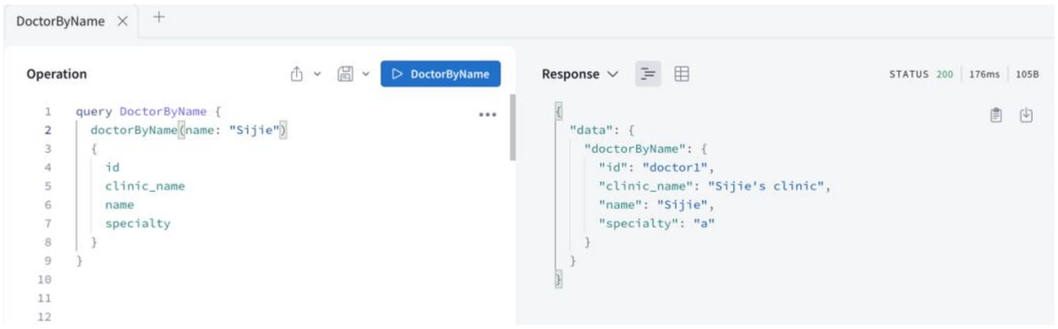
API Design: GraphQL API Testing

Testing

1. doctorByName

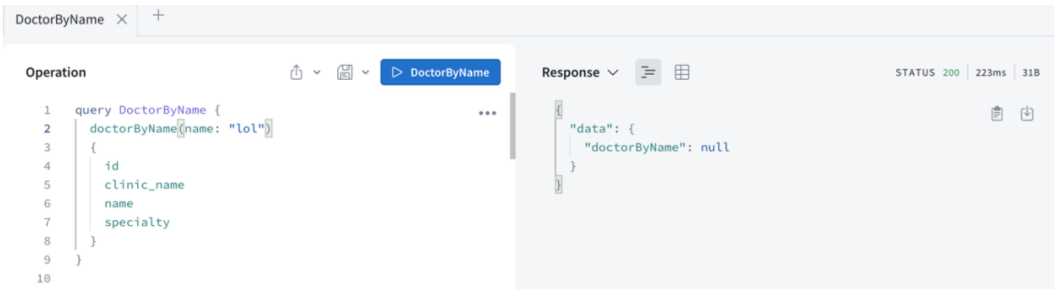
Happy Test



```
query DoctorByName {
  doctorByName(name: "Sijie")
  {
    id
    clinic_name
    name
    specialty
  }
}
```

```
{
  "data": {
    "doctorByName": {
      "id": "doctor1",
      "clinic_name": "Sijie's clinic",
      "name": "Sijie",
      "specialty": "a"
    }
  }
}
```

Error Test

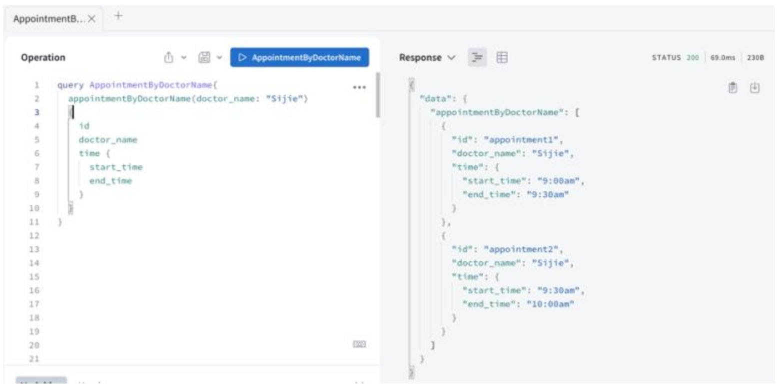


```
query DoctorByName {
  doctorByName(name: "lol")
  {
    id
    clinic_name
    name
    specialty
  }
}
```

```
{
  "data": {
    "doctorByName": null
  }
}
```

2. appointmentByDoctorName

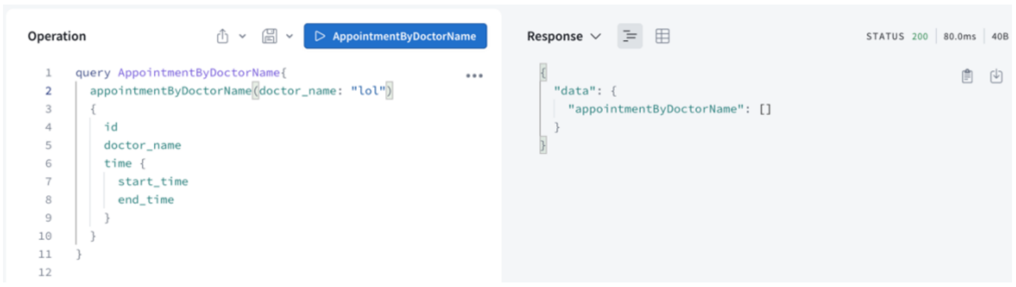
Happy Test



```
query AppointmentByDoctorName{
  appointmentByDoctorName(doctor_name: "Sijie")
  {
    id
    doctor_name
    time {
      start_time
      end_time
    }
  }
}
```

```
{
  "data": {
    "appointmentByDoctorName": [
      {
        "id": "appointment1",
        "doctor_name": "Sijie",
        "time": {
          "start_time": "9:00am",
          "end_time": "9:30am"
        }
      },
      {
        "id": "appointment2",
        "doctor_name": "Sijie",
        "time": {
          "start_time": "9:30am",
          "end_time": "10:00am"
        }
      }
    ]
  }
}
```

