Changes to last assignment:

1. One major change is the return type of DoctorByID, where the doctor's appointments are not listed anymore to make the return response more concise. Other queries and mutations are not changed.

2. Then the original database is designed to be as below:

In the original database, there will be 2 doctors and 4 appointments

```
let doctors =[{
        id: 0,
        name: "Alice",
        clinic: "Excel Dental",
        specialty: "Teeth Filling",
        availableSlots: [
            "9:00 - 9:30", "9:30 - 10:00", "10:30 - 11:00",
            "11:00 - 11:30", "11:30 - 12:00", "12:00 - 12:30", "12:30 - 13:00",
            "13:00 - 13:30", "13:30 - 14:00", "14:00 - 14:30", "14:30 - 15:00",
            "15:30 - 16:00", "16:00 - 16:30", "16:30 - 17:00",
        ]
    },{
        id: 1,
        name: "Makky",
        clinic: "UPMC",
        specialty: "Coughing",
        availableSlots: [
            "9:00 - 9:30", "9:30 - 10:00", "10:00 - 10:30", "10:30 - 11:00",
            "11:00 - 11:30", "11:30 - 12:00", "12:00 - 12:30", "12:30 - 13:00",
            "13:30 - 14:00", "14:00 - 14:30", "14:30 - 15:00",
            "15:30 - 16:00", "16:00 - 16:30", "16:30 - 17:00",
        ]
    }]
```

```
let appointments = [{
        id: 0,
        doctorID: 0,
        patient: "Bob",
        time: "15:00 - 15:30"
    },{
        id: 1,
        doctorID: 1,
        patient: "Bob",
        time: "15:00 - 15:30"
    },{
        id: 2,
        doctorID: 1,
        patient: "David",
        time: "13:00 - 13:30"
    },{
        id: 3,
        doctorID: 0,
        patient: "Helen",
        time: "10:00 - 10:30"
    }]

module.exports = {
    doctors,
    appointments
}
```

3. The test cases are changed with respect to the original database

Test Case Identifier	Test case Description	Inputs	Expected Output	Remarks

AppointmentByID Valid	Enter a valid appointID to see information about the appointment	appointmentID: 1	Status 200
			{
			“id” : 1
			“doctorID”: 1
			“patient”: “Bob”
			“time”: “15:00 - 15:30”
			}
				Happy path.

AppointmentByID Invalid	Enter a non-exist appointID	appointmentID: 4	Status 400
			{
			“error”: “Appointment not exist”
			}
				Return status 400 with error message “appointment not exist”

DoctorByID Valid	Enter a valid doctorID to see information about the doctor	doctorID: 1	Status 200
			{
			“id”: 1
			“name”: “Makky”
			“clinic”: UMPC”
			“specialty”: “Coughing”
			“availableSlots”: [
			“9:00 - 9:30”,
			“9:30 - 10:00”
			…
			]
			}
				Happy path.

DoctorByID Invalid	Enter a non-exist(negative) doctor ID to see information about the doctor	doctorID: -1	Status 400
			{
			“error”: “Doctor ID must be positive”
			}
				Return status 400 with error message “Doctor ID not exist”.

Available Slots Valid	Enter a valid doctorID to see its available time slots during the day.	doctorID: 1	Status 200
			{
			[
			“9:00 - 9:30”,
			“9:30 - 10:00”,
			“10:00 - 10:30”,
			…
			]
			}
				Happy path. Return all available time slots of the doctor.

Available Slots Invalid	Enter a non-exist doctorID to see its available time slots during the day	doctorID: 3	Status 400
			{
			“error”: “Doctor id not exist”
			}
				Return status 400 with error message “doctor not exist”.

Book Appointment Valid	Enter a valid doctorID, valid time slot and patient name to book an appointment.	doctorID: 1
		patient: “Kate”
		time: 11:00 - 11:30
			Status 200
			{
			“id”: 4
			“doctorID”: 1
			“patient”: “Kate”
			“time”: “9:30 - 10:00”
			}
				Happy path. Return created appointment.

Book Appointment Invalid	Enter a valid doctorID, invalid time slot and patient name to book an appointment.	doctorID:1
		patient: “Kate”
		time: 15:00 - 15:30
			Status 400
			{
			“error”: “Unavailable time slot”
			}
				Return status 400 with error message “Unavailable time slot”.

