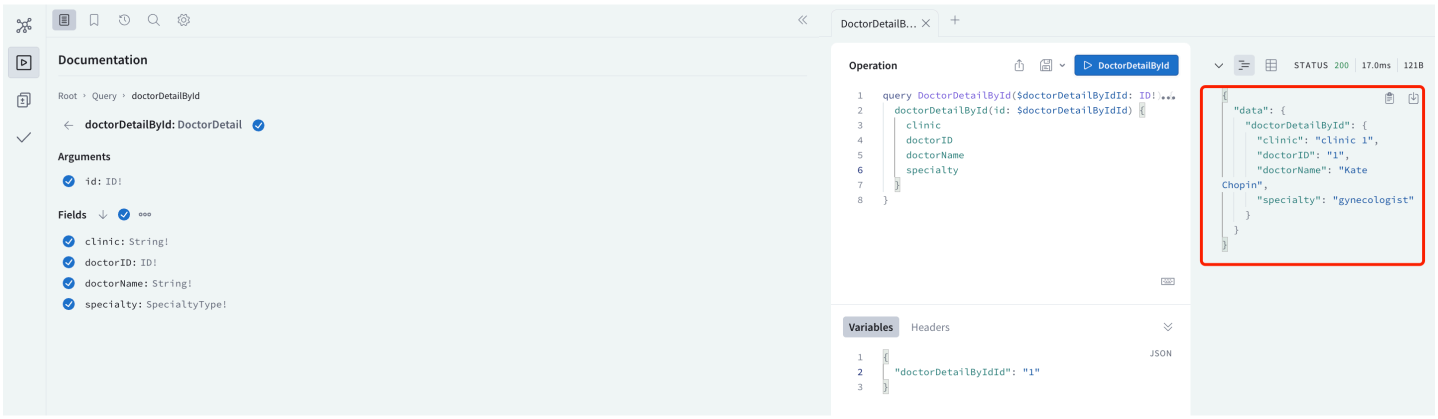
## *Testing*

***Name: doctorDetailById():*** Get doctor details (name, clinic name, specialty)

In order to test this query, I need to hardcode the doctor data in memory, and the happy path testing result is as below:



Documentation

Root › Query › doctorDetailById

doctorDetailById: DoctorDetail

Arguments

id: ID!

Fields

clinic: String!
doctorID: ID!
doctorName: String!
specialty: SpecialtyType!

Operation

```
query DoctorDetailById($doctorDetailByIdId: ID!)...
  doctorDetailById(id: $doctorDetailByIdId) {
    clinic
    doctorID
    doctorName
    specialty
  }
}
```

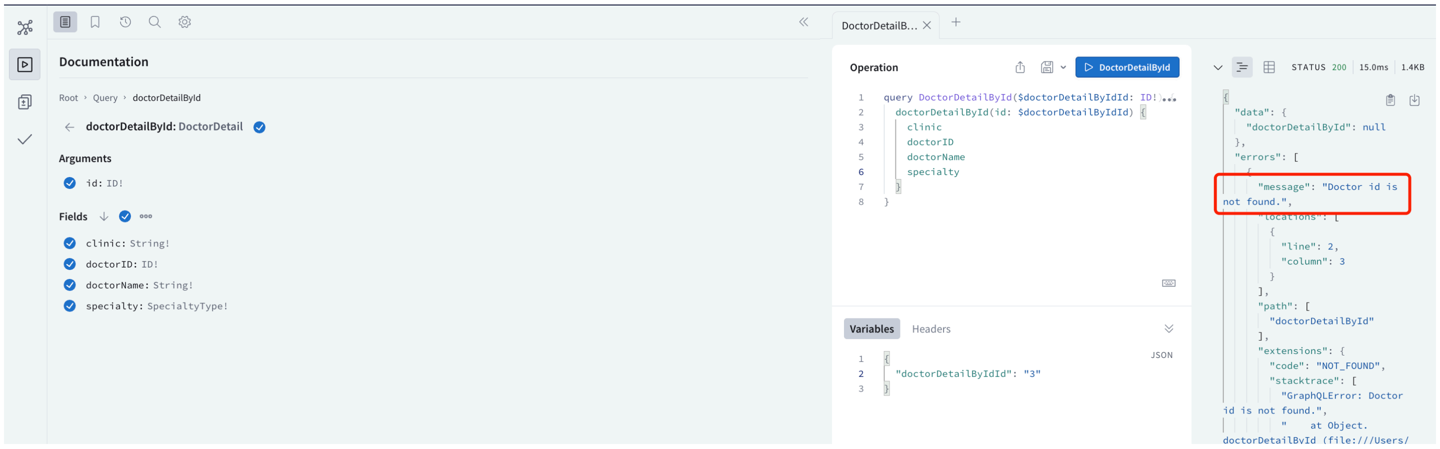
Variables

```
{
  "doctorDetailByIdId": "1"
}
```

STATUS 200 | 17.0ms | 121B

```
{
  "data": {
    "doctorDetailById": {
      "clinic": "clinic 1",
      "doctorID": "1",
      "doctorName": "Kate
Chopin",
      "specialty": "gynecologist"
    }
  }
}
```

Since the doctor id is the mandatory filed in this query and the input id should be valid, which means the input doctor id should be in the database, if I choose to enter a doctor id which is not in the database, my APIs has the customized exception handling shown as below:



Documentation

Root › Query › doctorDetailById

doctorDetailById: DoctorDetail

Arguments

id: ID!

Fields

clinic: String!
doctorID: ID!
doctorName: String!
specialty: SpecialtyType!