Error Test


Operation | AppointmentByDoctorName

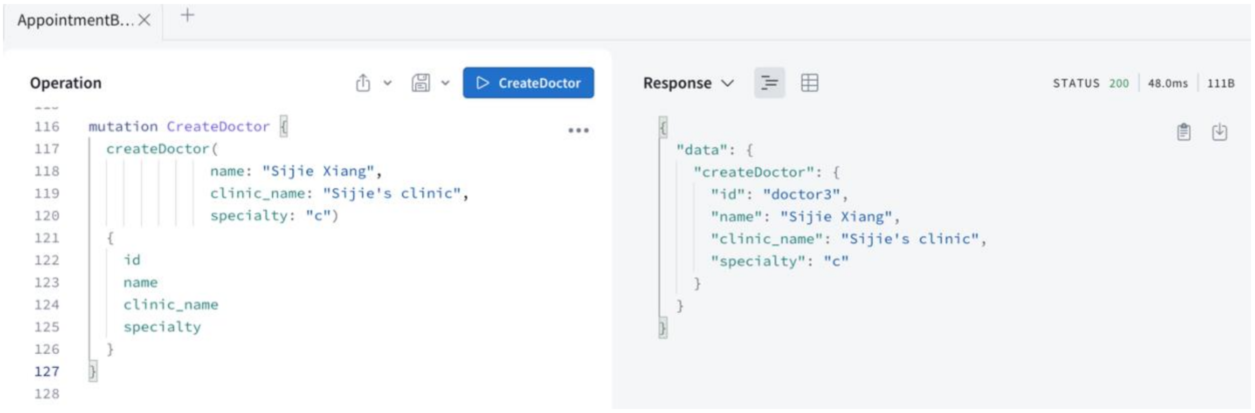
```
query AppointmentByDoctorName{
  appointmentByDoctorName(doctor_name: "lol")
  {
    id
    doctor_name
    time {
      start_time
      end_time
    }
  }
}
```

Response | STATUS 200 | 80.0ms | 40B

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

3. createDoctor

Happy Test

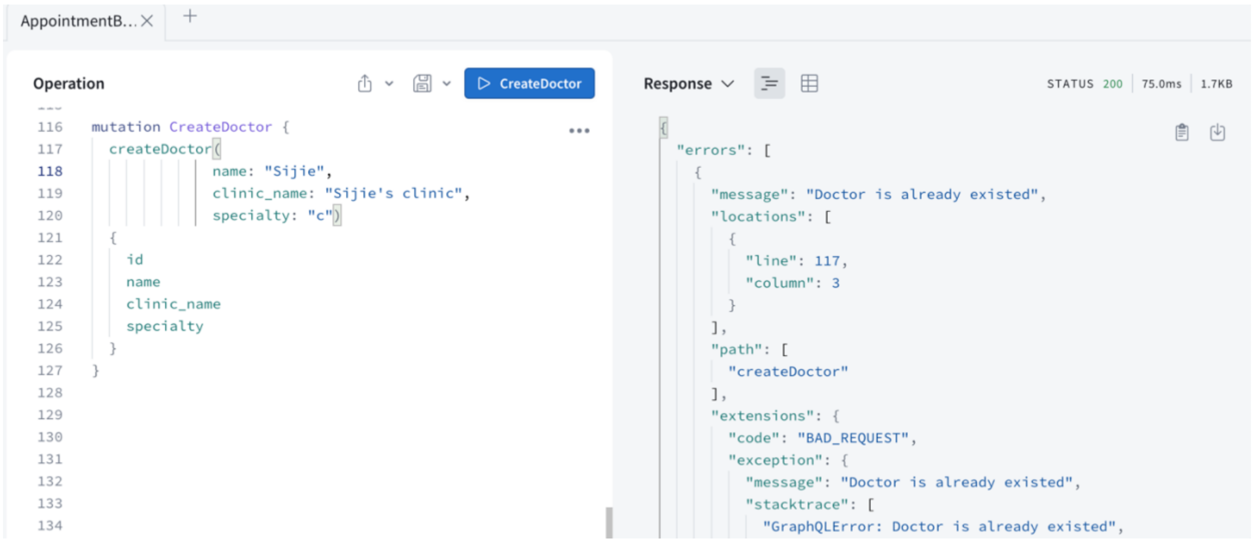

AppointmentB... | Operation | CreateDoctor

```
mutation CreateDoctor {
  createDoctor(
              name: "Sijie Xiang",
              clinic_name: "Sijie's clinic",
              specialty: "c")
  {
    id
    name
    clinic_name
    specialty
  }
}
```

Response | STATUS 200 | 48.0ms | 111B

```
{
  "data": {
    "createDoctor": {
      "id": "doctor3",
      "name": "Sijie Xiang",
      "clinic_name": "Sijie's clinic",
      "specialty": "c"
    }
  }
}
```

Error Test


AppointmentB... | Operation | CreateDoctor

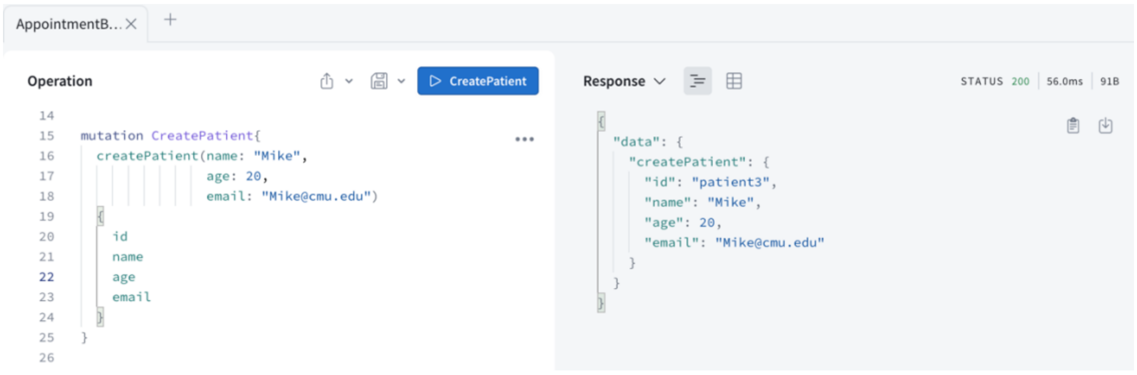
```
mutation CreateDoctor {
  createDoctor(
              name: "Sijie",
              clinic_name: "Sijie's clinic",
              specialty: "c")
  {
    id
    name
    clinic_name
    specialty
  }
}
```

Response | STATUS 200 | 75.0ms | 1.7KB

```
{
  "errors": [
    {
      "message": "Doctor is already existed",
      "locations": [
        {
          "line": 117,
          "column": 3
        }
      ],
      "path": [
        "createDoctor"
      ],
      "extensions": {
        "code": "BAD_REQUEST",
        "exception": {
          "message": "Doctor is already existed",
          "stacktrace": [
            "GraphQLError: Doctor is already existed",
```

