

Implementation Task:

Testing for each Testcase Identifier:

DoctorByIdHappyTest:

The screenshot displays a GraphQL IDE interface for a 'DoctorByIdHappyTest'. The 'Operation' tab on the left shows a query to fetch doctor information by ID. The 'Response' tab on the right shows the JSON output, indicating a successful status (200) and a null doctor value.

Operation

```
14  }
15  }
16
17  query GetDoctor($doctorId: ID!){
18    doctor(id: $doctorId){
19      id
20      name
21      clinic_name
22      speciality
23    }
24  }
25
26  query GetDoctorFreeSlots($doctorFreeslotsId: ID!){
27    doctor_freeslots(id: $doctorFreeslotsId) {
```

Variables

```
1  {
2    "doctorId": "6"
3  }
```

Response

```
{
  "data": {
    "doctor": null
  }
}
```

STATUS 200 | 11.0ms | 25B

DoctorByIdErrorTest:

The screenshot displays a GraphQL IDE interface for a 'DoctorByIdErrorTest'. The 'Operation' tab on the left shows a query to fetch doctor information by ID. The 'Response' tab on the right shows the JSON output, indicating a successful status (200) and a specific doctor object.

Operation

```
14  }
15  }
16
17  query GetDoctor($doctorId: ID!){
18    doctor(id: $doctorId){
19      id
20      name
21      clinic_name
22      speciality
23    }
24  }
25
26  query GetDoctorFreeSlots($doctorFreeslotsId: ID!){
27    doctor_freeslots(id: $doctorFreeslotsId) {
```

Variables

```
1  {
2    "doctorId": "1"
3  }
```

Response

```
{
  "data": {
    "doctor": {
      "id": "1",
      "name": "Mathila",
      "clinic_name": "apollo",
      "speciality": "general physician"
    }
  }
}
```

STATUS 200 | 2B

DoctorFreeSlotsHappyTest:

The screenshot displays a GraphQL client interface with a tab labeled "Doctors". The "Operation" panel on the left contains the following query:

```
23 | }  
24 | }  
25 |  
26 | query GetDoctorFreeSlots($doctorFreeslotsId: ID!){ ...  
27 |   doctor_freeslots(id: $doctorFreeslotsId) {  
28 |     timeslots_available {  
29 |       timeslot  
30 |     }  
31 |   }  
32 | }  
33 |  
34 |  
35 | mutation create_doctor($createinput: CreateDoctor){  
36 |   create_doctor(input: $createinput) {
```

Below the query, the "Variables" panel shows the input:

```
1 | {  
2 |   "doctorFreeslotsId": "1"  
3 | }  
4 |
```

The "Response" panel on the right shows the JSON output:

```
{  
  "data": {  
    "doctor_freeslots": {  
      "timeslots_available": [  
        {  
          "timeslot": "9"  
        },  
        {  
          "timeslot": "9:30"  
        },  
        {  
          "timeslot": "11"  
        },  
        {  
          "timeslot": "11:30"  
        },  
        {  
          "timeslot": "12"  
        },  
        {  
          "timeslot": "12:30"  
        },  
        {  
          "timeslot": "13"  
        },  
        {  
          "timeslot": "13:30"  
        }  
      ]  
    }  
  }  
}
```

At the top right of the response panel, the status is "STATUS 200", the time taken is "10.0ms", and the size is "327B".

DoctorFreeSlotsErrorTest:

The screenshot displays the same GraphQL client interface as above, but with a different query and response. The "Operation" panel shows the same query as in the first screenshot.

The "Variables" panel shows the input:

```
1 | {  
2 |   "doctorFreeslotsId": "8"  
3 | }  
4 |
```




The "Response" panel shows the JSON output:

```
{  
  "data": {  
    "doctor_freeslots": null  
  }  
}
```


At the top right of the response panel, the status is "STATUS 200", the time taken is "9.00ms", and the size is "35B".

