What is Graphql

- Specific application bulit on http about how you get around the server.
- The way you determine what data you get back from the endpoint is based on the query which you use

Difference between Restful API and Graphql

With Restful API

- We need a particular endpoint we need to ge the particular data
- We might get unnecessary information : `/authors` endpoint to get the information about the authors
- We need another endpoint for accessing more data : `/authors/:id/books`

WIth GraphQL

- We can get specific information

```
query {
authors {
name
books {
name
}
}
}
```

- We get the author name, books written by the author and books name
- We give the simple query to the server and the server parses that query
- We get only the information that we ask for

Setting up GraphQL

- Use npm to create package json file

```
npm init
```

- Change the package.json as follows

```
cat package.json
{
  "name": "practice",
  "version": "1.0.0",
  "description": "",
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
```

- Install all the dependencies needed

```
npm i express express-graphql graphql
```

- Install nodemon as dev dependency

```
npm i --save-dev nodemon
```

- Add a script in package.json

```
{
  "name": "practice",
  "version": "1.0.0",
  "description": "",
  "main": "server.js",
  "scripts": {
    "devStart": "nodemon server.js"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.17.1",
    "express-graphql": "^0.12.0",
    "graphql": "^15.5.1"
  },
```

```
  "devDependencies": {
    "nodemon": "^2.0.12"
  }
}
```

Create the server file

server.js

```
const express = require('express')
const app = express()
app.listen(5000., () => console.log('Server Running'))
```

- Run the server as

```
npm run devStart
```

- This will run our server on localhost port 5000

Adding GraphQL to our server

```
const express = require('express')
const { graphqlHTTP } = require('express-graphql');

const app = express()

app.use('/graphql', graphqlHTTP({
        graphiql: true
}))

app.listen(5000., () => console.log('Server Running'))
```

- Going to the url `/graphql` will tell us, it needs a schema
- Define a schema, which tells how all of our data is interacted

Creating a schema

Add Graphql Object type

```
const express = require('express')
const { graphqlHTTP } = require('express-graphql');

const {
        GraphQLSchema,
        GraphQLObjectType
} = require('graphql')

const app = express()

app.use('/graphql', graphqlHTTP({
        graphiql: true
}))

app.listen(5000., () => console.log('Server Running'))
```

Adding the schema

- We will create a dummy schema
- The first section will have getting of data

```
const express = require('express')
const { graphqlHTTP } = require('express-graphql');

const {
        GraphQLSchema,
        GraphQLObjectType,
        GraphQLString
} = require('graphql')

const app = express()

const schema = new GraphQLSchema({
        query: new GraphQLObjectType({
                name: 'Hello World',
                fields: () => ({
                        message: {
                                type: GraphQLString,
                                resolve: () => 'Hello World'
                        }
```

```
                })
        })
})

app.use('/graphql', graphqlHTTP({
        schema: schema,
        graphiql: true
}))

app.listen(5000., () => console.log('Server Running'))
```

- It will have a GraphqlObjectType
- It will have the name
- It will have fields as a function whichi will print the "Hello World"
- Running the server will give an interface and give an error

```
{
  "errors": [
    {
      "message": "Names must match /^[_a-zA-Z][_a-zA-Z0-9]*$/ but \"Hello World\"
does not."
    }
  ]
}
```

- This tells us that the name cannot have name
- Remove the space from the name

```
const express = require('express')
const { graphqlHTTP } = require('express-graphql');

const {
        GraphQLSchema,
        GraphQLObjectType,
        GraphQLString
} = require('graphql')

const app = express()

const schema = new GraphQLSchema({
```

```
        query: new GraphQLObjectType({
                name: 'HelloWorld',
                fields: () => ({
                        message: {
                                type: GraphQLString,
                                resolve: () => 'Hello World'
                        }
                })
        })
})

app.use('/graphql', graphqlHTTP({
        schema: schema,
        graphiql: true
}))

app.listen(5000., () => console.log('Server Running'))
```

- Query the server
- Checking the Docs will tell us that there is only one query
- That is the `HelloWorld`

```
query {
    message
}
```

- Output

```
{
  "data": {
    "message": "Hello World"
  }
}
```

- The query keyword is not mandatory
- GraphQL by default uses the query keyword
- We can build any type of object by passing it the fields
- Telling it what type those different fields are

