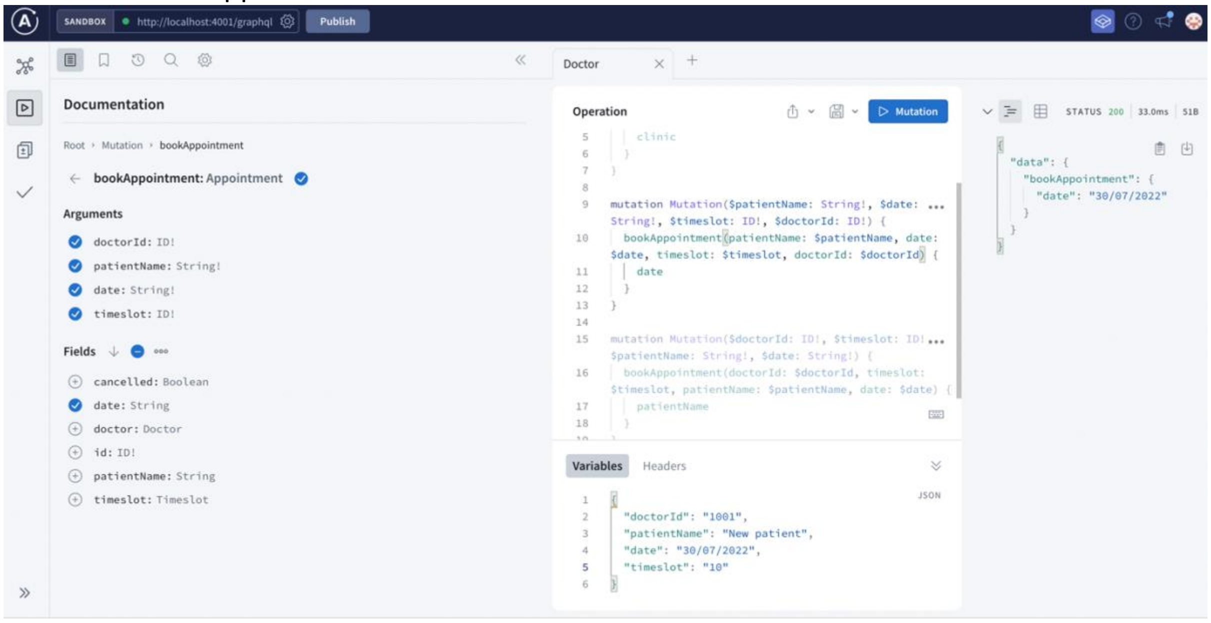
17625: API Design
Assignment 2.2: GraphQL API
Shashank Shekhar, sshekha3

1.Testing

- Identifier: BookAppointmentSuccess



Documentation

Root › Mutation › bookAppointment

bookAppointment: Appointment

Arguments

```
doctorId: ID!
patientName: String!
date: String!
timeslot: ID!
```

Fields

```
cancelled: Boolean
date: String
doctor: Doctor
id: ID!
patientName: String
timeslot: Timeslot
```

Operation

```
    clinic
  }
}

mutation Mutation($patientName: String!, $date: String!, $timeslot: ID!, $doctorId: ID!) {
  bookAppointment(patientName: $patientName, date: $date, timeslot: $timeslot, doctorId: $doctorId) {
    date
  }
}

mutation Mutation($doctorId: ID!, $timeslot: ID! $patientName: String!, $date: String!) {
  bookAppointment(doctorId: $doctorId, timeslot: $timeslot, patientName: $patientName, date: $date) {
    patientName
  }
```

Variables

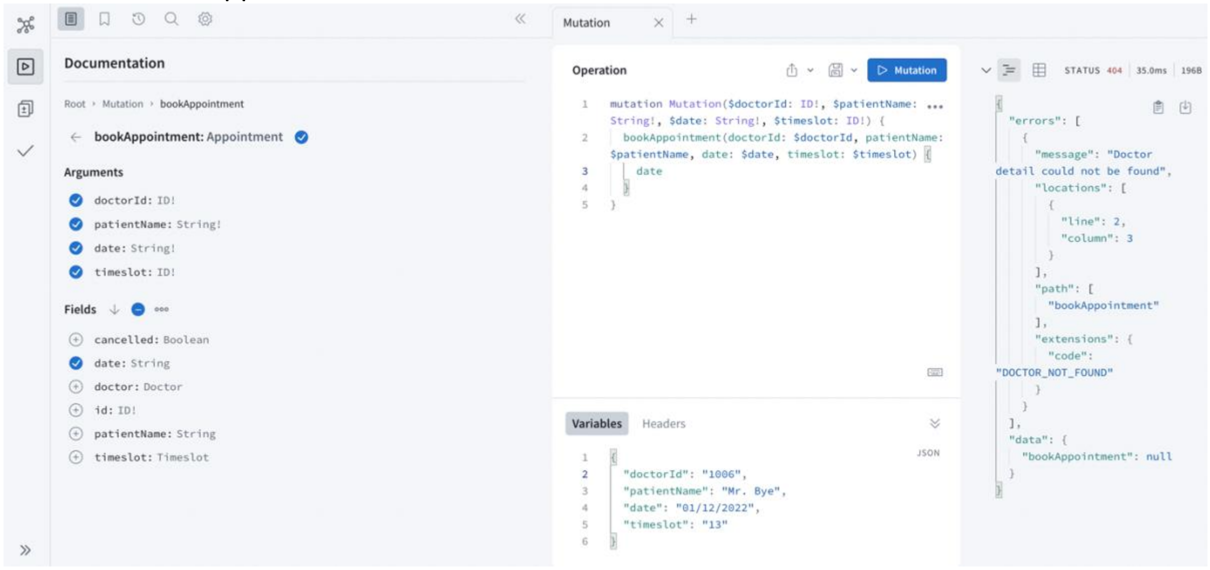
```
{
  "doctorId": "1001",
  "patientName": "New patient",
  "date": "30/07/2022",
  "timeslot": "10"
}
```

