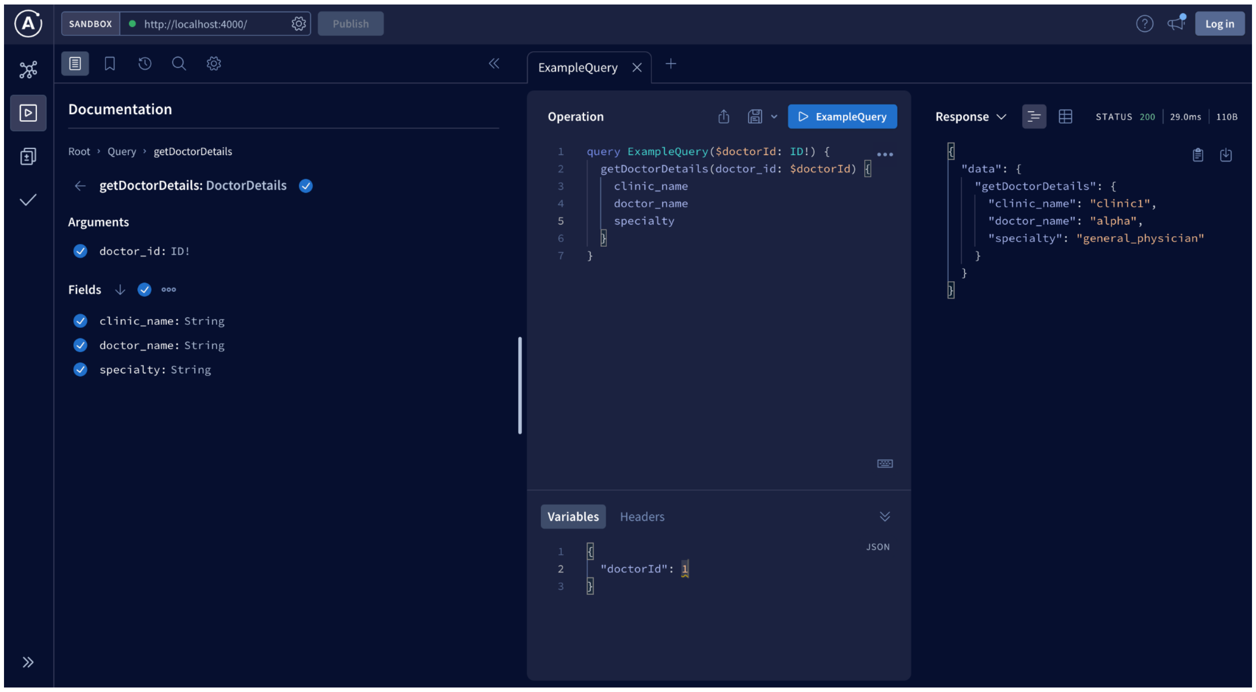
API Design
Assignment A2.2

Testing

Notes:
The test case identifiers follow the following format: [Q/M][int].[0/1]. Q/M stands for Query/Mutation. The following int is the serial number, and the binary digit after the dot stands for happy path test case (1) and error condition test case (0).
This API is for the use of the developers to perform CRUD operations.

Happy path test cases for the queries are mentioned below.

Q1.1. getDoctorDetailsHappy
Here the input is a valid doctor_id. The expected output is the doctor_name, clinic, and specialty.

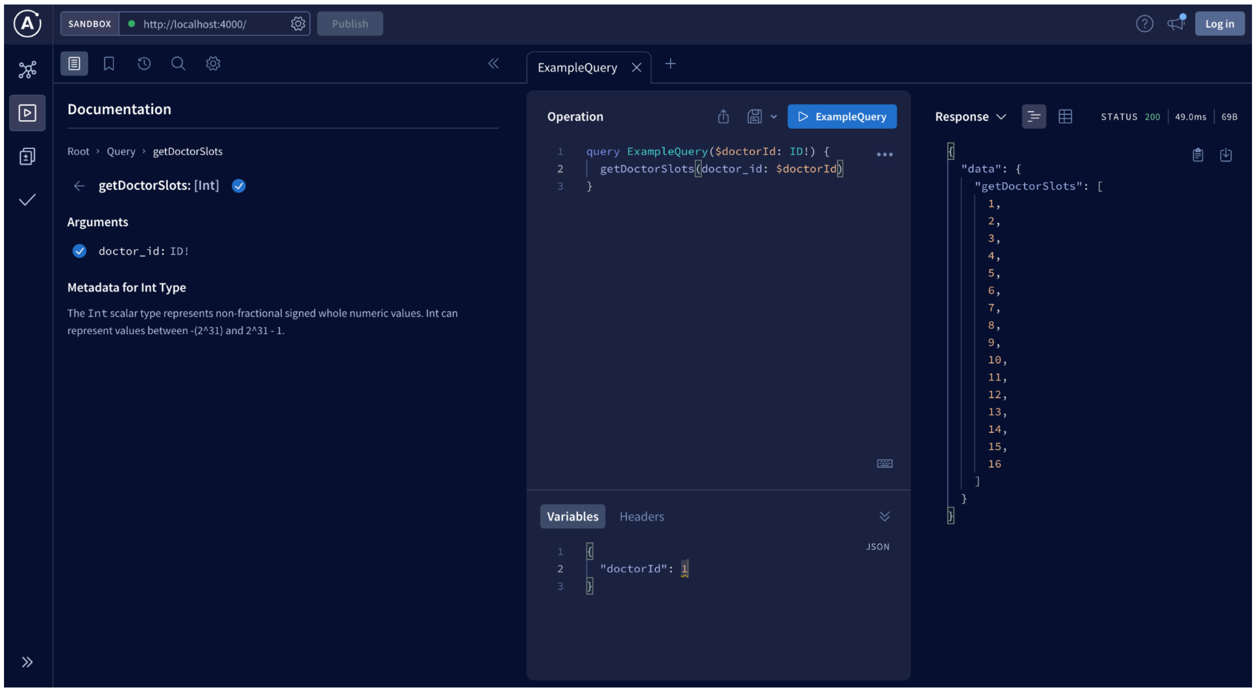


```
query ExampleQuery($doctorId: ID!) {
  getDoctorDetails(doctor_id: $doctorId) {
    clinic_name
    doctor_name
    specialty
  }
}
```

```
{
  "data": {
    "getDoctorDetails": {
      "clinic_name": "clinic1",
      "doctor_name": "alpha",
      "specialty": "general_physician"
    }
  }
}
```

```
{
  "doctorId": 1
}
```