- Resolving what these fields will do

Adding Database to the Server

Add name of some books

```
const express = require('express')
const { graphqlHTTP } = require('express-graphql');

const {
        GraphQLSchema,
        GraphQLObjectType,
        GraphQLString
} = require('graphql')

const app = express()

const authors = [
        { id: 1, name: 'JK Rowling' },
        { id: 2, name: 'JRR Tokein' },
        { id: 3, name: 'Brent Weeks' }
]

const books = [
        { id:1, name: 'Harry Potter', authorid: 1 },
        { id:2, name: 'Some other shit', authorid: 1 },
        { id:3, name: 'Baaler chal', authorid: 1 },
        { id:4, name: 'Boi er naam', authorid: 2 },
        { id:5, name: 'Chudir bhai', authorid: 2 },
        { id:6, name: 'chondro gupto', authorid: 2 },
        { id:7, name: 'Pod mere', authorid: 3 },
        { id:8, name: 'Gondo shukto', authorid: 3 }
]

const schema = new GraphQLSchema({
        query: new GraphQLObjectType({
                name: 'HelloWorld',
                fields: () => ({
                        message: {
```

```
                                type: GraphQLString,
                                resolve: () => 'Hello World'
                            }
                    })
            })
})

app.use('/graphql', graphqlHTTP({
        schema: schema,
        graphiql: true
}))

app.listen(5000., () => console.log('Server Running'))
```

Adding a Root Query Section

- Right now we can only query the hello world object from the messages field

```
const express = require('express')
const { graphqlHTTP } = require('express-graphql');

const {
    GraphQLSchema,
    GraphQLObjectType,
    GraphQLString,
    GraphQLList,
    GraphQLInt,
    GraphQLNonNUll
} = require('graphql')

const app = express()

const authors = [
    { id: 1, name: 'JK Rowling' },
    { id: 2, name: 'JRR Tokein' },
    { id: 3, name: 'Brent Weeks' }
]

const books = [
    { id:1, name: 'Harry Potter', authorid: 1 },
```

```
    { id:2, name: 'Some other shit', authorid: 1 },
    { id:3, name: 'Baaler chal', authorid: 1 },
    { id:4, name: 'Boi er naam', authorid: 2 },
    { id:5, name: 'Chudir bhai', authorid: 2 },
    { id:6, name: 'chondro gupto', authorid: 2 },
    { id:7, name: 'Pod mere', authorid: 3 },
    { id:8, name: 'Gondo shukto', authorid: 3 }
]

const BookType = new GraphQLObjectType({
    name: 'Book',
    description: 'This represents a book written by an author',
    fields: () => ({
        id: { type: GraphQLNonNull(GraphQLInt) },
        name: { type: GraphQLNonNull(GraphQLString) },
        authorId: { type: GraphQLNonNull(GraphQLInt) }
    })
})

const RootQueryTpye = new GraphQLObjectType ({
    name: 'Query',
    description: 'Root Query',
    fields: () => ({
        books: {
            type: new GraphQLList(BookType),
            description: 'List of Books',
            resolve: () => books
        })
    })
})


app.use('/graphql', graphqlHTTP({
    schema: schema,
    graphiql: true
}))

app.listen(5000., () => console.log('Server Running'))
```

- The root query creates the basic structure for our query

```
const RootQueryTpye = new GraphQLObjectType ({
    name: 'Query',
    description: 'Root Query',
    fields: () => ({
        books: {
            type: new GraphQLList(BookType),
            description: 'List of Books',
            resolve: () => books
        })
    })
})
```

- This will define the BooType object

```
const BookType = new GraphQLObjectType({
    name: 'Book',
    description: 'This represents a book written by an author',
    fields: () => ({
        id: { type: GraphQLNonNull(GraphQLInt) },
        name: { type: GraphQLNonNull(GraphQLString) },
        authorId: { type: GraphQLNonNull(GraphQLInt) }
    })
})
```

- Using `({})` as the body of the function will return whatever is the result of the function
- The parameters `id, name, authorId` will directly fetch from the database, as there are names of the type
- Datatype `GraphQLNotNull` is used, so that it does not take a null value
- Schema has to be added

```
const schema = new GraphQLSchema ({
        query: RootQueryType
})
```