ScheduleAppointmentHTest:

Doctors × +



Operation    schedule_appointment

```
41 }
42
43 mutation schedule_appointment($scheduleAppointmentId: ID!, $newapp: ScheduleAppointment){
44   schedule_appointment(id: $scheduleAppointmentId,
45   input: $newapp) {
46     id
47     name
48     timeslots_booked {
49       patient_name
50       timeslot
51     }
52     speciality
53 }
```

Variables Headers 

JSON


```
1 {
2   "scheduleAppointmentId": "1",
3   "newapp":{"patient_name":"newp","timeslot":"11"}
4 }
5
```




Response  

STATUS 200 | 41.0ms | 241B


```
{
  "data": {
    "schedule_appointment": {
      "id": "1",
      "name": "Mathila",
      "timeslots_booked": [
        {
          "patient_name": "Lalitha",
          "timeslot": "10"
        },
        {
          "patient_name": "sai",
          "timeslot": "10:30"
        },
        {
          "patient_name": "newp",
          "timeslot": "11"
        }
      ],
      "speciality": "general physician"
    }
  }
}
```

ScheduleAppointmentETest:

Doctors × +  Submit Operation (% + Enter)



Operation    schedule_appointment

```
41 }
42
43 mutation schedule_appointment($scheduleAppointmentId: ID!, $newapp: ScheduleAppointment){
44   schedule_appointment(id: $scheduleAppointmentId,
45   input: $newapp) {
46     id
47     name
48     timeslots_booked {
49       patient_name
50       timeslot
51     }
52     speciality
53 }
```

Variables Headers 

JSON

```
1 {
2   "scheduleAppointmentId": "10",
3   "newapp":{"patient_name":"newp","timeslot":"11"}
4 }
5
```

Response  

STATUS 200 | 12.0ms | 39B

```
{
  "data": {
    "schedule_appointment": null
  }
}
```

CancelAppointmentHTest:

Doctors

+

Operation

cancel_appointment

```
55 mutation cancel_appointment($cancelAppointmentId: ID!,  
$patientName: String!) {  
56   cancel_appointment(id: $cancelAppointmentId,  
    patient_name: $patientName) {  
57     id  
58     name  
59     clinic_name  
60     timeslots_booked {  
61       patient_name  
62       timeslot  
63     }  
64   }  
65 }  
66
```

Variables

Headers

JSON

```
1 {  
2   "cancelAppointmentId": "1",  
3   "patientName": "sai"  
4 }  
5
```

Response

STATUS 200 12.0ms 187B

```
{  
  "data": {  
    "cancel_appointment": {  
      "id": "1",  
      "name": "Mathila",  
      "clinic_name": "apollo",  
      "timeslots_booked": [  
        {  
          "patient_name": "Lalitha",  
          "timeslot": "10"  
        },  
        {  
          "patient_name": "newp",  
          "timeslot": "11"  
        }  
      ]  
    }  
  }  
}
```

CancelAppointmentETest:

Doctors

+

Operation

cancel_appointment

```
55 mutation cancel_appointment($cancelAppointmentId: ID!,  
$patientName: String!) {  
56   cancel_appointment(id: $cancelAppointmentId,  
    patient_name: $patientName) {  
57     id  
58     name  
59     clinic_name  
60     timeslots_booked {  
61       patient_name  
62       timeslot  
63     }  
64   }  
65 }  
66
```

Variables

Headers

JSON

```
1 {  
2   "cancelAppointmentId": "1",  
3   "patientName": "noName"  
4 }  
5
```

Response

STATUS 200 10.0ms 37B

```
{  
  "data": {  
    "cancel_appointment": null  
  }  
}
```

UpdateAppointmentHTest:

Doctors

×

+

Submit Operation (⌘ + Enter)

Operation

↑

↓

update

```
66
67 mutation update($updateAppointmentOldName2: String, ...
    $updateAppointmentNewName2: String){
68   update_appointment(oldName:
    $updateAppointmentOldName2,newName:
    $updateAppointmentNewName2) {
69     clinic_name
70     name
71     timeslots_booked {
72       patient_name
73       timeslot
74     }
75   }
76 }
```

Variables

Headers

⌵

```
1
2 "updateAppointmentOldName2":"Lalitha",
3 "updateAppointmentNewName2":"newName"
4
5
```

