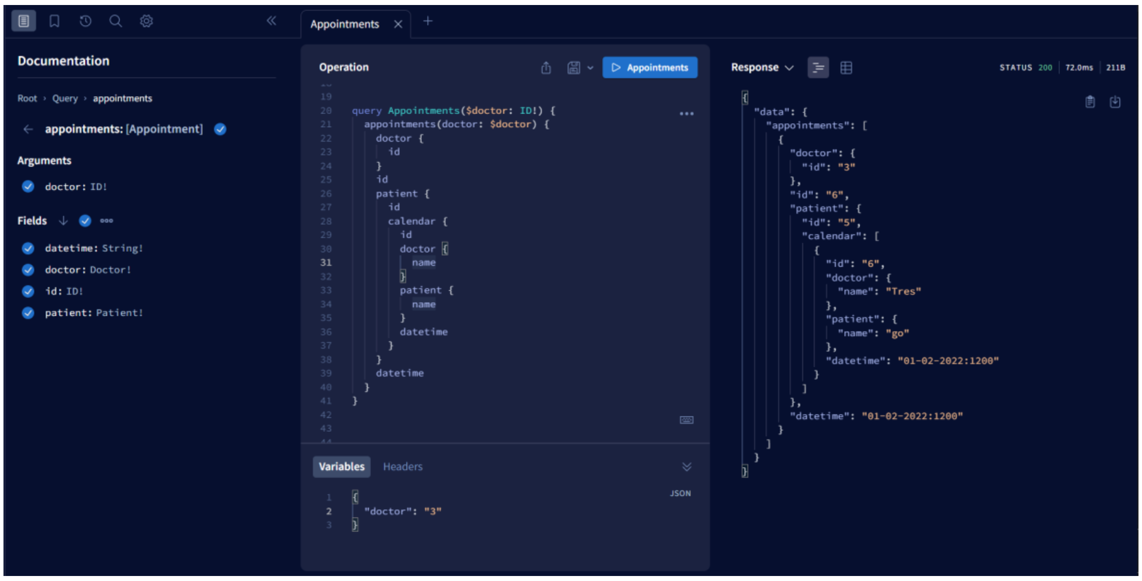
Assignment 2.2 – GraphQL API Design

API Tests

TestCase Identifier: Get Appointments

Description: Gets a list of today's appointments for specified doctor

Test Screen Shots:



```
query Appointments($doctor: ID!) {
  appointments(doctor: $doctor) {
    doctor {
      id
    }
    id
    patient {
      id
      calendar {
        id
        doctor {
          name
        }
        patient {
          name
        }
        datetime
      }
    }
    datetime
  }
}
```

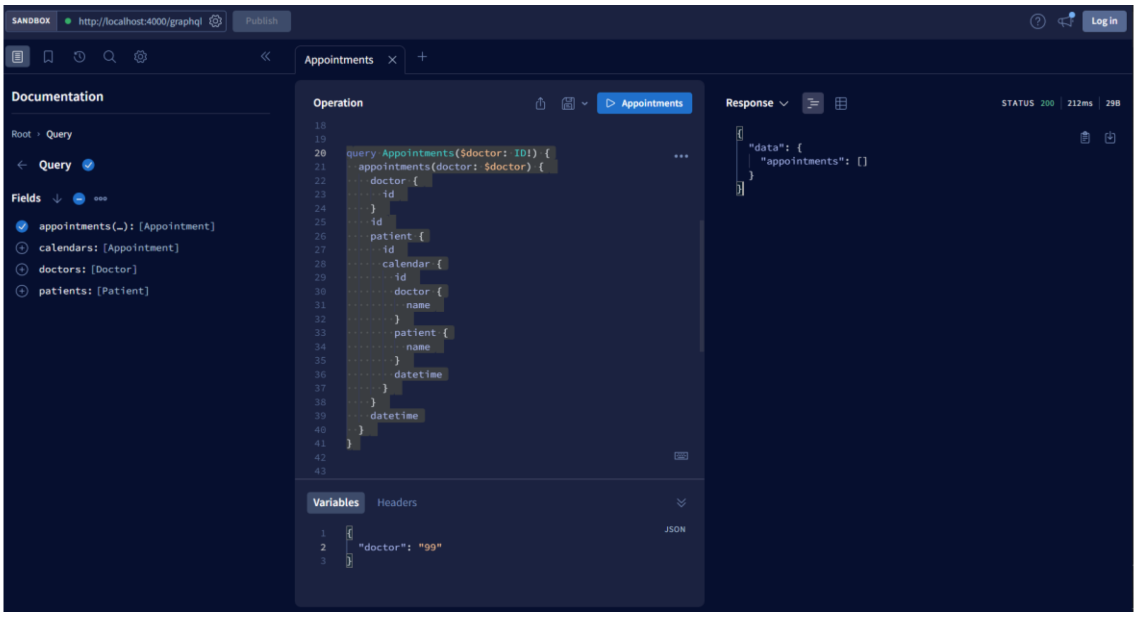
```
{
  "doctor": "3"
}
```

```
{
  "data": {
    "appointments": [
      {
        "doctor": {
          "id": "3"
        },
        "id": "6",
        "patient": {
          "id": "5",
          "calendar": [
            {
              "id": "6",
              "doctor": {
                "name": "Tres"
              },
              "patient": {
                "name": "go"
              },
              "datetime": "01-02-2022:1200"
            }
          ]
        },
        "datetime": "01-02-2022:1200"
      }
    ]
  }
}
```