STATUS 200 | 33.0ms | 51B

```
{
  "data": {
    "bookAppointment": {
      "date": "30/07/2022"
    }
  }
}
```

Figure 1: Successful appointment booking

- Identifier: BookAppointmentFail



Documentation

Root › Mutation › bookAppointment

bookAppointment: Appointment

Arguments

```
doctorId: ID!
patientName: String!
date: String!
timeslot: ID!
```

Fields

```
cancelled: Boolean
date: String
doctor: Doctor
id: ID!
patientName: String
timeslot: Timeslot
```

Operation

```
mutation Mutation($doctorId: ID!, $patientName: String!, $date: String!, $timeslot: ID!) {
  bookAppointment(doctorId: $doctorId, patientName: $patientName, date: $date, timeslot: $timeslot) {
    date
  }
}
```

Variables

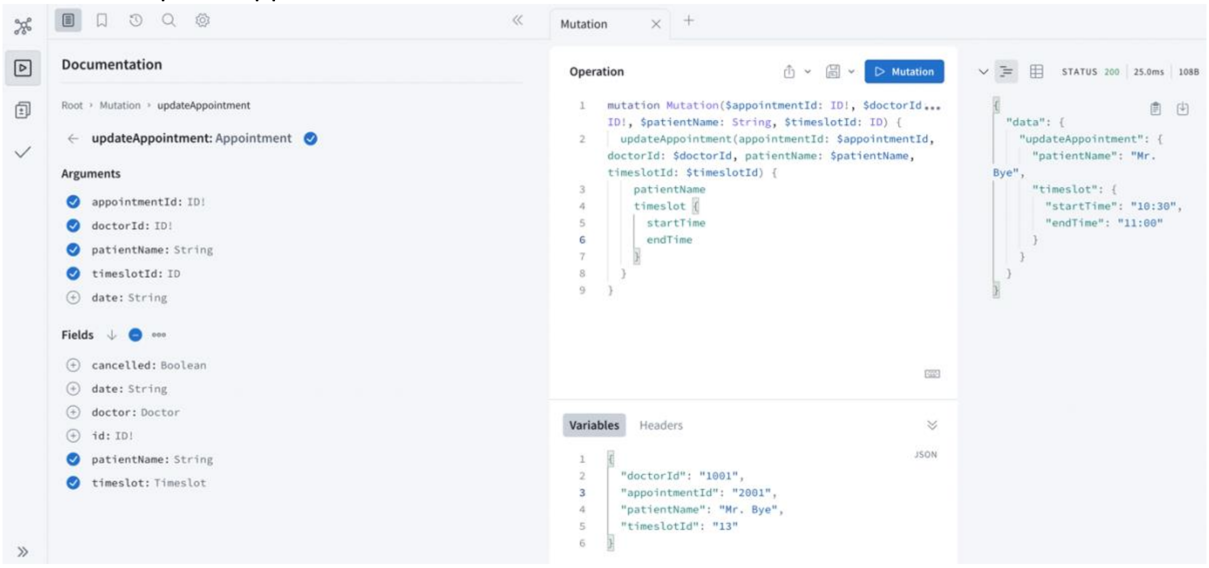
```
{
  "doctorId": "1006",
  "patientName": "Mr. Bye",
  "date": "01/12/2022",
  "timeslot": "13"
}
```

STATUS 404 | 35.0ms | 196B

```
{
  "errors": [
    {
      "message": "Doctor detail could not be found",
      "locations": [
        {
          "line": 2,
          "column": 3
        }
      ],
      "path": [
        "bookAppointment"
      ],
      "extensions": {
        "code": "DOCTOR_NOT_FOUND"
      }
    }
  ],
  "data": {
    "bookAppointment": null
  }
}
```

Figure 2: Unsuccessful appointment booking

- Identifier: UpdateAppointmentSuccess



Documentation

Root › Mutation › updateAppointment

updateAppointment: Appointment

Arguments

- appointmentId: ID!
- doctorId: ID!
- patientName: String
- timeslotId: ID
- date: String

Fields

- cancelled: Boolean
- date: String
- doctor: Doctor
- id: ID!
- patientName: String
- timeslot: Timeslot

Operation

```graphql
mutation Mutation($appointmentId: ID!, $doctorId...
ID!, $patientName: String, $timeslotId: ID) {
  updateAppointment(appointmentId: $appointmentId,
doctorId: $doctorId, patientName: $patientName,
timeslotId: $timeslotId) {
    patientName
    timeslot {
      startTime
      endTime
    }
  }
}
```