Operation

```
query DoctorDetailById($doctorDetailByIdId: ID!)...
  doctorDetailById(id: $doctorDetailByIdId) {
    clinic
    doctorID
    doctorName
    specialty
  }
}
```

Variables

```
{
  "doctorDetailByIdId": "3"
}
```

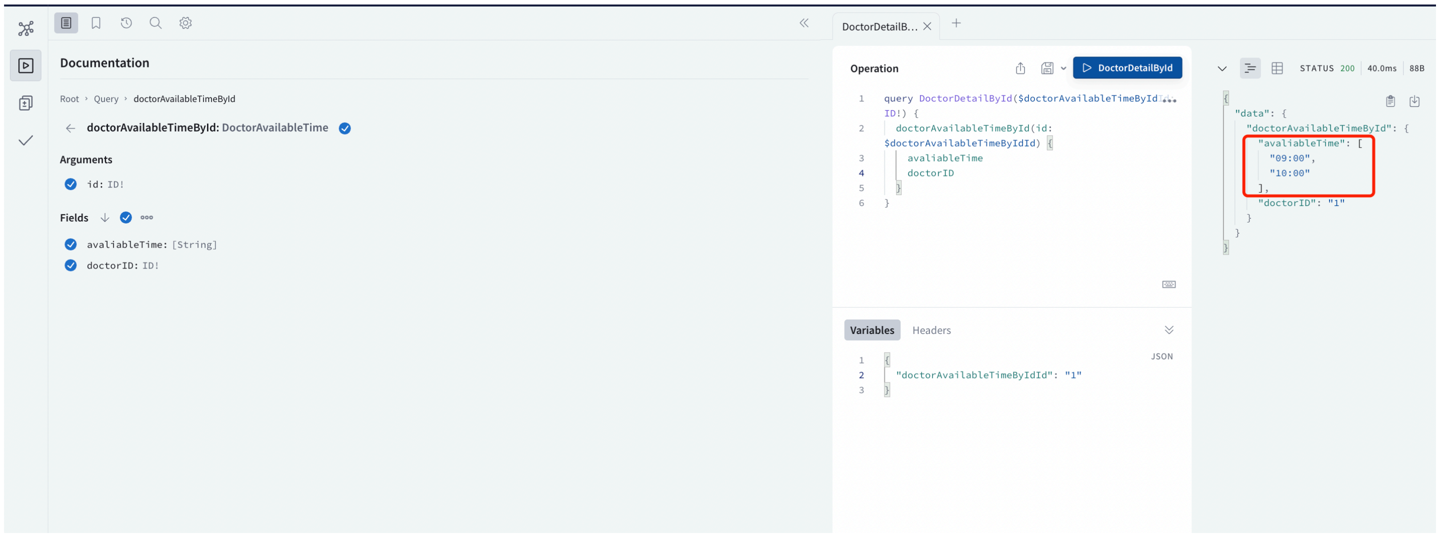
STATUS 200 | 15.0ms | 1.4KB

```
{
  "data": {
    "doctorDetailById": null
  },
  "errors": [
    {
      "message": "Doctor id is
not found.",
      "locations": [
        {
          "line": 2,
          "column": 3
        }
      ],
      "path": [
        "doctorDetailById"
      ],
      "extensions": {
        "code": "NOT_FOUND",
        "stacktrace": [
          "GraphQLError: Doctor
id is not found.",
          "    at Object.
doctorDetailById (file:///Users/
```

---

***Name: doctorAvailableTimeById():*** Get doctor's available timeslots for today

The input is the doctor id. The happy path is shown as below:



doctorAvailableTimeById: DoctorAvailableTime

Arguments

id: ID!

Fields

avaliableTime: [String]

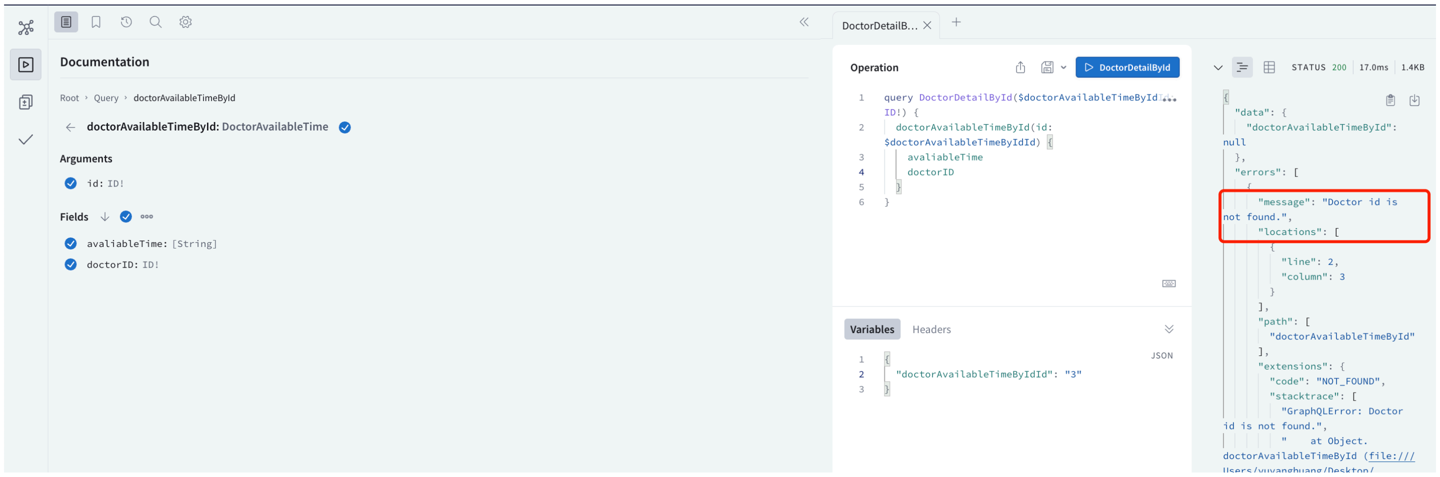
doctorID: ID!

```
query DoctorDetailById($doctorAvailableTimeByIdId: ID!) {
  doctorAvailableTimeById(id: $doctorAvailableTimeByIdId) {
    avaliableTime
    doctorID
  }
}
```

```
{
  "doctorAvailableTimeByIdId": "1"
}
```

```
{
  "data": {
    "doctorAvailableTimeById": {
      "avaliableTime": [
        "09:00",
        "10:00"
      ],
      "doctorID": "1"
    }
  }
}
```

Since the doctor id is the mandatory filed in this query and the input id should be valid, which means the input doctor id should be in the database, if I choose to enter a doctor id which is not in the database, my APIs has the customized exception handling shown as below:



doctorAvailableTimeById: DoctorAvailableTime

Arguments

id: ID!