TestCase Identifier: Get Appointment Invalid Doctor

Description: Gets list of appointments, but specified doctor does not exist

Test Screen Shots:



```
query Appointments($doctor: ID!) {
  appointments(doctor: $doctor) {
    doctor {
      id
    }
    id
    patient {
      id
      calendar {
        id
        doctor {
          name
        }
        patient {
          name
        }
        datetime
      }
    }
    datetime
  }
}
```

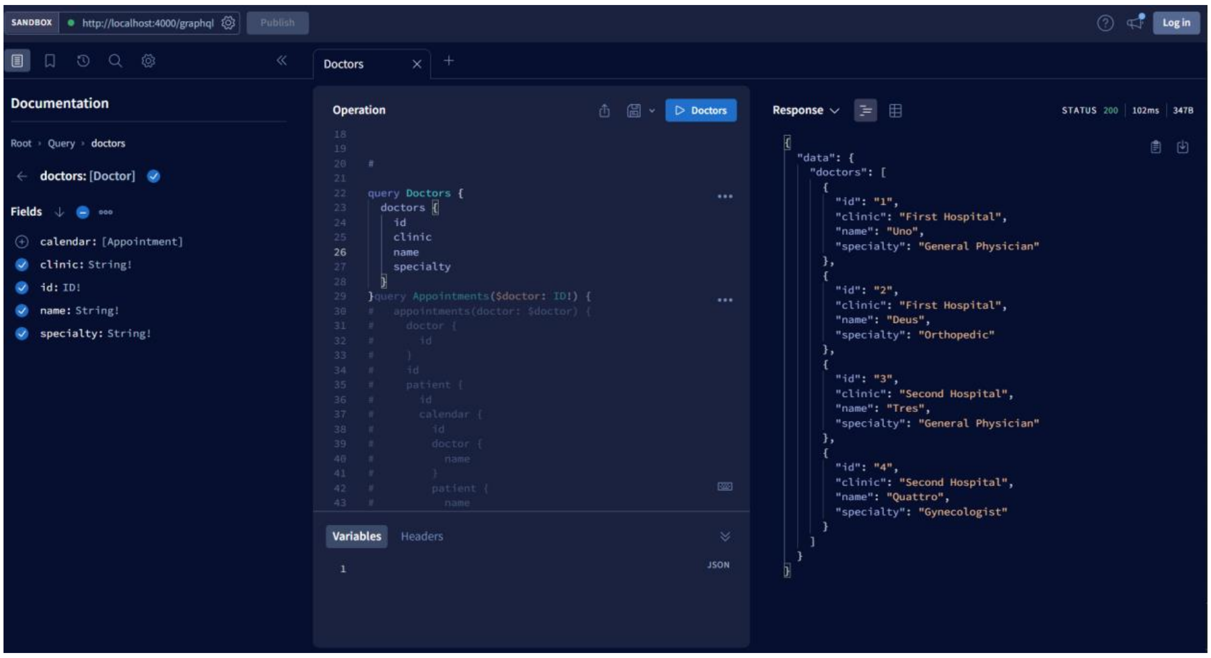
```
{
  "doctor": "99"
}
```

```
{
  "data": {
    "appointments": []
  }
}
```

TestCase Identifier: Get Doctor

Description: Gets list of doctors

Test Screen Shots:



```
calendar: [Appointment]
clinic: String!
id: ID!
name: String!
specialty: String!
```

