

## Testing

### Get\_Doctor\_Success

The screenshot shows the Apollo Studio interface with a successful GraphQL query execution. The query is:

```
1 query DoctorByID($doctorByIdId: ID!) {  
2   doctorByID(id: $doctorByIdId) {  
3     appointments {  
4       id  
5       time  
6     }  
7     clinicName  
8     id  
9     name  
10  }  
11 }  
12
```

The variables are:

```
1 {  
2   "doctorByIdId": "D_1"  
3 }
```

The response is:

```
{  
  "data": {  
    "doctorByID": {  
      "appointments": [  
        {  
          "id": "A_1",  
          "time": 16  
        },  
        {  
          "id": "A_2",  
          "time": 18  
        }  
      ],  
      "clinicName": "Clinic1",  
      "id": "D_1",  
      "name": "Doctor1"  
    }  
  }  
}
```

The status is 200, with 53.0ms execution time and 140B response size.

### Get\_Doctor\_Fail

The screenshot shows the Apollo Studio interface with a failed GraphQL query execution. The query is:

```
1 query DoctorByID($doctorByIdId: ID!) {  
2   doctorByID(id: $doctorByIdId) {  
3     appointments {  
4       id  
5       time  
6     }  
7     clinicName  
8     id  
9     name  
10  }  
11 }  
12
```

The variables are:

```
1 {  
2   "doctorByIdId": "-1"  
3 }
```

The response is:

```
{  
  "data": {  
    "doctorByID": null  
  }  
}
```

The status is 200, with 67.0ms execution time and 29B response size.

## Get\_Time\_Success

The screenshot shows the Apollo Studio interface with a successful GraphQL query. The left sidebar displays the schema path: Root > Query > availableTimeslotByID. The main panel shows the query operation and its variables.

**Query Operation:**

```
1 query Query($availableTimeslotByIdId: ID!) {  
2   availableTimeslotByID(id: $availableTimeslotByIdId)  
3 }
```

**Variables:**

```
1 {  
2   "availableTimeslotByIdId": "D_1"  
3 }
```

**Response:**

```
{  
  "data": {  
    "availableTimeslotByID": [  
      1,  
      2,  
      3,  
      4,  
      5,  
      6,  
      7,  
      8,  
      9,  
      10,  
      11,  
      12,  
      13,  
      14,  
      15,  
      17,  
      19,  
      20,  
      21,  
    ]  
  }  
}
```

**Status:** 200, 42.0ms, 166B

## Get\_Time\_Fail

The screenshot shows the Apollo Studio interface with a failed GraphQL query. The left sidebar displays the schema path: Root > Query > availableTimeslotByID. The main panel shows the query operation and its variables.

**Query Operation:**

```
1 query Query($availableTimeslotByIdId: ID!) {  
2   availableTimeslotByID(id: $availableTimeslotByIdId)  
3 }
```

**Variables:**

```
1 {  
2   "availableTimeslotByIdId": "-1"  
3 }
```

**Response:**

```
{  
  "data": {  
    "availableTimeslotByID": null  
  }  
}
```

**Status:** 200, 37.0ms, 40B

## Add\_Appointment\_Success

The screenshot shows the Apollo Studio interface with a successful GraphQL mutation. The left sidebar displays the 'Documentation' panel for the 'createAppointment' operation, showing the 'input: CreateAppointmentInput!' and its fields: 'doctorID: ID!', 'patientID: ID!', and 'time: Int'. The main panel shows the 'Operation' tab with the following query:

```
1 mutation CreateAppointment($input: CreateAppointmentInput!) {  
2   createAppointment(input: $input) {  
3     doctor {  
4       clinicName  
5       id  
6       name  
7     }  
9     patient {  
10      id  
11      name  
12    }  
13    time  
14  }  
15 }  
16
```

The 'Response' tab shows the JSON output:

```
{  
  "data": {  
    "createAppointment": {  
      "doctor": {  
        "clinicName": "Clinic1",  
        "id": "D_1",  
        "name": "Doctor1"  
      },  
      "id": "A_3",  
      "patient": {  
        "id": "P_1",  
        "name": "Patient1"  
      },  
      "time": 20  
    }  
  }  
}
```

The 'Variables' tab shows the input variables:

```
{  
  "input": {  
    "doctorID": "D_1",  
    "patientID": "P_1",  
    "time": 20  
  }  
}
```

The status bar at the top right indicates 'STATUS 200' and '33.0ms'.

## Add\_Appointment\_Fail

The screenshot shows the Apollo Studio interface with a failed GraphQL mutation. The left sidebar displays the 'Documentation' panel for the 'createAppointment' operation, showing the 'input: CreateAppointmentInput!' and its fields: 'doctorID: ID!', 'patientID: ID!', and 'time: Int'. The main panel shows the 'Operation' tab with the following query:

```
1 mutation CreateAppointment($input: CreateAppointmentInput!) {  
2   createAppointment(input: $input) {  
3     doctor {  
4       id  
5     }  
6     id  
7     patient {  
8       id  
9     }  
10    time  
11  }  
12 }
```

The 'Response' tab shows the JSON output:

```
{  
  "data": {  
    "createAppointment": {  
      "doctor": null,  
      "id": "-1",  
      "patient": null,  
      "time": -1  
    }  
  }  
}
```

The 'Variables' tab shows the input variables:

```
{  
  "input": {  
    "doctorID": "D_1",  
    "patientID": "P_1",  
    "time": 20  
  }  
}
```

The status bar at the top right indicates 'STATUS 200' and '64.0ms'.