Fields

avaliableTime: [String]

doctorID: ID!

```
query DoctorDetailById($doctorAvailableTimeByIdId: ID!) {
  doctorAvailableTimeById(id: $doctorAvailableTimeByIdId) {
    avaliableTime
    doctorID
  }
}
```

```
{
  "doctorAvailableTimeByIdId": "3"
}
```

```
{
  "data": {
    "doctorAvailableTimeById": null
  },
  "errors": [
    {
      "message": "Doctor id is not found.",
      "locations": [
        {
          "line": 2,
          "column": 3
        }
      ],
      "path": [
        "doctorAvailableTimeById"
      ],
      "extensions": {
        "code": "NOT_FOUND",
        "stacktrace": [
          "GraphQLError: Doctor id is not found.",
          "    at Object.doctorAvailableTimeById (file:///Users/yuyanghuang/Desktop/
```

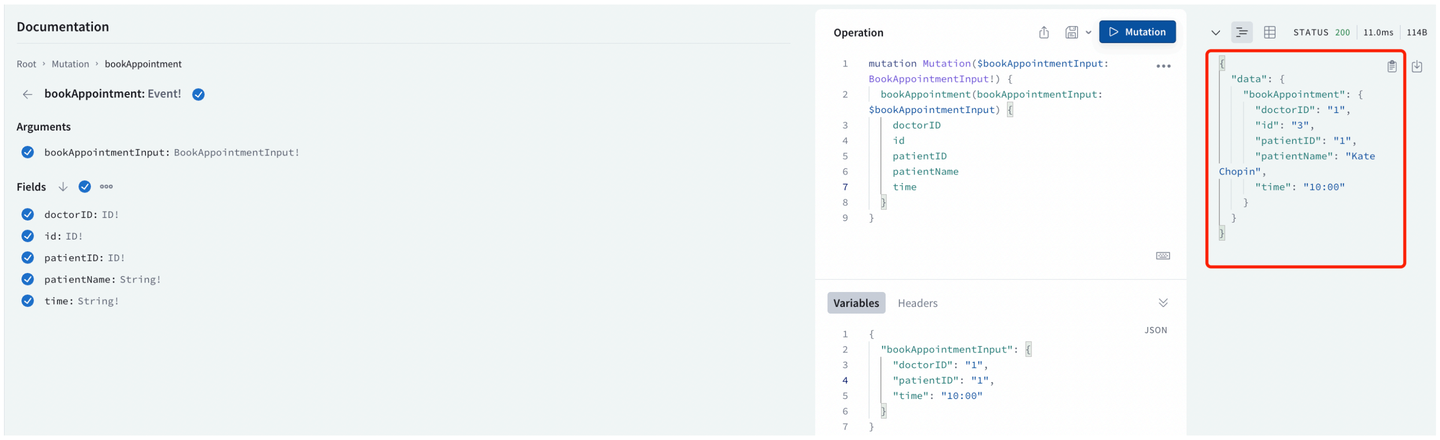
---

***Name: bookAppointment():*** Book an appointment with a doctor for today

Before executing the functions, I do 4 error checks:

- check doctor id: the input doctor id should be in the database
- check patitent id: the input patient id should be in the database
- check time format: the input time should be in "hh:mm" format
- check whether the doctor has time available for the input time.

Happy path:



bookAppointment: Event!

Arguments

```
bookAppointmentInput: BookAppointmentInput!
```

Fields

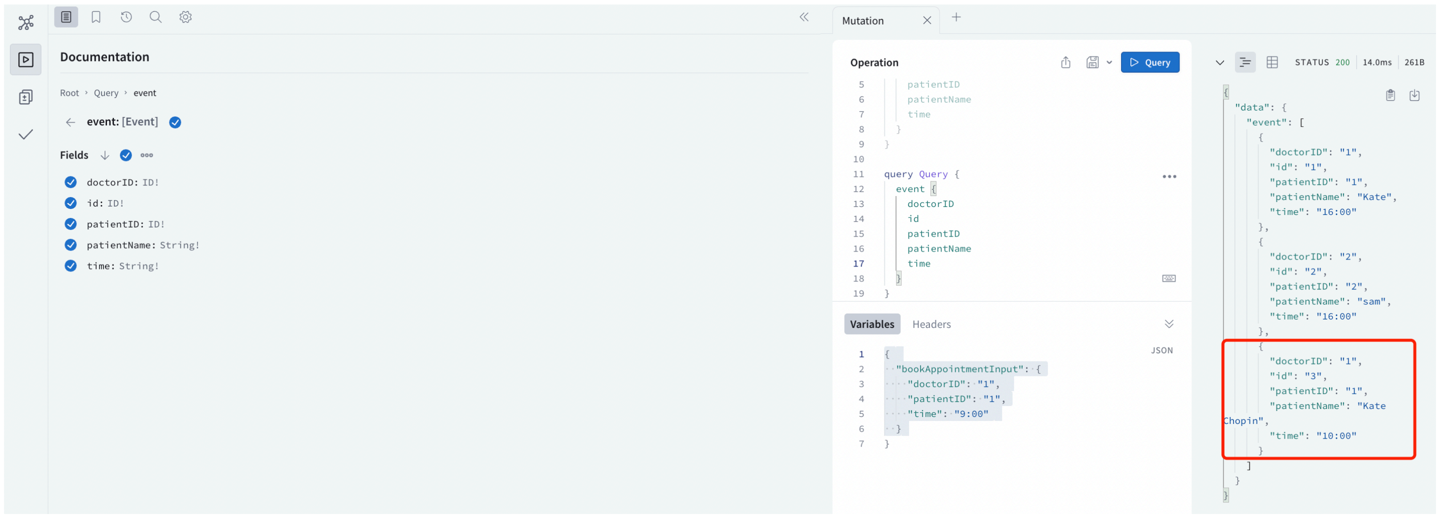
```
doctorID: ID!
id: ID!
patientID: ID!
patientName: String!
time: String!
```

```
mutation Mutation($bookAppointmentInput: BookAppointmentInput!) {
  bookAppointment(bookAppointmentInput: $bookAppointmentInput) {
    doctorID
    id
    patientID
    patientName
    time
  }
}
```

```
{
  "bookAppointmentInput": {
    "doctorID": "1",
    "patientID": "1",
    "time": "10:00"
  }
}
```

```
{
  "data": {
    "bookAppointment": {
      "doctorID": "1",
      "id": "3",
      "patientID": "1",
      "patientName": "Kate Chopin",
      "time": "10:00"
    }
  }
}
```

If the event is created we can see the newly added in event in the event list database:



event: [Event]

Fields

```
doctorID: ID!
id: ID!
patientID: ID!
patientName: String!
time: String!
```