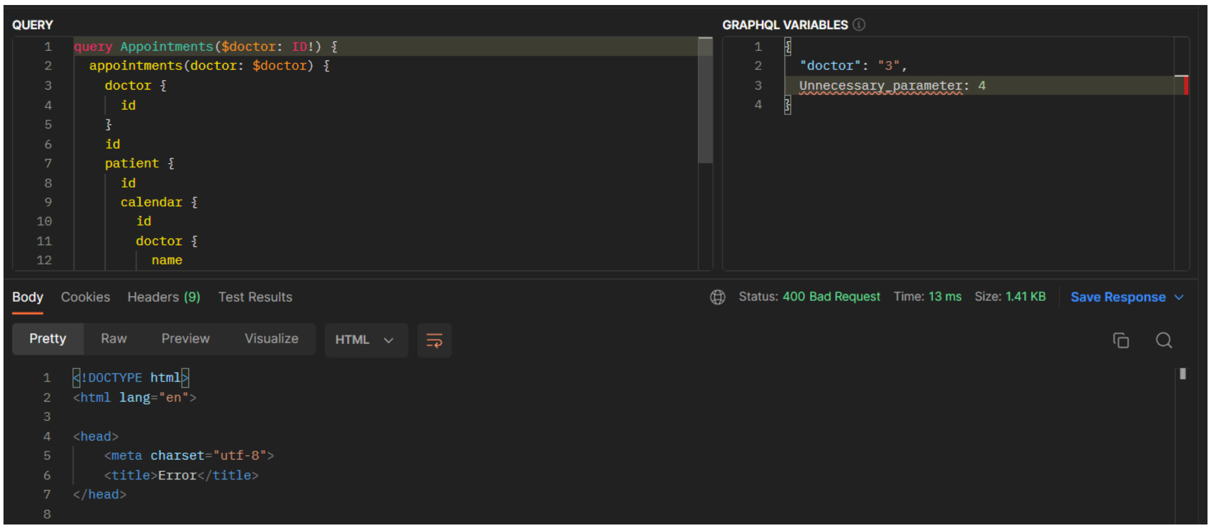
```
#

query Doctors {
  doctors {
    id
    clinic
    name
    specialty
  }
}query Appointments($doctor: ID!) {
#   appointments(doctor: $doctor) {
#     doctor {
#       id
#     }
#     id
#     patient {
#       id
#       calendar {
#         id
#         doctor {
#           name
#         }
#         patient {
#           name
```

```
{
  "data": {
    "doctors": [
      {
        "id": "1",
        "clinic": "First Hospital",
        "name": "Uno",
        "specialty": "General Physician"
      },
      {
        "id": "2",
        "clinic": "First Hospital",
        "name": "Deus",
        "specialty": "Orthopedic"
      },
      {
        "id": "3",
        "clinic": "Second Hospital",
        "name": "Tres",
        "specialty": "General Physician"
      },
      {
        "id": "4",
        "clinic": "Second Hospital",
        "name": "Quattro",
        "specialty": "Gynecologist"
      }
    ]
  }
}
```

TestCase Identifier: Get Doctor

Description: Gets list of doctors, but unwanted query is added

Test Screen Shots:



```
query Appointments($doctor: ID!) {
  appointments(doctor: $doctor) {
    doctor {
      id
    }
    id
    patient {
      id
      calendar {
        id
        doctor {
          name
```

```
{
  "doctor": "3",
  Unnecessary_parameter: 4
}
```