Cancel Appointment Valid	Enter a valid appointmentID to cancel it.	appointmentID: 0	Status 200
			{
			}
				Happy path.

Cancel	Enter an invalid	appointmentID: 4	Status 400	Return status 400 with error

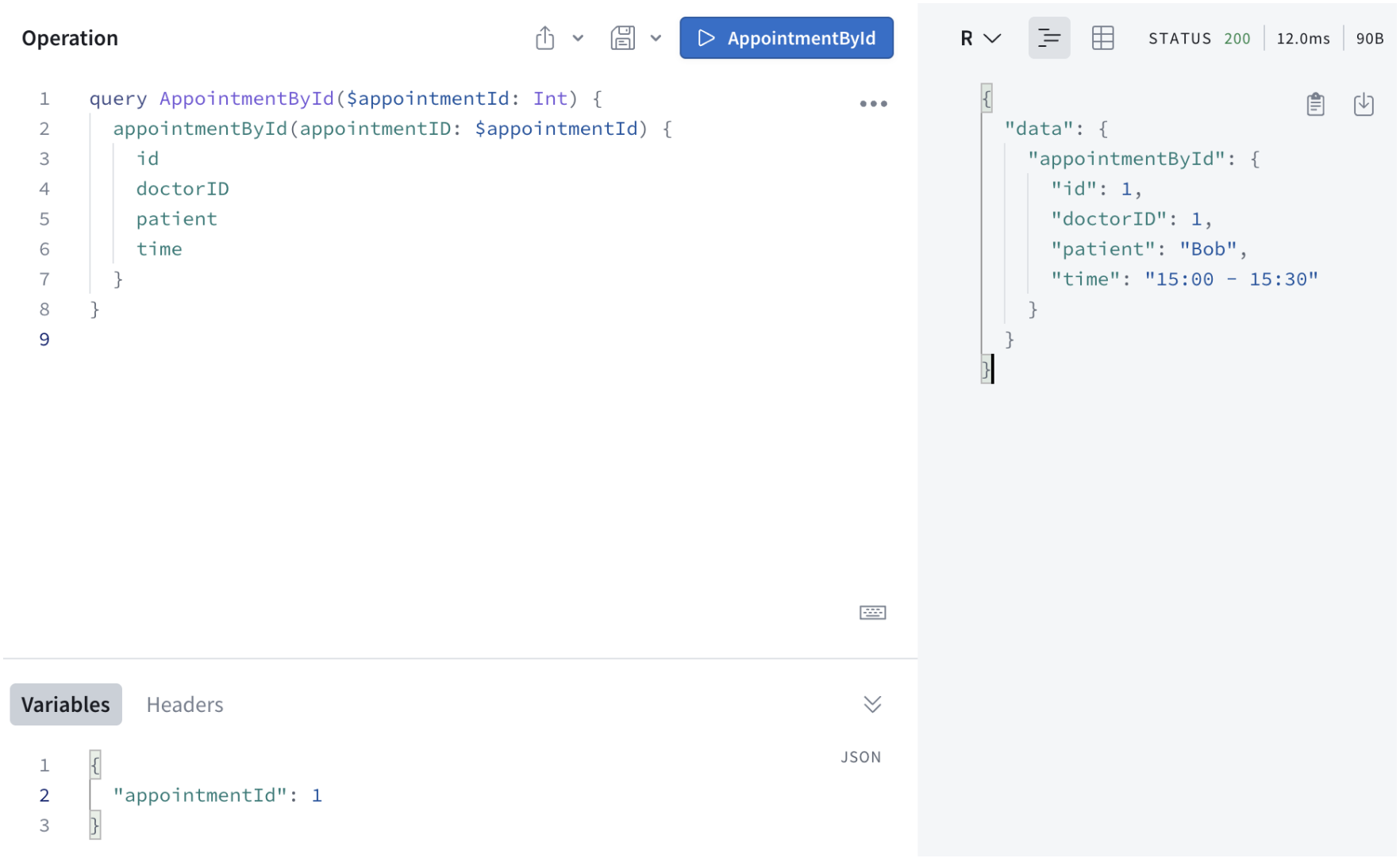
Appointment Invalid	appointment ID to cancel.		{
			“error”: “Appointment not exist”
			}
				message “appointment not exist”.