4. createPatient

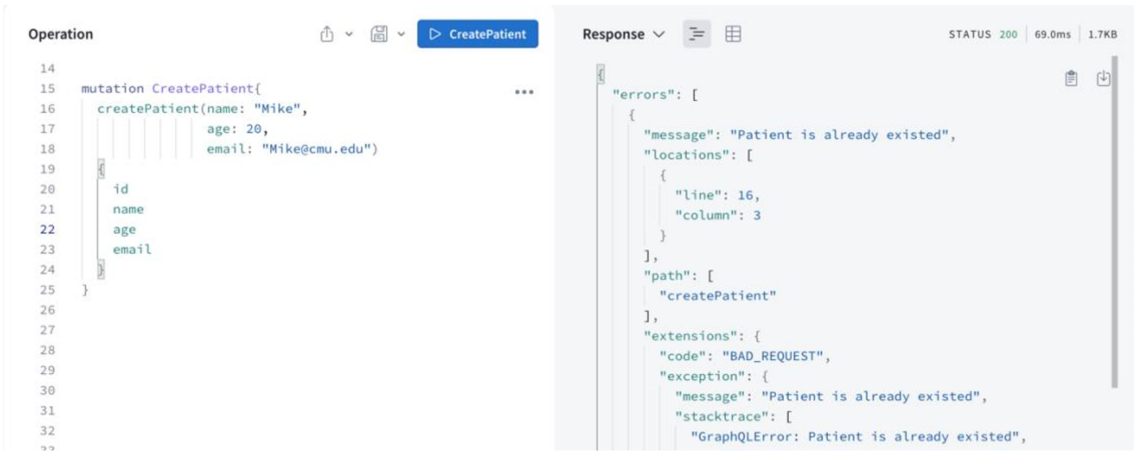
Happy Test



```
mutation CreatePatient{
  createPatient(name: "Mike",
                age: 20,
                email: "Mike@cmu.edu")
  {
    id
    name
    age
    email
  }
}
```

```
{
  "data": {
    "createPatient": {
      "id": "patient3",
      "name": "Mike",
      "age": 20,
      "email": "Mike@cmu.edu"
    }
  }
}
```

Error Test

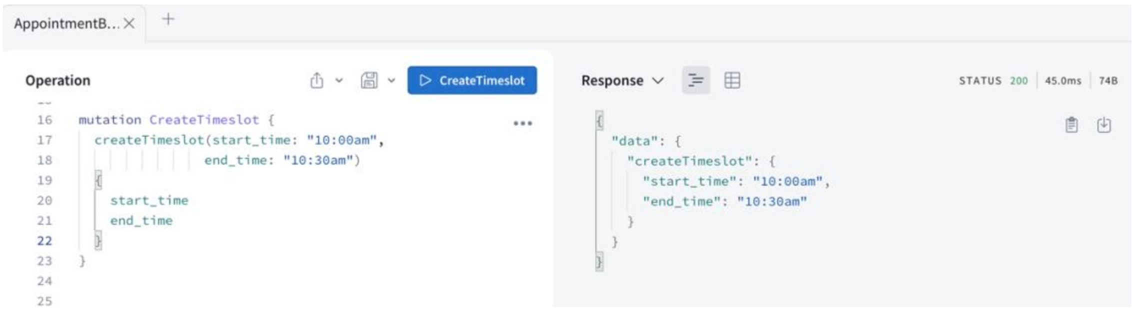


```
mutation CreatePatient{
  createPatient(name: "Mike",
                age: 20,
                email: "Mike@cmu.edu")
  {
    id
    name
    age
    email
  }
}
```

```
{
  "errors": [
    {
      "message": "Patient is already existed",
      "locations": [
        {
          "line": 16,
          "column": 3
        }
      ],
      "path": [
        "createPatient"
      ],
      "extensions": {
        "code": "BAD_REQUEST",
        "exception": {
          "message": "Patient is already existed",
          "stacktrace": [
            "GraphQLError: Patient is already existed",
```

5. createTimeslot

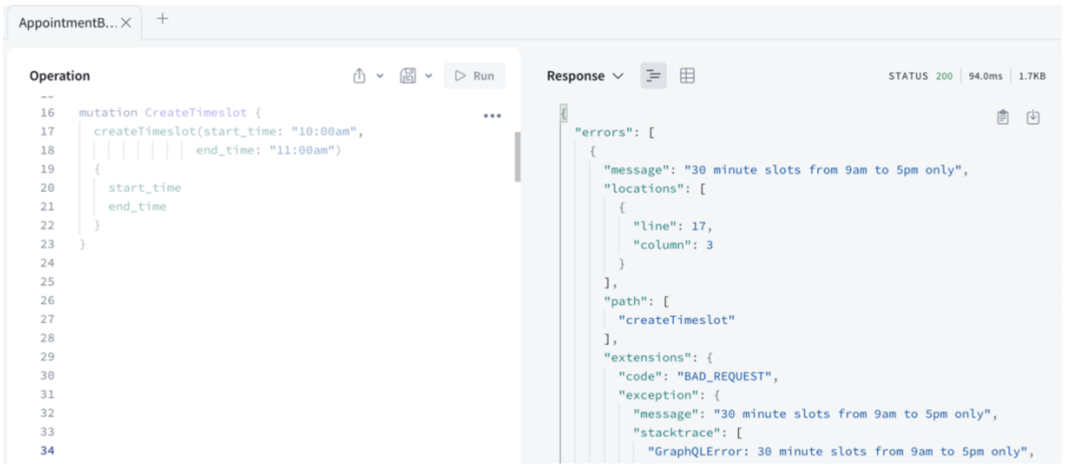
Happy Test



```
mutation CreateTimeslot {
  createTimeslot(start_time: "10:00am",
                 end_time: "10:30am")
  {
    start_time
    end_time
  }
}
```

```
{
  "data": {
    "createTimeslot": {
      "start_time": "10:00am",
      "end_time": "10:30am"
    }
  }
}
```