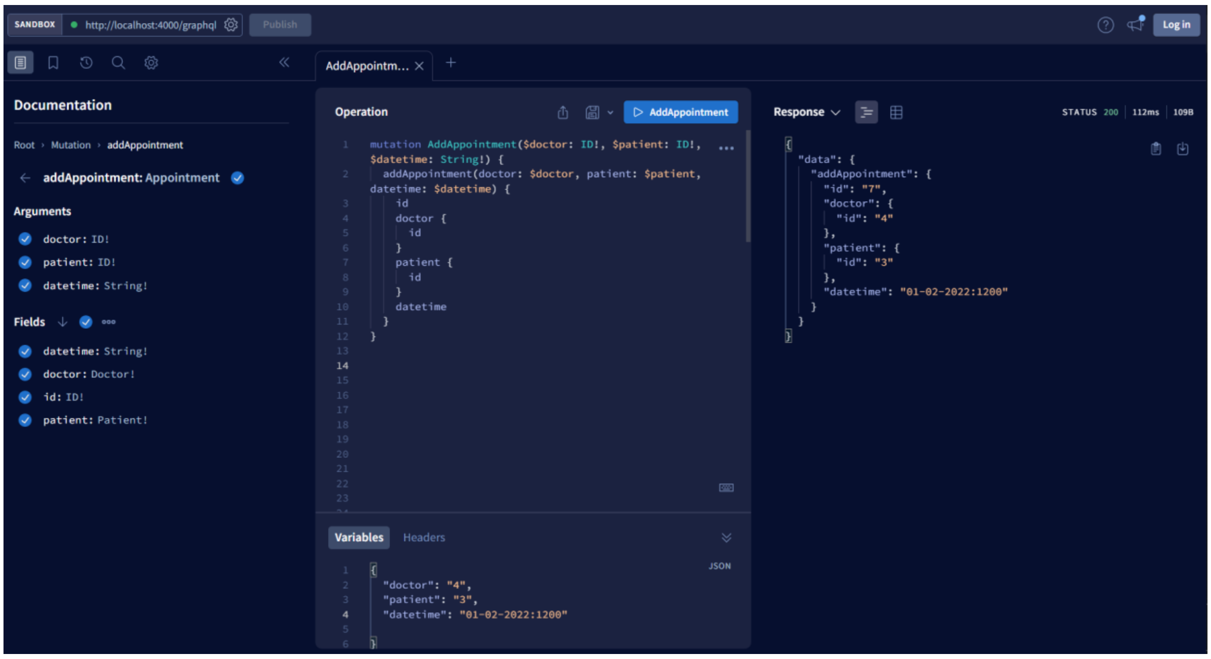
Status: 400 Bad Request Time: 13 ms Size: 1.41 KB

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="utf-8">
    <title>Error</title>
</head>

```

TestCase Identifier: Book Appointment

Description: Books an appointment for a certain doctor and patient at certain date-time

Test Screen Shots:



Root › Mutation › addAppointment

addAppointment: Appointment

Arguments

- doctor: ID!
- patient: ID!
- datetime: String!

Fields

- datetime: String!
- doctor: Doctor!
- id: ID!
- patient: Patient!

```
mutation AddAppointment($doctor: ID!, $patient: ID!,
$datetime: String!) {
  addAppointment(doctor: $doctor, patient: $patient,
datetime: $datetime) {
    id
    doctor {
      id
    }
    patient {
      id
    }
    datetime
  }
}
```

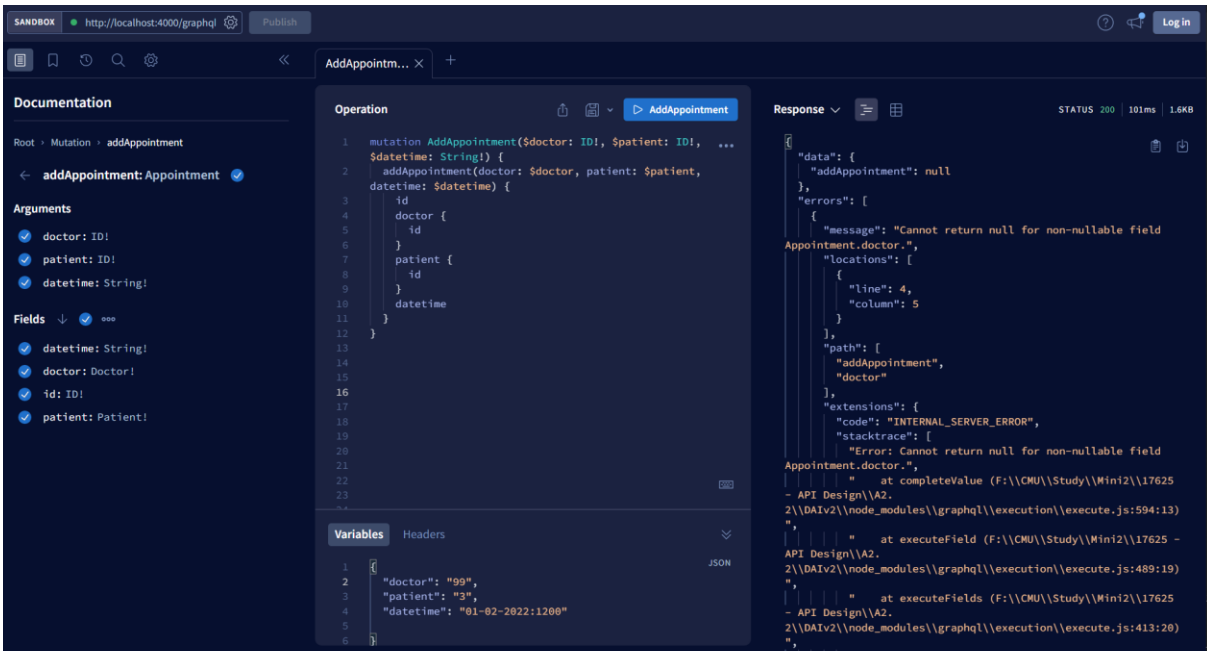
```
{
  "doctor": "4",
  "patient": "3",
  "datetime": "01-02-2022:1200"