Update Appointment Valid	Enter a valid appointmentID and a different time slot.	appointmentID: 1
		name: “Taylor”
			Status 200
			{
			“id”: 1
			“doctorID”: 1
			“patient”: “Taylor”
			“time”: “15:00 - 15:30”
			}
				Happy path. Return modified appointment.

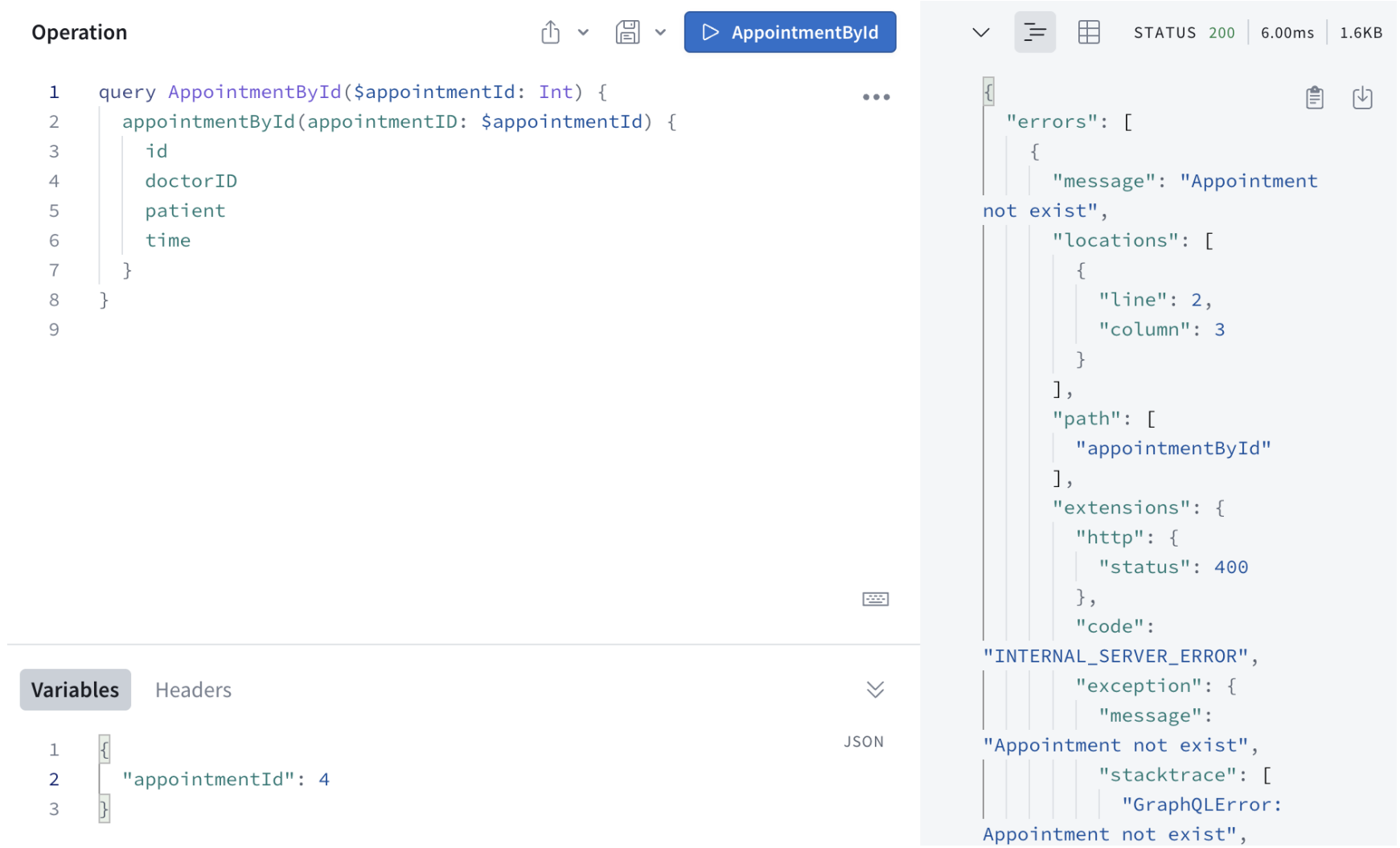
Update Appointment Invalid	Enter a valid appointmentID and the same time slot.	appointmentID: 2
		patient: “David”
			Status 400
			{
			“error”: “Patient not updated”
			}
				Return status 400 with error message “Patient not updated”.