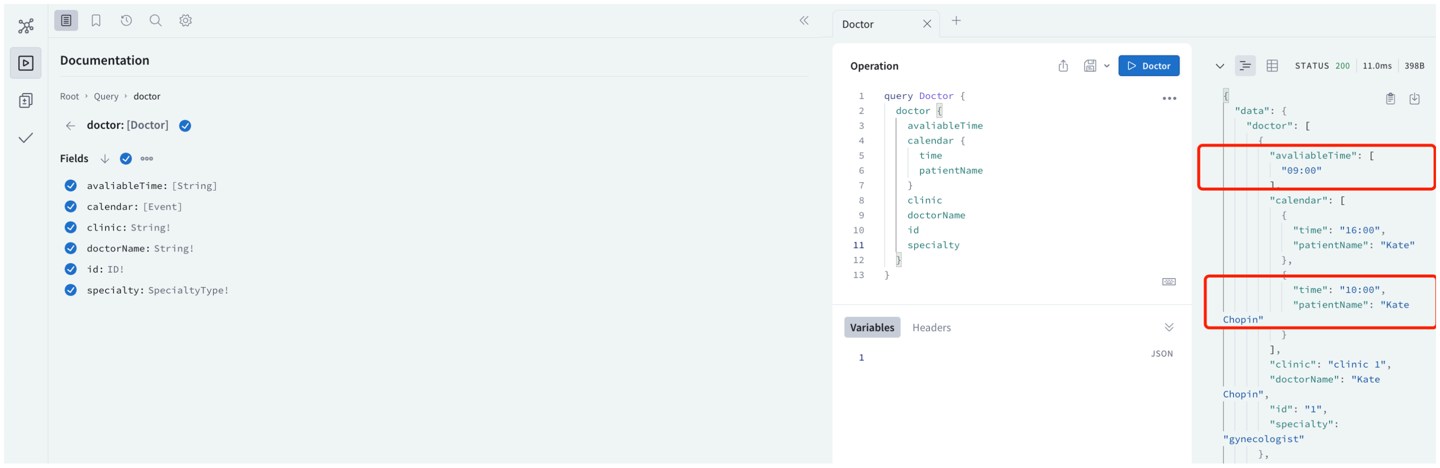
```
    patientID
    patientName
    time
  }
}

query Query {
  event {
    doctorID
    id
    patientID
    patientName
    time
  }
}
```

```
{
  "bookAppointmentInput": {
    "doctorID": "1",
    "patientID": "1",
    "time": "9:00"
  }
}
```

```
{
  "data": {
    "event": [
      {
        "doctorID": "1",
        "id": "1",
        "patientID": "1",
        "patientName": "Kate",
        "time": "16:00"
      },
      {
        "doctorID": "2",
        "id": "2",
        "patientID": "2",
        "patientName": "sam",
        "time": "16:00"
      },
      {
        "doctorID": "1",
        "id": "3",
        "patientID": "1",
        "patientName": "Kate Chopin",
        "time": "10:00"
      }
    ]
  }
}
```

And also the calendar(add event to doctor's calendar) and available time(remove booked timeslot from available time) field is updated based on the event:



doctor: [Doctor]

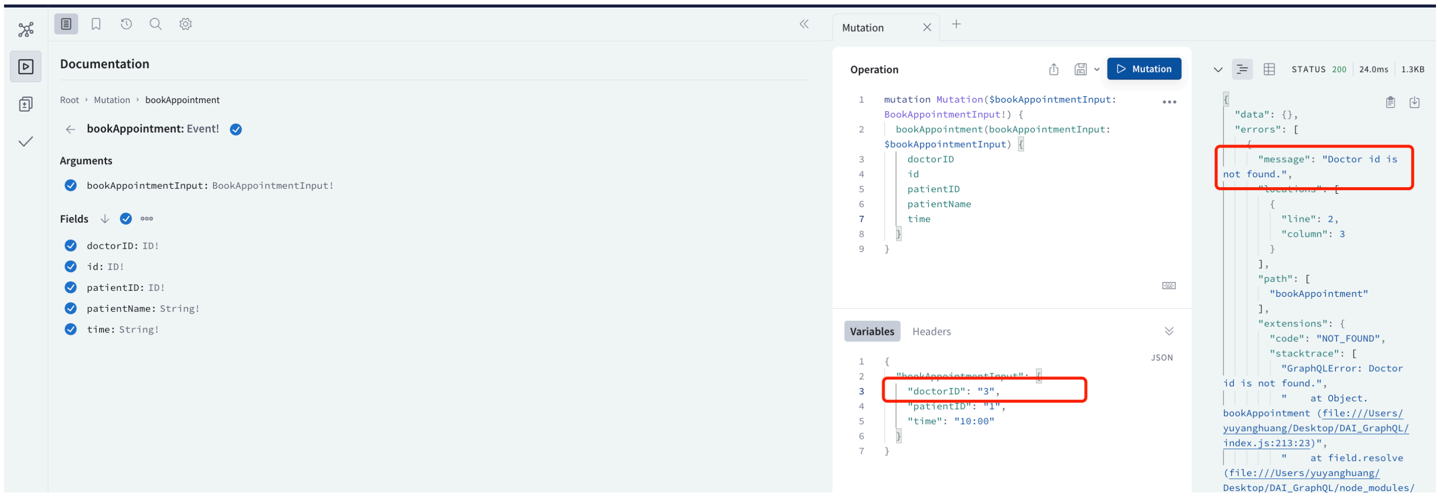
Fields

```
avaliableTime: [String]
calendar: [Event]
clinic: String!
doctorName: String!
id: ID!
specialty: SpecialtyType!
```

```
query Doctor {
  doctor {
    avaliableTime
    calendar {
      time
      patientName
    }
    clinic
    doctorName
    id
    specialty
  }
}
```

```
{
  "data": {
    "doctor": [
      {
        "avaliableTime": [
          "09:00"
        ],
        "calendar": [
          {
            "time": "16:00",
            "patientName": "Kate"
          },
          {
            "time": "10:00",
            "patientName": "Kate Chopin"
          }
        ],
        "clinic": "clinic 1",
        "doctorName": "Kate Chopin",
        "id": "1",
        "specialty": "gynecologist"
      },
```

Doctor id not valid(only have id with 1 or 2 in the database):



Documentation

Root › Mutation › bookAppointment

bookAppointment: Event!

Arguments

bookAppointmentInput: BookAppointmentInput!

Fields

doctorID: ID!
id: ID!
patientID: ID!
patientName: String!
time: String!