}
```

```
{
  "data": {
    "addAppointment": {
      "id": "7",
      "doctor": {
        "id": "4"
      },
      "patient": {
        "id": "3"
      },
      "datetime": "01-02-2022:1200"
    }
  }
}
```

TestCase Identifier: Book Appointment Invalid Doctor

Description: Books an appointment for a doctor and patient at date-time, but doctor does not exist

Test Screen Shots:



Root › Mutation › addAppointment

addAppointment: Appointment

Arguments

- doctor: ID!
- patient: ID!
- datetime: String!

Fields

- datetime: String!
- doctor: Doctor!
- id: ID!
- patient: Patient!

```
mutation AddAppointment($doctor: ID!, $patient: ID!,
$datetime: String!) {
  addAppointment(doctor: $doctor, patient: $patient,
datetime: $datetime) {
    id
    doctor {
      id
    }
    patient {
      id
    }
    datetime
  }
}
```

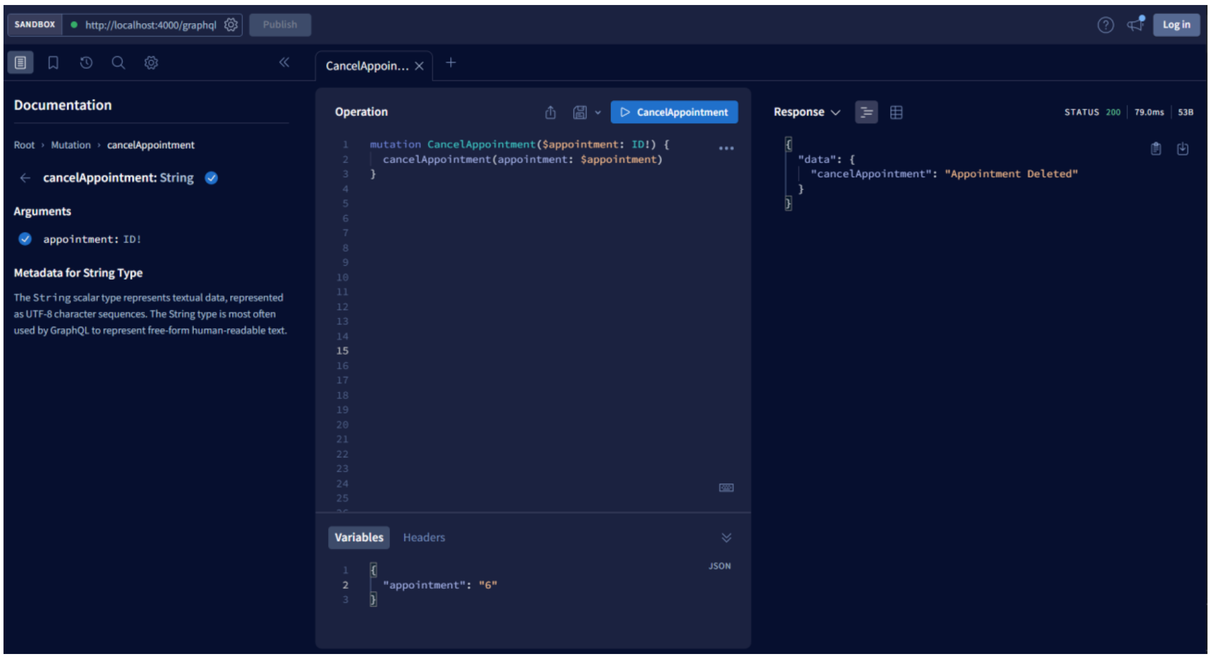
```
{
  "doctor": "99",
  "patient": "3",
  "datetime": "01-02-2022:1200"