Error Test

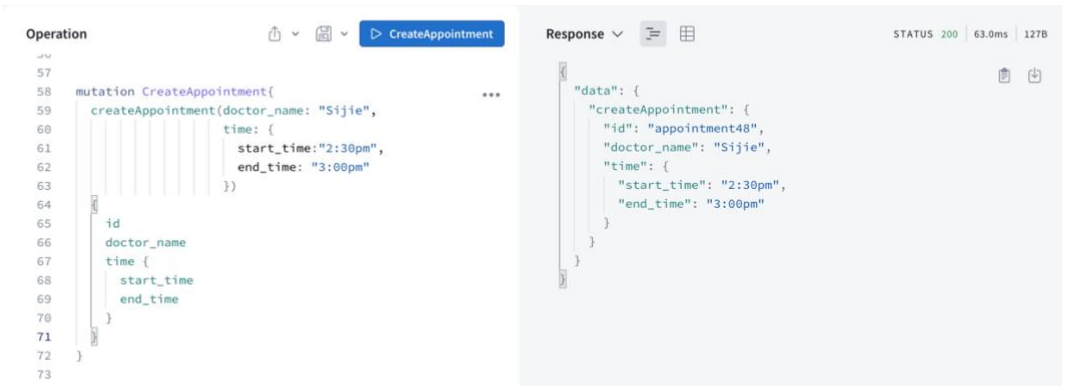


```
mutation CreateTimeslot {
  createTimeslot(start_time: "10:00am",
                 end_time: "11:00am")
  {
    start_time
    end_time
  }
}
```

```
{
  "errors": [
    {
      "message": "30 minute slots from 9am to 5pm only",
      "locations": [
        {
          "line": 17,
          "column": 3
        }
      ],
      "path": [
        "createTimeslot"
      ],
      "extensions": {
        "code": "BAD_REQUEST",
        "exception": {
          "message": "30 minute slots from 9am to 5pm only",
          "stacktrace": [
            "GraphQLError: 30 minute slots from 9am to 5pm only",
```

6. createAppointment

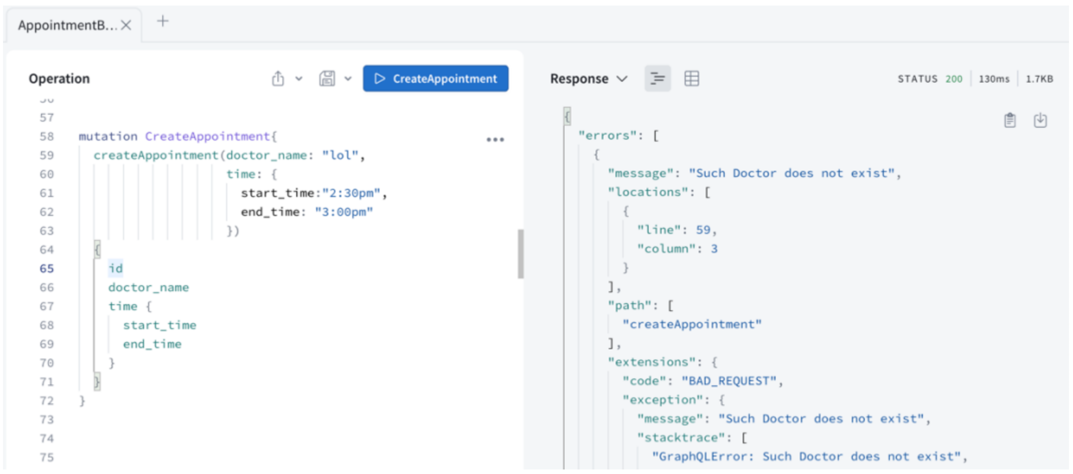
Happy Test



```
mutation CreateAppointment{
  createAppointment(doctor_name: "Sijie",
                    time: {
                      start_time:"2:30pm",
                      end_time: "3:00pm"
                    })
  {
    id
    doctor_name
    time {
      start_time
      end_time
    }
  }
}
```

```
{
  "data": {
    "createAppointment": {
      "id": "appointment48",
      "doctor_name": "Sijie",
      "time": {
        "start_time": "2:30pm",
        "end_time": "3:00pm"
      }
    }
  }
}
```

Error Test

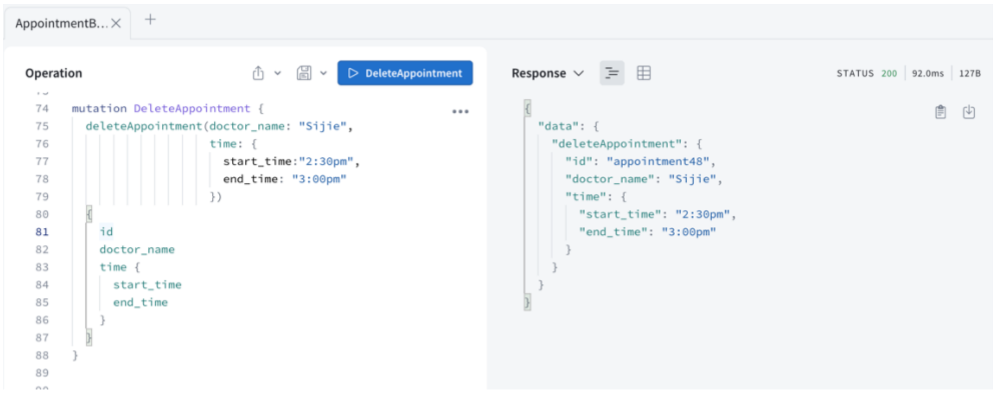


```
mutation CreateAppointment{
  createAppointment(doctor_name: "lol",
                    time: {
                      start_time:"2:30pm",
                      end_time: "3:00pm"
                    })
  {
    id
    doctor_name
    time {
      start_time
      end_time
    }
  }
}
```

```
{
  "errors": [
    {
      "message": "Such Doctor does not exist",
      "locations": [
        {
          "line": 59,
          "column": 3
        }
      ],
      "path": [
        "createAppointment"
      ],
      "extensions": {
        "code": "BAD_REQUEST",
        "exception": {
          "message": "Such Doctor does not exist",
          "stacktrace": [
            "GraphQLError: Such Doctor does not exist",
```