- The schema will query the `RootQueryType`

server.js

```
const express = require('express')
const { graphqlHTTP } = require('express-graphql');

const {
    GraphQLSchema,
    GraphQLObjectType,
    GraphQLString,
    GraphQLList,
    GraphQLInt,
    GraphQLNonNull
} = require('graphql')

const app = express()

const authors = [
    { id: 1, name: 'JK Rowling' },
    { id: 2, name: 'JRR Tokein' },
    { id: 3, name: 'Brent Weeks' }
]

const books = [
    { id:1, name: 'Harry Potter', authorid: 1 },
    { id:2, name: 'Some other shit', authorid: 1 },
    { id:3, name: 'Baaler chal', authorid: 1 },
    { id:4, name: 'Boi er naam', authorid: 2 },
    { id:5, name: 'Chudir bhai', authorid: 2 },
    { id:6, name: 'chondro gupto', authorid: 2 },
    { id:7, name: 'Pod mere', authorid: 3 },
    { id:8, name: 'Gondo shukto', authorid: 3 }
]

const BookType = new GraphQLObjectType({
    name: 'Book',
    description: 'This represents a book written by an author',
    fields: () => ({
        id: { type: GraphQLNonNull(GraphQLInt) },
        name: { type: GraphQLNonNull(GraphQLString) },
        authorId: { type: GraphQLNonNull(GraphQLInt) }
    })
})
```

```
const RootQueryType = new GraphQLObjectType ({
    name: 'Query',
    description: 'Root Query',
    fields: () => ({
        books: {
            type: new GraphQLList(BookType),
            description: 'List of Books',
            resolve: () => books
        }
    })
})

const schema = new GraphQLSchema ({
    query: RootQueryType
})


app.use('/graphql', graphqlHTTP({
    schema: schema,
    graphiql: true
}))

app.listen(5000., () => console.log('Server Running'))
```

- Refreshing the server will give us the documentation explorer containing the different query type

Sending Queries

- Get the names of all books

```
{
  books {
    name
  }
}
```

Output

```
{
  "data": {
    "books": [
      {
        "name": "Harry Potter"
      },
      {
        "name": "Some other shit"
      },
      {
        "name": "Baaler chal"
      },
      {
        "name": "Boi er naam"
      },
      {
        "name": "Chudir bhai"
      },
      {
        "name": "chondro gupto"
      },
      {
        "name": "Pod mere"
      },
      {
        "name": "Gondo shukto"
      }
    ]
  }
}
```


- Get the id and names of all books

query

```
{
  books {
    id
    name
  }
}
```

output


```
{
  "data": {
    "books": [
      {
        "id": 1,
        "name": "Harry Potter"
      },
      {
        "id": 2,
        "name": "Some other shit"
      },
      {
        "id": 3,
        "name": "Baaler chal"
      },
      {
        "id": 4,
        "name": "Boi er naam"
      },
      {
        "id": 5,
        "name": "Chudir bhai"
      },
      {
        "id": 6,
        "name": "chondro gupto"
      },
      {
        "id": 7,
        "name": "Pod mere"
      },
      {
        "id": 8,
        "name": "Gondo shukto"
      }
    ]
  }
}
```

- Get only the id

query

```
{
  books {
    id
  }
}
```

output

```
{
  "data": {
    "books": [
      {
        "id": 1
      },
      {
        "id": 2
      },
      {
        "id": 3
      },
      {
        "id": 4
      },
      {
        "id": 5
      },
      {
        "id": 6
      },
      {
        "id": 7
      },
      {
        "id": 8
      }
    ]
```

```
  }
}
```

Adding More Fields

- If we try to get the author name from the server

```
{
  books {
    id
    author {
      name
    }
  }
}
```

- We will get error saying cannot query field "author"
- We will add a new field to our BookType object called author

```
const BookType = new GraphQLObjectType({
    name: 'Book',
    description: 'This represents a book written by an author',
    fields: () => ({
        id: { type: GraphQLNonNull(GraphQLInt) },
        name: { type: GraphQLNonNull(GraphQLString) },
        authorId: { type: GraphQLNonNull(GraphQLInt) }
        author: {
            type: AuthorType
            resolve: (book) => {
                return authors.find(author => author.id == book.authorid)
            }
        }
    })
})
```

