**GraphQL**

**Guiding Tutorial -**

1. **Channel :**

[**https://www.youtube.com/@DanVega**](https://www.youtube.com/@DanVega)

1. **Video:**

[**https://www.youtube.com/watch?v=TVk2fMEezO4&list=RDCMUCc98QQw1D-y38wg6mO3w4MQ&index=3&ab\_channel=DanVega**](https://www.youtube.com/watch?v=TVk2fMEezO4&list=RDCMUCc98QQw1D-y38wg6mO3w4MQ&index=3&ab_channel=DanVega)

https://www.howtographql.com/basics/0-introduction/

**GraphQL යනු,**

GraphQL යනු REST,SOAP වැනි API ලියන තවත් එක් technique එකක් වන අතර මෙහි ඇති විශේෂත්වය වනුයේ single end point එකක් හරහා multiple consumers ලට dynamically ව ඔහුන්ට අවශ්‍ය data පමණක් access කර ගන්න පුළුවන් විදිහට මෙම API ලියන එකයි.Rest හෝ SOAP වලිදී end point ගොඩක් ලියනවා වෙනුවට GraphQL වලදී single end point එකකින් මේකේ වැඩේ කර ගත හැක.

**Ex:**

There is an API end point which prove a Student Object based on given ID.

Student [ID, StudentFirstName, StudentLastName, Age]

Consumer 01 : I want full student object.

Consumer 02 : I want only student first name and age.

Consumer 03 : I want only student last name.

Rest or SOAP වලදී නම් එක්කෝ end point 3 ක් දෙන්න වෙනවා.එහම් නැත්තන් දෙන full student object එකෙන් ඕනේ කරන data ටික විතරක් consumer ට filer කරලා ගන්න වෙනවා.නමුත් GraphQL වලදී එක end point එකක් හදන අතර consumer ට පුලුවන් එයාට ඕනේ කරන data ටික request කරන්න.ඉතකොට ඒ data ටික විටරක් API එකෙන් provide කරනවා.

**As a summary.**

1. **New API standard that was invented and open source by Facebook.**
2. **Enable declarative data fetching.**

Client ට පුලුවන් decide කරන මොන data ද API එකෙන් fetch කරන්න ඕනේ කියල.නමුත් rest API වලදී නම් API එකෙන් දෙන response එක එහෙම්මම client ට access කරලා එකෙන් අවශ්‍ය data ටික ගන්න තමයි සිදු වෙන්නේ.එහෙම නැතුව client ට බෑ API එකෙන්ම dynamically ව ඕනේ කරන data ටික විටරක් ගන්න.එනම් එකම API end point එකෙන් client ට ඒ ඒ අවස්තාවට අනුව dynamically ව client ට ඔනේකරන data ටිකගන්න බැ.

1. **GraphQL sever exposes single endpoint and responds to queries.**

Multiple End Point හදලා fixed data structure එකක් provide කරනවා වෙනුවට single end point එකක් හරහා එක multiple consumers ලට ඔහුන්ට අවශ්‍ය data පමණක් access කර ගන්න පුළුවන් විදිහට provide කරන technology එකකි.

**Schema define කිරීමේදී පහත පරදී keyword use වේ.**

**Query** - to retrieve data (Read).

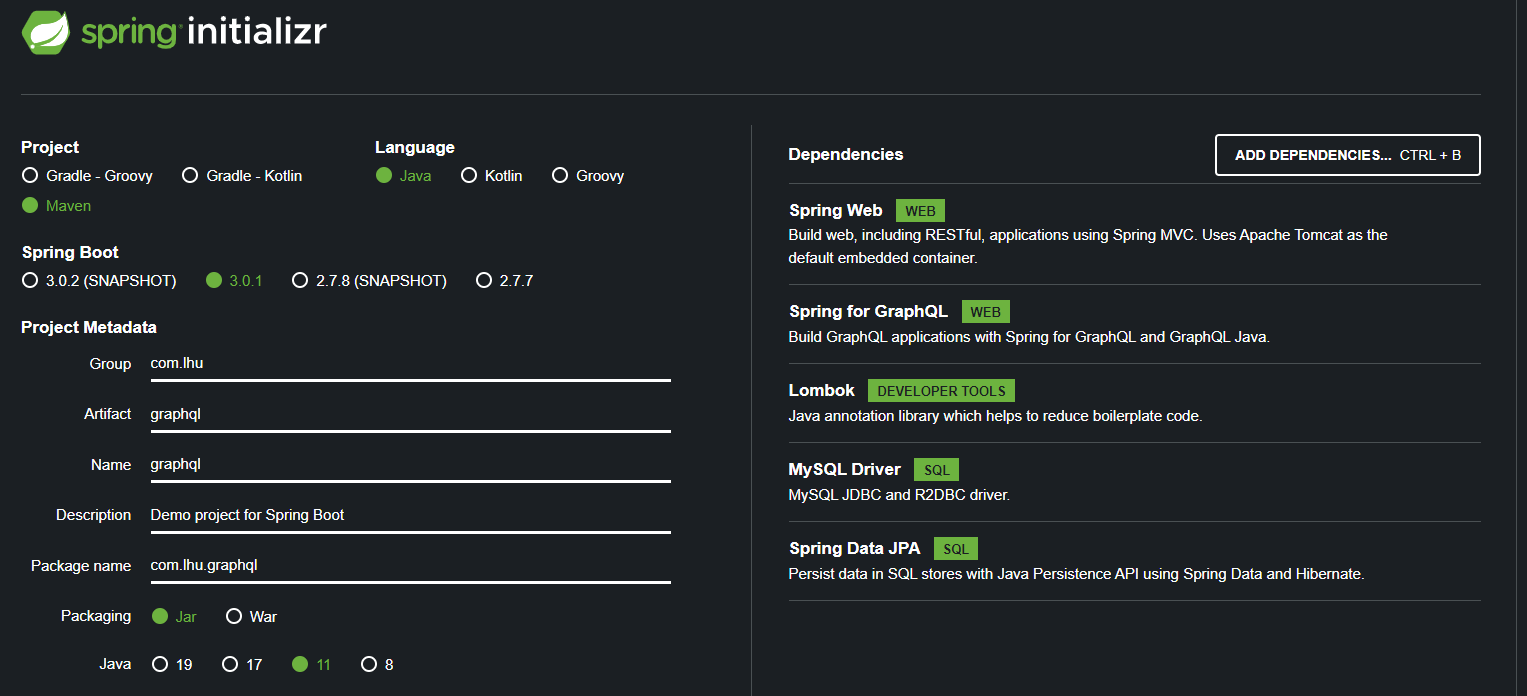
**Mutation** - to change the data (Create, Update, Delete).