Q2.1. getDoctorSlotsHappy
Here the input is a valid doctor_id. The expected output is the array of available_timeslots.



getDoctorSlots: [Int]

Arguments

doctor_id: ID!

Metadata for Int Type

The Int scalar type represents non-fractional signed whole numeric values. Int can represent values between -(2^31) and 2^31 - 1.

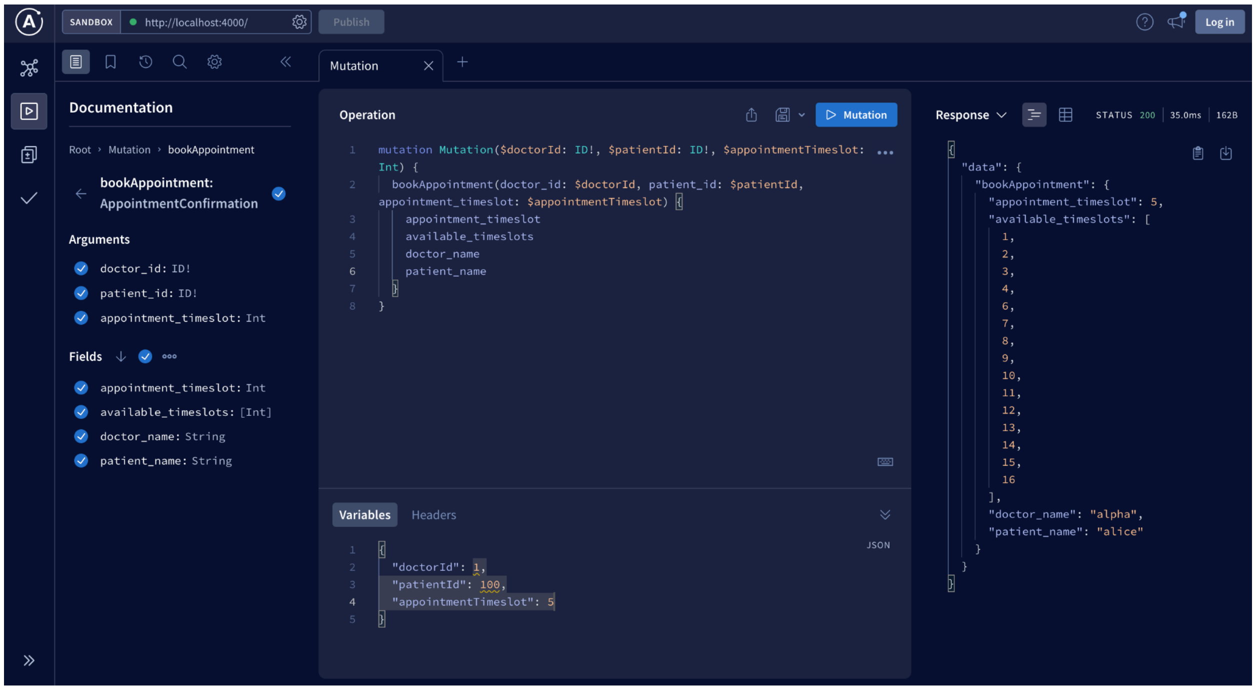
```
query ExampleQuery($doctorId: ID!) {
  getDoctorSlots(doctor_id: $doctorId)
}
```

```
{
  "doctorId": 1
}
```

```
{
  "data": {
    "getDoctorSlots": [
      1,
      2,
      3,
      4,
      5,
      6,
      7,
      8,
      9,
      10,
      11,
      12,
      13,
      14,
      15,
      16
    ]
  }
}
```

Happy path test cases for the mutations are mentioned below.

