Graphql:

Query language (alternate for rest services) that helps to fetch data from db to client application.

Facebook framework.

With Low internet connection you can retrieve the data fast since it can get the particular relevant data rather than bringing the whole table or mongodb object.

They control the data that they get not the server so they are fast and stable.

They are organised in terms of types and fields not endpoints.

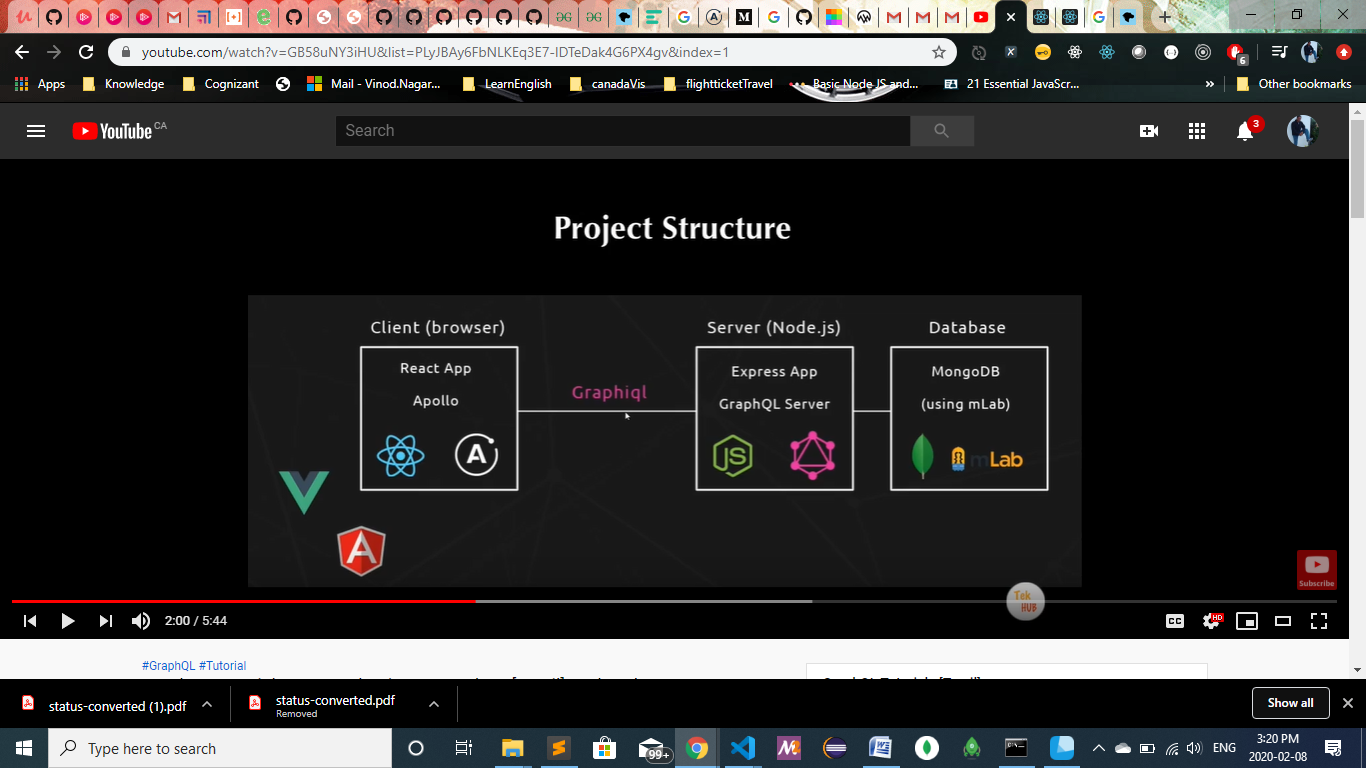
Can access the full capabilities of data from single endpoint.