**Subscription** - Which allows to create a connection to read data and keep that connection for when data changes (15.00).

**Project Creation.**

Created project GitHub Link -

Create spring boot project by adding bellow dependencies.



**Application.property**

server.port=2020

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.datasource.url=jdbc:mysql://localhost:3306/graphql?createDatabaseIfNotExist=true

spring.datasource.username=root

spring.datasource.password=root

#spring.jpa.show-sql=true

spring.jpa.hibernate.ddl-auto=create

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect

spring.graphql.graphiql.enabled=true

spring.graphql.schema.printer.enabled=true

#spring.graphql.cors.allowed-origins=\*

**Schema.graphqls**

type Student {

id: ID!

firstName: String

lastName: String

age: Int

gender: Gender

studentClass: StudentClass

}

enum Gender{

MALE,

FEMALE

}

type StudentClass {

id: ID!

grade: String

teacherName: String

students: [Student]

}

type Query {

allStudent: [Student]!

findStudent(id: ID!): Student

allStudentClass: [StudentClass]!

findStudentClass(id: ID!): StudentClass

allStudentClassPage(page: Int!, size: Int!): [StudentClass]!

}

type Mutation {

createStudentClass(grade: String!,teacherName: String!): StudentClass!

updateStudentClass(id: ID!,grade: String!,teacherName: String!): StudentClass!

deleteStudentClass(id: ID!): Boolean!

}

**Schema define කිරීමේදී පහත පරදී keyword use වේ.**

**Query** - to retrieve data (Read).

**Mutation** - to change the data (Create, Update, Delete).

**Subscription** - Which allows to create a connection to read data and keep that connection for when data changes (15.00).

**Controller layer.**

**01.Query**

@SchemaMapping(typeName = "Query", value = "allStudentClass")

  // @QueryMapping

  public List<StudentClass> allStudentClass() {

    return studentClassService.allStudentClass();

  }

If it is query and we give method name as query value, then we can simply use

@QueryMapping

**Pagination**

@QueryMapping

  public Page<StudentClass> allStudentClassPage(@Argument Integer page, @Argument Integer size) {

    PageRequest pageRequest = PageRequest.of(page, size);

    return studentClassService.allStudentClassPage(pageRequest);

  }

**02.Mutation**

 @SchemaMapping(typeName = "Mutation", value = "createStudentClass")

  // @MutationMapping

  public StudentClass createStudentClass(@Argument String grade, @Argument String teacherName) {

    StudentClass studentClass =

        StudentClass.builder().grade(grade).teacherName(teacherName).build();

    return studentClassService.createStudentClass(studentClass);

  }

**Test Application**

Run the application

You can test GraphQL application using UI or Postman( or similar IDE).

1. **Using UI**

Hit on browser.