- New author field is created with the type as `AuthorType`
- This will resolve to a function
- The function takes a parameter `book`
- Now it returns the author
- It checks if the author id is equal to the book id or not

- Now create the AuthorType object

```
onst AuthorType = new GraphQLObjectType({
    name: 'Author',
    description: 'This represents an author of a book',
    fields: () => ({
        id: { type: GraphQLNonNull(GraphQLInt) },
        name: { type: GraphQLNonNull(GraphQLString) }
    })
})
```

- It will have the author id and the author name
- Now querying author name with id will give us the results

query

```
{
  books {
    id
    author {
      name
    }
  }
}
```

output

```
{
  "data": {
    "books": [
      {
        "id": 1,
        "author": {
          "name": "JK Rowling"
        }
      },
      {
        "id": 2,
        "author": {
          "name": "JK Rowling"
        }
```

```
      },
      {
        "id": 3,
        "author": {
          "name": "JK Rowling"
        }
      },
      {
        "id": 4,
        "author": {
          "name": "JRR Tokein"
        }
      },
      {
        "id": 5,
        "author": {
          "name": "JRR Tokein"
        }
      },
      {
        "id": 6,
        "author": {
          "name": "JRR Tokein"
        }
      },
      {
        "id": 7,
        "author": {
          "name": "Brent Weeks"
        }
      },
      {
        "id": 8,
        "author": {
          "name": "Brent Weeks"
        }
      }
    ]
  }
}
```

Adding More Queries

Adding Query for Getting Authors

- Add a new query along with the book query

```
const RootQueryType = new GraphQLObjectType ({
        name: 'Query',
        description: 'Root Query',
        fields: () => ({
                books: {
                        type: new GraphQLList(BookType),
                        description: 'List of Books',
                        resolve: () => books
                },
                author: {
                        type: GraphQLList(AuthorType),
                        description: 'List of all Authors',
                        resolve: () => authors
                }
        })
})
```

- Now we can query the authors name

query

```
{
 authors {
  name
}
}
```

output

```
{
  "data": {
    "authors": [
      {
```

```
        "name": "JK Rowling"
      },
      {
        "name": "JRR Tokein"
      },
      {
        "name": "Brent Weeks"
      }
    ]
  }
}
```

Adding books field for author

- If we query books under authors, we won't get result
- We need to get list of books with the author name

```
const AuthorType = new GraphQLObjectType({
    name: 'Author',
    description: 'This represents an author of a book',
    fields: () => ({
        id: { type: GraphQLNonNull(GraphQLInt) },
        name: { type: GraphQLNonNull(GraphQLString) },
        books: {
            type: new GraphQLList(BookType),
            resolve: (author) => {
                return books.filter(book => books.id == authors.id)
            }
        }
    })
})
```

- This books field will be of type `GraphQLList`
- It will take `author` as the parent parameter
- Then it will resolve a function
- This function will filter by matching the book.id with the authors.id
- Now we can query the author name with a list of books written by that author

query

```
{
  authors {
    id
    name
    books {
      name
    }
  }
}
```

output

```
{
  "data": {
    "authors": [
      {
        "id": 1,
        "name": "JK Rowling",
        "books": [
          {
            "name": "Harry Potter"
          },
          {
            "name": "Some other shit"
          },
          {
            "name": "Baaler chal"
          },
          {
            "name": "Boi er naam"
          },
          {
            "name": "Chudir bhai"
          },
          {
            "name": "chondro gupto"
          },
          {
            "name": "Pod mere"
          },
```

```
      {
        "name": "Gondo shukto"
      }
    ]
  },
  {
    "id": 2,
    "name": "JRR Tokein",
    "books": [
      {
        "name": "Harry Potter"
      },
      {
        "name": "Some other shit"
      },
      {
        "name": "Baaler chal"
      },
      {
        "name": "Boi er naam"
      },
      {
        "name": "Chudir bhai"
      },
      {
        "name": "chondro gupto"
      },
      {
        "name": "Pod mere"
      },
      {
        "name": "Gondo shukto"
      }
    ]
  },
  {
    "id": 3,
    "name": "Brent Weeks",
    "books": [
      {
```

```
            "name": "Harry Potter"
          },
          {
            "name": "Some other shit"
          },
          {
            "name": "Baaler chal"
          },
          {
            "name": "Boi er naam"
          },
          {
            "name": "Chudir bhai"
          },
          {
            "name": "chondro gupto"
          },
          {
            "name": "Pod mere"
          },
          {
            "name": "Gondo shukto"
          }
        ]
      }
    ]
  }
}
```

