

## API Design: GraphQL API Testing

### Testing

#### 1. doctorByName

##### Happy Test

The screenshot shows a GraphQL IDE interface with a tab labeled "DoctorByName". The "Operation" pane on the left contains the following query:

```
1 query DoctorByName {  
2   doctorByName(name: "Sijie")  
3   {  
4     id  
5     clinic_name  
6     name  
7     specialty  
8   }  
9 }  
10  
11  
12
```

The "Response" pane on the right shows the JSON response:

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

At the top right, the status is "STATUS 200" with a response time of "176ms" and a size of "105B".

##### Error Test

The screenshot shows a GraphQL IDE interface with a tab labeled "DoctorByName". The "Operation" pane on the left contains the following query:

```
1 query DoctorByName {  
2   doctorByName(name: "lol")  
3   {  
4     id  
5     clinic_name  
6     name  
7     specialty  
8   }  
9 }  
10
```

The "Response" pane on the right shows the JSON response:

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

At the top right, the status is "STATUS 200" with a response time of "223ms" and a size of "31B".

#### 2. appointmentByDoctorName

##### Happy Test

The screenshot shows a GraphQL IDE interface with a tab labeled "AppointmentB...". The "Operation" pane on the left contains the following query:

```
1 query AppointmentByDoctorName {  
2   appointmentByDoctorName(doctor_name: "Sijie")  
3   {  
4     id  
5     doctor_name  
6     time {  
7       start_time  
8       end_time  
9     }  
10  }  
11 }  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21
```

The "Response" pane on the right shows the JSON response:

```
{  
  "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"  
        }  
      }  
    ]  
  }  
}
```

At the top right, the status is "STATUS 200" with a response time of "69.8ms" and a size of "230B".

##### Error Test

Operation AppointmentByDoctorName

```
1 query AppointmentByDoctorName{
2   appointmentByDoctorName(doctor_name: "lol")
3 }
4 {
5   id
6   doctor_name
7   time {
8     start_time
9     end_time
10  }
11 }
12 }
```

Response STATUS 200 | 80.0ms | 40B

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

### 3. createDoctor

## Happy Test

AppointmentB... +

Operation CreateDoctor

```
116 mutation CreateDoctor {
117   createDoctor(
118     name: "Sijie Xiang",
119     clinic_name: "Sijie's clinic",
120     specialty: "c")
121   {
122     id
123     name
124     clinic_name
125     specialty
126   }
127 }
128 }
```

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

```
116 mutation CreateDoctor {
117   createDoctor(
118     name: "Sijie",
119     clinic_name: "Sijie's clinic",
120     specialty: "c")
121   {
122     id
123     name
124     clinic_name
125     specialty
126   }
127 }
128 }
```

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

AppointmentB... X +




Operation    CreatePatient

```
14
15 mutation CreatePatient{
16   createPatient(name: "Mike",
17                age: 20,
18                email: "Mike@cmu.edu")
19 }
20
21 id
22 name
23 age
24 email
25
26
```



Response   STATUS 200 | 56.0ms | 91B

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

## Error Test

Operation    CreatePatient

```
14
15 mutation CreatePatient{
16   createPatient(name: "Mike",
17                age: 20,
18                email: "Mike@cmu.edu")
19 }
20
21 id
22 name
23 age
24 email
25
26
27
28
29
30
31
32
33
```




Response   STATUS 200 | 69.0ms | 1.7KB

```
{
  "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

AppointmentB... X +

Operation    CreateTimeslot

```
16 mutation CreateTimeslot {
17   createTimeslot(start_time: "10:00am",
18                 end_time: "10:30am")
19 }
20
21 start_time
22 end_time
23
24
25
```

Response   STATUS 200 | 45.0ms | 74B

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

## Error Test

AppointmentB... X +

Operation

```

16 mutation CreateTimeslot {
17   createTimeslot(start_time: "10:00am",
18                 end_time: "11:00am")
19   {
20     start_time
21     end_time
22   }
23 }
24
25
26
27
28
29
30
31
32
33
34

```

Run

Response