Variables

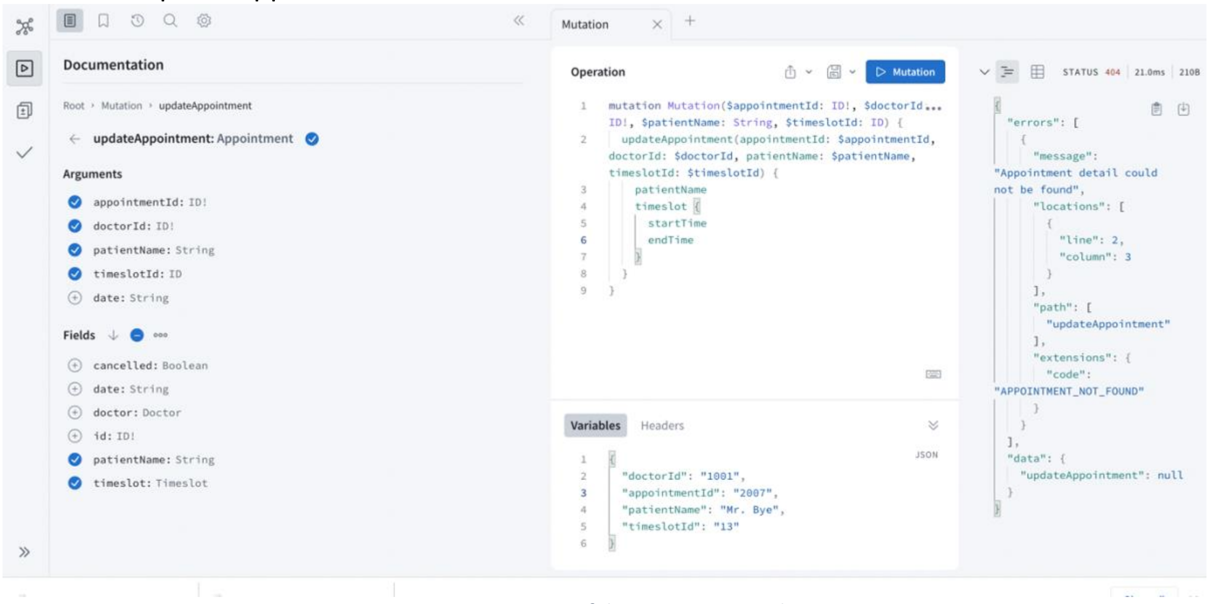
```json
{
  "doctorId": "1001",
  "appointmentId": "2001",
  "patientName": "Mr. Bye",
  "timeslotId": "13"
}
```

STATUS 200 | 25.0ms | 108B

```json
{
  "data": {
    "updateAppointment": {
      "patientName": "Mr.
Bye",
      "timeslot": {
        "startTime": "10:30",
        "endTime": "11:00"
      }
    }
  }
}
```

Figure 3: Successful appointment update

- Identifier: UpdateAppointmentFail



Documentation

Root › Mutation › updateAppointment

updateAppointment: Appointment

Arguments

- appointmentId: ID!
- doctorId: ID!
- patientName: String
- timeslotId: ID
- date: String

Fields

- cancelled: Boolean
- date: String
- doctor: Doctor
- id: ID!
- patientName: String
- timeslot: Timeslot

Operation

```graphql
mutation Mutation($appointmentId: ID!, $doctorId...
ID!, $patientName: String, $timeslotId: ID) {
  updateAppointment(appointmentId: $appointmentId,
doctorId: $doctorId, patientName: $patientName,
timeslotId: $timeslotId) {
    patientName
    timeslot {
      startTime
      endTime
    }
  }
}
```

Variables

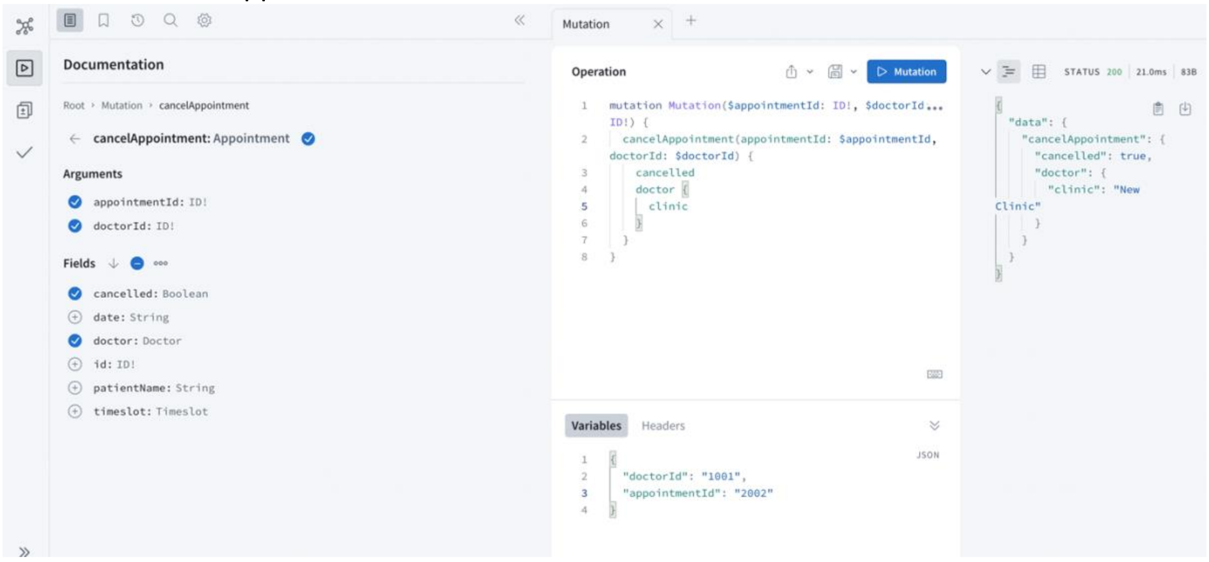
```json
{
  "doctorId": "1001",
  "appointmentId": "2007",
  "patientName": "Mr. Bye",
  "timeslotId": "13"
}
```

STATUS 404 | 21.0ms | 210B

```json
{
  "errors": [
    {
      "message":
"Appointment detail could
not be found",
      "locations": [
        {
          "line": 2,
          "column": 3
        }
      ],
      "path": [
        "updateAppointment"
      ],
      "extensions": {
        "code":
"APPOINTMENT_NOT_FOUND"
      }
    }
  ],
  "data": {
    "updateAppointment": null
  }
}
```

Figure 4: Unsuccessful appointment update

- Identifier: CancelAppointmentSuccess



```
cancelAppointment: Appointment
appointmentId: ID!
doctorId: ID!
cancelled: Boolean
date: String
doctor: Doctor
id: ID!
patientName: String
timeslot: Timeslot
```

```
mutation Mutation($appointmentId: ID!, $doctorId...
ID!) {
  cancelAppointment(appointmentId: $appointmentId,
doctorId: $doctorId) {
    cancelled
    doctor {
      clinic
    }
  }
}
```