M1.1. bookAppointmentHappy
Here the input is a valid doctor_id, a valid patient_id, and an appointment_timeslot which is present in the available_timeslots array. The expected output is the doctor_name, the new available_timeslots array from which the appointment_timeslot has been removed, patient_name, and appointment_timeslot.



bookAppointment: AppointmentConfirmation

Arguments

- doctor_id: ID!
- patient_id: ID!
- appointment_timeslot: Int

Fields

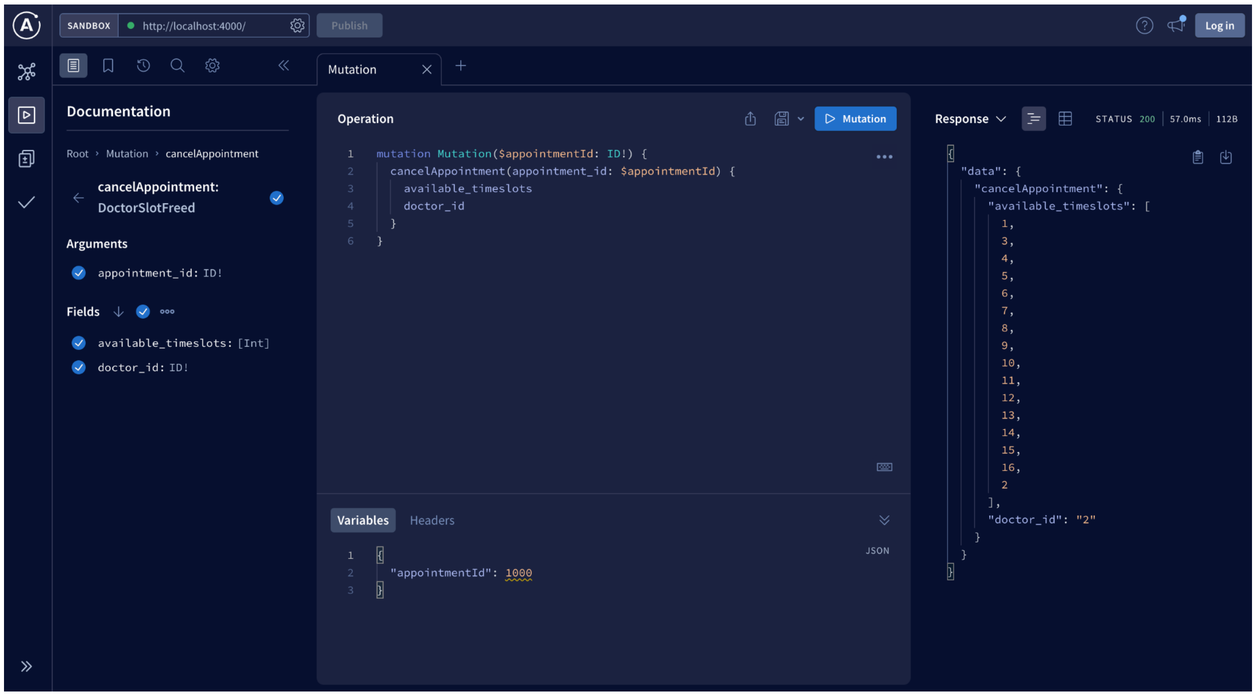
- appointment_timeslot: Int
- available_timeslots: [Int]
- doctor_name: String
- patient_name: String

```graphql
mutation Mutation($doctorId: ID!, $patientId: ID!, $appointmentTimeslot: Int) {
  bookAppointment(doctor_id: $doctorId, patient_id: $patientId, appointment_timeslot: $appointmentTimeslot) {
    appointment_timeslot
    available_timeslots
    doctor_name
    patient_name
  }
}
```

```json
{
  "doctorId": 1,
  "patientId": 100,
  "appointmentTimeslot": 5
}
```

```json
{
  "data": {
    "bookAppointment": {
      "appointment_timeslot": 5,
      "available_timeslots": [
        1,
        2,
        3,
        4,
        6,
        7,
        8,
        9,
        10,
        11,
        12,
        13,
        14,
        15,
        16
      ],
      "doctor_name": "alpha",
      "patient_name": "alice"
    }
  }
}
```

M2.1. cancelAppointmentHappy
Here the input is a valid appointment_id. The expected output is the doctor_id, and the new available_timeslots array to which the appointment_timeslot has been added back.



cancelAppointment: DoctorSlotFreed

Arguments

appointment_id: ID!