Test results:

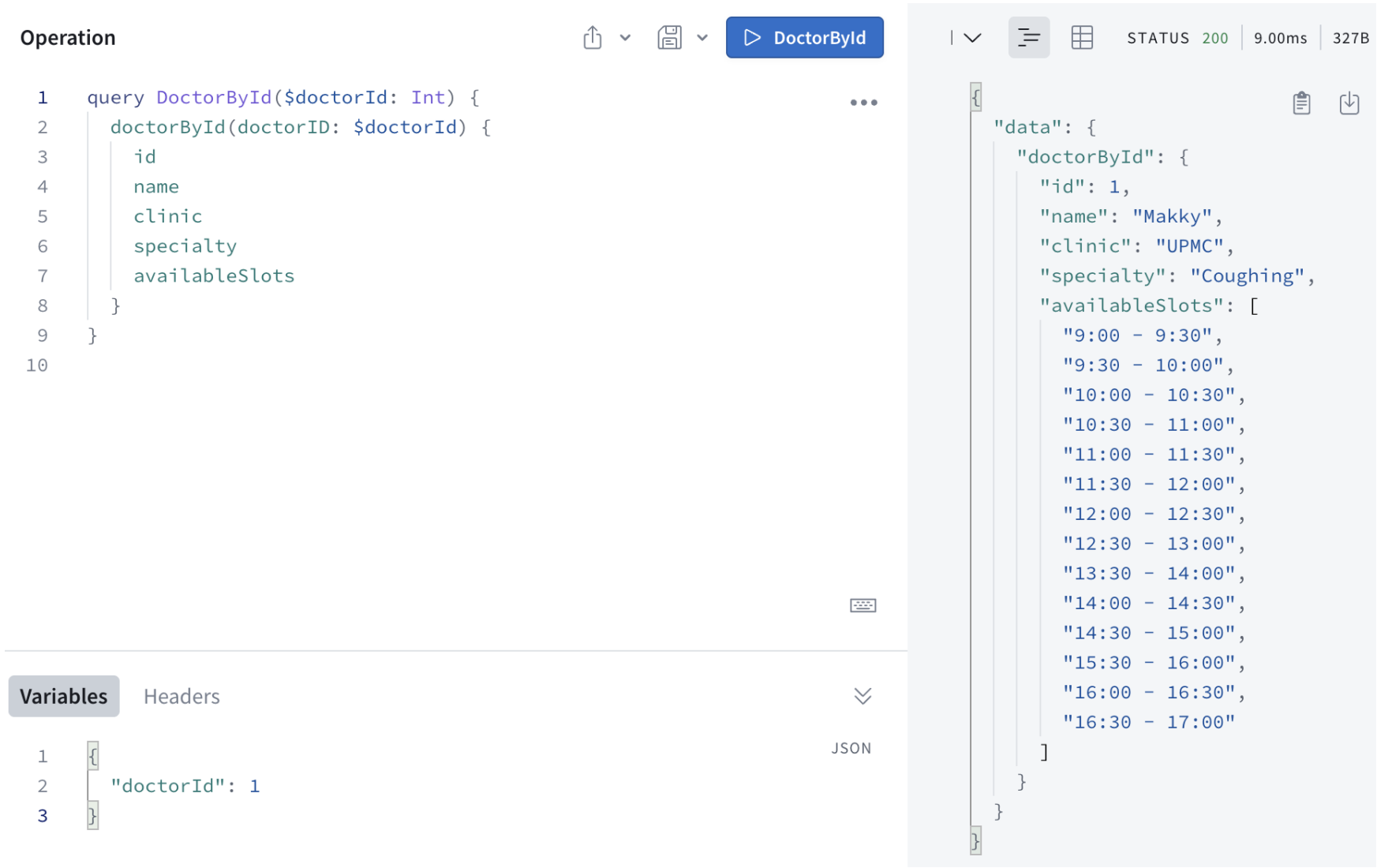
AppointmentByID Valid



AppointmentByID Invalid



DoctorByID Valid



DoctorByID Invalid

Operation

DoctorById

```
query DoctorById($doctorId: Int) {
  doctorById(doctorID: $doctorId) {
    id
    name
    clinic
    specialty
    availableSlots
  }
}
```