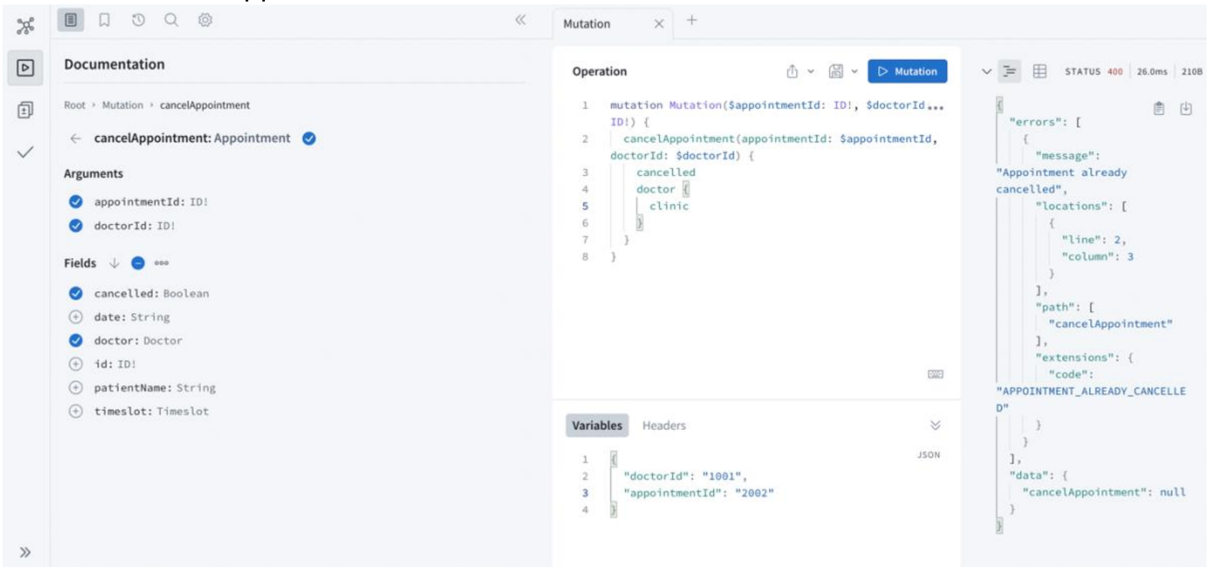
```
{
  "doctorId": "1001",
  "appointmentId": "2002"
}
```

STATUS 200 | 21.0ms | 83B

```
{
  "data": {
    "cancelAppointment": {
      "cancelled": true,
      "doctor": {
        "clinic": "New
Clinic"
      }
    }
  }
}
```

Figure 5: Successful appointment cancellation

- Identifier: CancelAppointmentFail



```
cancelAppointment: Appointment
appointmentId: ID!
doctorId: ID!
cancelled: Boolean
date: String
doctor: Doctor
id: ID!
patientName: String
timeslot: Timeslot
```

```
mutation Mutation($appointmentId: ID!, $doctorId...
ID!) {
  cancelAppointment(appointmentId: $appointmentId,
doctorId: $doctorId) {
    cancelled
    doctor {
      clinic
    }
  }
}
```

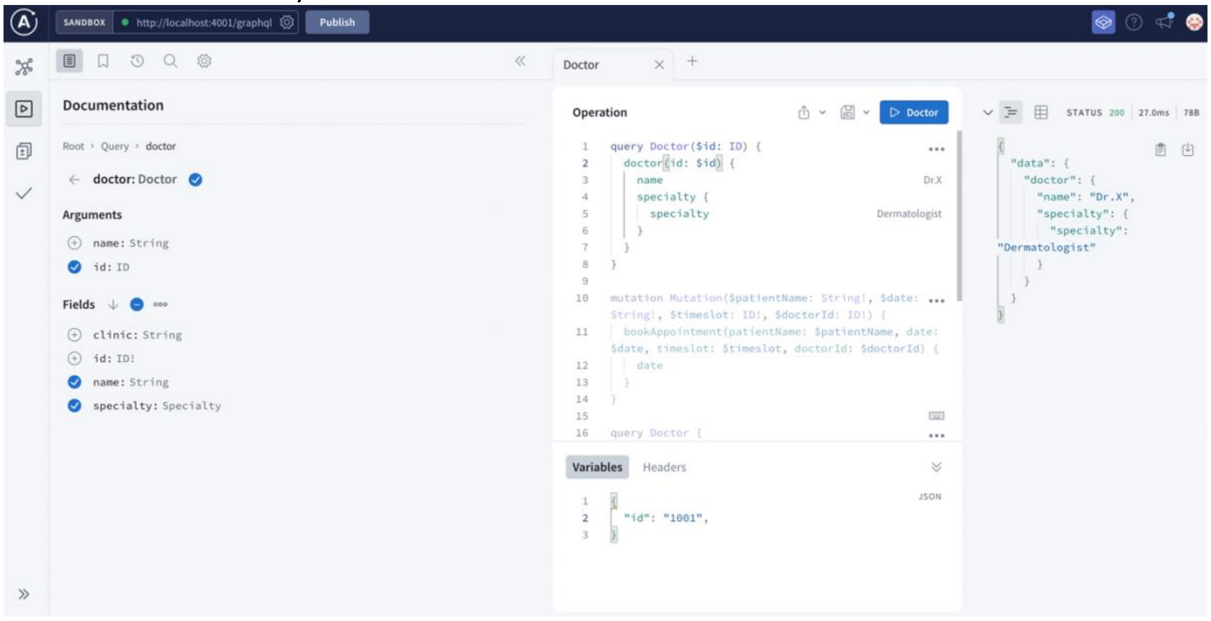
```
{
  "doctorId": "1001",
  "appointmentId": "2002"
}
```