Operation

```
mutation Mutation($bookAppointmentInput: BookAppointmentInput!) {
  bookAppointment(bookAppointmentInput: $bookAppointmentInput) {
    doctorID
    id
    patientID
    patientName
    time
  }
}
```

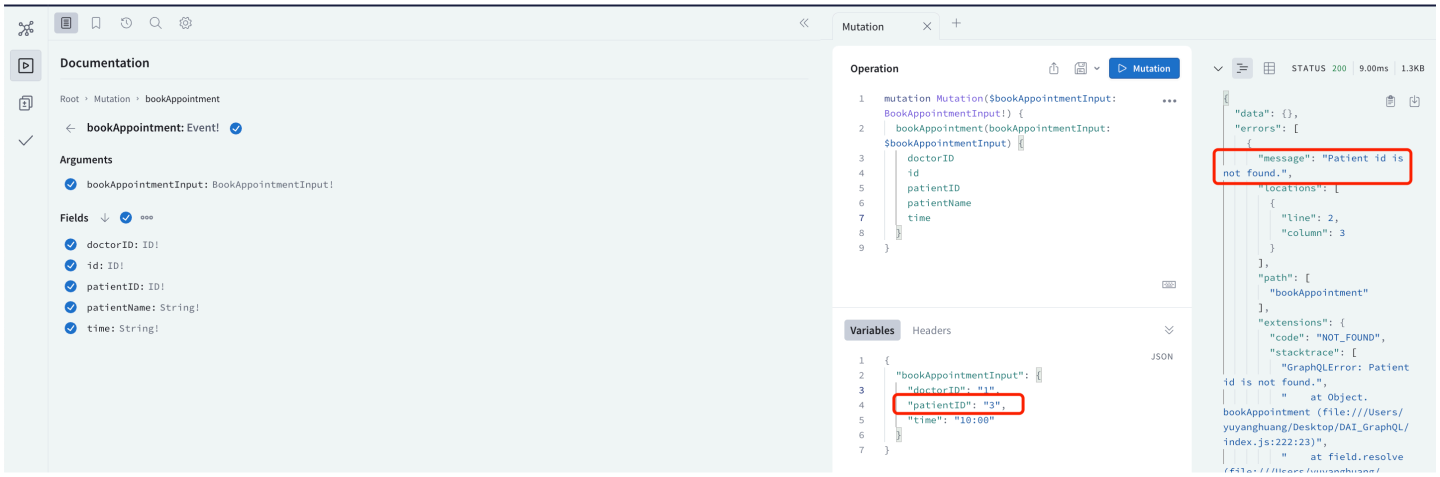
Variables

```
{
  "bookAppointmentInput": {
    "doctorID": "3",
    "patientID": "1",
    "time": "10:00"
  }
}
```

STATUS 200 24.0ms 1.3KB

```
{
  "data": {},
  "errors": [
    {
      "message": "Doctor id is not found.",
      "locations": [
        {
          "line": 2,
          "column": 3
        }
      ],
      "path": [
        "bookAppointment"
      ],
      "extensions": {
        "code": "NOT_FOUND",
        "stacktrace": [
          "GraphQLError: Doctor id is not found.",
          "    at Object.bookAppointment (file:///Users/yuyanghuang/Desktop/DAI_GraphQL/index.js:213:23)",
          "    at field.resolve (file:///Users/yuyanghuang/Desktop/DAI_GraphQL/node_modules/
```

Patient id not valid(only have id with 1 or 2 in the database):



Documentation

Root › Mutation › bookAppointment

bookAppointment: Event!

Arguments

bookAppointmentInput: BookAppointmentInput!

Fields

doctorID: ID!
id: ID!
patientID: ID!
patientName: String!
time: String!

Operation

```
mutation Mutation($bookAppointmentInput: BookAppointmentInput!) {
  bookAppointment(bookAppointmentInput: $bookAppointmentInput) {
    doctorID
    id
    patientID
    patientName
    time
  }
}
```

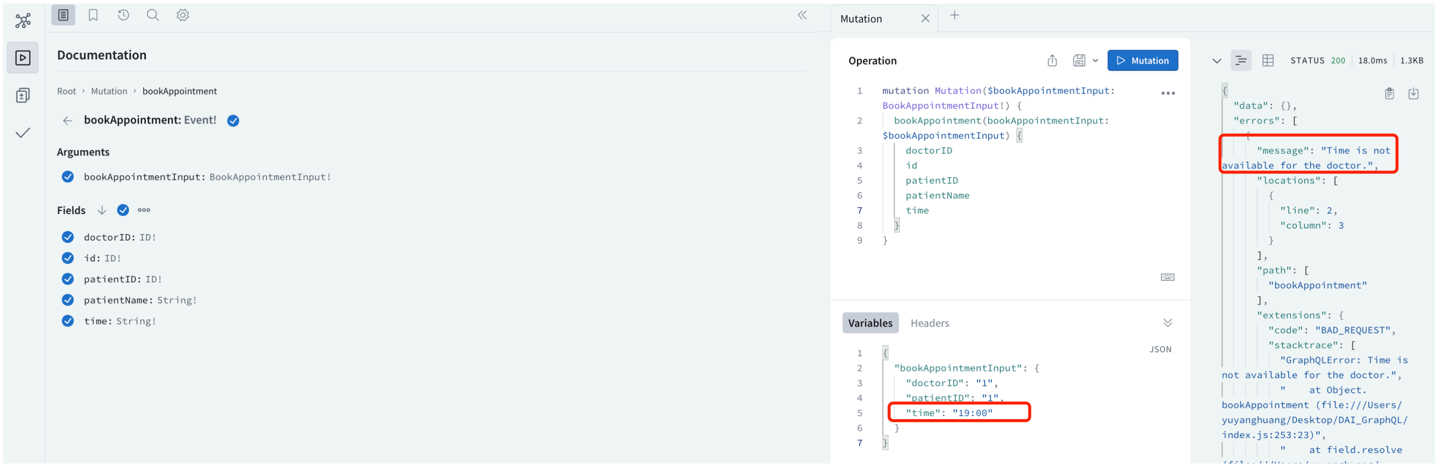
Variables

```
{
  "bookAppointmentInput": {
    "doctorID": "1",
    "patientID": "3",
    "time": "10:00"
  }
}
```

STATUS 200 9.00ms 1.3KB

```
{
  "data": {},
  "errors": [
    {
      "message": "Patient id is not found.",
      "locations": [
        {
          "line": 2,
          "column": 3
        }
      ],
      "path": [
        "bookAppointment"
      ],
      "extensions": {
        "code": "NOT_FOUND",
        "stacktrace": [
          "GraphQLError: Patient id is not found.",
          "    at Object.bookAppointment (file:///Users/yuyanghuang/Desktop/DAI_GraphQL/index.js:222:23)",
          "    at field.resolve
```

Time is not available for the given doctor:



Documentation

Root › Mutation › bookAppointment

bookAppointment: Event!