<http://localhost:2020/graphiql?path=/graphql>

# Welcome to GraphiQL

#

# GraphiQL is an in-browser tool for writing, validating, and

# testing GraphQL queries.

#

# Type queries into this side of the screen, and you will see intelligent

# typeaheads aware of the current GraphQL type schema and live syntax and

# validation errors highlighted within the text.

#

# GraphQL queries typically start with a "{" character. Lines that start

# with a # are ignored.

#

# An example GraphQL query might look like:

#

#     {

#       field(arg: "value") {

#         subField

#       }

#     }

#

# Keyboard shortcuts:

#

#   Prettify query:  Shift-Ctrl-P (or press the prettify button)

#

#  Merge fragments:  Shift-Ctrl-M (or press the merge button)

#

#        Run Query:  Ctrl-Enter (or press the play button)

#

#    Auto Complete:  Ctrl-Space (or just start typing)

#

1. **Getting all Student (Without pagination)**

query{

  allStudent{

    id

    firstName

    lastName

    age

    studentClass{

      id

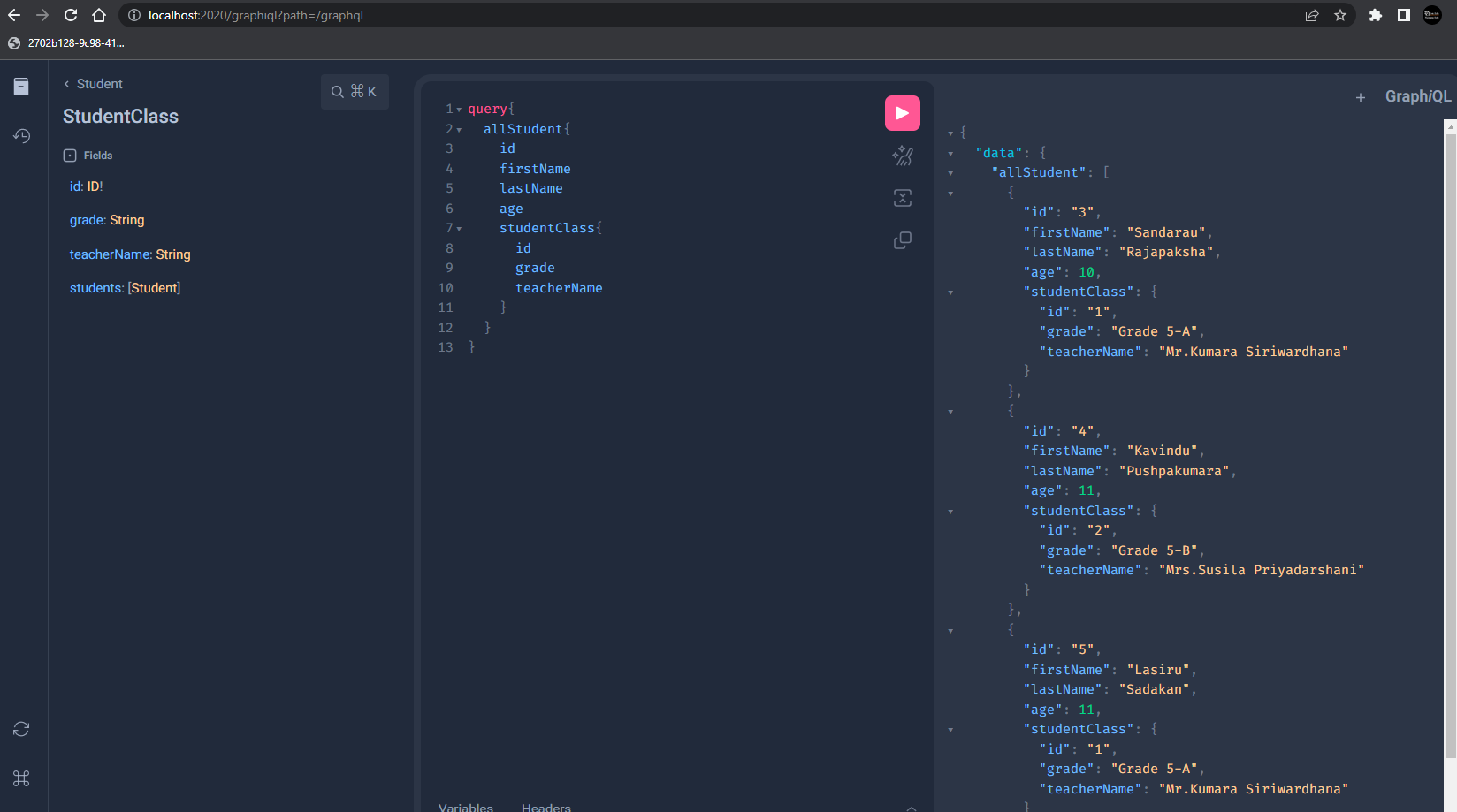
      grade

      teacherName

    }

  }

}



1. **Get a specific Student**

query{