STATUS 400 | 26.0ms | 210B

```
{
  "errors": [
    {
      "message":
"Appointment already
cancelled",
      "locations": [
        {
          "line": 2,
          "column": 3
        }
      ],
      "path": [
        "cancelAppointment"
      ],
      "extensions": {
        "code":
"APPOINTMENT_ALREADY_CANCELLE
D"
      }
    }
  ],
  "data": {
    "cancelAppointment": null
  }
}
```

Figure 6: Unsuccessful appointment cancellation

- Identifier: DoctorQuerySuccess



SANDBOX http://localhost:4001/graphql Publish

Documentation

Root › Query › doctor

doctor: Doctor

Arguments

```
name: String
id: ID
```

Fields

```
clinic: String
id: ID!
name: String
specialty: Specialty
```

Operation

```
query Doctor($id: ID) {
  doctor(id: $id) {
    name
    specialty {
      specialty
    }
  }
}

mutation Mutation($patientName: String!, $date: String!, $timeslot: ID!, $doctorId: ID!) {
  bookAppointment(patientName: $patientName, date: $date, timeslot: $timeslot, doctorId: $doctorId) {
    date
  }
}

query Doctor {
```

Variables

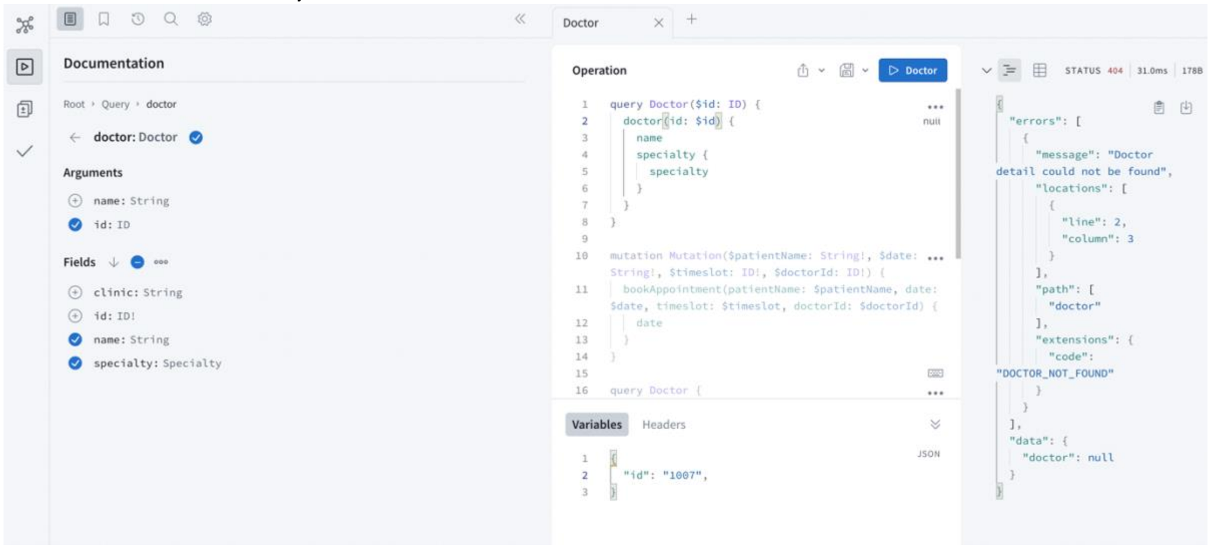
```
{
  "id": "1001",
}
```

STATUS 200 | 27.0ms | 78B

```
{
  "data": {
    "doctor": {
      "name": "Dr.X",
      "specialty": {
        "specialty":
"Dermatologist"
      }
    }
  }
}
```

Figure 7: Successful doctor query

- Identifier: DoctorQueryFail



Documentation

Root › Query › doctor

doctor: Doctor

Arguments

```
name: String
id: ID
```

Fields

```
clinic: String
id: ID!
name: String
specialty: Specialty
```

Operation

```
query Doctor($id: ID) {
  doctor(id: $id) {
    name
    specialty {
      specialty
    }
  }
}

mutation Mutation($patientName: String!, $date: String!, $timeslot: ID!, $doctorId: ID!) {
  bookAppointment(patientName: $patientName, date: $date, timeslot: $timeslot, doctorId: $doctorId) {
    date
  }
}

query Doctor {
```

Variables

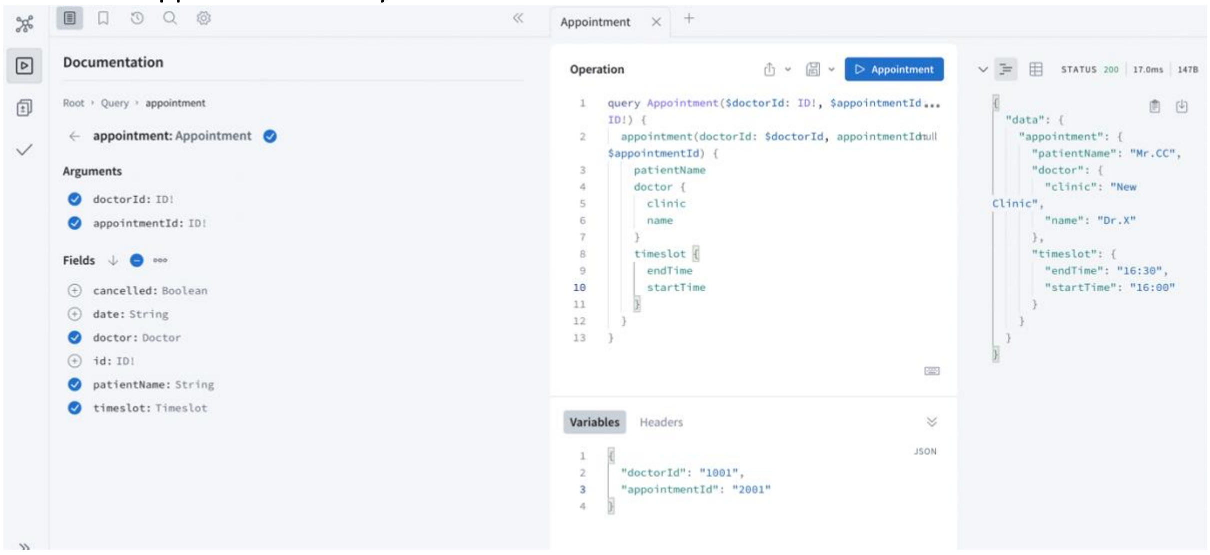
```
{
  "id": "1007",
}
```

STATUS 404 | 31.0ms | 178B

```
{
  "errors": [
    {
      "message": "Doctor
detail could not be found",
      "locations": [
        {
          "line": 2,
          "column": 3
        }
      ],
      "path": [
        "doctor"
      ],
      "extensions": {
        "code":
"DOCTOR_NOT_FOUND"
      }
    }
  ],
  "data": {
    "doctor": null
  }
}
```