Passing Argument to GraphQL Method

```
const RootQueryType = new GraphQLObjectType ({
    name: 'Query',
    description: 'Root Query',
    fields: () => ({
        book: {
            type: BookType,
            description: 'A single book',
            args: {
                id: { type: GraphQLInt }
```

```
        },
        resolve: (parent, args) => books.find(book => book.id == args.id)
    },
    books: {
        type: new GraphQLList(BookType),
        description: 'List of Books',
        resolve: () => books
    },
    authors: {
        type: GraphQLList(AuthorType),
        description: 'List of all Authors',
        resolve: () => authors
    }
```

- Add one field for a single book
- Add the type as BookType and not GraphQLList type
- The resolve function takes two arguments, (parent) which we don't need and (args)
- The `args` parameter is defined as the id of the book which is a GraphQLInt
- The resolve() function will find the book.id which matches with the queried book id
- Now we can query a single book with the id

query

```
{
  book(id: 1) {
    name
  }
}
```

output

```
{
  "data": {
    "book": {
      "name": "Harry Potter"
    }
  }
}
```

- Also get the author of the book

query

```
{
  book(id: 1) {
    name
    author{
      name
    }
  }
}
```

output

```
{
  "data": {
    "book": {
      "name": "Harry Potter",
      "author": {
        "name": "JK Rowling"
      }
    }
  }
}
```

Getting a single author

```
const RootQueryType = new GraphQLObjectType ({
    name: 'Query',
    description: 'Root Query',
    fields: () => ({
        book: {
            type: BookType,
            description: 'A single book',
            args: {
                id: { type: GraphQLInt }
            },
            resolve: (parent, args) => books.find(book => book.id == args.id)
        },
        books: {
            type: new GraphQLList(BookType),
```

```
            description: 'List of Books',
            resolve: () => books
        },
        authors: {
            type: GraphQLList(AuthorType),
            description: 'List of all Authors',
            resolve: () => authors
        },

        author: {
            type: AuthorType,
            description: 'A single author',
            args: {
                id: { type: GraphQLInt }
            },
            resolve: (parent, args) => authors.find(author => author.id === 
args.id)
        }
    })
})
```

- Add a field author
- It will take the id of the author as `GraphQLInt`
- It will match with the author.id in the list with the author id passed `args.id`
- We can get a single author using author id

query

```
{
  author(id: 2) {
    name
  }
}
```

output

```
{
  "data": {
    "author": {
      "name": "JRR Tokein"
    }
```

```
  }
}
```

- Field is returning a function that returns an object
- Everything is defined in a function so that they can reference each other before they are defined

Data Mutation

- Mutation is GraphQL version of using POST, PUT and DELETE on a REST API server
- Mutations create changes to the existing data

Updating the schema

- The schema takes a mutation field

```
const schema = new GraphQLSchema ({
        query: RootQueryType,
        mutation: RootMutationType
})
```

- The schema will have a mutation with the type `RootMutationType` which we will create

```
const RootMutationType = new GraphQLObjectType ({
        name: 'Mutation',
        description: 'Root Mutation',
        fields: () => ({
                addBook: {
                        type: BookType,
                        description: 'Add a book',
                        args: {
                                name: { type: GraphQLNonNull(GraphQLString) },
                                authorid: { type: GraphQLNonNull(GraphQLInt) }
                        },
                        resolve: (parent, args) => {
                                const book = { id: books.length + 1, name:
args.name, authorid: args.authorid }
                                books.push(book)
                                return book
                        }
```

```
                }
        })
})
```