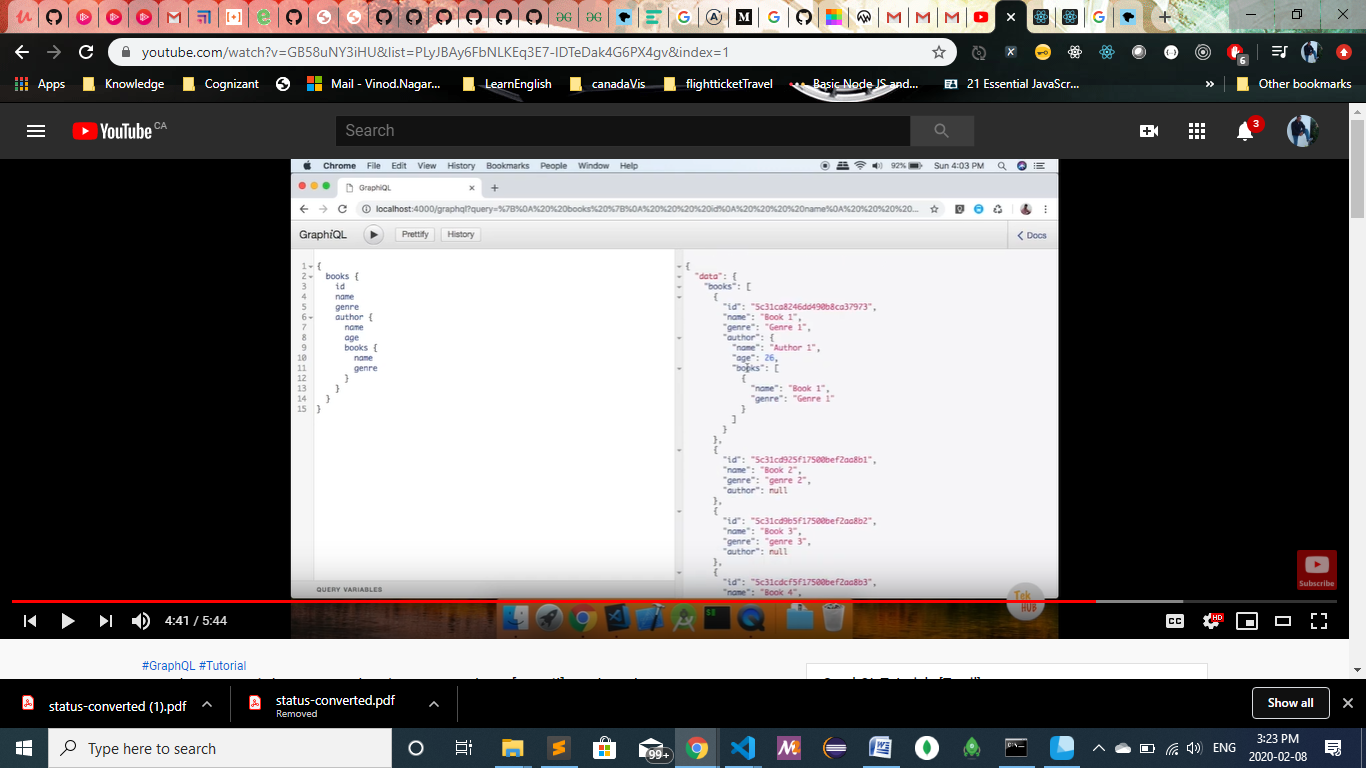
Can use types to avoid writing manual parsing code.

We basially write the schema and relational logics inside the graphql.

Schema : barebone of how the data looks.

Graphiql:one of the interface , playground for querying





2) setting up of expressjs

npm init -y

npm install express --save

npm install -g nodemon

app.js

const express = require('express');

const app = express();

app.listen(4000, () => (

    console.log('Server listening on port 4000')

))

npx is the alternate for script tags wherein you are mentioning it runs under your node modules locally .