Fields

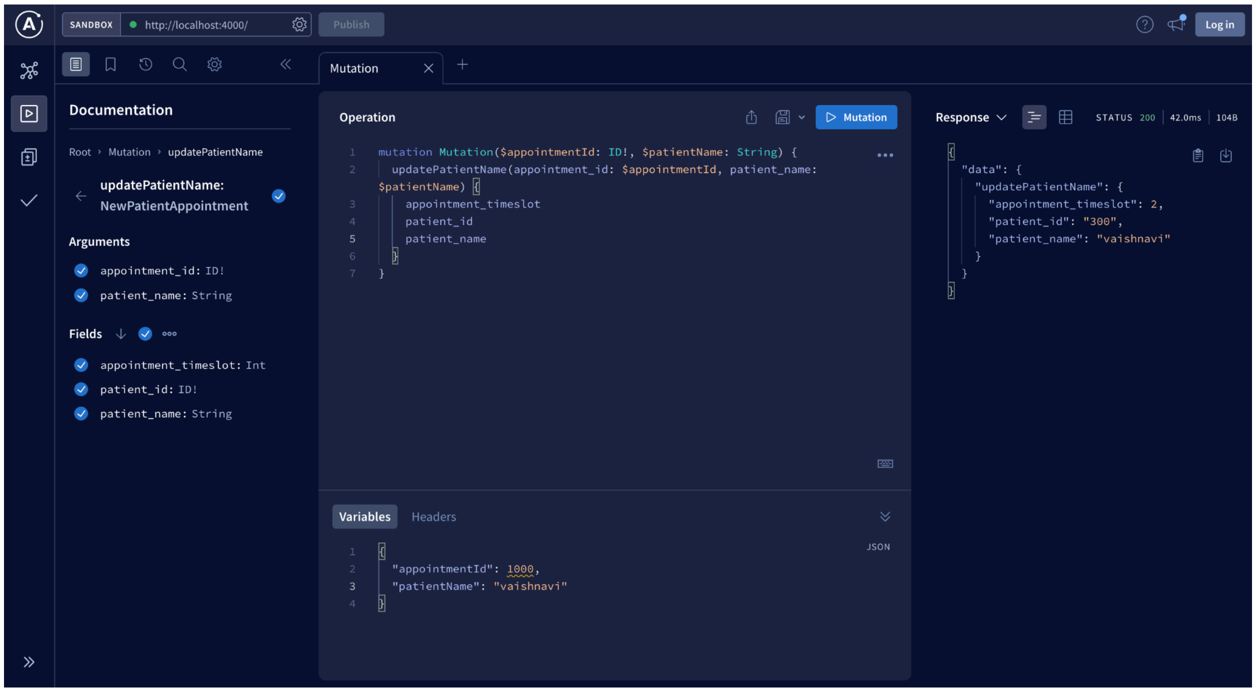
available_timeslots: [Int]
doctor_id: ID!

```
mutation Mutation($appointmentId: ID!) {
  cancelAppointment(appointment_id: $appointmentId) {
    available_timeslots
    doctor_id
  }
}
```

```
{
  "appointmentId": 1000
}
```

```
{
  "data": {
    "cancelAppointment": {
      "available_timeslots": [
        1,
        3,
        4,
        5,
        6,
        7,
        8,
        9,
        10,
        11,
        12,
        13,
        14,
        15,
        16,
        2
      ],
      "doctor_id": "2"
    }
  }
}
```

M3.1. updatePatientNameHappy
Here the input is a valid appointment_id, and the new patient_name. The expected output is a new patient_id for this new patient, the new patient_name, and the appointment_timeslot.



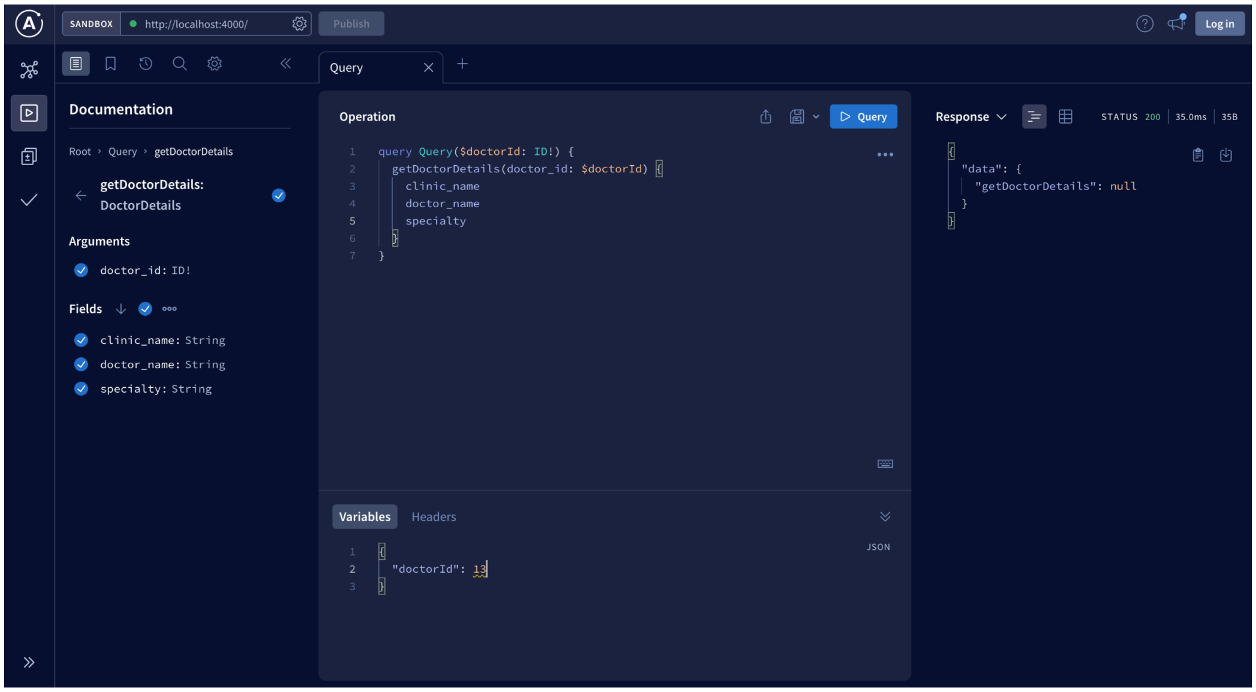
```
mutation Mutation($appointmentId: ID!, $patientName: String) {
  updatePatientName(appointment_id: $appointmentId, patient_name:
$patientName) {
    appointment_timeslot
    patient_id
    patient_name
  }
}
```

```
{
  "appointmentId": 1000,
  "patientName": "vaishnavi"
}
```

```
{
  "data": {
    "updatePatientName": {
      "appointment_timeslot": 2,
      "patient_id": "300",
      "patient_name": "vaishnavi"
    }
  }
}
```

