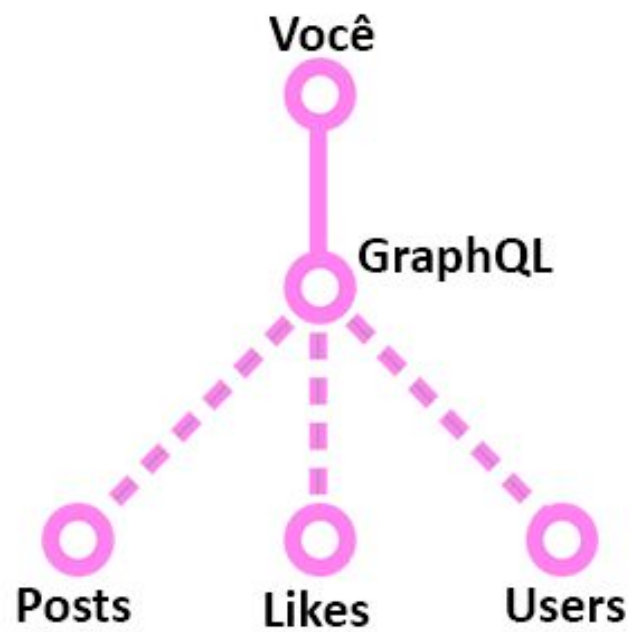
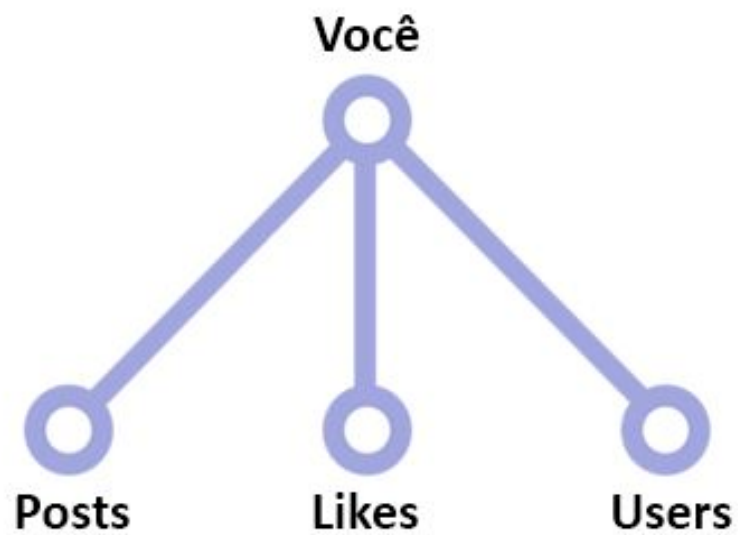