3)Graphql setup with schema:

npm install graphql express-graphql --save

app.js

const express = require('express');

const graphqlHttp = require('express-graphql');

const app = express();

app.use('/graphql', graphqlHttp({}));

app.listen(4000, () => (

    console.log('Server listening on port 4000')

))

schema->schema.js

const graphql = require('graphql');

const { GraphQLObjectType ,GraphQLString} = graphql;

const BookType = new GraphQLObjectType({

    name: 'Book',

    fields: () => ({

        id: (type: GraphQLString),

        name: (type: GraphQLString),

        genre: (type:GraphQLString)

    })

});

Schema has name & fields. Fields is mentioned in function because it might be dependent on some other field which will be declared in the future and is in turn dependent on this. (interdependent)

# 4)  GraphQL Query

rootQuery: how the user is going to interact with the schema. (basically equivalent to get request)

App.js

const express = require('express');

const graphqlHttp = require('express-graphql');

const schema = require('./schema/schema');

const app = express();

app.use('/graphql', graphqlHttp({

    schema: schema,

    graphiql:true

}));

app.listen(4000, () => (

    console.log('Server listening on port 4000')

))

Schema->schema.js

const graphql = require('graphql');

const { GraphQLObjectType, GraphQLString, GraphQLSchema } = graphql;

const \_ = require('lodash');

//dummy data

var books = [

    { name: 'book1', genre: 'genre 1', id: '1' },

    { name: 'book2', genre: 'genre 2', id: '2' },

    { name: 'book3', genre: 'genre 3', id: '3' },

    { name: 'book4', genre: 'genre 4', id: '4' }

];

const BookType = new GraphQLObjectType({

    name: 'Book',

    fields: () => ({

        id: {type: GraphQLString},

        name: {type: GraphQLString},

        genre: {type:GraphQLString}

    })

});

const RootQuery = new GraphQLObjectType({

    name: 'RootQueryType',

    fields: {

        book: {

            type: BookType,

            args: { id: { type: GraphQLString }},

            resolve(parent, args) {

                //code to get data from db/other source

                return books.find(book => book.id === args.id);

               // return \_.find(books, { id: args.id });

            }

        }

    }

});