- The mutation type will take a book name and the authorid from the argument and the book id will be one more than the length of the book lists length
- Then will push the book object to the array
- It will also return the book object, since the mutation function returns a `BookType` object
- Add and query the name of the added book

query

```
mutation {
  addBook(name: "Fifty shades of chodon", authorid: 4) {
    id
    name
  }
}
```

output

```
{
  "data": {
    "addBook": {
      "id": 9,
      "name": "Fifty shades of chodon"
    }
  }
}
```

- Check the book is added

```
{
  books {
    id
    name
  }
}
```

```
{
  "data": {
    "books": [
      {
        "id": 1,
        "name": "Harry Potter"
      },
      {
        "id": 2,
        "name": "Some other shit"
      },
      {
        "id": 3,
        "name": "Baaler chal"
      },
      {
        "id": 4,
        "name": "Boi er naam"
      },
      {
        "id": 5,
        "name": "Chudir bhai"
      },
      {
        "id": 6,
        "name": "chondro gupto"
      },
      {
        "id": 7,
        "name": "Pod mere"
      },
      {
        "id": 8,
        "name": "Gondo shukto"
      },
      {
        "id": 9,
        "name": "Fifty shades of chodon"
      }
    ]
```

```
  }
}
```

Adding author

- Add new mutation `addAuthor`

```
const RootMutationType = new GraphQLObjectType ({
        name: 'Mutation',
        description: 'Root Mutation',
        fields: () => ({
                addBook: {
                        type: BookType,
                        description: 'Add a book',
                        args: {
                                name: { type: GraphQLNonNull(GraphQLString) },
                                authorid: { type: GraphQLNonNull(GraphQLInt) }
                        },
                        resolve: (parent, args) => {
                                const book = { id: books.length + 1, name:
args.name, authorid: args.authorid }
                                books.push(book)
                                return book
                        }
                },

                addAuthor: {
                        type: AuthorType,
                        description: 'Add an author',
                        args: {
                                name: { type: GraphQLNonNull(GraphQLString) }
                        },
                        resolve: (parent, args) => {
                                const author = { id: authors.length + 1, name:
args.name }
                                authors.push(author)
                                return author
                        }
                }
```

```
        })
})
```

- It takes the author name as argument
- Then the id is added as one more than the lenght of the array
- The name is added from the argument
- The author type object is then pushed to the array
- Then the author return type is returned
- Add an author and query its id and name

query

```
mutation {
  addAuthor(name:"Chodon Churdhuri") {
    id
    name
  }
}
```

output

```
{
  "data": {
    "addAuthor": {
      "id": 4,
      "name": "Chodon Churdhuri"
    }
  }
}
```

- Check if the author is added

```
{
  authors {
    id
    name
    books {
      name
```

```
    }
  }
}
```


```
{
  "data": {
    "authors": [
      {
        "id": 1,
        "name": "JK Rowling",
        "books": [
          {
            "name": "Harry Potter"
          },
          {
            "name": "Some other shit"
          },
          {
            "name": "Baaler chal"
          },
          {
            "name": "Boi er naam"
          },
          {
            "name": "Chudir bhai"
          },
          {
            "name": "chondro gupto"
          },
          {
            "name": "Pod mere"
          },
          {
            "name": "Gondo shukto"
          }
        ]
      },
      {
        "id": 2,
        "name": "JRR Tokein",
```

```
      "books": [
        {
          "name": "Harry Potter"
        },
        {
          "name": "Some other shit"
        },
        {
          "name": "Baaler chal"
        },
        {
          "name": "Boi er naam"
        },
        {
          "name": "Chudir bhai"
        },
        {
          "name": "chondro gupto"
        },
        {
          "name": "Pod mere"
        },
        {
          "name": "Gondo shukto"
        }
      ]
    },
    {
      "id": 3,
      "name": "Brent Weeks",
      "books": [
        {
          "name": "Harry Potter"
        },
        {
          "name": "Some other shit"
        },
        {
          "name": "Baaler chal"
        },
```

```
        {
          "name": "Boi er naam"
        },
        {
          "name": "Chudir bhai"
        },
        {
          "name": "chondro gupto"
        },
        {
          "name": "Pod mere"
        },
        {
          "name": "Gondo shukto"
        }
      ]
    },
    {
      "id": 4,
      "name": "Chodon Churdhuri",
      "books": [
        {
          "name": "Harry Potter"
        },
        {
          "name": "Some other shit"
        },
        {
          "name": "Baaler chal"
        },
        {
          "name": "Boi er naam"
        },
        {
          "name": "Chudir bhai"
        },
        {
          "name": "chondro gupto"
        },
        {
```

```
            "name": "Pod mere"
          },
          {
            "name": "Gondo shukto"
          }
        ]
      }
    ]
  }
}
```