}
```

```
{
  "data": {
    "addAppointment": null
  },
  "errors": [
    {
      "message": "Cannot return null for non-nullable field
Appointment.doctor.",
      "locations": [
        {
          "line": 4,
          "column": 5
        }
      ],
      "path": [
        "addAppointment",
        "doctor"
      ],
      "extensions": {
        "code": "INTERNAL_SERVER_ERROR",
        "stacktrace": [
          "Error: Cannot return null for non-nullable field
Appointment.doctor.",
          "    at completeValue (F:\\CMU\\Study\\Mini2\\17625
- API Design\\A2.
2\\DAIv2\\node_modules\\graphql\\execution\\execute.js:594:13)
",
          "    at executeField (F:\\CMU\\Study\\Mini2\\17625 -
API Design\\A2.
2\\DAIv2\\node_modules\\graphql\\execution\\execute.js:489:19)
",
          "    at executeFields (F:\\CMU\\Study\\Mini2\\17625
- API Design\\A2.
2\\DAIv2\\node_modules\\graphql\\execution\\execute.js:413:20)
",
```

TestCase Identifier: Cancel Appointment

Description: Cancels an existing appointment

Test Screen Shots:



cancelAppointment: String

Arguments

appointment: ID!

Metadata for String Type

The String scalar type represents textual data, represented as UTF-8 character sequences. The String type is most often used by GraphQL to represent free-form human-readable text.

```
mutation CancelAppointment($appointment: ID!) {
  cancelAppointment(appointment: $appointment)
}
```

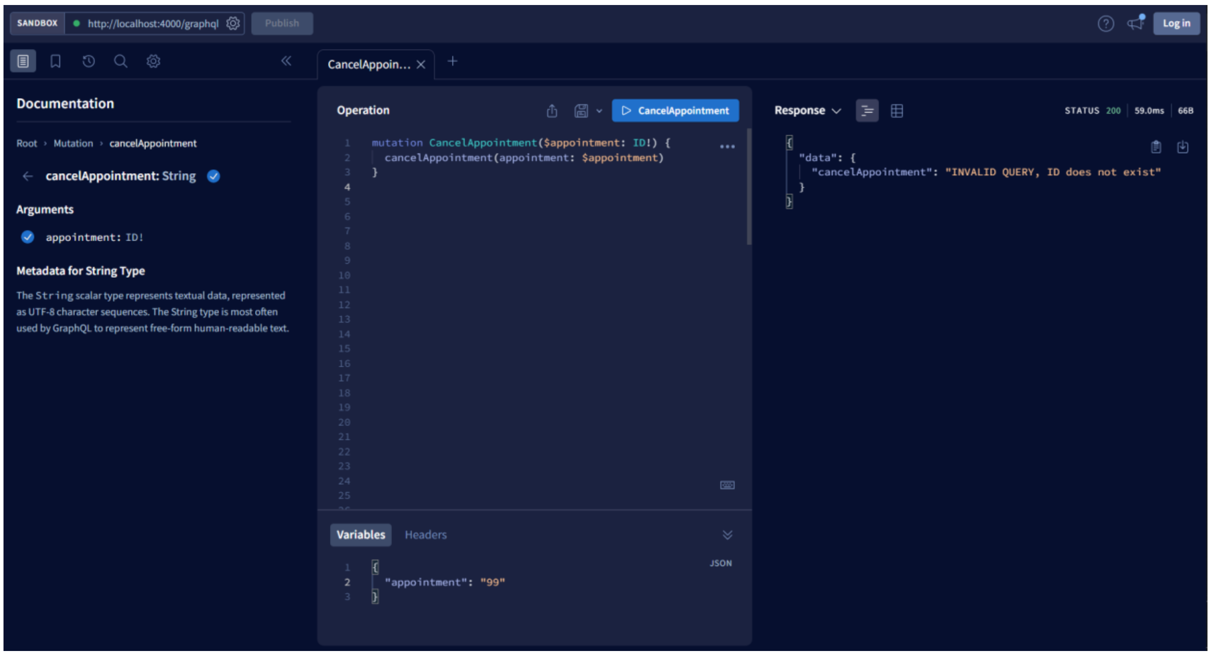
```
{
  "appointment": "6"
}
```

```
{
  "data": {
    "cancelAppointment": "Appointment Deleted"
  }
}
```

TestCase Identifier: Cancel Appointment, Invalid Appointment

Description: Cancels an appointment, but the appointment does not exist

Test Screen Shots:



cancelAppointment: String

Arguments

appointment: ID!

Metadata for String Type

The String scalar type represents textual data, represented as UTF-8 character sequences. The String type is most often used by GraphQL to represent free-form human-readable text.