GraphQL

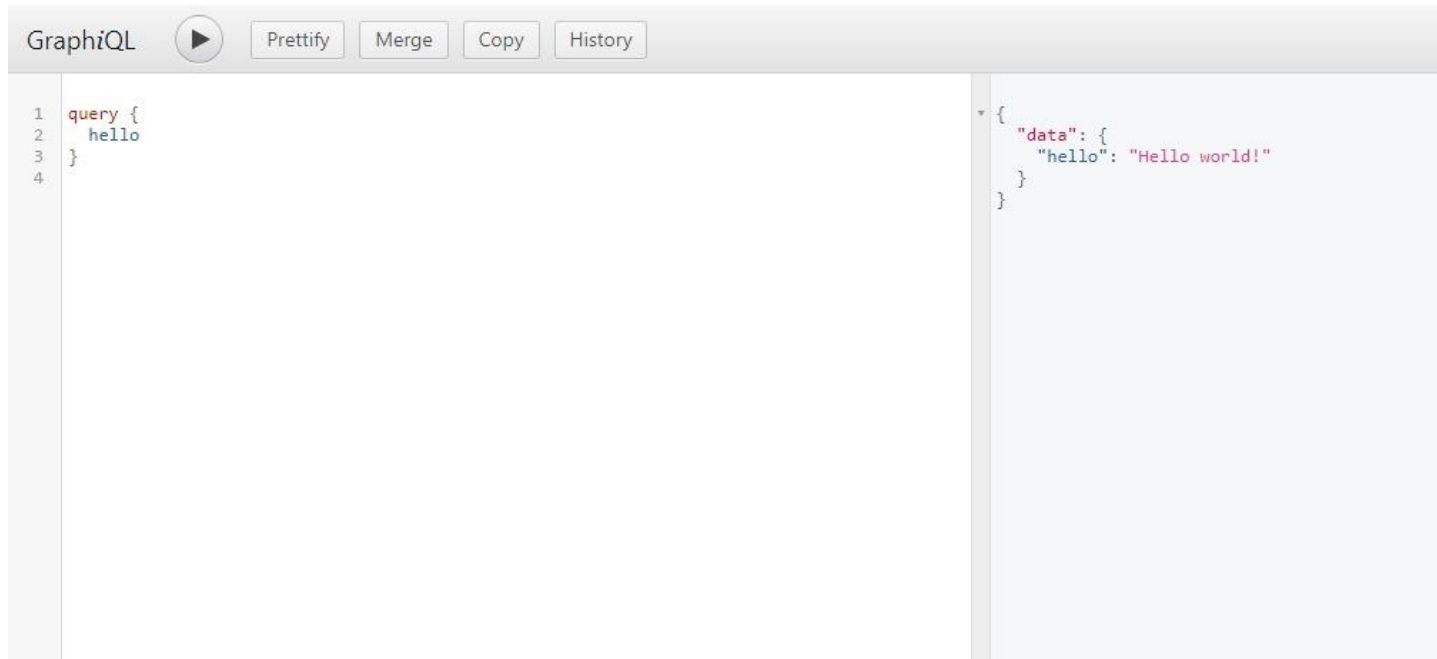
GraphQL

- GraphQL significa **Graph Query Language** linguagem de query assim como SQL (Structured Query Language) porém seu uso não envolve implementar banco de dados, mas sim, **definir dados** seja para API ou servidor.
- É uma linguagem de consulta e ambiente de execução voltada a servidores para as interfaces de programação de aplicações (APIs) cuja prioridade é fornecer exatamente os dados que os clientes solicitam e nada além.



GraphiQL

Interface gráfica de servidor GraphQL que pode ser usado para fazer consultas, pode ser desabilitada caso desejado.



Criação do GraphQL

Foi desenvolvido pelo Facebook, que foi o primeiro a usá-lo para aplicações mobile em 2012. Dessa forma, a linguagem oferece a habilidade de modelar dados usando schemas. A especificação do GraphQL foi transformada em open source em 2015. Agora, ela é supervisionada pela GraphQL Foundation.

schema

```
type User {  
  id: ID  
  
  name: String  
  
  email: String  
  
  bio: String  
  
}
```

Tipos do GraphQL

Os tipos são usados na definição de Schemas e na chamada de variáveis.

- **Int**: Número Inteiro.
- **Float**: Número com casas decimais, com duas casas decimais de precisão..
- **String**: Sequência de caracteres UTF-8.
- **Boolean**: true or false.
- **ID**: O tipo ID representa um identificador exclusivo, geralmente usado para buscar novamente um objeto.

Resolvers

Uma vez definido o Schema estaria pendente as funções que são chamadas para realmente executar os campos definidos, tais funções seriam os resolvers, destacando que a consulta realizada por usuários é através de tais resolvers.

DEFININDO RESOLVERS

```
extend type Query {  
  users: [User]  
  login(email: String!, password: String!): User  
}
```

CHAMANDO RESOLVERS USERS

```
{  
  users {  
    id,  
    nome,  
    email,  
    password  
  }  
}
```

[] - O atributo entre colchetes representa que o mesmo é um array.

Query e Mutation

Dados de um servidor GraphQL são consumidos através de queries feitas pelo cliente. Assim como query é usada para **buscar** dados, mutation é usada para **criar, alterar ou deletar** dados do banco de dados, tais Query's e Mutation's usam os dados através de resolvers explicados anteriormente.

OBS: A **Query** é o padrão de qualquer consulta, então caso uma consulta não possua variáveis a palavra query no início pode ser removida.