Arguments

bookAppointmentInput: BookAppointmentInput!

Fields

doctorID: ID!
id: ID!
patientID: ID!
patientName: String!
time: String!

Operation

```
mutation Mutation($bookAppointmentInput: BookAppointmentInput!) {
  bookAppointment(bookAppointmentInput: $bookAppointmentInput) {
    doctorID
    id
    patientID
    patientName
    time
  }
}
```

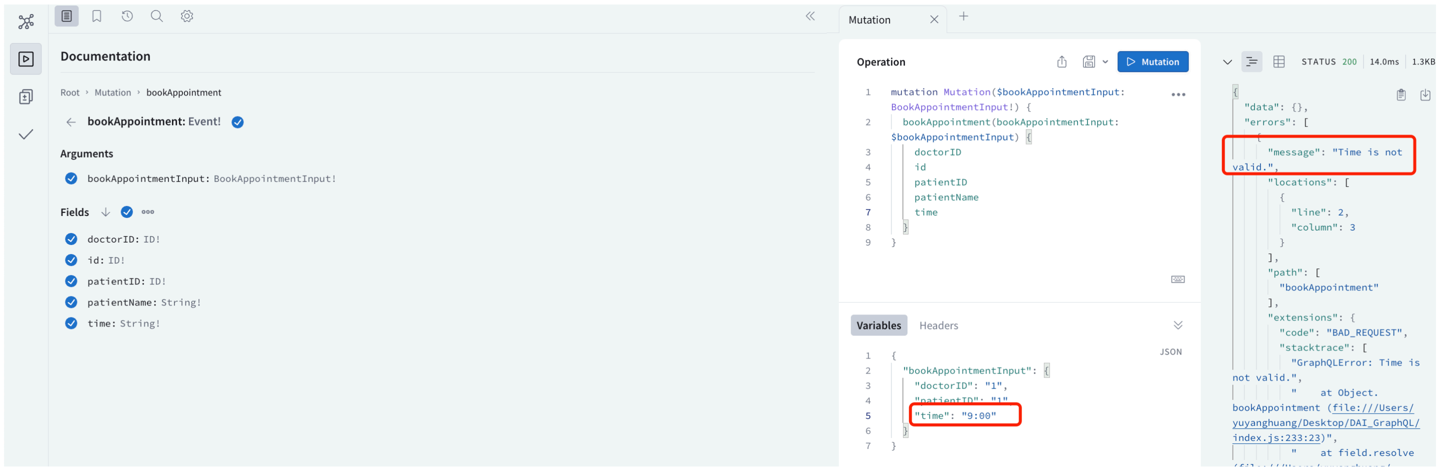
Variables

```
{
  "bookAppointmentInput": {
    "doctorID": "1",
    "patientID": "1",
    "time": "19:00"
  }
}
```

STATUS 200 18.0ms 1.3KB

```
{
  "data": {},
  "errors": [
    {
      "message": "Time is not available for the doctor.",
      "locations": [
        {
          "line": 2,
          "column": 3
        }
      ],
      "path": [
        "bookAppointment"
      ],
      "extensions": {
        "code": "BAD_REQUEST",
        "stacktrace": [
          "GraphQLError: Time is not available for the doctor.",
          "    at Object.bookAppointment (file:///Users/yuyanghuang/Desktop/DAI_GraphQL/index.js:253:23)",
          "    at field.resolve
```

Input time format is wrong:



Documentation

Root › Mutation › bookAppointment

bookAppointment: Event!

Arguments

bookAppointmentInput: BookAppointmentInput!

Fields

doctorID: ID!
id: ID!
patientID: ID!
patientName: String!
time: String!

Mutation

Operation

```
mutation Mutation($bookAppointmentInput:
BookAppointmentInput!) {
  bookAppointment(bookAppointmentInput:
$bookAppointmentInput) {
    doctorID
    id
    patientID
    patientName
    time
  }
}
```

Variables | Headers | JSON

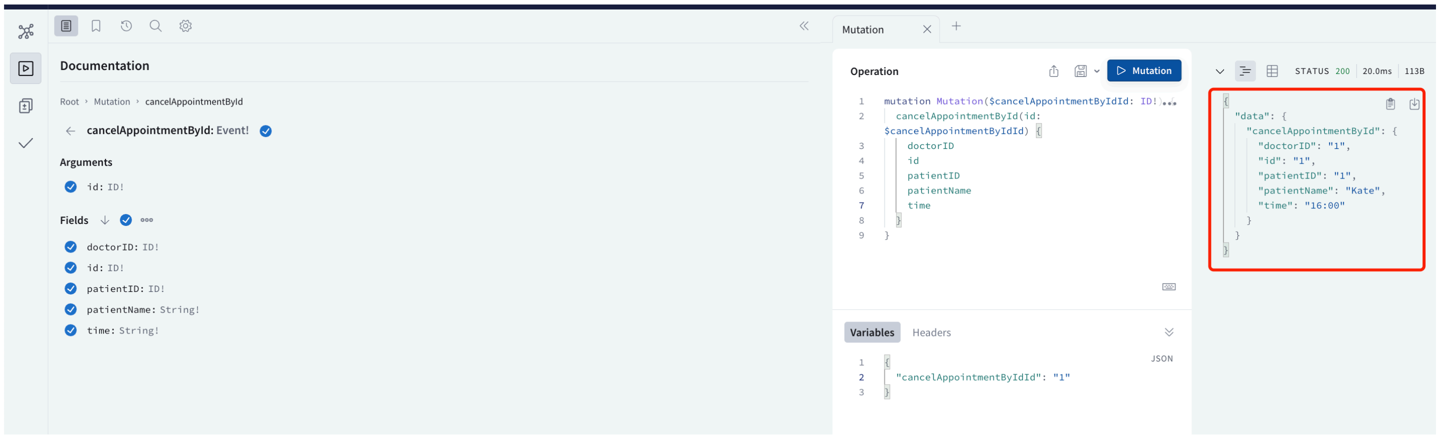
```
{
  "bookAppointmentInput": {
    "doctorID": "1",
    "patientID": "1",
    "time": "9:00"
  }
}
```

STATUS 200 | 14.0ms | 1.3KB

```
{
  "data": {},
  "errors": [
    {
      "message": "Time is not
valid.",
      "locations": [
        {
          "line": 2,
          "column": 3
        }
      ],
      "path": [
        "bookAppointment"
      ],
      "extensions": {
        "code": "BAD_REQUEST",
        "stacktrace": [
          "GraphQLError: Time is
not valid.",
          "    at Object.
bookAppointment (file:///Users/
yuyanghuang/Desktop/DAI_GraphQL/
index.js:233:23)",
          "    at field.resolve
```

***Name: cancelAppointmentById():*** Cancel an appointment(event) by id

Happy path:



Documentation

Root › Mutation › cancelAppointmentById

cancelAppointmentById: Event!