  findStudent(id:4){

    id

    firstName

    lastName

    age

    studentClass{

      id

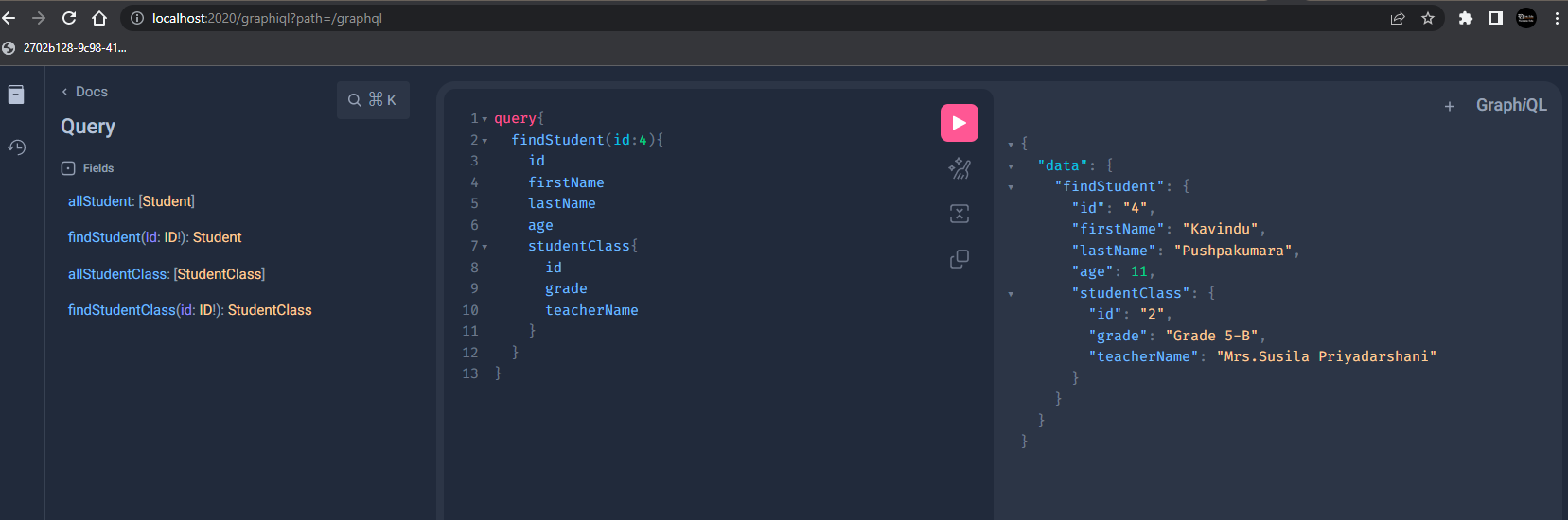
      grade

      teacherName

    }

  }

}



1. **Getting all student classes (Without pagination)**

query{

  allStudentClass{

    id

    grade

    teacherName

    students{

      id

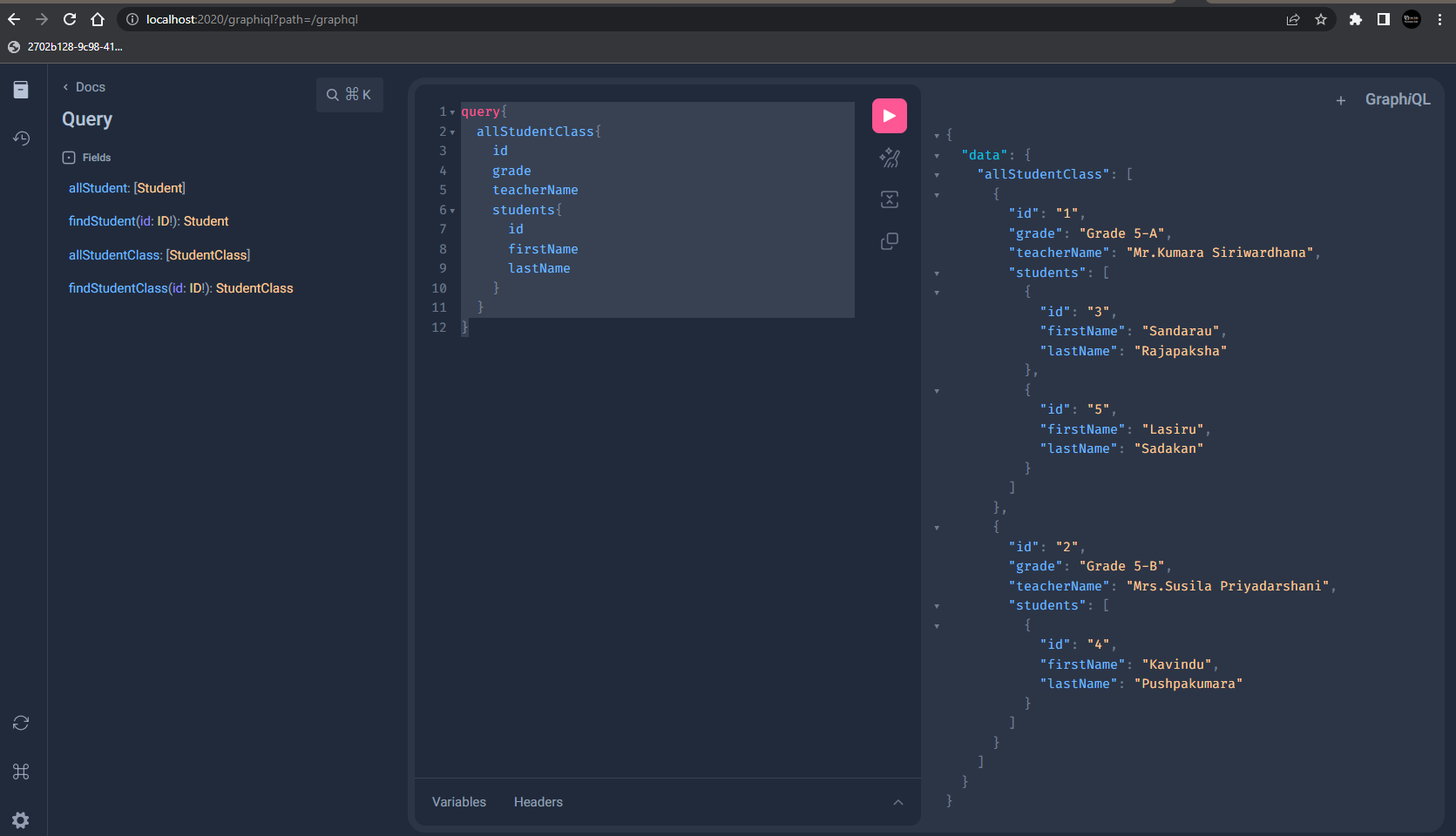
      firstName

      lastName

    }

  }

}



1. **Getting a specific student classed**

query{

  findStudentClass(id:2){

    id

    grade

    teacherName

    students{

      id

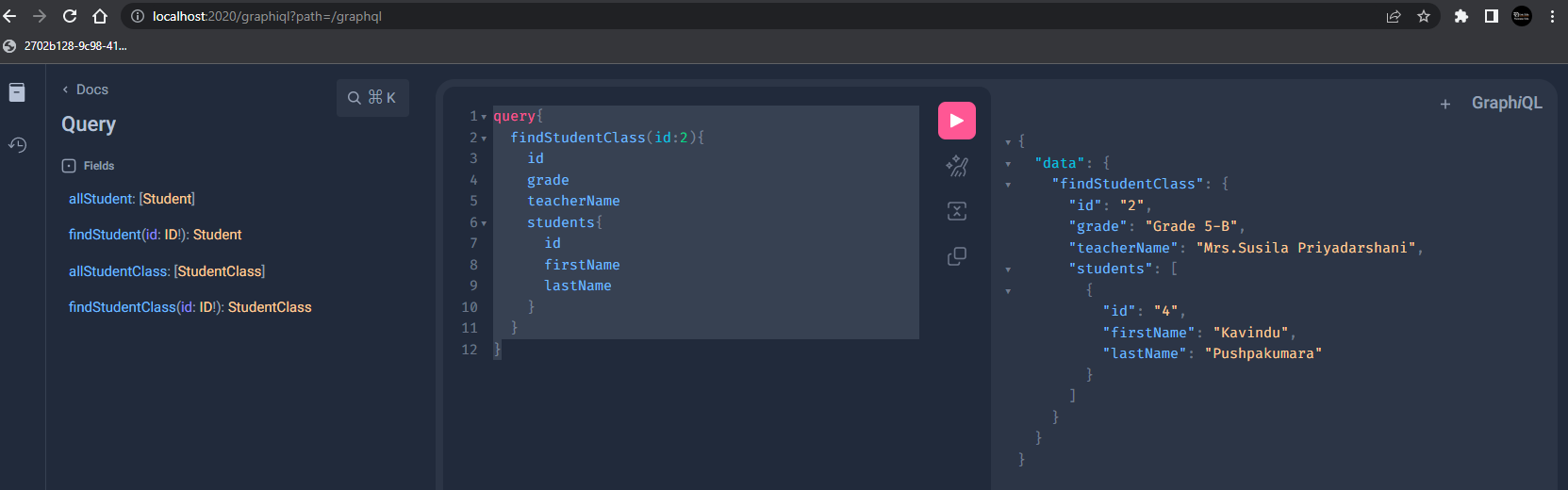