Error condition test cases for the queries are mentioned below.

Q1.0. getDoctorDetailsUnhappy
Here the input is an invalid doctor_id. The expected output is an error.

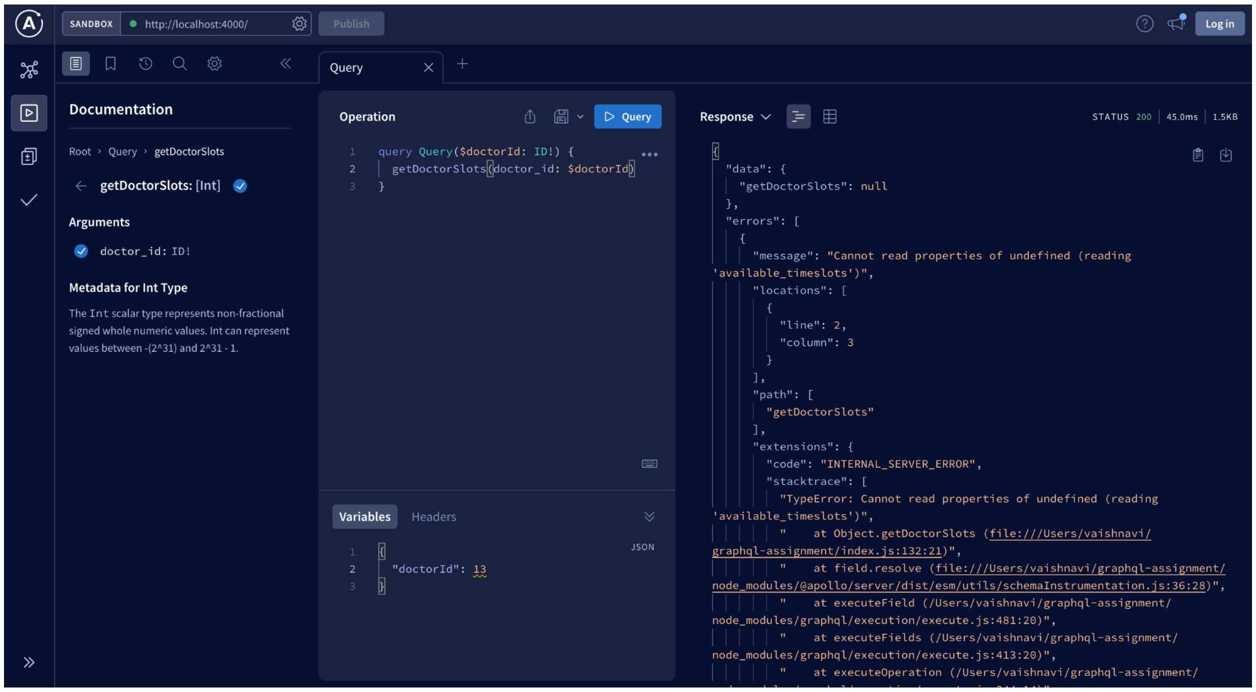


```
query Query($doctorId: ID!) {
  getDoctorDetails(doctor_id: $doctorId) {
    clinic_name
    doctor_name
    specialty
  }
}
```

```
{
  "doctorId": 13
}
```

```
{
  "data": {
    "getDoctorDetails": null
  }
}
```

Q2.0. getDoctorSlotsUnhappy
Here the input is an invalid doctor_id. The expected output is an error.