7. deleteAppointment

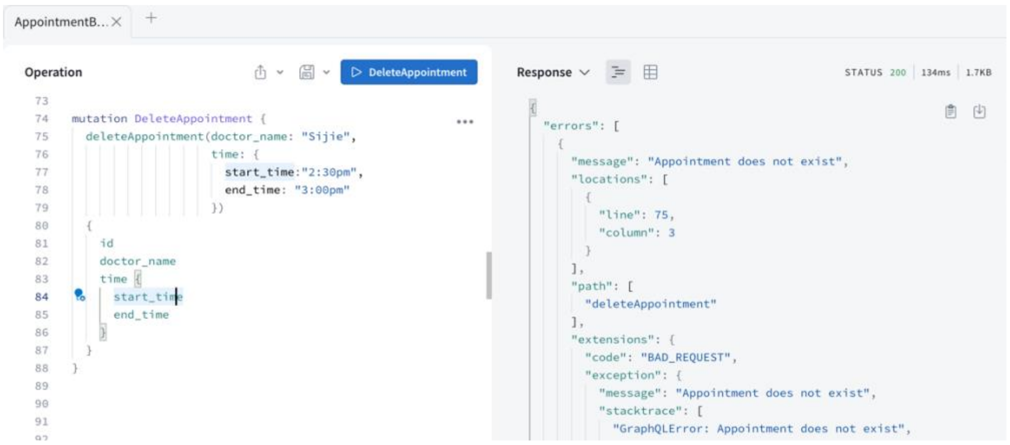
Happy Test



```
mutation DeleteAppointment {
  deleteAppointment(doctor_name: "Sijie",
                    time: {
                      start_time:"2:30pm",
                      end_time: "3:00pm"
                    })
  {
    id
    doctor_name
    time {
      start_time
      end_time
    }
  }
}
```

```
{
  "data": {
    "deleteAppointment": {
      "id": "appointment48",
      "doctor_name": "Sijie",
      "time": {
        "start_time": "2:30pm",
        "end_time": "3:00pm"
      }
    }
  }
}
```

Error Test

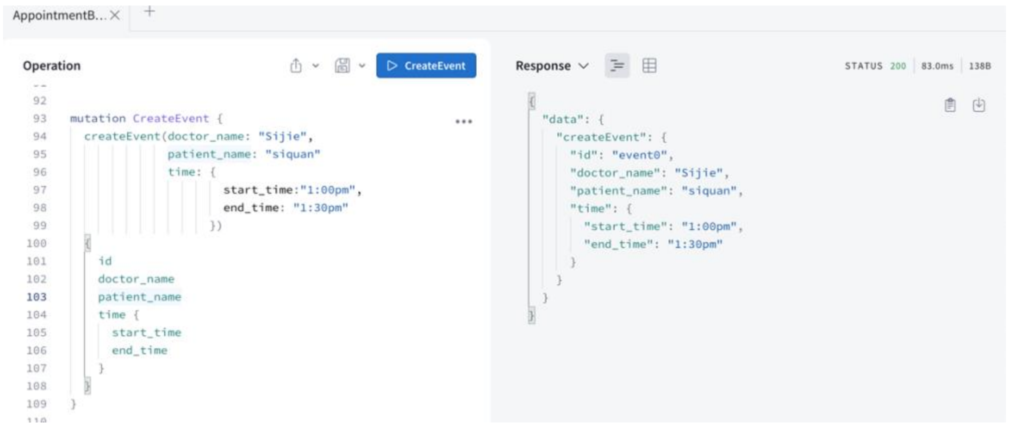


```
mutation DeleteAppointment {
  deleteAppointment(doctor_name: "Sijie",
                    time: {
                      start_time:"2:30pm",
                      end_time: "3:00pm"
                    })
  {
    id
    doctor_name
    time {
      start_time
      end_time
    }
  }
}
```

```
{
  "errors": [
    {
      "message": "Appointment does not exist",
      "locations": [
        {
          "line": 75,
          "column": 3
        }
      ],
      "path": [
        "deleteAppointment"
      ],
      "extensions": {
        "code": "BAD_REQUEST",
        "exception": {
          "message": "Appointment does not exist",
          "stacktrace": [
            "GraphQLError: Appointment does not exist",
```

8. createEvent

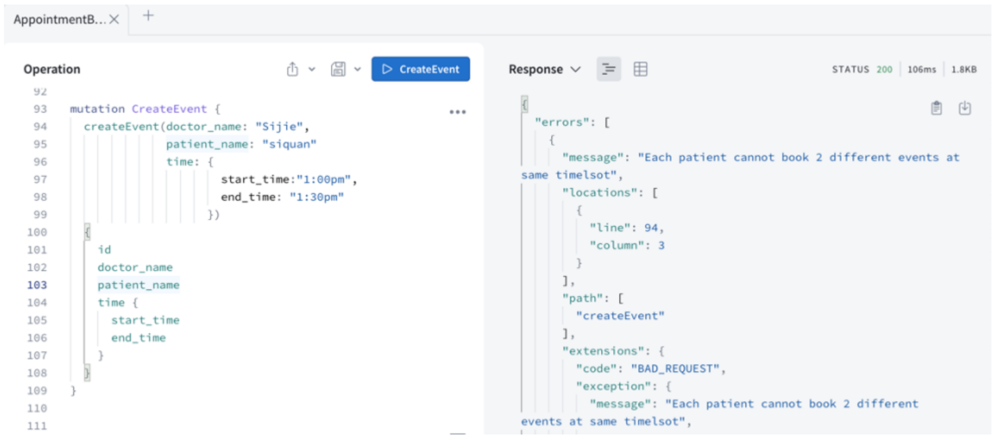
Happy Test



```
mutation CreateEvent {
  createEvent(doctor_name: "Sijie",
              patient_name: "siquan",
              time: {
                start_time:"1:00pm",
                end_time: "1:30pm"
              })
  {
    id
    doctor_name
    patient_name
    time {
      start_time
      end_time
    }
  }
}
```

```
{
  "data": {
    "createEvent": {
      "id": "event0",
      "doctor_name": "Sijie",
      "patient_name": "siquan",
      "time": {
        "start_time": "1:00pm",
        "end_time": "1:30pm"
      }
    }
  }
}
```