      firstName

      lastName

    }

  }

}



1. **Creating a student class**

mutation{

  createStudentClass(grade:"Grade 6-A",teacherName:"Mr.fernando"){

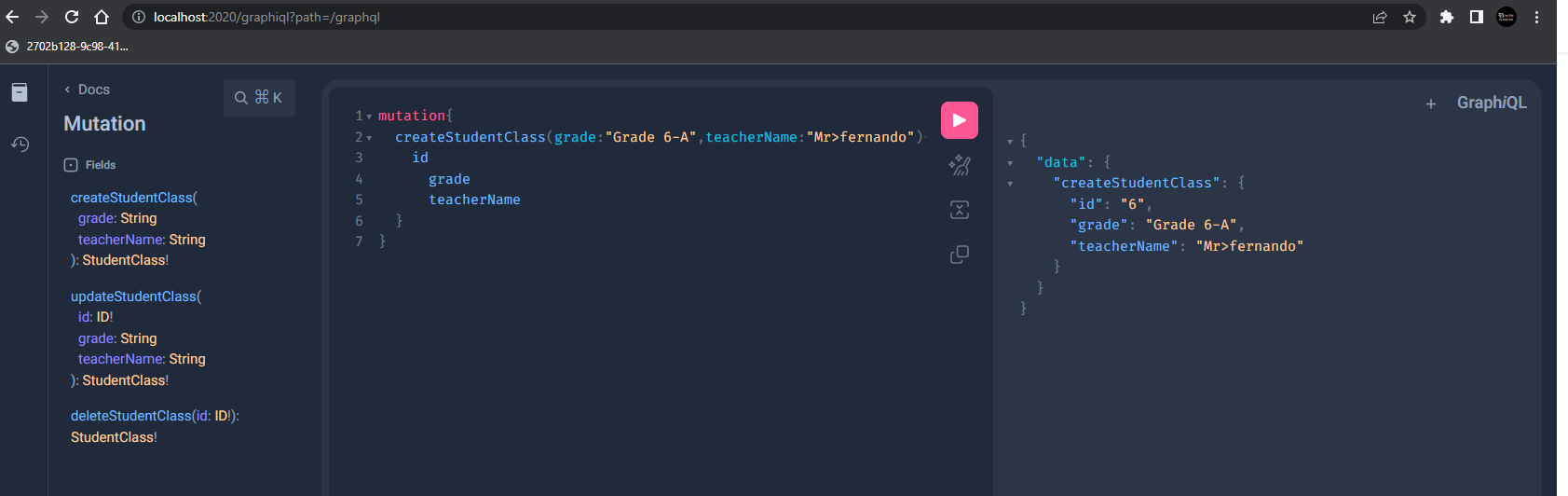
    id

      grade

      teacherName

  }

}



1. **Updating a student class**

mutation{

  updateStudentClass(id:6,grade:"Grade 6-A",teacherName:"Mr.fernando"){

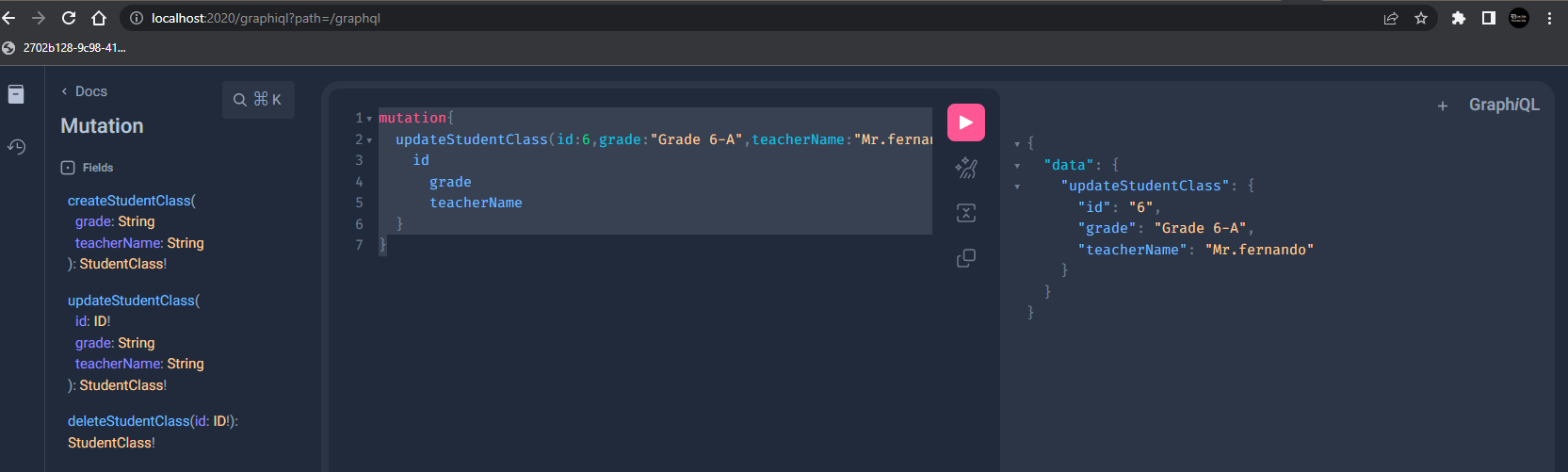
    id

      grade

      teacherName

  }

}

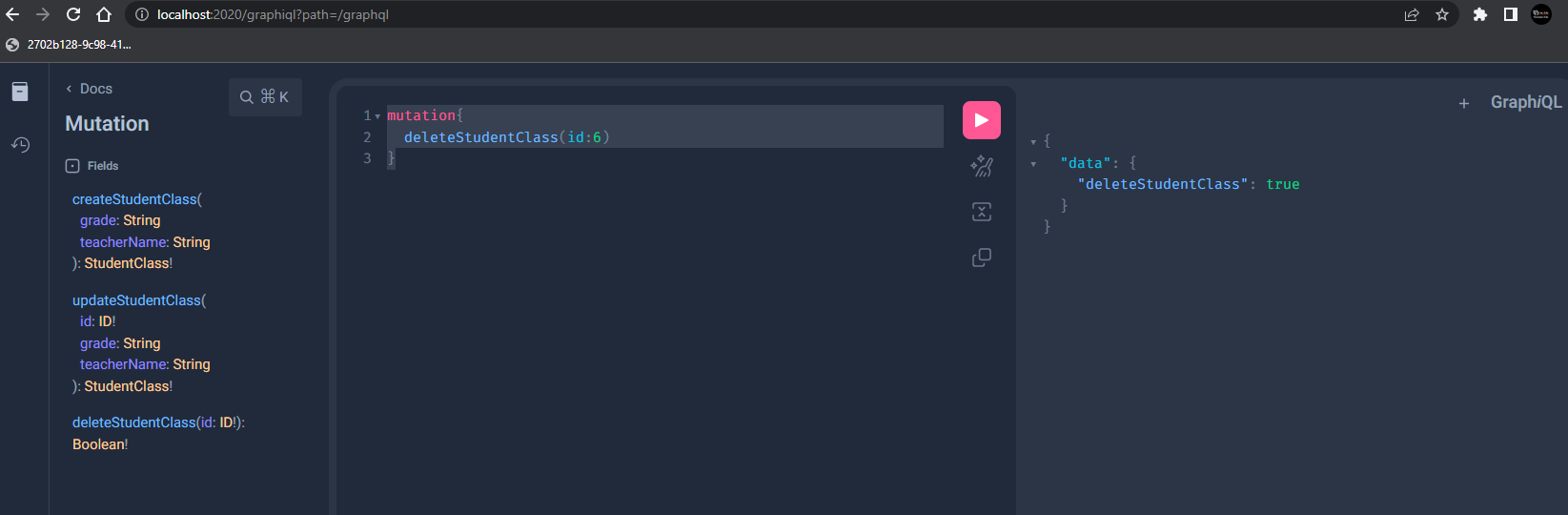


1. **Deleting a student class**

mutation{

  deleteStudentClass(id:6)