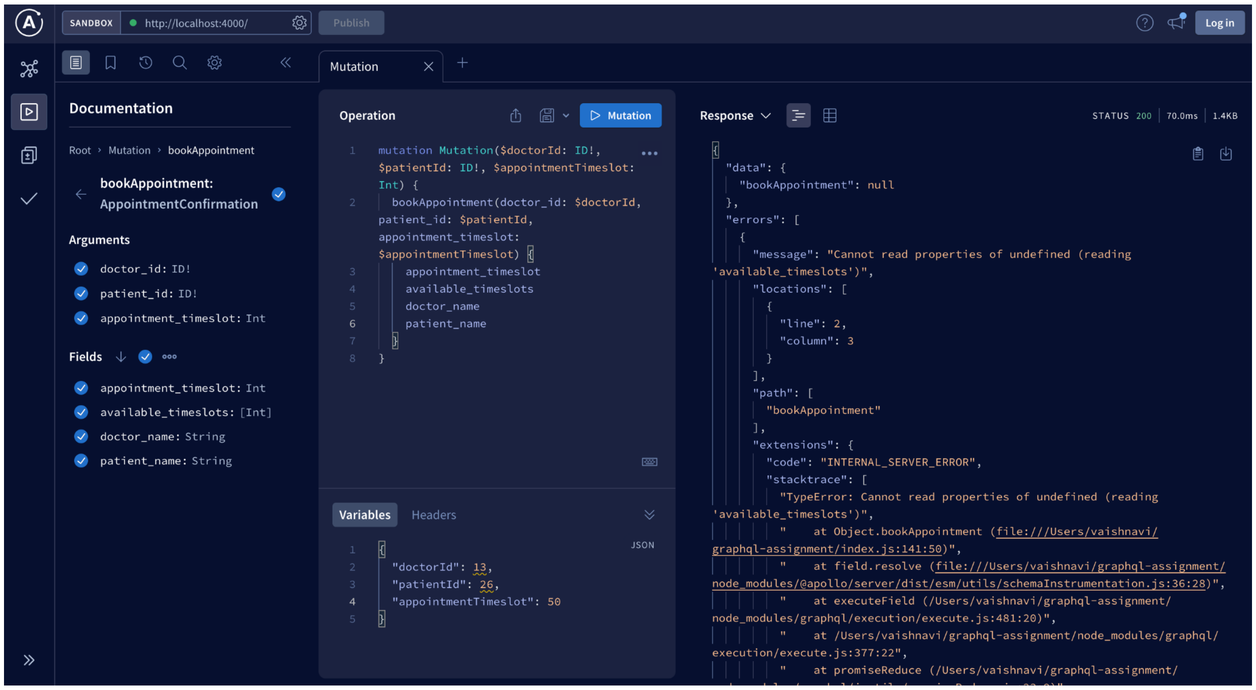
```
query Query($doctorId: ID!) {
  getDoctorSlots(doctor_id: $doctorId)
}
```

```
{
  "doctorId": 13
}
```

```
{
  "data": {
    "getDoctorSlots": null
  },
  "errors": [
    {
      "message": "Cannot read properties of undefined (reading
'available_timeslots')",
      "locations": [
        {
          "line": 2,
          "column": 3
        }
      ],
      "path": [
        "getDoctorSlots"
      ],
      "extensions": {
        "code": "INTERNAL_SERVER_ERROR",
        "stacktrace": [
          "TypeError: Cannot read properties of undefined (reading
'available_timeslots')",
          "    at Object.getDoctorSlots (file:///Users/vaishnavi/
graphql-assignment/index.js:132:21)",
          "    at field.resolve (file:///Users/vaishnavi/graphql-assignment/
node_modules/@apollo/server/dist/esm/utils/schemaInstrumentation.js:36:28)",
          "    at executeField (/Users/vaishnavi/graphql-assignment/
node_modules/graphql/execution/execute.js:481:20)",
          "    at executeFields (/Users/vaishnavi/graphql-assignment/
node_modules/graphql/execution/execute.js:413:20)",
          "    at executeOperation (/Users/vaishnavi/graphql-assignment/
```

Error condition test cases for the mutations are mentioned below.

M1.0. bookAppointmentUnhappy
Here the input is an invalid doctor_id, an invalid patient_id, and an appointment_timeslot which is not present in the available_timeslots array. The expected output is an error.