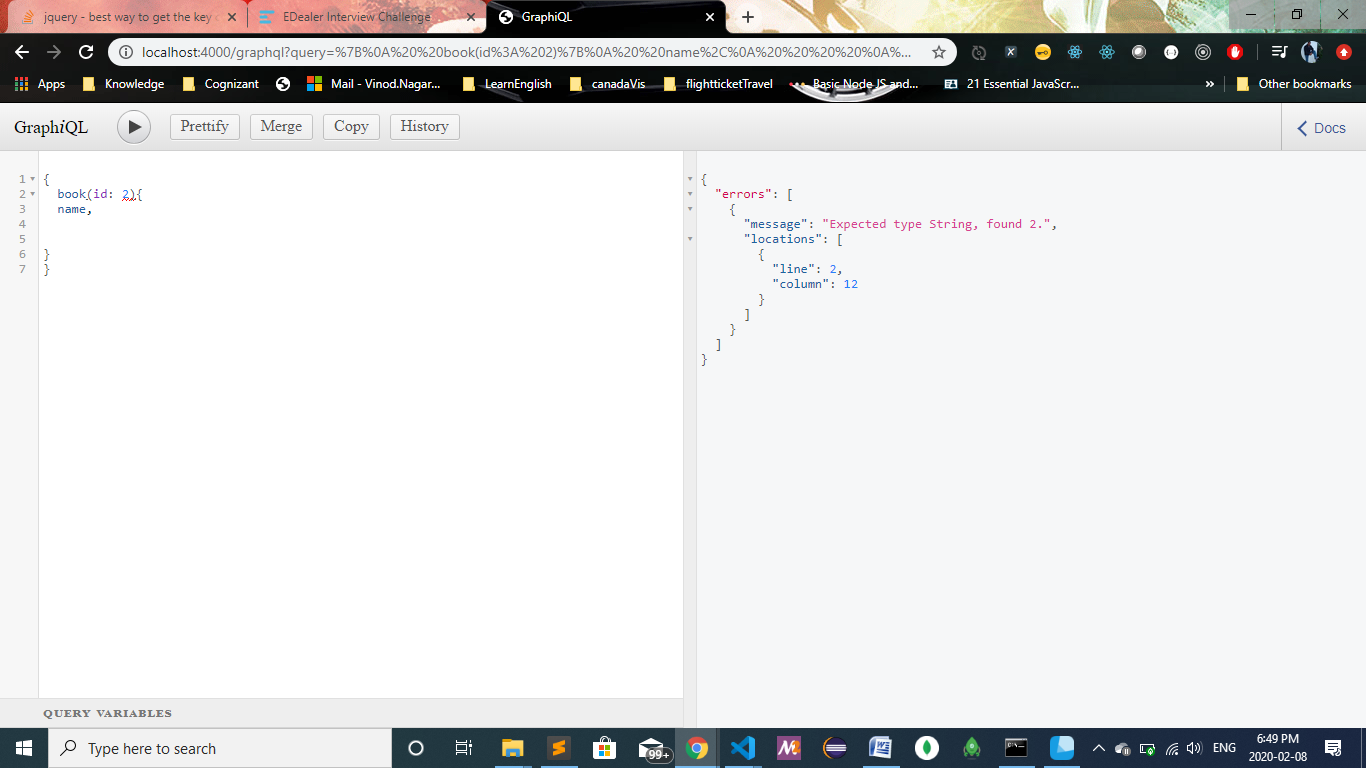
module.exports = new GraphQLSchema({

    query: RootQuery

    //query for get and mutations for insert

});

5) Type Relations & GraphQL List



For type conversion :

We have GraphQLId for parsing the values to string.

Schema->schema.js

const graphql = require('graphql');

const { GraphQLObjectType, GraphQLString, GraphQLSchema ,GraphQLID} = graphql;

const \_ = require('lodash');

//dummy data

var books = [

    { name: 'book1', genre: 'genre 1', id: '1' },

    { name: 'book2', genre: 'genre 2', id: '2' },

    { name: 'book3', genre: 'genre 3', id: '3' },

    { name: 'book4', genre: 'genre 4', id: '4' }

];

const BookType = new GraphQLObjectType({

    name: 'Book',

    fields: () => ({

        id: {type: GraphQLID},

        name: {type: GraphQLString},

        genre: {type:GraphQLString}

    })

});

const RootQuery = new GraphQLObjectType({

    name: 'RootQueryType',

    fields: {

        book: {

            type: BookType,

            args: { id: { type: GraphQLID }},

            resolve(parent, args) {

                //code to get data from db/other source

                return books.find(book => book.id === args.id);

               // return \_.find(books, { id: args.id });

            }

        }

    }

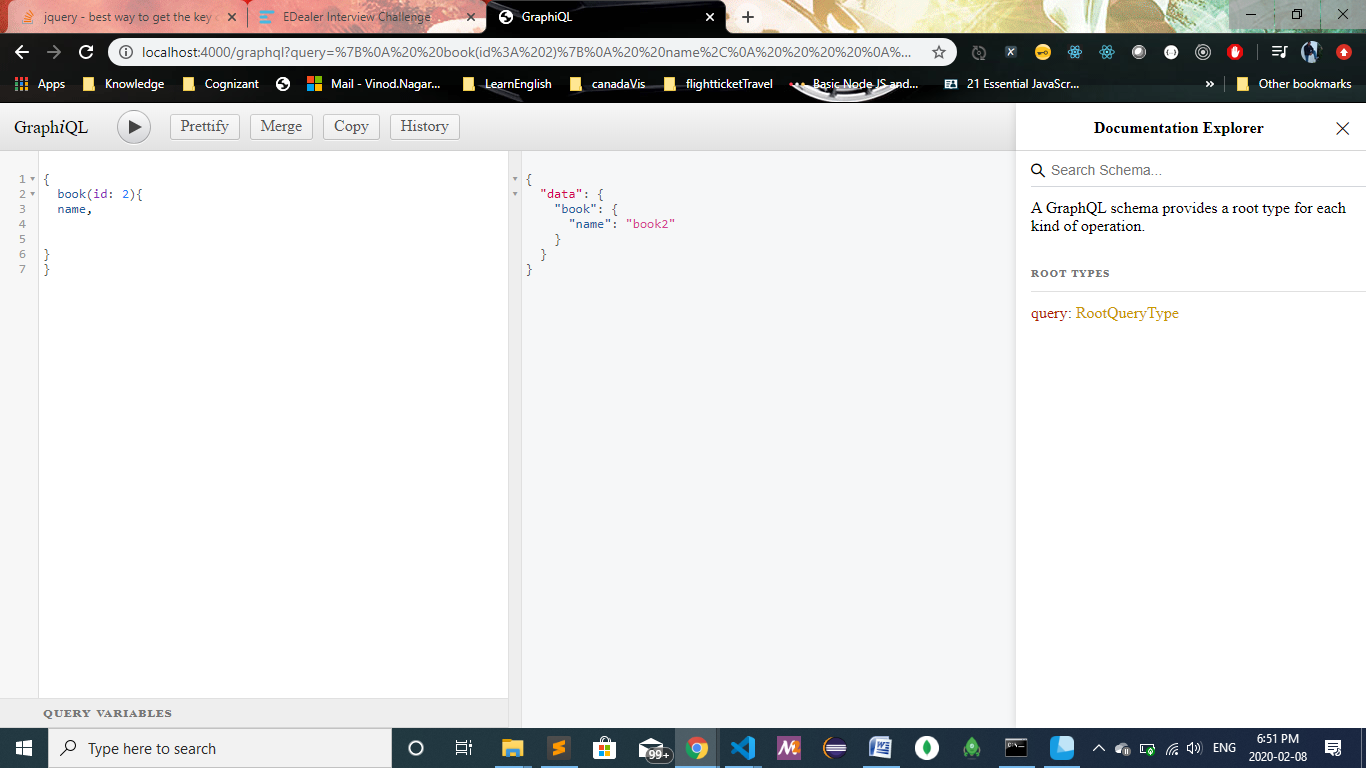
});