```

{
  "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",

```

STATUS 200 | 94.0ms | 1.7KB

## 6. createAppointment

### Happy Test

AppointmentB... X +

Operation

```

57
58 mutation CreateAppointment{
59   createAppointment(doctor_name: "Sijie",
60                   time: {
61                     start_time:"2:30pm",
62                     end_time: "3:00pm"
63                   })
64
65   id
66   doctor_name
67   time {
68     start_time
69     end_time
70   }
71 }
72
73

```

CreateAppointment

Response

```

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

```

STATUS 200 | 63.0ms | 127B

### Error Test

AppointmentB... X +

Operation

```

57
58 mutation CreateAppointment{
59   createAppointment(doctor_name: "lol",
60                   time: {
61                     start_time:"2:30pm",
62                     end_time: "3:00pm"
63                   })
64
65   id
66   doctor_name
67   time {
68     start_time
69     end_time
70   }
71 }
72
73
74
75

```

CreateAppointment

Response

```

{
  "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",

```

STATUS 200 | 130ms | 1.7KB

## 7. deleteAppointment

## Happy Test

The screenshot shows a GraphQL IDE interface with a tab labeled "AppointmentB...". The "Operation" pane on the left contains the following query:

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

The "Response" pane on the right shows the JSON response:

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

At the top right of the response pane, the status is "STATUS 200", the time is "92.0ms", and the size is "127B".

## Error Test

The screenshot shows the same GraphQL IDE interface. The "Operation" pane contains the same query as before. The "Response" pane shows an error response:

```
{
  "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",
          ]
        }
      }
    }
  ]
}
```

At the top right of the response pane, the status is "STATUS 200", the time is "134ms", and the size is "1.7KB".

## 8. createEvent

## Happy Test

The screenshot shows a GraphQL IDE interface with a tab labeled "AppointmentB...". The "Operation" pane on the left contains the following query:

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

The "Response" pane on the right shows the JSON response:

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

At the top right of the response pane, the status is "STATUS 200", the time is "83.0ms", and the size is "138B".

## Error Test

AppointmentB...X +

Operation

```

92
93 mutation CreateEvent {
94   createEvent(doctor_name: "Sijie",
95     patient_name: "Siquan"
96     time: {
97       start_time: "1:00pm",
98       end_time: "1:30pm"
99     })
100 }
101 {
102   id
103   doctor_name
104   patient_name
105   time {
106     start_time
107     end_time
108   }
109 }
110
111

```

Response

STATUS 200 106ms 1.8KB

```

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

```

## 9. deleteEvent

### Happy Test

AppointmentB...X +

Operation

```

112
113 mutation DeleteEvent {
114   deleteEvent(patient_name: "Siquan"
115     time: {
116       start_time: "1:00pm",
117       end_time: "1:30pm"
118     })
119 }
120 {
121   id
122   doctor_name
123   patient_name
124   time {
125     start_time
126     end_time
127   }
128 }
129

```

Response

STATUS 200 60.0ms 138B

```

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

```

### Error Test

AppointmentB...X +

Operation

```

112
113 mutation DeleteEvent {
114   deleteEvent(patient_name: "Siquan"
115     time: {
116       start_time: "1:00pm",
117       end_time: "1:30pm"
118     })
119 }
120 {
121   id
122   doctor_name
123   patient_name
124   time {
125     start_time
126     end_time
127   }
128 }
129

```

Response

STATUS 200 83.0ms 1.6KB

```





{
  "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

AppointmentB... × +

Operation     UpdateEvent





```
130 mutation UpdateEvent {  
131   updateEvent(doctor_name: "Sijie"  
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
```

Response    STATUS 200 91.0ms 140B




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

## Error Test

AppointmentB... × +

Operation     UpdateEvent

```
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  
149
```

Response    STATUS 200 50.0ms 1.6KB

```
{  
  "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 queries 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  
}
```