Error Test

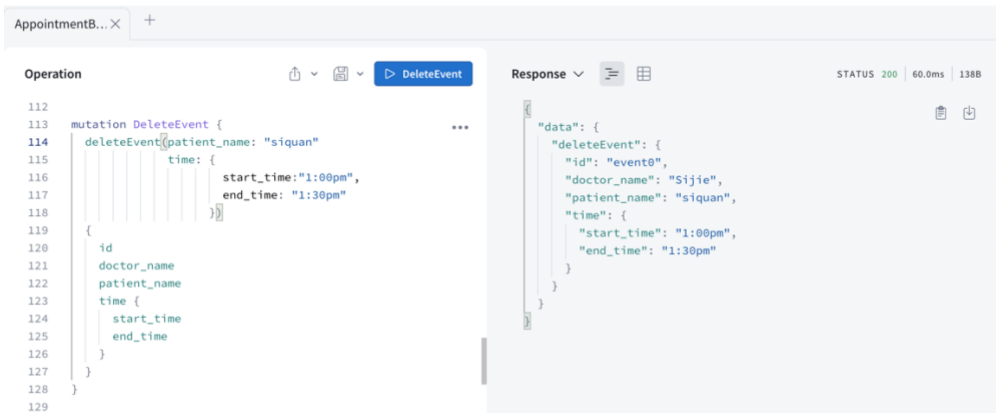


```
mutation CreateEvent {
  createEvent(doctor_name: "Sijie",
              patient_name: "siquan"
              time: {
                    start_time:"1:00pm",
                    end_time: "1:30pm"
                    })
  {
    id
    doctor_name
    patient_name
    time {
      start_time
      end_time
    }
  }
}
```

```
{
  "errors": [
    {
      "message": "Each patient cannot book 2 different events at
same timelsot",
      "locations": [
        {
          "line": 94,
          "column": 3
        }
      ],
      "path": [
        "createEvent"
      ],
      "extensions": {
        "code": "BAD_REQUEST",
        "exception": {
          "message": "Each patient cannot book 2 different
events at same timelsot",
```

9. deleteEvent

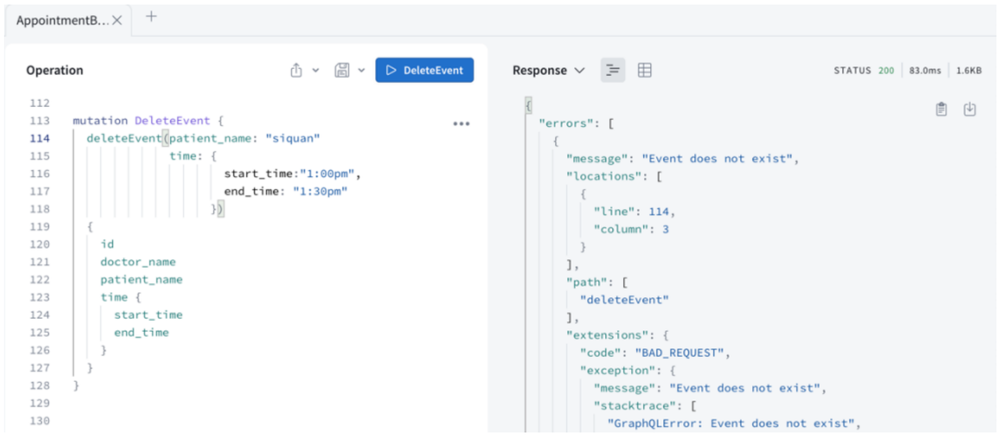
Happy Test



```
mutation DeleteEvent {
  deleteEvent(patient_name: "siquan"
              time: {
                    start_time:"1:00pm",
                    end_time: "1:30pm"
                    })
  {
    id
    doctor_name
    patient_name
    time {
      start_time
      end_time
    }
  }
}
```

```
{
  "data": {
    "deleteEvent": {
      "id": "event0",
      "doctor_name": "Sijie",
      "patient_name": "siquan",
      "time": {
        "start_time": "1:00pm",
        "end_time": "1:30pm"
      }
    }
  }
}
```

Error Test

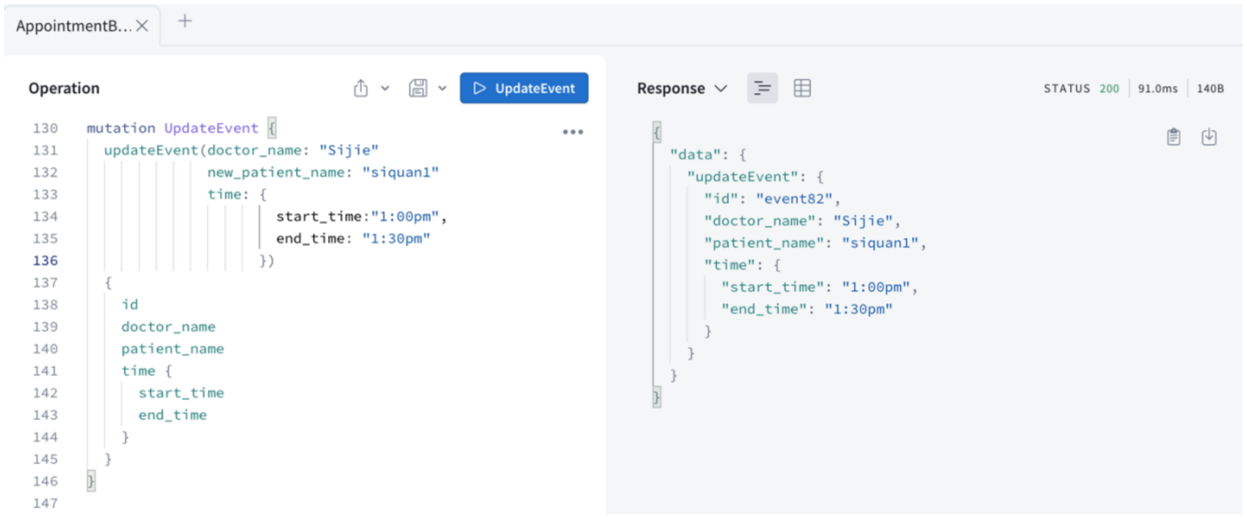


```
mutation DeleteEvent {
  deleteEvent(patient_name: "siquan"
              time: {
                    start_time:"1:00pm",
                    end_time: "1:30pm"
                    })
  {
    id
    doctor_name
    patient_name
    time {
      start_time
      end_time
    }
  }
}
```

```
{
  "errors": [
    {
      "message": "Event does not exist",
      "locations": [
        {
          "line": 114,
          "column": 3
        }
      ],
      "path": [
        "deleteEvent"
      ],
      "extensions": {
        "code": "BAD_REQUEST",
        "exception": {
          "message": "Event does not exist",
          "stacktrace": [
            "GraphQLError: Event does not exist",
```