GraphQL Server Code

```
const express = require('express')
const { graphqlHTTP } = require('express-graphql');

const {
    GraphQLSchema,
    GraphQLObjectType,
    GraphQLString,
    GraphQLList,
    GraphQLInt,
    GraphQLNonNull
} = require('graphql')

const app = express()

const authors = [
    { id: 1, name: 'JK Rowling' },
    { id: 2, name: 'JRR Tokein' },
    { id: 3, name: 'Brent Weeks' }
]

const books = [
    { id: 1, name: 'Harry Potter', authorid: 1 },
    { id: 2, name: 'Some other shit', authorid: 1 },
    { id: 3, name: 'Baaler chal', authorid: 1 },
    { id: 4, name: 'Boi er naam', authorid: 2 },
    { id: 5, name: 'Chudir bhai', authorid: 2 },
```

```
    { id: 6, name: 'chondro gupto', authorid: 2 },
    { id: 7, name: 'Pod mere', authorid: 3 },
    { id: 8, name: 'Gondo shukto', authorid: 3 }
]

const BookType = new GraphQLObjectType({
    name: 'Book',
    description: 'This represents a book written by an author',
    fields: () => ({
        id: { type: GraphQLNonNull(GraphQLInt) },
        name: { type: GraphQLNonNull(GraphQLString) },
        authorId: { type: GraphQLNonNull(GraphQLInt) },
        author: {
            type: AuthorType,
            resolve: (book) => {
                return authors.find(author => author.id == book.authorid)
            }
        }
    })
})


const AuthorType = new GraphQLObjectType({
    name: 'Author',
    description: 'This represents an author of a book',
    fields: () => ({
        id: { type: GraphQLNonNull(GraphQLInt) },
        name: { type: GraphQLNonNull(GraphQLString) },
        books: {
            type: new GraphQLList(BookType),
            resolve: (author) => {
                return books.filter(book => books.id == authors.id)
            }
        }
    })
})

const RootQueryType = new GraphQLObjectType ({
    name: 'Query',
    description: 'Root Query',
```

```
    fields: () => ({
        book: {
            type: BookType,
            description: 'A single book',
            args: {
                id: { type: GraphQLInt }
            },
            resolve: (parent, args) => books.find(book => book.id == args.id)
        },
        books: {
            type: new GraphQLList(BookType),
            description: 'List of Books',
            resolve: () => books
        },
        authors: {
            type: GraphQLList(AuthorType),
            description: 'List of all Authors',
            resolve: () => authors
        },

        author: {
            type: AuthorType,
            description: 'A single author',
            args: {
                id: { type: GraphQLInt }
            },
            resolve: (parent, args) => authors.find(author => author.id ===
args.id)
        }
    })
})

const RootMutationType = new GraphQLObjectType ({
    name: 'Mutation',
    description: 'Root Mutation',
    fields: () => ({
        addBook: {
            type: BookType,
            description: 'Add a book',
            args: {
```

```
                name: { type: GraphQLNonNull(GraphQLString) },
                authorid: { type: GraphQLNonNull(GraphQLInt) }
            },
            resolve: (parent, args) => {
                const book = { id: books.length + 1, name: args.name, authorid:
args.authorid }
                books.push(book)
                return book
            }
        },

        addAuthor: {
            type: AuthorType,
            description: 'Add an author',
            args: {
                name: { type: GraphQLNonNull(GraphQLString) }
            },
            resolve: (parent, args) => {
                const author = { id: authors.length + 1, name: args.name }
                authors.push(author)
                return author
            }
        }

    })
})

const schema = new GraphQLSchema ({
    query: RootQueryType,
    mutation: RootMutationType
})


app.use('/graphql', graphqlHTTP({
    schema: schema,
    graphiql: true
}))

app.listen(5000., () => console.log('Server Running'))
```