Arguments

id: ID!

Fields

doctorID: ID!
id: ID!
patientID: ID!
patientName: String!
time: String!

Mutation

Operation

```
mutation Mutation($cancelAppointmentByIdId: ID!) {
  cancelAppointmentById(id:
$cancelAppointmentByIdId) {
    doctorID
    id
    patientID
    patientName
    time
  }
}
```

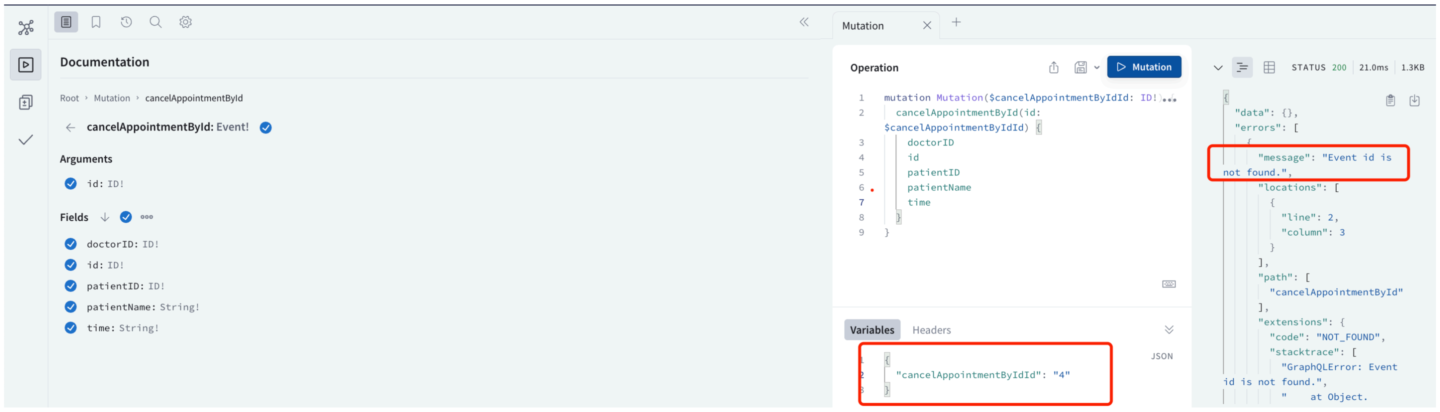
Variables | Headers | JSON

```
{
  "cancelAppointmentByIdId": "1"
}
```

STATUS 200 | 20.0ms | 113B

```
{
  "data": {
    "cancelAppointmentById": {
      "doctorID": "1",
      "id": "1",
      "patientID": "1",
      "patientName": "Kate",
      "time": "16:00"
    }
  }
}
```

Cancel an appointment with an invalid Id, the error handling is customized:



Documentation

Root › Mutation › cancelAppointmentById

cancelAppointmentById: Event!

Arguments

id: ID!

Fields

doctorID: ID!
id: ID!
patientID: ID!
patientName: String!
time: String!

Mutation

Operation

```
mutation Mutation($cancelAppointmentByIdId: ID!) {
  cancelAppointmentById(id:
$cancelAppointmentByIdId) {
    doctorID
    id
    patientID
    patientName
    time
  }
}
```

Variables | Headers | JSON

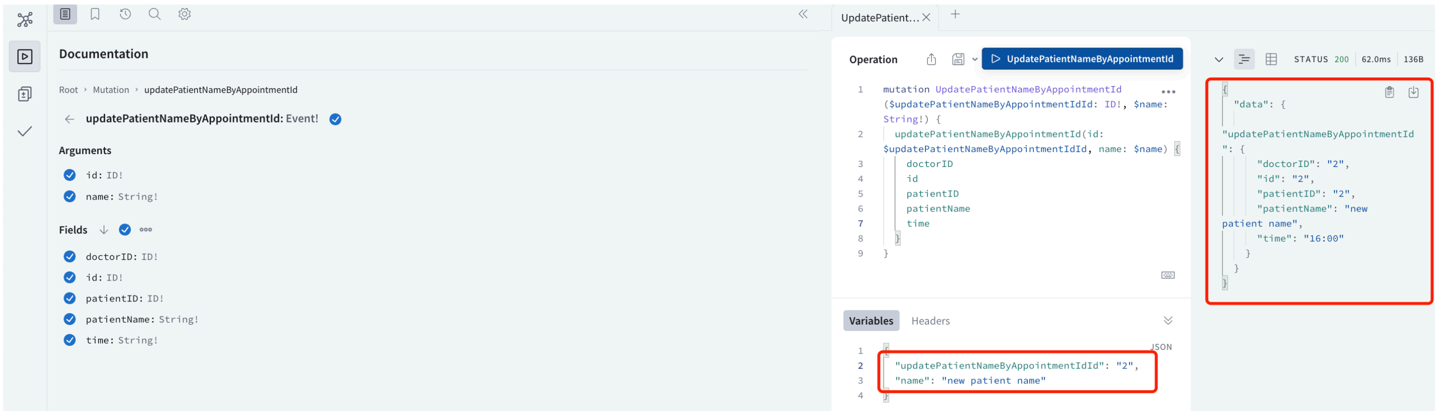
```
{
  "cancelAppointmentByIdId": "4"
}
```

STATUS 200 | 21.0ms | 1.3KB

```
{
  "data": {},
  "errors": [
    {
      "message": "Event id is
not found.",
      "locations": [
        {
          "line": 2,
          "column": 3
        }
      ],
      "path": [
        "cancelAppointmentById"
      ],
      "extensions": {
        "code": "NOT_FOUND",
        "stacktrace": [
          "GraphQLError: Event
id is not found.",
          "    at Object.
```

***Name: updatePatientNameByAppointmentId():*** Update name of the patient for an appointment

Happy path:


Documentation

Root › Mutation › updatePatientNameByAppointmentId

updatePatientNameByAppointmentId: Event!