10. updateEvent

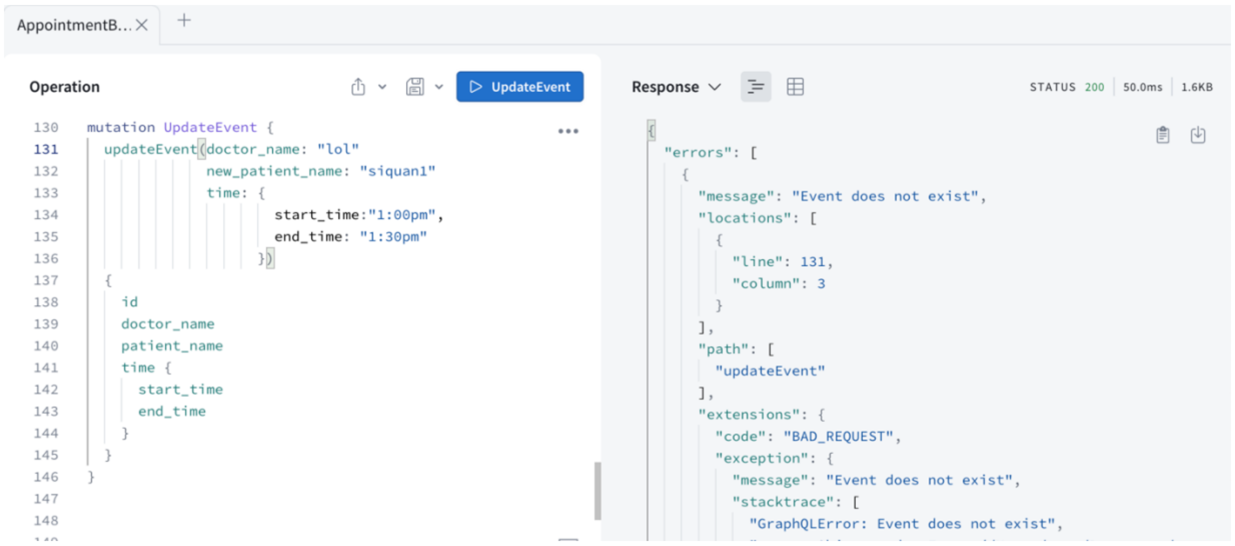
Happy Test



```
mutation UpdateEvent {
  updateEvent(doctor_name: "Sijie"
              new_patient_name: "siquan1"
              time: {
                  start_time:"1:00pm",
                  end_time: "1:30pm"
                })
  {
    id
    doctor_name
    patient_name
    time {
      start_time
      end_time
    }
  }
}
```

```
{
  "data": {
    "updateEvent": {
      "id": "event82",
      "doctor_name": "Sijie",
      "patient_name": "siquan1",
      "time": {
        "start_time": "1:00pm",
        "end_time": "1:30pm"
      }
    }
  }
}
```

Error Test



```
130  mutation UpdateEvent {
131    updateEvent(doctor_name: "lol"
132                new_patient_name: "siquan1"
133                time: {
134                    start_time:"1:00pm",
135                    end_time: "1:30pm"
136                  })
137    {
138      id
139      doctor_name
140      patient_name
141      time {
142        start_time
143        end_time
144      }
145    }
146  }
147
148
```

```
{
  "errors": [
    {
      "message": "Event does not exist",
      "locations": [
        {
          "line": 131,
          "column": 3
        }
      ],
      "path": [
        "updateEvent"
      ],
      "extensions": {
        "code": "BAD_REQUEST",
        "exception": {
          "message": "Event does not exist",
          "stacktrace": [
            "GraphQLError: Event does not exist",
```

Reflection

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

I have thought about just using 3 types such as Event, Patient, and Doctor, but such design barely reflects the real situation where in large applications type Timeslot and Appointment also are extremely important. Timeslot limits the duration of event or appointment whereas appointments allow patients to see availability of doctors. Therefore, with my current design, the entire application can be easily extended and truly reflects how the real-world functions. In addition, separation of concern is a huge factor when I design those schemas for types and resources because they are fundamental to future implementation and testing steps.

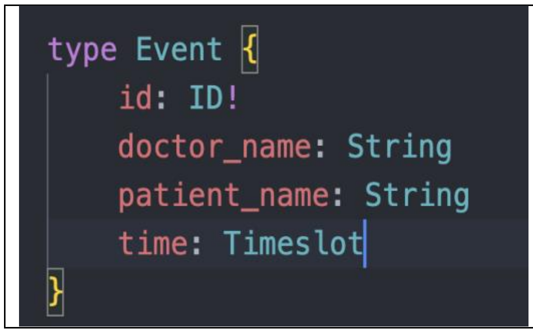
2. Consider the case where, in future, the 'Event' structure is changed to have more fields e.g reference to patient details, consultation type (first time/follow-up etc.) and others.

   - What changes will the clients (API consumer) need to make to their existing queries (if any).