module.exports = new GraphQLSchema({

    query: RootQuery

    //query for get and mutations for insert

});



const graphql = require('graphql');

const { GraphQLObjectType, GraphQLString, GraphQLSchema ,GraphQLID,GraphQLInt,GraphQLList} = graphql;

const \_ = require('lodash');

//dummy data

let books = [

    { name: 'book1', genre: 'genre 1', id: '1',authorId:'1' },

    { name: 'book2', genre: 'genre 2', id: '2' ,authorId:'2' },

    { name: 'book3', genre: 'genre 3', id: '3', authorId: '3' },

    { name: 'book4', genre: 'genre 4', id: '4', authorId: '2' },

    { name: 'book5', genre: 'genre 5', id: '5', authorId: '3' },

    { name: 'book6', genre: 'genre 6', id: '6' ,authorId:'3' },

];

let authors = [

    { name: 'author 1', age: '25', id: '1' },

    { name: 'author 2', age: '26', id: '2' },

    { name: 'author 3', age: '27', id: '3' }

];

const BookType = new GraphQLObjectType({

    name: 'Book',

    fields: () => ({

        id: {type: GraphQLID},

        name: {type: GraphQLString},

        genre: { type: GraphQLString },

        author: {

            type: AuthorType,

               resolve(parent, args) {

                return authors.find(author=>author.id===parent.authorId)

            }

        }

    })

});

const AuthorType = new GraphQLObjectType({

    name: 'Author',

    fields: () => ({

        id: { type: GraphQLID },

        name: { type: GraphQLString },

        age: { type: GraphQLInt },

        books: {

            type: new GraphQLList(BookType),

            resolve(parent, args) {

                console.log(parent.id);

                console.log(books);

                return books.filter(book => book.authorId === parent.id);

            }

        }

    })

});

const RootQuery = new GraphQLObjectType({

    name: 'RootQueryType',

    fields: {

        book: {

            type: BookType,

            args: { id: { type: GraphQLID }},

            resolve(parent, args) {

                //code to get data from db/other source

                return books.find(book => book.id === args.id);

               // return \_.find(books, { id: args.id });

            }

        },

        author: {

            type: AuthorType,

            args: { id: { type: GraphQLID } },

            resolve(parent, args) {

                return authors.find(author => (args.id === author.id));

            }

        },

        books: {

            type: new GraphQLList(BookType),

            resolve(parent, args) {

                return books;

            }

        },

        authors: {

            type: new GraphQLList(AuthorType),

            resolve(parent, args) {

                return authors;

            }

        }

    }

});