}



1. **Getting all Student (With pagination)**

query{

  allStudentClassPage(page:0,size:2){

     id

    grade

    teacherName

    students{

      id

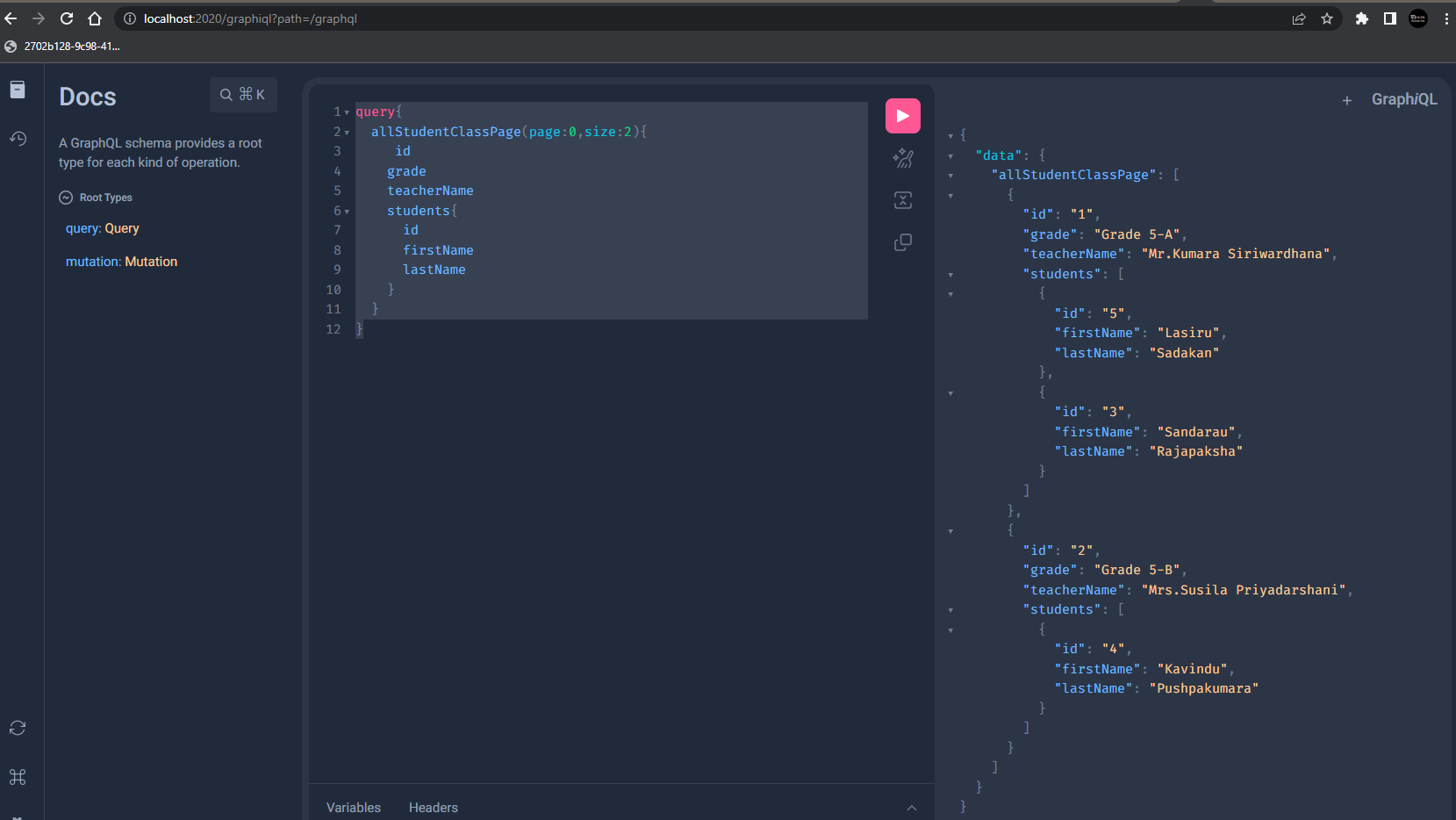
      firstName

      lastName

    }

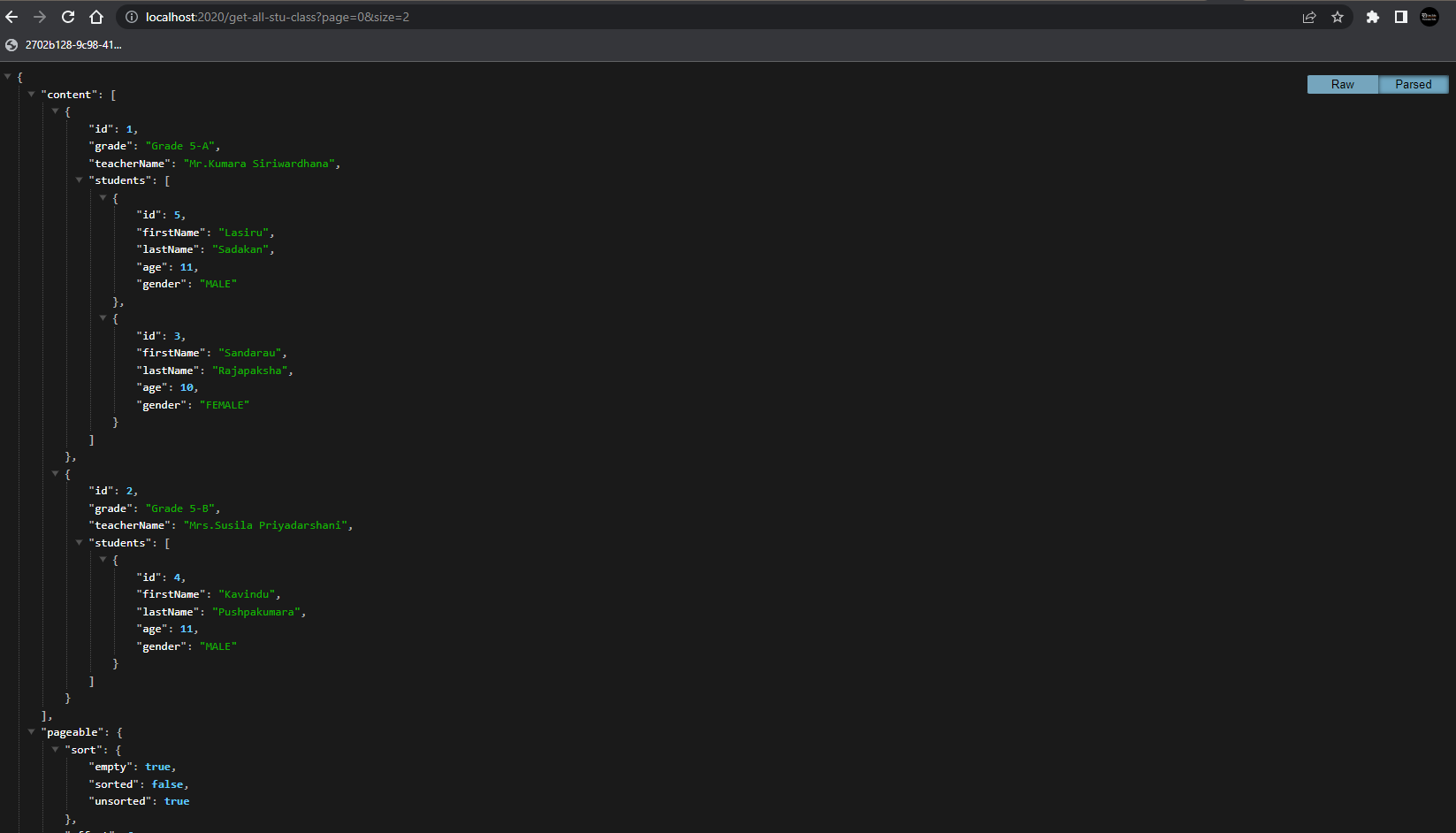
  }

}