bookAppointment: AppointmentConfirmation

Arguments

- doctor_id: ID!
- patient_id: ID!
- appointment_timeslot: Int

Fields

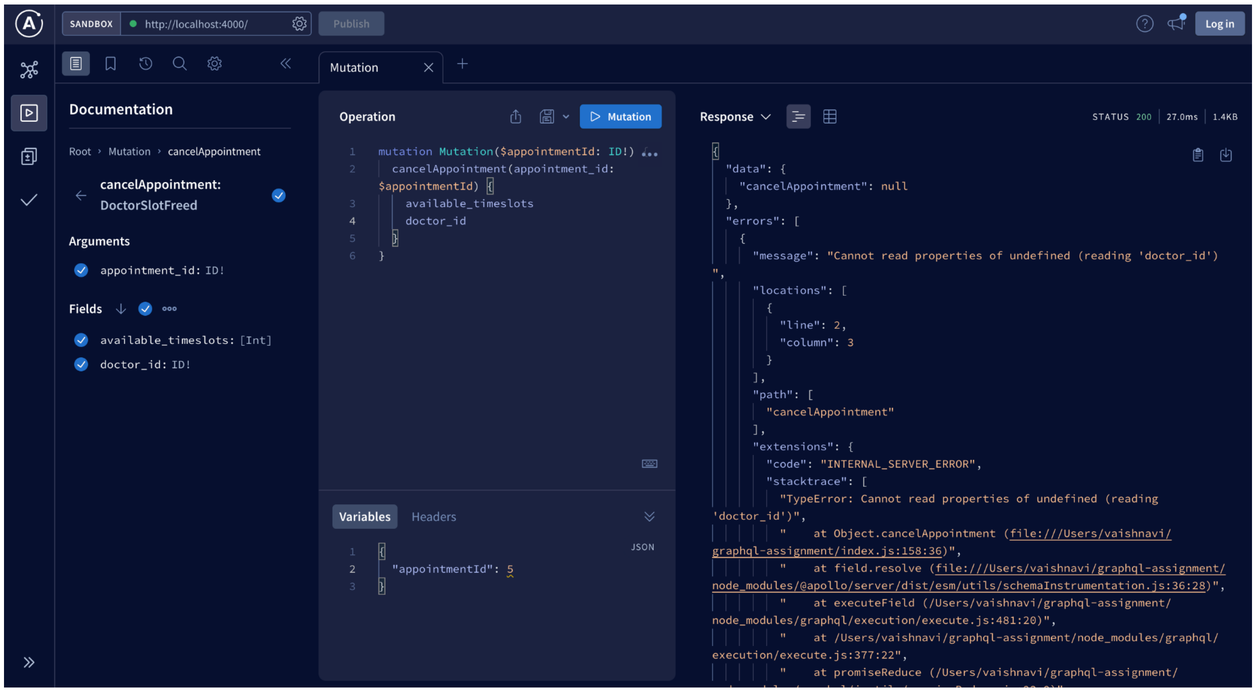
- appointment_timeslot: Int
- available_timeslots: [Int]
- doctor_name: String
- patient_name: String

```
mutation Mutation($doctorId: ID!,
$patientId: ID!, $appointmentTimeslot:
Int) {
  bookAppointment(doctor_id: $doctorId,
patient_id: $patientId,
appointment_timeslot:
$appointmentTimeslot) {
    appointment_timeslot
    available_timeslots
    doctor_name
    patient_name
  }
}
```

```
{
  "doctorId": 13,
  "patientId": 26,
  "appointmentTimeslot": 50
}
```

```
{
  "data": {
    "bookAppointment": null
  },
  "errors": [
    {
      "message": "Cannot read properties of undefined (reading
'available_timeslots')",
      "locations": [
        {
          "line": 2,
          "column": 3
        }
      ],
      "path": [
        "bookAppointment"
      ],
      "extensions": {
        "code": "INTERNAL_SERVER_ERROR",
        "stacktrace": [
          "TypeError: Cannot read properties of undefined (reading
'available_timeslots')",
          "    at Object.bookAppointment (file:///Users/vaishnavi/
graphql-assignment/index.js:141:50)",
          "    at field.resolve (file:///Users/vaishnavi/graphql-assignment/
node_modules/@apollo/server/dist/esm/utils/schemaInstrumentation.js:36:28)",
          "    at executeField (/Users/vaishnavi/graphql-assignment/
node_modules/graphql/execution/execute.js:481:20)",
          "    at /Users/vaishnavi/graphql-assignment/node_modules/graphql/
execution/execute.js:377:22",
          "    at promiseReduce (/Users/vaishnavi/graphql-assignment/
```

M2.0. cancelAppointmentUnhappy
Here the input is an invalid appointment_id. The expected output is an error.



Documentation

Root › Mutation › cancelAppointment

cancelAppointment: DoctorSlotFreed

Arguments

appointment_id: ID!

Fields

available_timeslots: [Int]