```
mutation CancelAppointment($appointment: ID!) {
  cancelAppointment(appointment: $appointment)
}
```

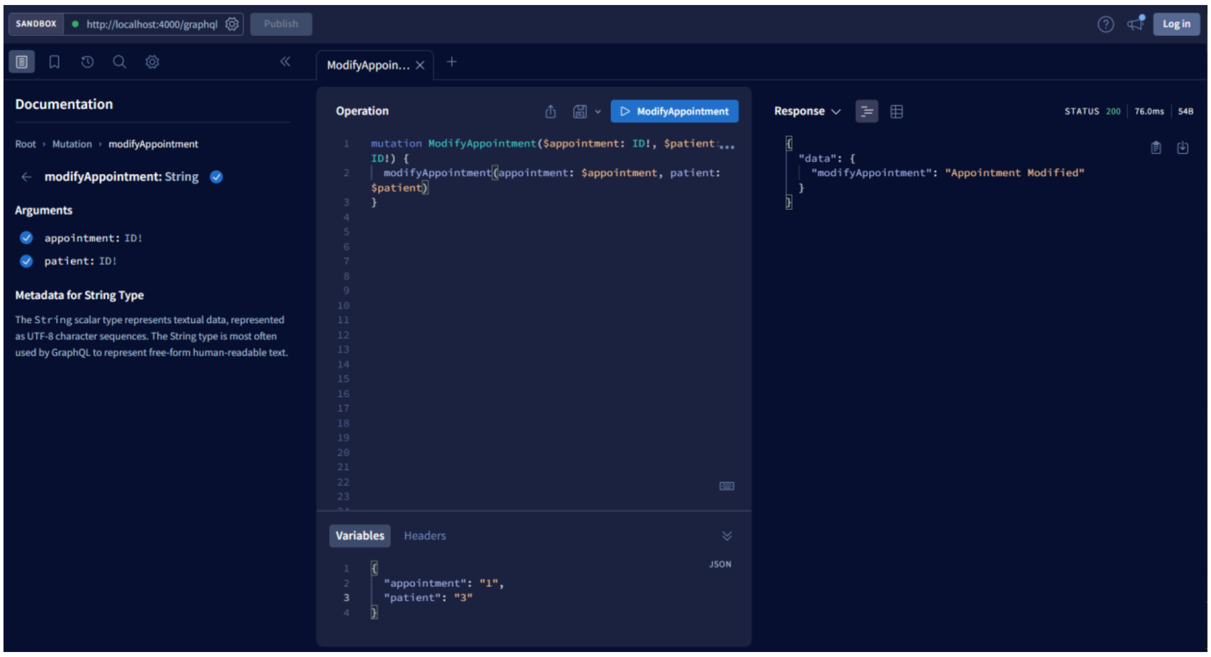
```
{
  "appointment": "99"
}
```

```
{
  "data": {
    "cancelAppointment": "INVALID QUERY, ID does not exist"
  }
}
```

TestCase Identifier: Modify Appointment

Description: Changes Patient in existing appointment

Test Screen Shots:



SANDBOX http://localhost:4000/graphql Publish Log in

ModifyAppoin...

Documentation

Root › Mutation › modifyAppointment

modifyAppointment: String

Arguments

appointment: ID!

patient: ID!

Metadata for String Type

The String scalar type represents textual data, represented as UTF-8 character sequences. The String type is most often used by GraphQL to represent free-form human-readable text.

Operation

```
mutation ModifyAppointment($appointment: ID!, $patient:...
ID!) {
  modifyAppointment(appointment: $appointment, patient:
$patient)
}
```

Variables Headers JSON