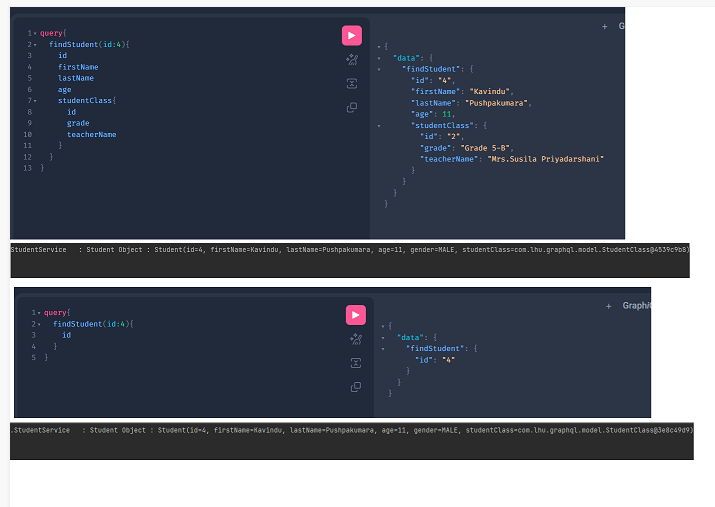


**To Test Rest Pagination API**

<http://localhost:2020/get-all-stu-class?page=0&size=2>



**GraphQL වලදී consumer requires කරන data filter කරලා දෙන්නේ database level එකෙන් නොවන අතර, එය controller layer එකේදී graphQL libraries මගින් කර දෙනු ලබයි.**



**Test using Postman**