Arguments

id: ID!
name: String!

Fields

doctorID: ID!
id: ID!
patientID: ID!
patientName: String!
time: String!

UpdatePatient...

Operation — UpdatePatientNameByAppointmentId

```
mutation UpdatePatientNameByAppointmentId
($updatePatientNameByAppointmentIdId: ID!, $name:
String!) {
  updatePatientNameByAppointmentId(id:
$updatePatientNameByAppointmentIdId, name: $name) {
    doctorID
    id
    patientID
    patientName
    time
  }
}
```

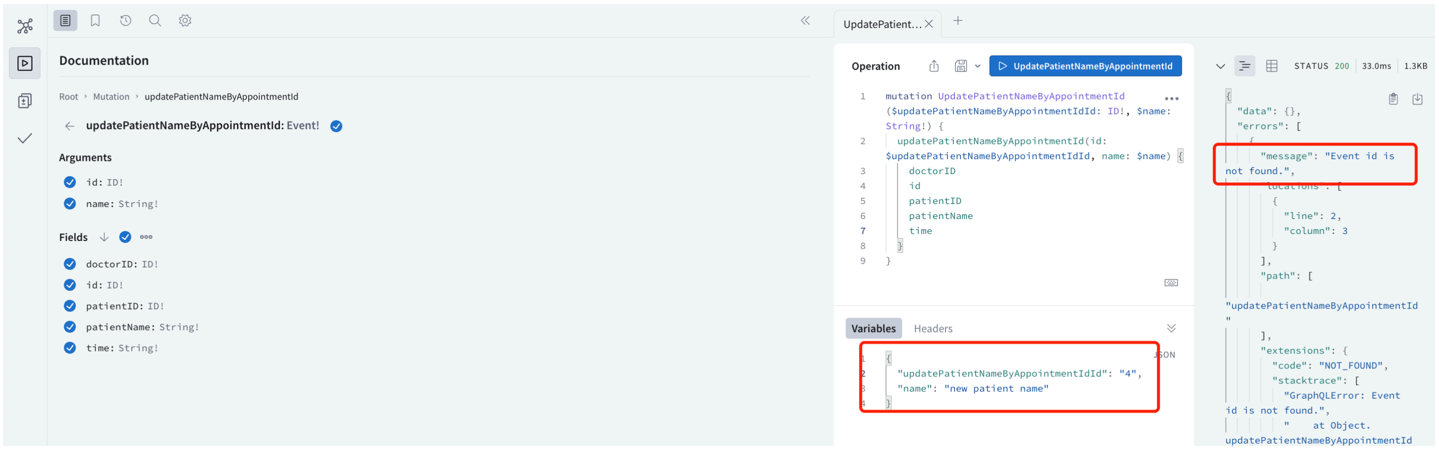
Variables  Headers

```
{
  "updatePatientNameByAppointmentIdId": "2",
  "name": "new patient name"
}
```

STATUS 200 | 62.0ms | 136B

```
{
  "data": {
    "updatePatientNameByAppointmentId": {
      "doctorID": "2",
      "id": "2",
      "patientID": "2",
      "patientName": "new patient name",
      "time": "16:00"
    }
  }
}
```

Input Invalid appointment id and the error message is customized:


Documentation

Root › Mutation › updatePatientNameByAppointmentId

updatePatientNameByAppointmentId: Event!

Arguments

id: ID!
name: String!

Fields

doctorID: ID!
id: ID!
patientID: ID!
patientName: String!
time: String!

UpdatePatient...

Operation — UpdatePatientNameByAppointmentId

```
mutation UpdatePatientNameByAppointmentId
($updatePatientNameByAppointmentIdId: ID!, $name:
String!) {
  updatePatientNameByAppointmentId(id:
$updatePatientNameByAppointmentIdId, name: $name) {
    doctorID
    id
    patientID
    patientName
    time
  }
}
```

Variables  Headers

```
{
  "updatePatientNameByAppointmentIdId": "4",
  "name": "new patient name"
}
```

STATUS 200 | 33.0ms | 1.3KB

```
{
  "data": {},
  "errors": [
    {
      "message": "Event id is not found.",
      "locations": [
        {
          "line": 2,
          "column": 3
        }
      ],
      "path": [
        "updatePatientNameByAppointmentId"
      ],
      "extensions": {
        "code": "NOT_FOUND",
        "stacktrace": [
          "GraphQLError: Event id is not found.",
          "    at Object.
updatePatientNameByAppointmentId
```

## *Reflection*

- What were some of the alternative schema and query design options you considered? Why did you choose the selected options?

  Currently, I am seperating patient, event, and doctor schema which is declared as below:

```
type Doctor {
    id: ID!
    doctorName: String!
    clinic: String!
    specialty: SpecialtyType!
    avaliableTime: [String]
    calendar: [Event]
}

type Event {
    id: ID!
    doctorID: ID!
    patientID: ID!
    patientName: String!
    time: String!
}

type Patient{
    id: ID!
    patientName: String!
}
```

The reason I am sperating these three type is that it can be easily stored in the database with three tables. one table called doctor, the one called patient and a joined table named event with the patient id and event id as the foreign keys. In contrast to those which have too many fileds in one type, this kind of schema can be easy to change and modify.

- Consider the case where, in future, the 'Event' structure is changed to have more fields e.g reference to patient details, consultation type (first time/follow-up etc.) and others.
  ○ What changes will the clients (API consumer) need to make to their existing queries (if any).

  Answer: If the client wants to include more fields to the patients, I think the first thing we need to do is to modify the patient schema. Also I need to add an new enum type named **consultationType**. As for the query API I think there is no other changes need to be made, since I already seperate the patient schema from the other schemas. They are relatively low-coupled and will not cause too many changes.

  ○ How will you accommodate the changes in your existing Schema and Query types?

  Answer: updated schema is shown as below:

  ```
  type Patient{
      id: ID!
      patientName: String!
      consultation: consultationType!
  }
  enum consultationType {
      first time
      follow-up
  }
  ```

  For the query, there is no changes with my design.

- Describe two GraphQL best practices that you have incorporated in your API design.
  1. I design and express the schema using diagram which lays a good foundation for future developemt.
  2. I design query and mutation in the way that the each query do just one thing and those queries that

are self-explanatory.  For the mutations, mutations are named as verbs which is shown in the code.