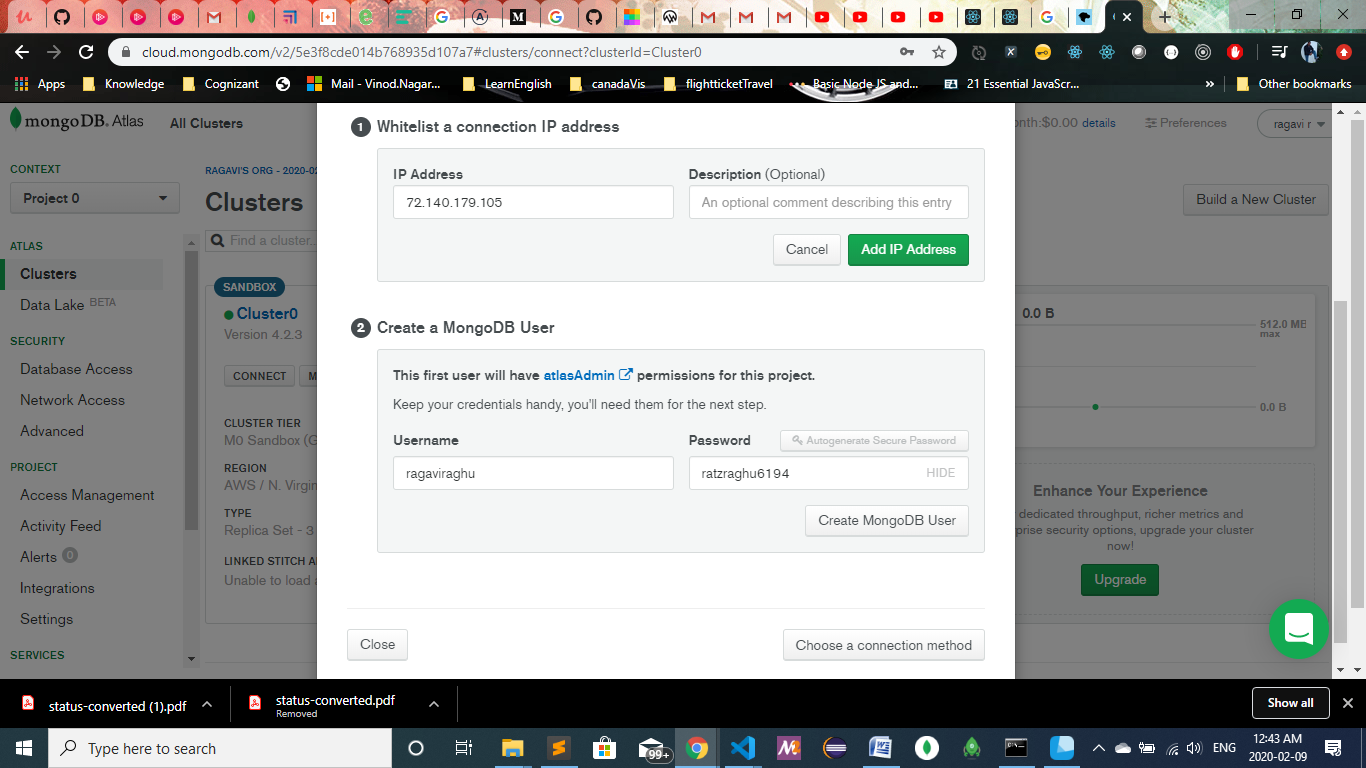
module.exports = new GraphQLSchema({

    query: RootQuery

    //query for get and mutations for insert

});

Mongodb creation in mlab



7)Mongodb Integration with graphql

mongodb+srv://ragaviraghu:ratzraghu6194@cluster0-bp3fh.mongodb.net/test?retryWrites=true&w=majority/testapp

model ->book.js

const mongoose = require('mongoose');

const Schema = mongoose.Schema;

const bookSchema = new Schema({

    name: String,

    authorId: String,

    genre:String

}, {

    writeConcern: {

      w: 'majority',

      j: true,

      wtimeout: 1000

    }

  });

module.exports = mongoose.model('Book', bookSchema);

author.js

const mongoose = require('mongoose');

const Schema = mongoose.Schema;

const authorSchema = new Schema({

    name: String,

age:Number,

}, {

    writeConcern: {

      w: 'majority',

      j: true,

      wtimeout: 1000

    }

  });

module.exports = mongoose.model('Author', authorSchema);

app.js

const express = require('express');

const graphqlHttp = require('express-graphql');

const schema = require('./schema/schema');

const mongoose = require('mongoose');

mongoose.connect('mongodb+srv://ragaviraghu:ratzraghu6194@cluster0-bp3fh.mongodb.net/test?retryWrites=true&w=majority/testapp',{useNewUrlParser:true,useUnifiedTopology: true})

mongoose.connection.once('open', () => {

    console.log('db connected');