Figure 8: Unsuccessful doctor query

- Identifier: AppointmentQuerySuccess



Documentation

Root › Query › appointment

appointment: Appointment

Arguments

- doctorId: ID!
- appointmentId: ID!

Fields

- cancelled: Boolean
- date: String
- doctor: Doctor
- id: ID!
- patientName: String
- timeslot: Timeslot

Operation

```
query Appointment($doctorId: ID!, $appointmentId...
ID!) {
  appointment(doctorId: $doctorId, appointmentId:
$appointmentId) {
    patientName
    doctor {
      clinic
      name
    }
    timeslot {
      endTime
      startTime
    }
  }
}
```

Variables

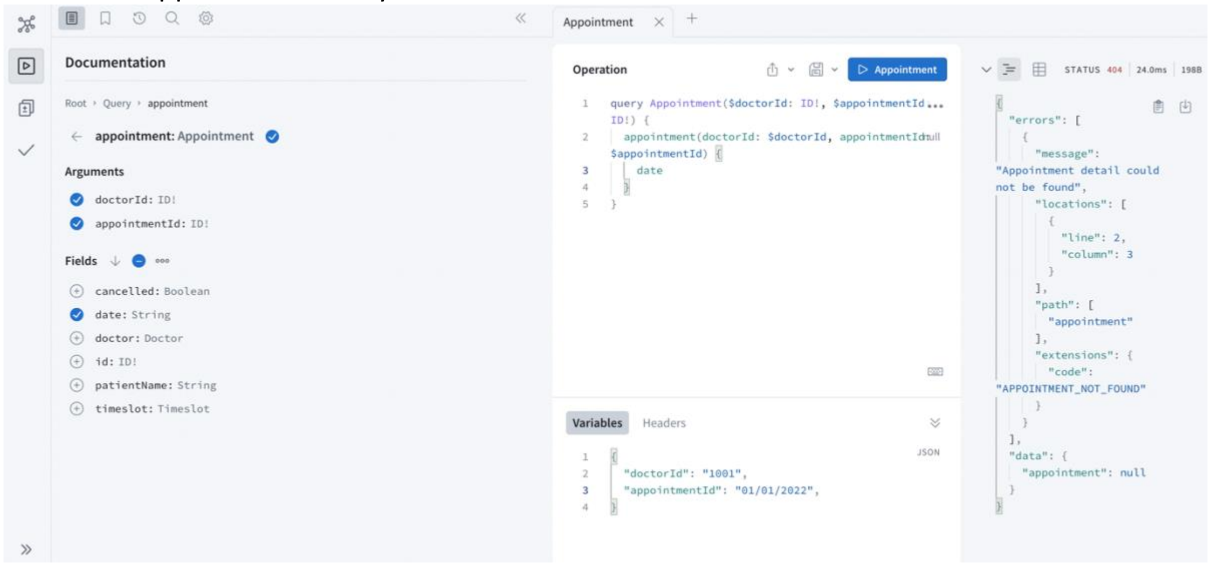
```
{
  "doctorId": "1001",
  "appointmentId": "2001"
}
```

STATUS 200 | 17.0ms | 147B

```
{
  "data": {
    "appointment": {
      "patientName": "Mr.CC",
      "doctor": {
        "clinic": "New
Clinic",
        "name": "Dr.X"
      },
      "timeslot": {
        "endTime": "16:30",
        "startTime": "16:00"
      }
    }
  }
}
```

Figure 9: Successful appointment query

- Identifier: AppointmentQueryFail



Documentation

Root › Query › appointment

appointment: Appointment

Arguments

- doctorId: ID!
- appointmentId: ID!

Fields

- cancelled: Boolean
- date: String
- doctor: Doctor
- id: ID!
- patientName: String
- timeslot: Timeslot

Operation

```
query Appointment($doctorId: ID!, $appointmentId...
ID!) {
  appointment(doctorId: $doctorId, appointmentId:
$appointmentId) {
    date
  }
}
```

