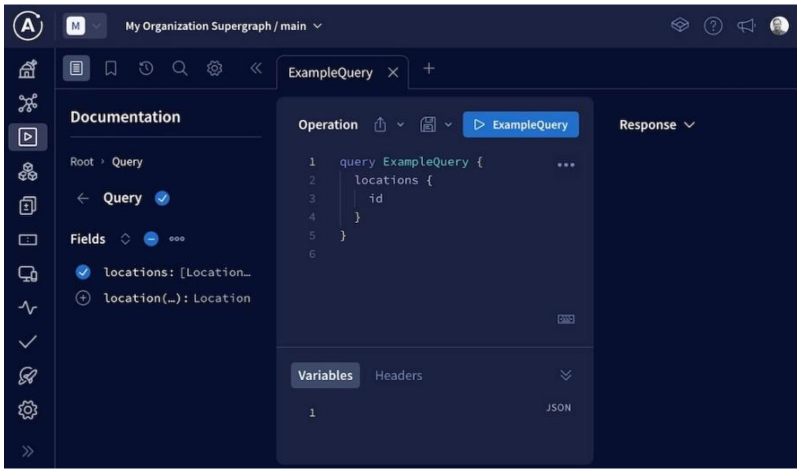
https://www.pluralsight.com/guides/how-to-set-up-graphql-in-a-react-app

Ways to implement GraphQL in a react project/app.

- **Using Apollo Client**: Apollo Client is a popular GraphQL client library that integrates seamlessly with React. It provides React hooks and components to interact with GraphQL APIs.
- **Relay Modern**: Developed by Facebook, Relay Modern is another GraphQL client for React applications. It offers advanced optimizations and a declarative approach to fetching data.
- **Using Fetch API**: You can use the Fetch API to make GraphQL queries and mutations directly. However, this approach might require more manual handling of data and caching compared to dedicated GraphQL clients.
- **React-Apollo**: If you want to use Apollo Client, you can also use the react-apollo package, which provides higher-order components (HOCs) and render props to integrate GraphQL with your React components.
- **React Hooks**: With the introduction of React Hooks, you can write custom hooks to encapsulate GraphQL logic and easily use them within your components.
- **Graphql-request**: A lightweight GraphQL client that works with React and can be used with React Hooks or traditional lifecycle methods.
- **URQL**: A highly customizable and lightweight GraphQL client that works well with React and other frameworks.
- **Apollo Boost**: A simpler version of Apollo Client that includes some preconfigured settings, ideal for smaller projects or quick prototyping.
- **Code-First Approach**: You can use libraries like type-graphql or graphql-codegen to generate GraphQL types and operations directly from your TypeScript or JavaScript code.
- **Apollo Server + React**: If you are building a full-stack application, you can use Apollo Server for your GraphQL backend and Apollo Client in your React frontend.
- **Express-GraphQL**: enables developers to easily implement and serve GraphQL APIs within their Node.js applications, making it a popular choice for building efficient and flexible server-side data APIs in combination with the powerful capabilities of GraphQL.

Of these ways to implement graphql the most used is Apollo-Client. That is because of its:

- Efficient Data Fetching
- Real-time Data with GraphQL Subscriptions
- Powerful DevTools



```
query ExampleQuery {
  locations {
    id
  }
}
```

locations: [Location…
location(…): Location

A complete getting started guide can be found in the following link.

https://www.apollographql.com/docs/react/get-started

Although Apollo Client is the most commonly used, in this project the Express-GraphQL way was used.

It utilizes the graphical environment provided by graphql and the express server service of node.js.

Prerequisites: node.js installed

Firstly open terminal and type: *npm init -y*

(this starts a project in react)

After that: *npm install express express-graphql graphql --save*

Inside the folder of the project create a server.js file and type the code as follows:

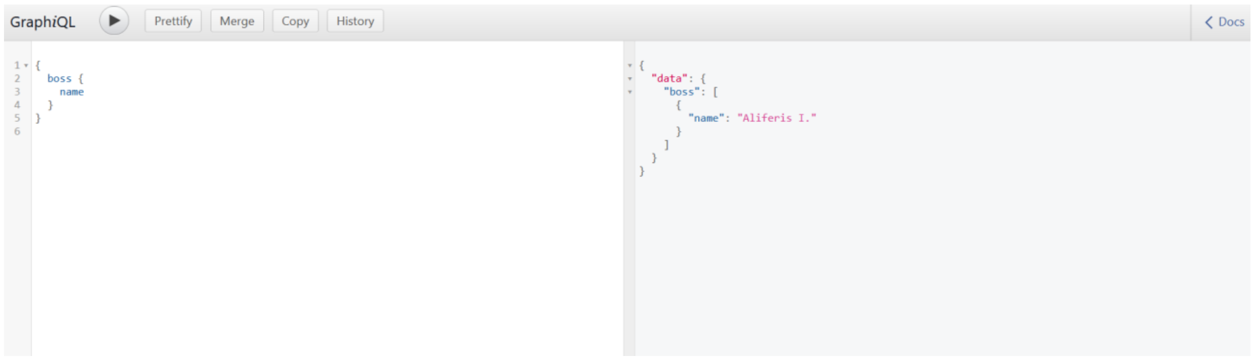
```
const express = require('express');
const { graphqlHTTP } = require('express-graphql');
const { buildSchema } = require('graphql');

// Create a server:
const app = express();

// Create a schema and a root resolver:
const schema = buildSchema(`
    type Boss {
        name: String!
        possition: String!
    }

    type Intern{
        name: String!
        possition: String!
    }

    type Query {
        interns :[Intern],
        boss: [Boss]
    }
`);

const rootValue = {
    boss:[
        {
            name: "Aliferis I.",
            position: "Project coordinator",
        }
    ],
    interns: [
        {
            name: "Erasmia T.",
            position: "Business Plan",
        },
        {
            name: "Ilias D.",
            position: "Dev-Ops",
        },
        {
            name: "Maritina F.",
            position: "Business Plan",
        },
        {
            name: "Tataridis P.",
            position: "Front-End",
        }
    ]
};

// Use those to handle incoming requests:
app.use(graphqlHTTP({
    schema,
    rootValue,
    graphiql: true,
}));

// Start the server:
app.listen(4000, () => console.log("Server started on port 4000"));
```

Finally open the terminal, navigate to the folder path and type: *node server.js*

You should see the response: Server started on port 4000

Go to your preferred browser and enter http://localhost:4000

The Graphical ui will appear. There you will be able to see all the queries you can make to your schema.



```
{
  boss {
    name
  }
}
```

```
{
  "data": {
    "boss": [
      {
        "name": "Aliferis I."
      }
    ]
  }
}
```