})

const app = express();

app.use('/graphql', graphqlHttp({

    schema: schema,

    graphiql:true

}));

app.listen(4000, () => (

    console.log('Server listening on port 4000')

))

schema->schema.js

const graphql = require('graphql');

const { GraphQLObjectType, GraphQLString, GraphQLSchema ,GraphQLID,GraphQLInt,GraphQLList} = graphql;

const \_ = require('lodash');

const Book = require('../model/book');

const Author = require('../model/author');

//dummy data

/\*let books = [

    { name: 'book1', genre: 'genre 1', id: '1',authorId:'1' },

    { name: 'book2', genre: 'genre 2', id: '2' ,authorId:'2' },

    { name: 'book3', genre: 'genre 3', id: '3', authorId: '3' },

    { name: 'book4', genre: 'genre 4', id: '4', authorId: '2' },

    { name: 'book5', genre: 'genre 5', id: '5', authorId: '3' },

    { name: 'book6', genre: 'genre 6', id: '6' ,authorId:'3' },

];

let authors = [

    { name: 'author 1', age: '25', id: '1' },

    { name: 'author 2', age: '26', id: '2' },

    { name: 'author 3', age: '27', id: '3' }

];\*/

const BookType = new GraphQLObjectType({

    name: 'Book',

    fields: () => ({

        id: {type: GraphQLID},

        name: {type: GraphQLString},

        genre: { type: GraphQLString },

        author: {

            type: AuthorType,

               resolve(parent, args) {

               // return authors.find(author=>author.id===parent.authorId)

            }

        }

    })

});

const AuthorType = new GraphQLObjectType({

    name: 'Author',

    fields: () => ({

        id: { type: GraphQLID },

        name: { type: GraphQLString },

        age: { type: GraphQLInt },

        books: {

            type: new GraphQLList(BookType),

            resolve(parent, args) {

                console.log(parent.id);

                console.log(books);

                //return books.filter(book => book.authorId === parent.id);

            }

        }

    })

});

const RootQuery = new GraphQLObjectType({

    name: 'RootQueryType',

    fields: {

        book: {

            type: BookType,

            args: { id: { type: GraphQLID }},

            resolve(parent, args) {

                //code to get data from db/other source

                //return books.find(book => book.id === args.id);

               // return \_.find(books, { id: args.id });

            }

        },

        author: {

            type: AuthorType,

            args: { id: { type: GraphQLID } },

            resolve(parent, args) {

               // return authors.find(author => (args.id === author.id));

            }

        },

        books: {

            type: new GraphQLList(BookType),

            resolve(parent, args) {

               // return books;

            }

        },

        authors: {

            type: new GraphQLList(AuthorType),

            resolve(parent, args) {

              //  return authors;

            }

        }

    }

});

const Mutation = new GraphQLObjectType({

    name: 'Mutation',

    fields: {

        addAuthor: {

            type: AuthorType,

            args: {

                name:{type:GraphQLString},

                age:{type:GraphQLInt}

            },

                resolve(parent, args){

        let author= new Author({

            name: args.name,

            age:args.age

        });

              return  author.save();

            }

        },

        addBook: {

            type:BookType,

            args: {

                name: { type: GraphQLString },

                genre: { type: GraphQLString },

                authorId:{type:GraphQLID}

            },

            resolve(parent, args) {

                let book = new Book({

                    name: args.name,

                    genre: args.genre,

                    authorId: args.authorId

                });

                return book.save();

            }

        }

    }

});

module.exports = new GraphQLSchema({

    query: RootQuery,

    mutation:Mutation

    //query for get and mutations for insert

});