Variables Headers

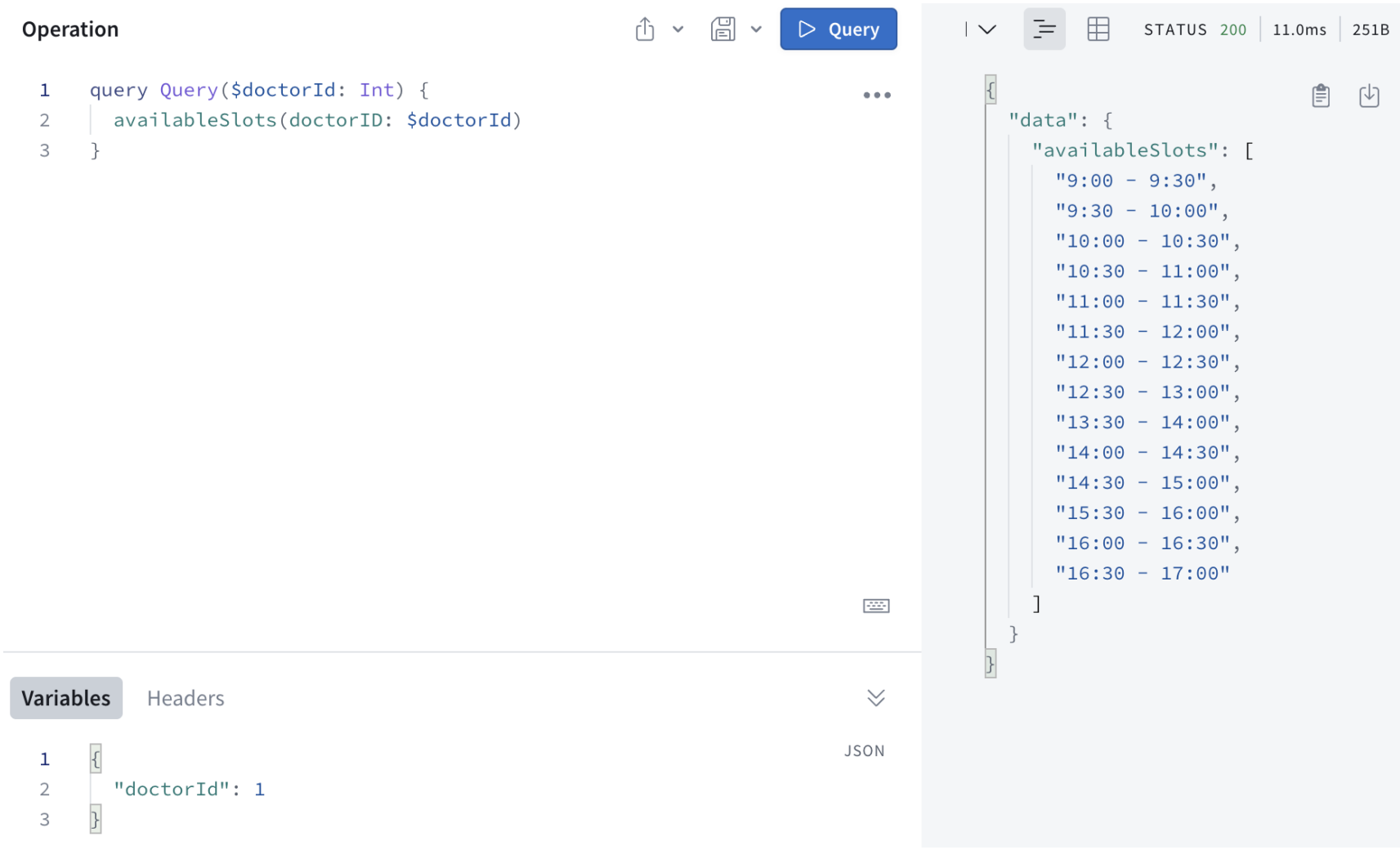
JSON

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

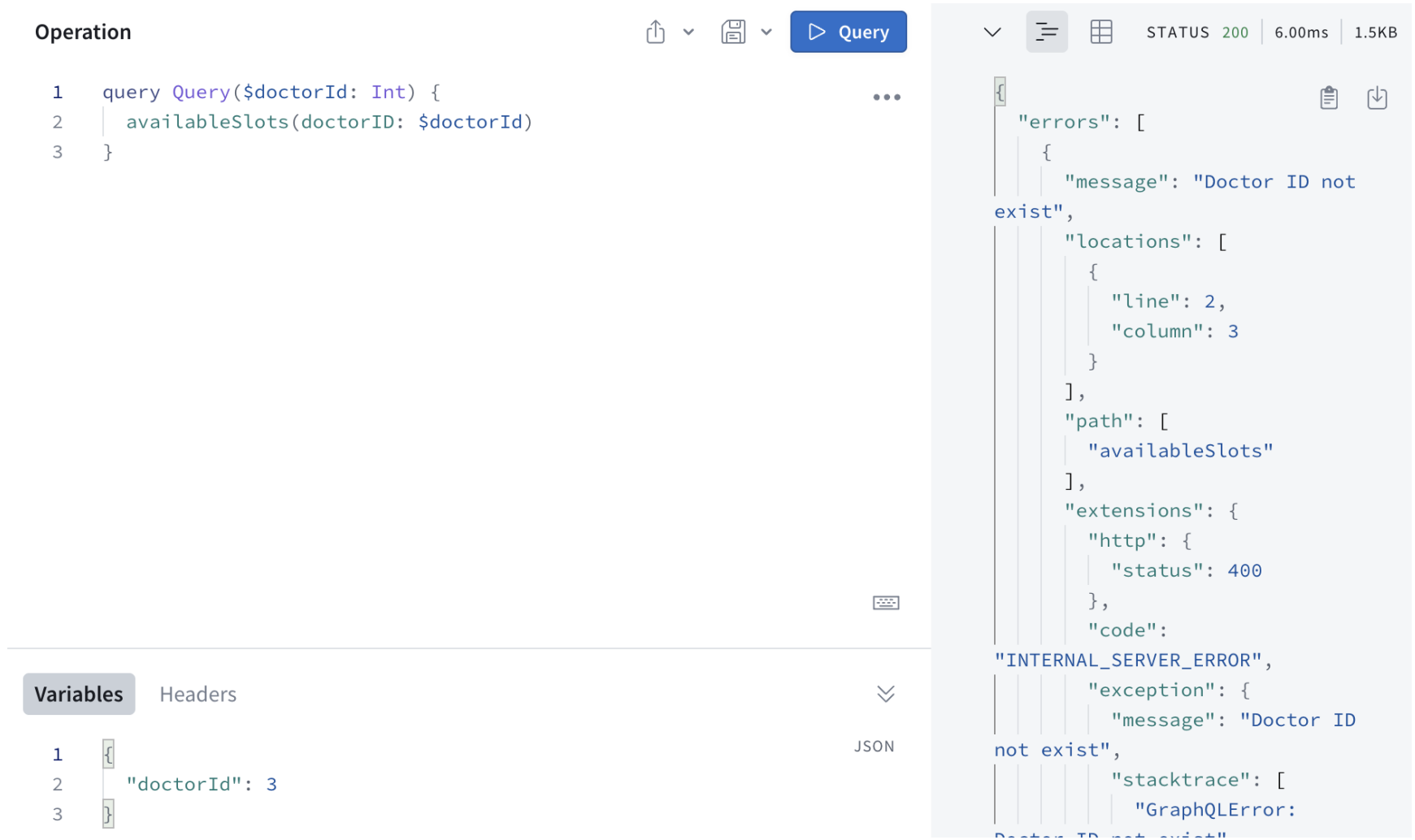
STATUS 200 | 11.0ms | 1.5KB


```
{
  "errors": [
    {
      "message": "Doctor ID not exist",
      "locations": [
        {
          "line": 2,
          "column": 3
        }
      ],
      "path": [
        "doctorById"
      ],
      "extensions": {
        "http": {
          "status": 400
        },
        "code": "INTERNAL_SERVER_ERROR",
        "exception": {
          "message": "Doctor ID not exist",
          "stacktrace": [
            "GraphQLError:
```