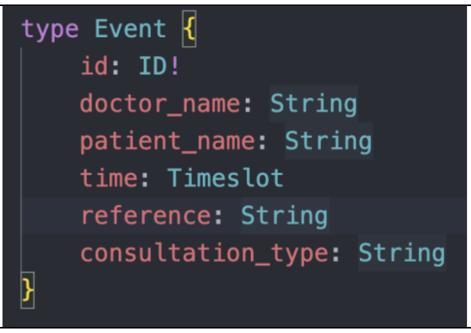
     Endpoint will be kept the same. In internal schema.graphql, reference field and consultation type field will be added under type Event and client will also need to include those fields in their existing queries if they want to see the change. Again, GraphQl here gives clients the power to ask for exactly what they need and nothing more.

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

     It depends how much information that client wants to add under type Event. If just to add 2 more fields such as little reference to patient details and consultation type, the following update can work

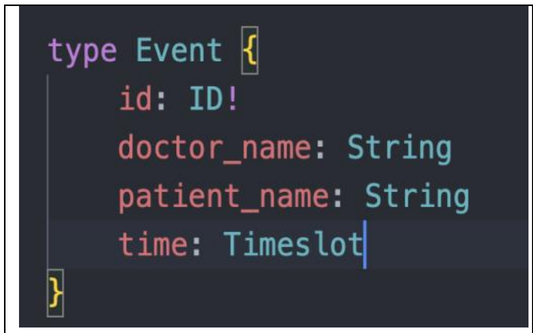


```
type Event {
    id: ID!
    doctor_name: String
    patient_name: String
    time: Timeslot
}
```

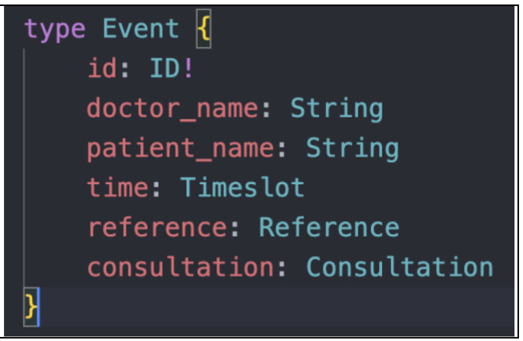


```
type Event {
    id: ID!
    doctor_name: String
    patient_name: String
    time: Timeslot
    reference: String
    consultation_type: String
}
```

However, the optimal way would be creating additional types such as Reference and Consultation in which more information can be included under those schema types. Still, they will be under type Event like the following. As for Query, we need to create new referenceInput and consultationInput to pass in as params.



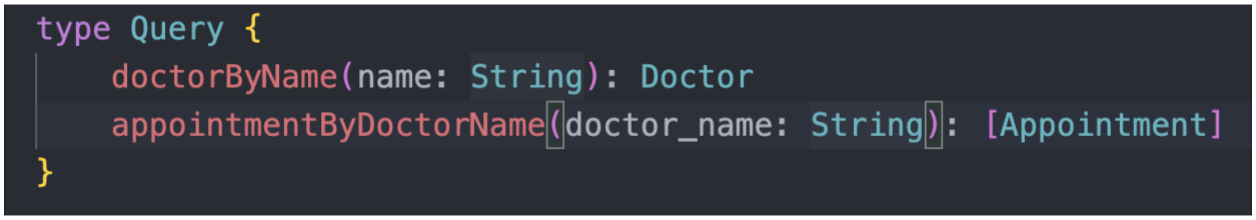
```
type Event {
    id: ID!
    doctor_name: String
    patient_name: String
    time: Timeslot
}
```



```
type Event {
    id: ID!
    doctor_name: String
    patient_name: String
    time: Timeslot
    reference: Reference
    consultation: Consultation
}
```

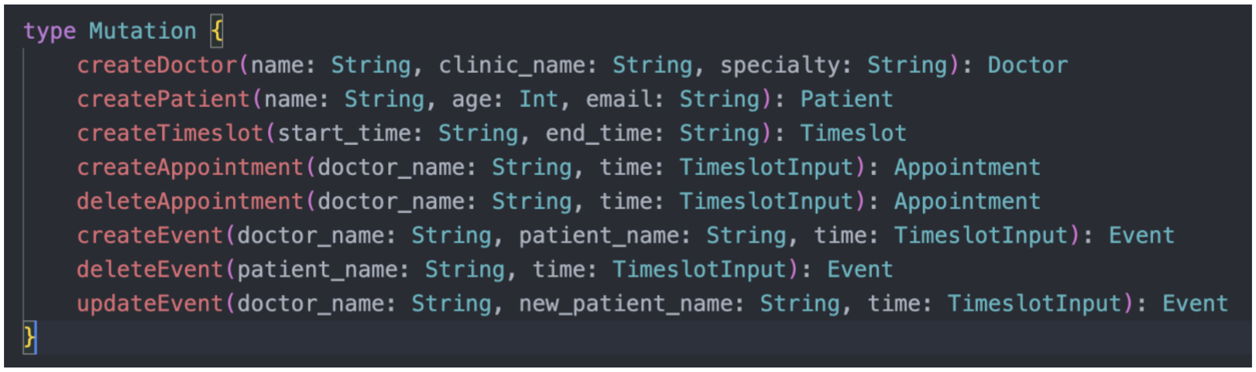
3. Describe **two** GraphQL best practices that you have incorporated in your API design.

- Avoid writing quires name like getDoctor or getAllAppointmentsByDoctorName because queries are always used to get something. Instead, use self-explanatory queries name such as doctorByName and appointmentByDoctorName



```
type Query {
    doctorByName(name: String): Doctor
    appointmentByDoctorName(doctor_name: String): [Appointment]
}
```

- Name mutations as verbs such as createDoctor, deleteAppointment, and updateEvent



```
type Mutation {
    createDoctor(name: String, clinic_name: String, specialty: String): Doctor
    createPatient(name: String, age: Int, email: String): Patient
    createTimeslot(start_time: String, end_time: String): Timeslot
    createAppointment(doctor_name: String, time: TimeslotInput): Appointment
    deleteAppointment(doctor_name: String, time: TimeslotInput): Appointment
    createEvent(doctor_name: String, patient_name: String, time: TimeslotInput): Event
    deleteEvent(patient_name: String, time: TimeslotInput): Event
    updateEvent(doctor_name: String, new_patient_name: String, time: TimeslotInput): Event
}
```