Graphene Documentation

Release 1.0

Syrus Akbary

Mar 22, 2018

Contents

Contents:

CHAPTER 1

Introduction tutorial - Graphene and Sequelize

Graphene has a number of additional features that are designed to make working with Sequelize *really simple*.

Our primary focus here is to give a good understanding of how to connect models from Sequelize to Graphene object types.

A good idea is to check the graphene documentation first.

- Node or Typescript(any)
- Graphene-JS

Project setup

```
yarn add graphene-sequelize
# or
npm install graphene-sequelize
```

Defining our models

Let's get started with these models:

```
import * as Sequelize from "sequelize";

// We define the Sequelize Models
export const sequelize = new Sequelize('database', 'username', 'password', {
    dialect: 'sqlite',
    storage: 'db.sqlite',
});

export const Project = sequelize.define('project', {
```

```
    title: Sequelize.STRING,
    description: Sequelize.TEXT
})

export const Task = sequelize.define('task', {
    title: Sequelize.STRING,
    description: Sequelize.TEXT,
    deadline: Sequelize.DATE
})
```

GraphQL presents your objects to the world as a graph structure rather than a more hierarchical structure to which you may be accustomed. In order to create this representation, Graphene needs to know about each *type* of object which will appear in the graph.

This graph also has a *root type* through which all access begins. This is the `Query` class below.

This means, for each of our models, we are going to create a type, decorating with `SequelizeObjectType`

After we've done that, we will list those types as fields in the `Query` class.

```
import { ObjectType, Field, NonNull } from "graphene-js";
import { SequelizeObjectType } from "graphene-sequelize";

@SequelizeObjectType({model: Project})
class ProjectType {
    // We can add here additional files
}

@SequelizeObjectType({model: Task})
class TaskType {
    // We can add here additional files
}

@ObjectType()
class Query {
    @Field(ProjectType, { args: {id: NonNull(String)}})
    getProject({id}) {
        return Project.findById(id);
    }
}

schema = new Schema({query: Query})
```

Querying

Then we can start querying our schema:

```
var result = await schema.execute('{ getProject(id: "1") { id, title } }')
console.log(result.data.getProject)
```

Congrats! You got your first graphene sequelize schema working!