## Cancel\_Appointment\_Success

The screenshot shows the Apollo Studio interface with the 'CancelAppointment' mutation selected. The left sidebar displays the documentation for the mutation, including its arguments and fields. The main panel shows the GraphQL operation and its variables. The right sidebar shows the JSON response, which includes the doctor's ID, the appointment's ID, the patient's ID, and the time.

**Documentation**

Root > Mutation > cancelAppointment

**cancelAppointment:** Appointment

**Arguments**

- input: CancelAppointmentInput!

**Fields**

- doctor: Doctor
- id: ID!
- patient: Patient
- time: Int

**Operation**

```
1 mutation CancelAppointment($input:  
2   CancelAppointmentInput!) {  
3   cancelAppointment(input: $input) {  
4     doctor {  
5       id  
6     }  
7     patient {  
8       id  
9     }  
10    time  
11  }
```

**Variables**

```
1 {  
2   "input": {  
3     "appointmentID": "A_1"  
4   }  
5 }
```

**JSON**

```
{  
  "data": {  
    "cancelAppointment": {  
      "doctor": {  
        "id": "D_1"  
      },  
      "id": "A_1",  
      "patient": {  
        "id": "P_1"  
      },  
      "time": 16  
    }  
  }  
}
```

## Cancel\_Appointment\_Fail

The screenshot shows the Apollo Studio interface with the 'CancelAppointment' mutation selected. The left sidebar displays the documentation for the mutation, including its arguments and fields. The main panel shows the GraphQL operation and its variables. The right sidebar shows the JSON response, which includes the doctor's ID, the appointment's ID, the patient's ID, and the time.

**Documentation**

Root > Mutation > cancelAppointment

**cancelAppointment:** Appointment

**Arguments**

- input: CancelAppointmentInput!

**Fields**

- doctor: Doctor
- id: ID!
- patient: Patient
- time: Int

**Operation**

```
1 mutation CancelAppointment($input:  
2   CancelAppointmentInput!) {  
3   cancelAppointment(input: $input) {  
4     doctor {  
5       id  
6     }  
7     patient {  
8       id  
9     }  
10    time  
11  }
```

**Variables**

```
1 {  
2   "input": {  
3     "appointmentID": "A_1"  
4   }  
5 }
```

**JSON**

```
{  
  "data": {  
    "cancelAppointment": {  
      "doctor": null,  
      "id": "-1",  
      "patient": null,  
      "time": -1  
    }  
  }  
}
```

## Update\_Appointment\_Success

The screenshot shows the Apollo Studio interface with a successful GraphQL mutation. The left sidebar displays the 'updateAppointment' mutation under the 'updateAppointment: Appointment' section. The 'Arguments' section shows 'input: UpdateAppointmentInput!'. The 'Fields' section shows 'doctor: Doctor', 'id: ID!', 'patient: Patient', and 'time: Int'. The 'Operation' tab shows the following query:

```
1 mutation UpdateAppointment($input:  
2   UpdateAppointmentInput!) {  
3   updateAppointment(input: $input) {  
4     doctor {  
5       id  
6     }  
7     patient {  
8       id  
9       name  
10    }  
11    time  
12  }
```

The 'Variables' tab shows the following variables:

```
1 {  
2   "input": {  
3     "appointmentID": "A_1",  
4     "patientName": "Patient1_NewName"  
5   }  
6 }
```

The 'JSON' tab shows the response:

```
{  
  "data": {  
    "updateAppointment": {  
      "doctor": {  
        "id": "D_1"  
      },  
      "id": "A_1",  
      "patient": {  
        "id": "P_1",  
        "name": "Patient1_NewName"  
      },  
      "time": 16  
    }  
  }  
}
```

The status bar indicates 'STATUS 200' and '172ms'.

## Update\_Appointment\_Fail

The screenshot shows the Apollo Studio interface with a failed GraphQL mutation. The left sidebar displays the 'updateAppointment' mutation under the 'updateAppointment: Appointment' section. The 'Arguments' section shows 'input: UpdateAppointmentInput!'. The 'Fields' section shows 'doctor: Doctor', 'id: ID!', 'patient: Patient', and 'time: Int'. The 'Operation' tab shows the following query:

```
1 mutation UpdateAppointment($input:  
2   UpdateAppointmentInput!) {  
3   updateAppointment(input: $input) {  
4     doctor {  
5       id  
6     }  
7     patient {  
8       id  
9     }  
10    time  
11  }  
12 }
```

The 'Variables' tab shows the following variables:

```
1 {  
2   "input": {  
3     "appointmentID": "A_3",  
4     "patientName": "Patient3_NewName"  
5   }  
6 }
```

The 'JSON' tab shows the response:

```
{  
  "data": {  
    "updateAppointment": {  
      "doctor": null,  
      "id": "-1",  
      "patient": null,  
      "time": -1  
    }  
  }  
}
```

The status bar indicates 'STATUS 200' and '62.0ms'.

## Reflection

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

An alternative schema is to store available timeslots as a field for each doctor instead of calculating the available timeslots upon query.

The alternative schema is not chosen because every time an appointment is cancel, we'll have to update the available timeslots, which will take more time and is redundant.

- **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).**

The clients won't need to change the old code for existing queries since the API will be backward compatible.

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

New fields (consultation type) need to be added to existing schema. Mutation for creating event will need to be changed to accommodate the new fields.

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

1. While designing queries, use object types instead of object Ids.

Instead of IDs, I used Patient and Doctor objects in the appointment type, so that client can get all information in one query.

2. Naming conventions

All mutations are named as verbs (CreateAppointment, CancelAppointment and UpdateAppointment), and all the queries are self-explanatory (doctorById, availableTimeslotById).