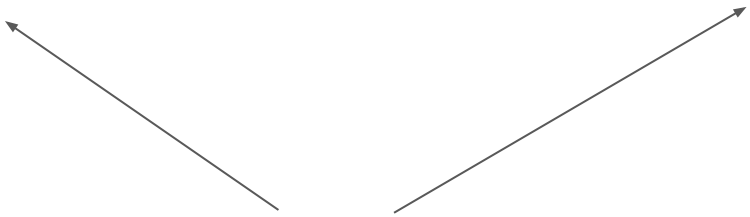
Query's

READ ALL

```
{  
  users {  
    id  
    name  
    email  
  }  
}
```

READ BY ID

```
query {  
  userById (id: 1) {  
    id,  
    name,  
    email  
  }  
}
```



Como ambos não usam variáveis (vão ser explicados mais à frente) o uso da palavra query não interfere no resultado

Mutation's

CREATE

```
mutation {  
  createUser(name: "João", email: "...", bio: "...") {  
    id  
    name  
    email  
    bio  
  }  
}
```

UPDATE

```
mutation {  
  updateUser(id: "123", ...) {  
    id  
    name  
    email  
    bio  
  }  
}
```

DELETE BY ID

```
mutation {  
  deleteUser(id: "123")  
}
```

Usando Variáveis

Variáveis são usadas no GraphQL com intuito de facilitar o uso dos dados.

CONSULTA

```
query ($id: Int!) {  
  userById(id: $id) {  
    id,  
    name,  
    email  
  }  
}
```

VARIÁVEIS

```
{ "id": 1 }
```

Usando Variáveis

As exclamações (!) após o tipo da variável informam que a mesma é obrigatória, caso não seja setada nas variáveis irá provocar erro.

CONSULTA

```
mutation ($name: String!, $email: String!, $password: String!) {  
  addUser(name: $name, email: $email, password: $password) {  
    id,  
    name,  
    email,  
    password  
  }  
}
```

VARIÁVEIS

```
{  
  "name": "Aluno",  
  "email": "aluno@gmail.com",  
  "password": "123"  
}
```

Usando Variáveis

Resultado da consulta mostrada no slide anterior.

GraphiQL  Prettify Merge Copy History

```
1 ▾ mutation ($name: String!, $email: String!, $password: String!) {  
2 ▾   addUser(name: $name, email: $email, password: $password) {  
3     id,  
4     name,  
5     email,  
6     password  
7   }  
8 }
```

QUERY VARIABLES

```
1 ▾ {  
2   "name": "Aluno",  
3   "email": "aluno@gmail.com",  
4   "password": "123"  
5 }
```

```
▾ {  
▾   "data": {  
▾     "addUser": {  
       "id": 29,  
       "name": "Aluno",  
       "email": "aluno@gmail.com",  
       "password": "123"  
     }  
   }  
}
```