```
{
  "appointment": "1",
  "patient": "3"
}
```

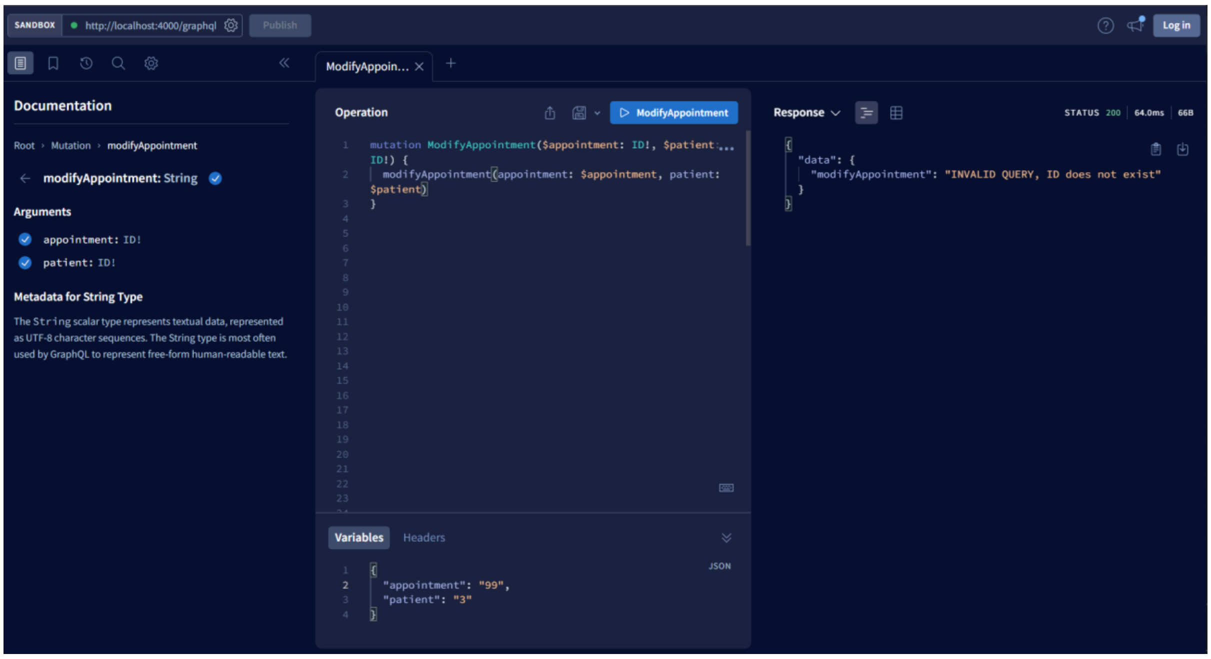
Response STATUS 200 | 76.0ms | 54B

```
{
  "data": {
    "modifyAppointment": "Appointment Modified"
  }
}
```

TestCase Identifier: Modify Appointment, Invalid Query

Description: Changes patient in existing appointment, but appointment does not exist

Test Screen Shots:



SANDBOX http://localhost:4000/graphql Publish Log in

ModifyAppoin...

Documentation

Root › Mutation › modifyAppointment

modifyAppointment: String

Arguments

appointment: ID!

patient: ID!

Metadata for String Type

The String scalar type represents textual data, represented as UTF-8 character sequences. The String type is most often used by GraphQL to represent free-form human-readable text.

Operation

```
mutation ModifyAppointment($appointment: ID!, $patient:...
ID!) {
  modifyAppointment(appointment: $appointment, patient:
$patient)
}
```

Variables Headers JSON

```
{
  "appointment": "99",
  "patient": "3"
}
```

Response STATUS 200 | 64.0ms | 66B

```
{
  "data": {
    "modifyAppointment": "INVALID QUERY, ID does not exist"
  }
}
```

Reflections

Alternative Query and Schema design options:

The schema could be designed to be more extensible by using a “User” category that then branches into “doctor” and “patient” sub-categories.

For the queries as well, it would be better to create queries that can modify or delete appointments based on any unique identifiers (doctor-patient-datetime pairs) rather than picking up the appointment ID.

However, given the limited scope of the assignment, adding this additional extensibility could increase the complexity of the code several times. Also, the incorporation of additional elements can create points of failure/breakage. Hence, given the limited scope of the assignment, the simplest approach seems the most suitable.

Event structure changes:

Since I am already using a Patient object as part of the Event or Appointment object, my API will be able to accommodate additional information added about patients. There will be no changes from the client side as the existing queries will work without any issues.

If fields like consultation type are added to the event itself, then those changes will have to be made in the schema and the resolvers for the appointments. From the client side, the existing queries will still work without any issues, but those wanting to use these new fields will have to add them to their queries.

Overall, the existence of patient and doctor objects is a good design philosophy, as we can keep changes in these two stakeholders separate from the Event/Appointment objects.

GraphQL best practices used:

I have used the appropriate naming conventions for naming mutations/queries. For example: addAppointment, modifyAppointment, cancelAppointment mutations.

I have used the “ID” datatype as the primary key for all the classes in the schema (DoctorID, PatientID, AppointmentID),

I have also incorporated extensible elements like the usage of object datatypes in the schema instead of using scalars only. For example: the appointment class uses objects of the Doctor and Patient classes to represent the two parties in the appointment. This practice ensures that the appointment class is free from any changes that might happen to these parties in the future, and queries will not need to be changed.