JSON

Response

⌵

⌵

⌵

```
{
  "data": {
    "update_appointment": {
      "clinic_name": "apollo",
      "name": "Mathila",
      "timeslots_booked": [
        {
          "patient_name": "newName",
          "timeslot": "10"
        },
        {
          "patient_name": "newp",
          "timeslot": "11"
        }
      ]
    }
  }
}
```

STATUS 200 | 9.00ms | 178B

UpdateAppointmentETest:

Doctors

×

+

Submit Operation (⌘ + Enter)

Operation

↑

↓

update

```
66
67 mutation update($updateAppointmentOldName2: String, ...
    $updateAppointmentNewName2: String){
68   update_appointment(oldName:
    $updateAppointmentOldName2,newName:
    $updateAppointmentNewName2) {
69     clinic_name
70     name
71     timeslots_booked {
72       patient_name
73       timeslot
74     }
75   }
76 }
```

Variables

Headers

⌵

```
1
2 "updateAppointmentOldName2":"hi",
3 "updateAppointmentNewName2":"newName"
4
5
```

JSON

Response

⌵

⌵

⌵

```
{
  "data": {
    "update_appointment": null
  }
}
```

STATUS 200 | 6.00ms | 37B

Extra Queries and Mutations Tests:

Operation

Doctors

1query Doctors {
2 doctors {
3 id
4 name
5 clinic_name
6 speciality
7 timeslots_available {
8 timeslot
9 }
10 timeslots_booked {
11 patient_name
12 timeslot
13 }
14 }
15}
16
17query GetDoctor(\$doctorId: ID!){
18 doctor(id: \$doctorId) {
19 name
20 clinic_name
21 speciality
22 }
23 }
24
25query GetDoctorFreeSlots
 (\$doctorFreeSlotId: ID!){
 ...
26 }

Variables Headers

Response

STATUS 200 | 36.0ms | 706B

1{
2 "data": {
3 "doctors": [
4 {
5 "id": "1",
6 "name": "Mathila",
7 "clinic_name": "apollo",
8 "speciality": "general physician",
9 "timeslots_available": [
10 {
11 "timeslot": "9"
12 },
13 {
14 "timeslot": "9:30"
15 },
16 {
17 "timeslot": "11"
18 },
19 {
20 "timeslot": "11:30"
21 },
22 {
23 "timeslot": "12"
24 },
25 {
26 "timeslot": "12:30"
27 },
28 {
29 "timeslot": "13"
30 }
31]
32 }
33 }
34 }
35 }

Operation

GetDoctor

16
17query GetDoctor(\$doctorId: ID!){
18 doctor(id: \$doctorId) {
19 name
20 clinic_name
21 speciality
22 }
23 }

Variables Headers

Response

STATUS 200 | 7.00ms | 95B

1{
2 "data": {
3 "doctor": {
4 "name": "Mathila",
5 "clinic_name": "apollo",
6 "speciality": "general physician"
7 }
8 }
9 }

Variables

1{
2 "doctorId": "1"
3 }
4 }

JSON

Reflection Task:

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

Answer: The alternative schema and query design that was considered is to have separate doctors and events details and map a variable in between them. But the current schema and queries are considered to make the API easy to understand and implement. By having all the details in a single object is going to make the traversal and understanding easy.

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.

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

Answer: Queries would remain the same but variables will be added in `timeslots_booked`. There would be changes in `timeslots_booked` object which in turn has `timeslot` and `patient_name` as a list of objects.

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

Answer: The changes are accommodated in the schema to add the additional fields. Queries would remain the same for most of them mentioned.

The existing schema would get changed to this:

```
type Details {
  patient_name: String
  patient_age: Int
  patient_contact: String
}
type Booked {
  timeslot: String
  consultation_type: String
  patient_details: [Details]
}
type Doctor {
  id: ID!
  name: String!
  clinic_name: String!
  speciality: String!
  timeslots_available: [Free]
  timeslots_booked: [Booked]
}
```

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

GraphQL variables were used to provide arguments, naming all queries and querying the data that is only needed are few of the GraphQL best practices that are incorporated in this API design.