

Graphql - breve guida

February 17, 2020

A cosa serve

È un linguaggio per API, permette di

- definire e esporre
- interrogare

delle API, in modo database-agnostic e permettendo di:

- annidare più livelli di query
- utilizzare paginazione, filtri sui risultati
- eseguire operazioni in realtime
- integrarsi con servizi di terze parti, anche legacy

Pre-requisiti

- Node.js
-

Creare un server GraphQL e interrogarlo

Per far partire un server GraphQL è necessario:

- avere un server Node.js

```
yarn init -y
mkdir src
touch src/index.js
```

- server \Rightarrow graphql(-yoga)

```

const {GraphQLServer} = require(  graphql  - y o g a  )
... //data
const resolvers = {
  ...
}
const server = new GraphQLServer({
  typeDefs:      ./src/schema.graphql ,
  resolvers,
})
server.start(() => console.log('Server is running on
http://localhost:4000'))

```

- fornire al server la definizione dell'API (GraphQL schema, scritto in SDL), file schema.graphql:

```

type Query {
  field: type+modifiers
  #-->i field possono essere di tipo scalare o oggetto
  , anche Custom
}
type Mutation {
  operation(par1: type+modifiers, par2:      ): return
  type+modifiers
}
type Subscription {
  ???
}
type Custom {
}

```

- fornire l'implementazione delle operazioni (resolver), file index.js:

```

const resolvers = {
  Query: {
    field: (param) => return obj
  }

  Custom: {
    field: (param) => return,
  }

  Mutation: {
    operation: (parent, args) => {

      return
    }
  }
}

```

- Accedendo al server è poi possibile interrogarlo in GraphQL (una richiesta per scheda, su GraphQL Playground):

```
query {  
  field {  
    fields voluti,  
  }  
}  
  
mutation {  
  post(  
    param1:  
    param2:  
  ) {  
    id  
  }  
}
```

Utilizzare Prisma per collegare un server GraphQL and un database

Prisma è un mezzo di collegamento tra il server GraphQL (\rightarrow js) e il database, autogenerato a partire da una descrizione dei dati dell'app e un collegamento ad un database.

Set-up

Comandi:

```
#creare i file di configurazione necessari  
mkdir prisma  
touch prisma/prisma.yml #config  
touch prisma/datamodel.prisma #data model dell'app  
  
#installare le dipendenze per Prisma  
yarn global add prisma  
yarn add prisma-client-lib  
  
#connettere ad un database  
prisma deploy #e seguire le istruzioni  
  
#far partire il client  
prisma generate
```

Se il datamodel viene aggiornato è sufficiente ri-eseguire `prisma deploy` per far ripartire il server (aggiornato) senza dover manualmente aggiornare il client con `generate`.

Connettere il server GraphQL a Prisma (index.js):

```
import {prisma} from "../generated/prisma-client"
...
const server = new GraphQLServer({
  typeDefs,
  resolvers,
  context: {prisma},
})
```

App data model

```
type Link {
  id: ID! @id
  createdAt: DateTime! @createdAt
  description: String!
  url: String!
}
```

Quelle precedute da @ sono direttive che comandano a GraphQL di autogenerare e gestire i campi (che quindi saranno in sola lettura) secondo certe funzionalità. Prisma fornisce inoltre la funzione `context.prisma.$exists.model(where)` che, per ogni modello (entità, `type` del datamodel), permette di sapere se esiste un elemento che rispetti la condizione `where`.

Configurazione

`prisma deploy` genera un Prisma client in base al data model indicato in `prisma.yml`:

```
#endpoint http per le API Prisma
endpoint: '' #verrà aggiornato automaticamente da prisma

#file che contiene il data model
datamodel: datamodelprisma

#linguaggio e posizione del client Prisma generato:
generate:
  - generator: javascript-client
    output: ../src/generated/prisma-client #esempio
```

Richiedere i dati tramite Prisma (resolver)

Ogni funzione GraphQL riceve **4 parametri**:

- **root**: parent, il risultato della precedente chiamata resolver
- **args**: argomenti veri e propri della funzione
- **context**: contesto di esecuzione dei resolver, oggetto in cui ogni resolver può leggere e scrivere

- info

Context contiene anche un'istanza del client Prisma: `context.prisma`

```
const resolvers = {
  Query: {
    info: () => 'This is the API of a Hackernews Clone',
    feed: (root , args , context , info) => {
      return context.prisma.links() },
  },
  Mutation: {
    post: (root, args, context) => { return context.
prisma.createLink({
      url: args.url,
      description: args.description , })
    },
  },
}
```

Prisma autogenera operazioni CRUD per ogni oggetto, e fornisce anche una sua interfaccia per visualizzare il risultato dell'esecuzione di queste operazioni (→ i dati del db) all'indirizzo <https://app.prisma.io/<USERNAME>/services>.