Variables

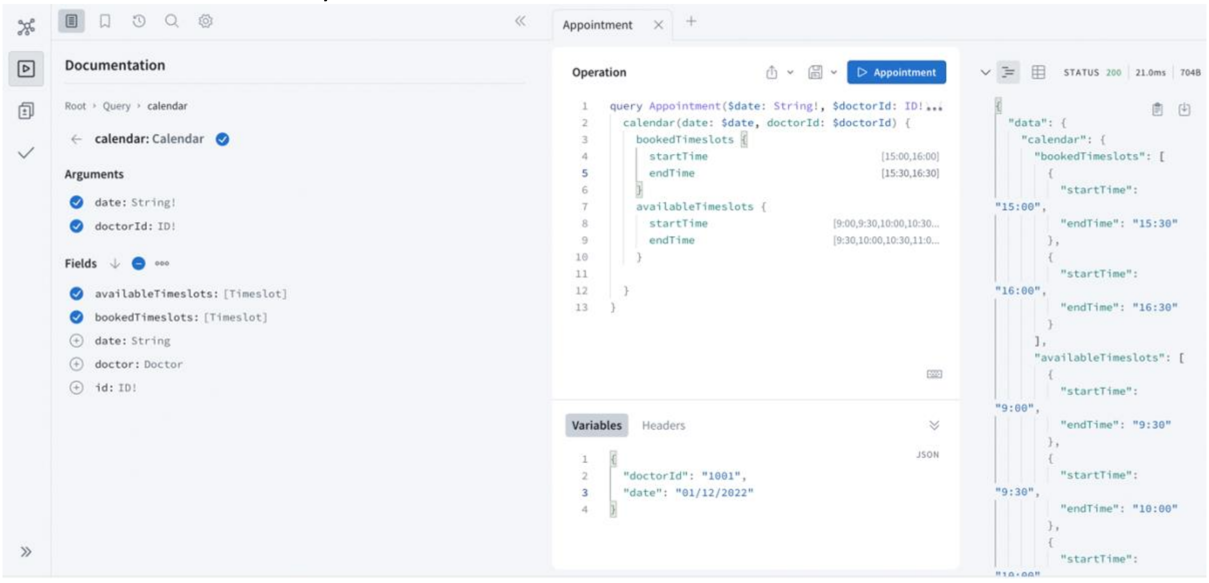
```
{
  "doctorId": "1001",
  "appointmentId": "01/01/2022",
}
```

STATUS 404 | 24.0ms | 198B

```
{
  "errors": [
    {
      "message":
"Appointment detail could
not be found",
      "locations": [
        {
          "line": 2,
          "column": 3
        }
      ],
      "path": [
        "appointment"
      ],
      "extensions": {
        "code":
"APPOINTMENT_NOT_FOUND"
      }
    }
  ],
  "data": {
    "appointment": null
  }
}
```

Figure 10: Unsuccessful appointment query

- Identifier: CalendarQuerySuccess


Documentation

Root › Query › calendar

calendar: Calendar ✓

Arguments

✓ date: String!
✓ doctorId: ID!

Fields

✓ availableTimeslots: [Timeslot]
✓ bookedTimeslots: [Timeslot]
date: String
doctor: Doctor
id: ID!

Appointment

Operation

```
query Appointment($date: String!, $doctorId: ID!...
  calendar(date: $date, doctorId: $doctorId) {
    bookedTimeslots {
      startTime                          [15:00,16:00]
      endTime                            [15:30,16:30]
    }
    availableTimeslots {
      startTime                [9:00,9:30,10:00,10:30...
      endTime                  [9:30,10:00,10:30,11:0...
    }

  }
}
```

Variables | Headers

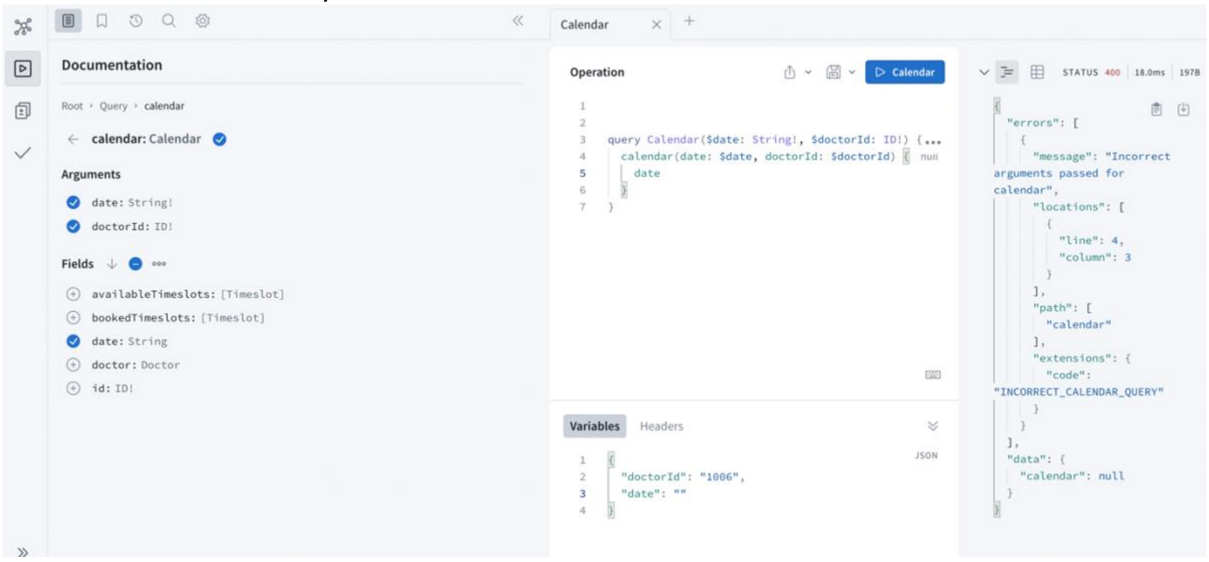
```
{
  "doctorId": "1001",
  "date": "01/12/2022"
}
```

STATUS 200 | 21.0ms | 704B

```
{
  "data": {
    "calendar": {
      "bookedTimeslots": [
        {
          "startTime":
"15:00",
          "endTime": "15:30"
        },
        {
          "startTime":
"16:00",
          "endTime": "16:30"
        }
      ],
      "availableTimeslots": [
        {
          "startTime":
"9:00",
          "endTime": "9:30"
        },
        {
          "startTime":
"9:30",
          "endTime": "10:00"
        },
        {
          "startTime":
```

Figure 11: Successful calendar query

- Identifier: CalendarQueryFail


Documentation

Root › Query › calendar

calendar: Calendar ✓

Arguments

✓ date: String!
✓ doctorId: ID!

Fields

availableTimeslots: [Timeslot]
bookedTimeslots: [Timeslot]
✓ date: String
doctor: Doctor
id: ID!