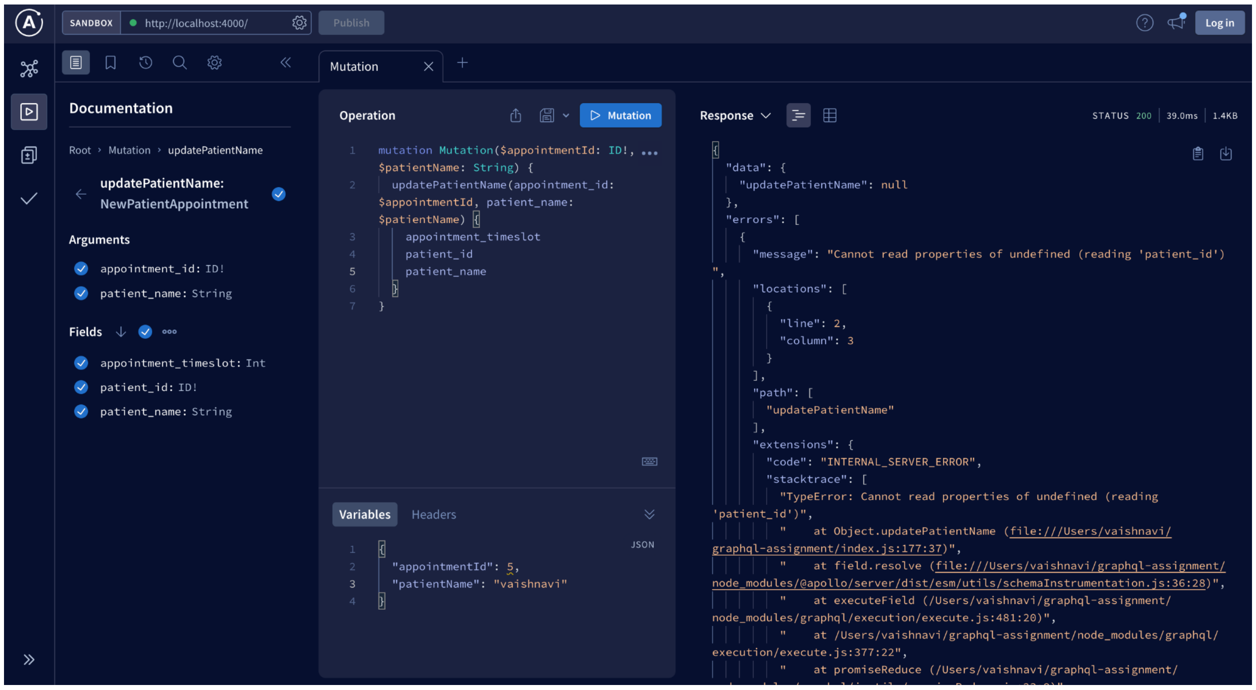
doctor_id: ID!

```
mutation Mutation($appointmentId: ID!) {
  cancelAppointment(appointment_id: $appointmentId) {
    available_timeslots
    doctor_id
  }
}
```

```
{
  "appointmentId": 5
}
```

```
{
  "data": {
    "cancelAppointment": null
  },
  "errors": [
    {
      "message": "Cannot read properties of undefined (reading 'doctor_id')
",
      "locations": [
        {
          "line": 2,
          "column": 3
        }
      ],
      "path": [
        "cancelAppointment"
      ],
      "extensions": {
        "code": "INTERNAL_SERVER_ERROR",
        "stacktrace": [
          "TypeError: Cannot read properties of undefined (reading
'doctor_id')",
          "    at Object.cancelAppointment (file:///Users/vaishnavi/
graphql-assignment/index.js:158:36)",
          "    at field.resolve (file:///Users/vaishnavi/graphql-assignment/
node_modules/@apollo/server/dist/esm/utils/schemaInstrumentation.js:36:28)",
          "    at executeField (/Users/vaishnavi/graphql-assignment/
node_modules/graphql/execution/execute.js:481:20)",
          "    at /Users/vaishnavi/graphql-assignment/node_modules/graphql/
execution/execute.js:377:22",
          "    at promiseReduce (/Users/vaishnavi/graphql-assignment/
```

M3.0. updatePatientNameUnhappy
Here the input is an invalid appointment_id, and the new patient_name. The expected output is an error.



Documentation

Root › Mutation › updatePatientName

updatePatientName: NewPatientAppointment

Arguments

appointment_id: ID!

patient_name: String

Fields

appointment_timeslot: Int

patient_id: ID!

patient_name: String

```
mutation Mutation($appointmentId: ID!, $patientName: String) {
  updatePatientName(appointment_id: $appointmentId, patient_name: $patientName) {
    appointment_timeslot
    patient_id
    patient_name
  }
}
```

```
{
  "appointmentId": 5,
  "patientName": "vaishnavi"
}
```

```
{
  "data": {
    "updatePatientName": null
  },
  "errors": [
    {
      "message": "Cannot read properties of undefined (reading 'patient_id')
",
      "locations": [
        {
          "line": 2,
          "column": 3
        }
      ],
      "path": [
        "updatePatientName"
      ],
      "extensions": {
        "code": "INTERNAL_SERVER_ERROR",
        "stacktrace": [
          "TypeError: Cannot read properties of undefined (reading
'patient_id')",
          "    at Object.updatePatientName (file:///Users/vaishnavi/
graphql-assignment/index.js:177:37)",
          "    at field.resolve (file:///Users/vaishnavi/graphql-assignment/
node_modules/@apollo/server/dist/esm/utils/schemaInstrumentation.js:36:28)",
          "    at executeField (/Users/vaishnavi/graphql-assignment/
node_modules/graphql/execution/execute.js:481:20)",
          "    at /Users/vaishnavi/graphql-assignment/node_modules/graphql/
execution/execute.js:377:22",
          "    at promiseReduce (/Users/vaishnavi/graphql-assignment/
```

Reflection

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

Some alternative design options considered were having separate entities or types for clinics and specialties as well. However, I decided to not go with that option because those elements can be efficiently captured in the Doctor entity itself, which is what I have done. Another alternative considered was not having a Patient entity at all, as its information can be conveyed through just the Doctor and Appointment entities. However, I decided to not go with that option either because it might have made the schema harder to understand, and the queries more complex. That is why, I decided that having three entities or types namely the Doctor, Patient, and Appointment is the best option given the mentioned factors.

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).
My design already incorporates the Patient as an object instead of just the patient name, so the only changes required from the client's side would be for them to select which fields they want, and they have additional options now like patient details, consultation type (first time/follow-up etc.).

How will you accommodate the changes in your existing Schema and Query types?
I can accommodate the changes in my existing design by adding these additional patient details and consultation type (first time/follow-up etc.) fields to my type definitions and the Patient and Doctor entities in my original ER diagram.

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

1. I have expressed my schema in an accepted 'language' such as the ER diagram shown in the A2.1 document.
2. I have used ID type for unique identifiers and non-nullable types for mandatory fields.
3. While designing queries, I have use object types instead of object Ids.