A **P O L L O**

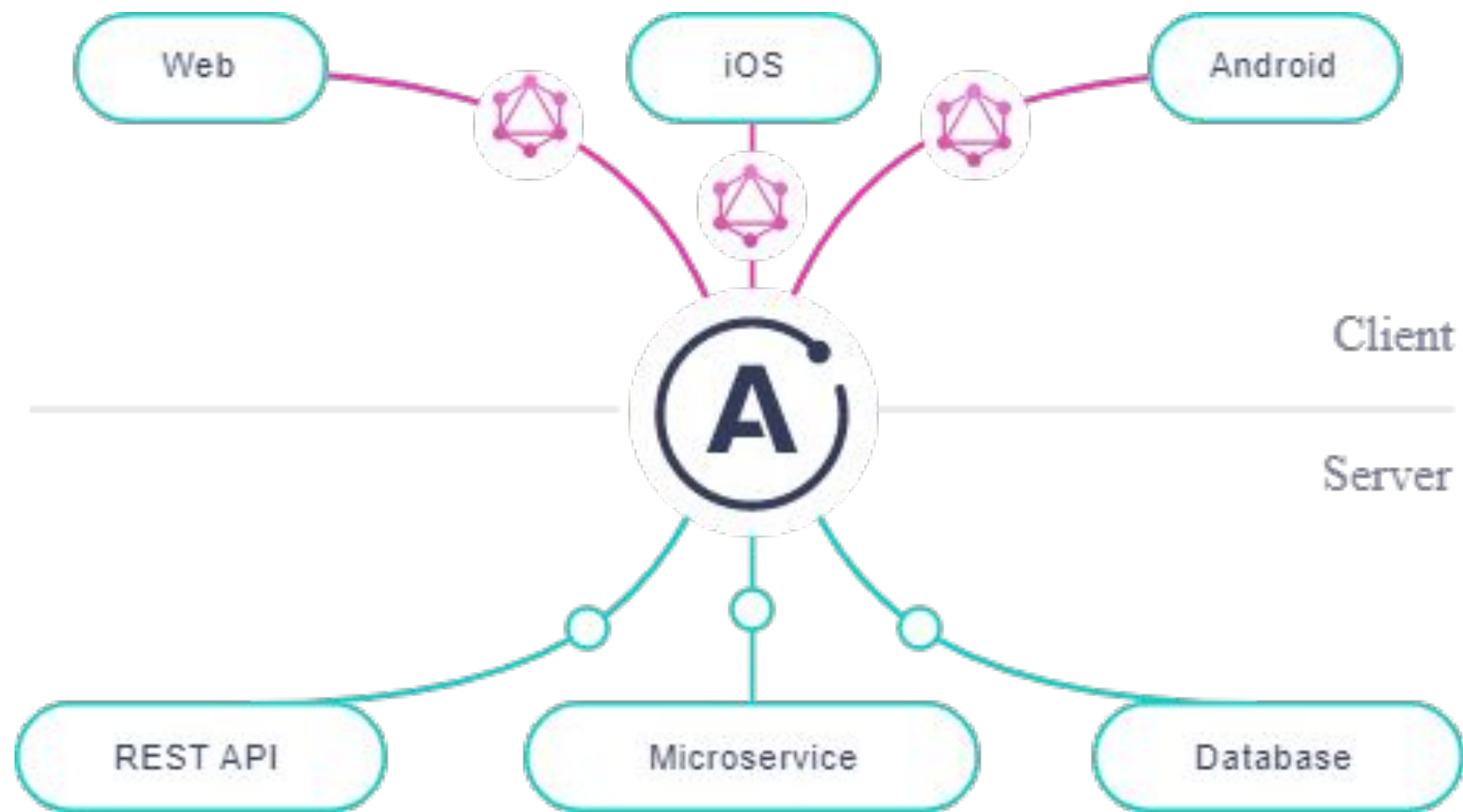
Apollo

É um conjunto de ferramentas e produtos criados pelo time de desenvolvimento do Meteor para trabalhar com GraphQL.

A implementação líder de GraphQL de código aberto com 17 milhões de downloads por mês e a única solução de gerenciamento de nuvem de ponta a ponta para GraphQL

Apollo

O Apollo rastreia seus esquemas GraphQL em um registro para criar uma fonte central de verdade para tudo em seu supergrafo. O Studio permite que os desenvolvedores explorem dados, colaborem em consultas, observem o uso e forneçam alterações de esquema com agilidade e confiança.



Exemplo de implementação de CRUD

API (express-graphql + pg-promise):

- Github: <https://github.com/lukkerr/GraphQL-Example2>
- Aplicação Hospedada: <http://graphql-example2.herokuapp.com/graphql>

Front-end (angular + apollo-angular):

- Github: <https://github.com/lukkerr/Apollo-Client-Graphql>
- Aplicação Hospedada: <http://apollo-client-graphql.vercel.app/>

Hello World

Site Oficial: <https://graphql.org/code/>

Instalando GraphQL (implantação express-graphql):

1. Tutorial de instalação do GraphQL
2. Tutorial de criação de Hello World

Exemplo de Hello World:

- Github: <https://github.com/WillmaTayanne/GraphQL-Example1>
- Aplicação: <https://graphql-example1.herokuapp.com/graphql>