Calendar

Operation

```


query Calendar($date: String!, $doctorId: ID!) {...
  calendar(date: $date, doctorId: $doctorId) {  null
    date
  }
}
```

Variables | Headers

```
{
  "doctorId": "1006",
  "date": ""
}
```

STATUS 400 | 18.0ms | 197B

```
{
  "errors": [
    {
      "message": "Incorrect
arguments passed for
calendar",
      "locations": [
        {
          "line": 4,
          "column": 3
        }
      ],
      "path": [
        "calendar"
      ],
      "extensions": {
        "code":
"INCORRECT_CALENDAR_QUERY"
      }
    }
  ],
  "data": {
    "calendar": null
  }
}
```

Figure 12: Unsuccessful calendar query

2. Reflection

I. **What were some of the alternative schema and query design options you considered? Why did you choose the selected options?**
Answer:

While developing the schema and query for the given problem, the focus was kept on the ease of data retrieval and delivery. The data was designed as it would exist in a real-world environment as entities having relations. As such, the query design was developed keeping in mind how to connect those entities. For instance, the *appointments* data includes id for *doctor* but the query for appointments includes the resolved field of doctor, matching all the data for that doctor to the id and then delivering it to the end user. Similarly, the mutations were also developed keeping this design choice in mind.
Another factor that influenced this design choice was the requirements of the system. The queries and mutations are designed as such that they can be easily conveyed to the client and are easy to adopt. At a given instance, the user might need all or some data from a query as per the requirement, as such all the data regarding a particular field has been provided. For instance, in when fetching calendar information, we provide all the available and booked timeslots, and along with that the start time and end time of each timeslot is also included. The customer might choose to see these details for instance in a dashboard to quickly see the schedule for a given date.
The queries and mutations also have been parameterized to enable accurate data retrieval. Using the unique identity in arguments help exact data match and to return accurate response codes. It is designed with expectations that each user interface will have access to these ids which it will use to query the data.
Alternative design choices which were considered included not including nested details of individual entities. This would have increased query time but have sacrificed the purpose of developing an expansive query API. It would have required multiple queries to fetch multiple data which is now being handled on the server end. This also helps in evaluating entity relationships on the server side rather than leaving its responsibility on the client side.

II. **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.**

A. **What changes will the clients (API consumer) need to make to their existing queries (if any).**
Answer:
In case of any modifications, to the 'Event' structure, or in this assignment known as 'Appointment', the clients would need to change their queries to include a *patient* and *consultationType* key in their query. This change can be communicated easily through updates and will not break any changes. Since, these fields will be added as an additional field, the client need not worry about backward compatibility as previous implementations where these fields are not used in any query will continue to work as normal. New fields, where

they require this data can be queried with updated query structure. For instance, an updated query might have the following structure:

```
query GetAppointmentDetail {
	Appointment(appointmentId: ID!, doctorId: ID!) {
		date
		doctor {
			name
			clinic
			specialty
		}
		patientName
		patientDetail{
			age
			id
			address
		}
		consultationType
		timeslot {
			startTime
			endTime
		}
	}
}
```

B. **How will you accommodate the changes in your existing Schema and Query types?**
Answer:
The ‘Event’ structure in the implementation is being referred to as ‘Appointment’. In case of any augmentation to the ‘Appointment’ entity when we augment reference to patient details, expectation is that the patient detail will be stored as a separate table in the database which can have fields like “id”, “name”, “address”, “age”, etc. As such, a new schema type for patient will be developed like this:

```
type Patient {
	id: ID!
	name: String
	age: Int
	address: String
}
```

The ‘Appointment’ or ‘Event’ schema will then include the reference to ‘Patient’ schema to cater to the new requirements as shown below:

```
type Appointment {
	id: ID!
	doctor: Doctor
	patientName: String
	patient: Patient
	timeslot: Timeslot
	date: String
	cancelled: Boolean
}
```

Similarly, if consultation types are included in the schema, an ‘enum’ type schema needs to be developed as follows:

```
enum Consultation {
      FIRST_TIME
      FOLLOW_UP
}
```

To integrate this into our ‘Event’ or ‘Appointment’ schema, the schema will be updated as follows:

```
type Appointment {
      id: ID!
      doctor: Doctor
      patientName: String
      patient: Patient
      consultationType: Consultation
      timeslot: Timeslot
      date: String
      cancelled: Boolean
}
```

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

Answer:

Following best practices were incorporated while designing the GraphQL API for the assignment:

a. Use of ID type for unique identifiers: The data used in this assignment has been stored in the form of table in a database where each field has a primary key. To use this primary key, ID type provided by GraphQL has been used to help in unique identification of data and its retrieval. For e.g.,

```
type Specialty {
      id: ID!
      specialty: String
}
```

```
type Calendar {
      id: ID!
      date: String
      doctor: Doctor
      bookedTimeslots: [Timeslot]
      availableTimeslots: [Timeslot]
}
```

b. Object types are used to reduce queries: The schema and resolvers have been designed such that they reduce the effort for the client to retrieve data. For instance, when we query and event, called appointment, which has doctor associated with it, all the details related to the doctor are also returned instead of returning just a reference or id for the doctor. The schema below illustrates this point:

```
type Appointment {
      id: ID!
      doctor: Doctor
      patientName: String
      timeslot: Timeslot
      date: String
      cancelled: Boolean
}
```

The highlighted field ‘Doctor’ is of the type ‘Doctor’ which in turn has schema as follows:

```
type Doctor {
      id: ID!
      name: String
      specialty: Specialty
      clinic: String
}
```

This enables the client to retrieve the entire doctor information from the appointment query if they need to by expanding the doctor field to include doctor detail such as name, specialty, clinic, etc.. The query can be written as follows:

```
query GetAppointmentDetail {
      Appointment(appointmentId: ID!, doctorId: ID!) {
            date
            doctor {
                  name
                  clinic
                  specialty
            }
            patientName
            timeslot {
                  startTime
                  endTime
            }
      }
}
```