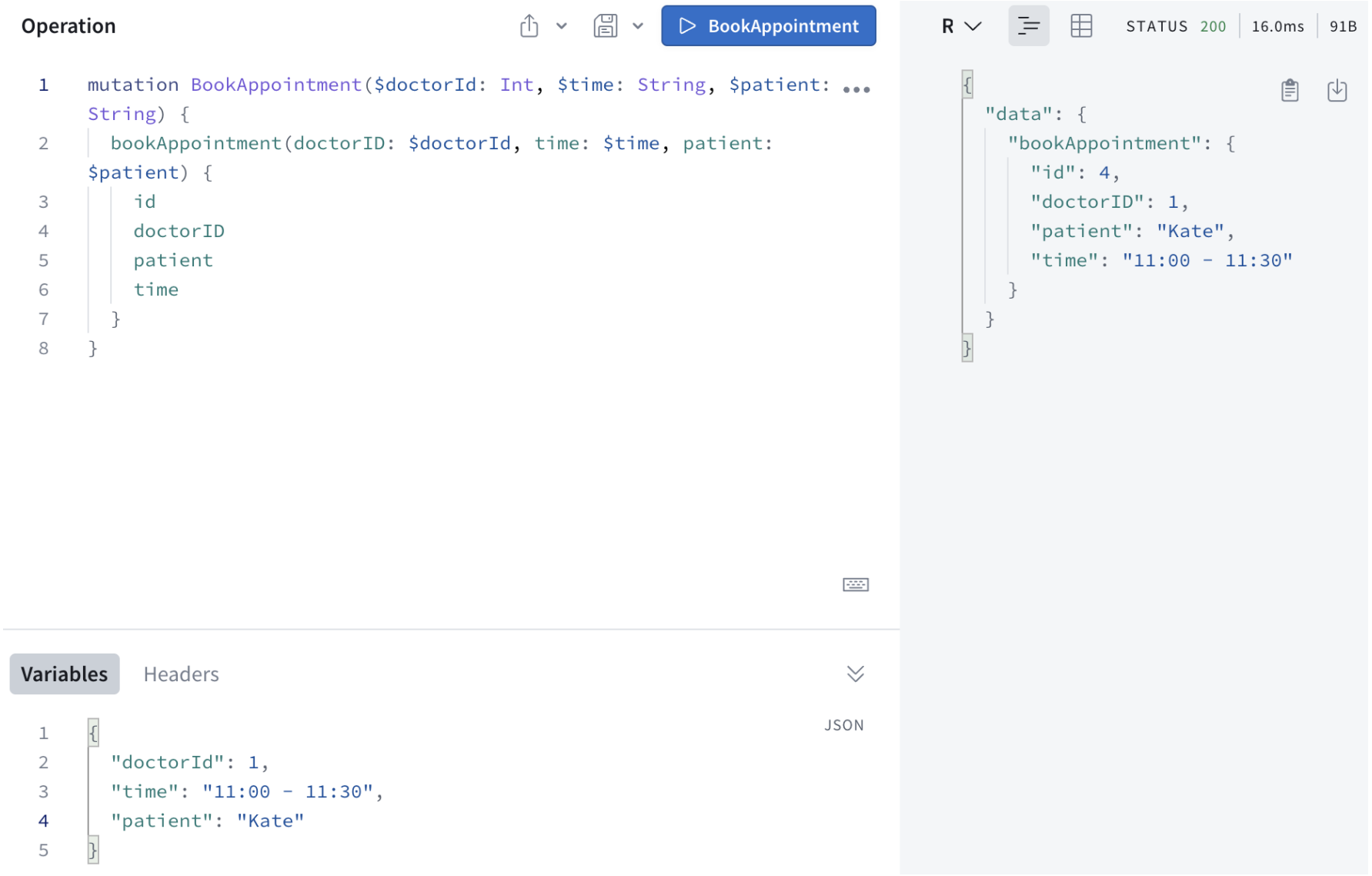
Available Slots Valid



Available Slots Invalid



Book Appointment Valid



Book Appointment Invalid

Operation

BookAppointment

```
mutation BookAppointment($doctorId: Int, $time: String, $patient: String) {
  bookAppointment(doctorID: $doctorId, time: $time, patient: $patient) {
    id
    doctorID
    patient
    time
  }
}
```

Variables Headers

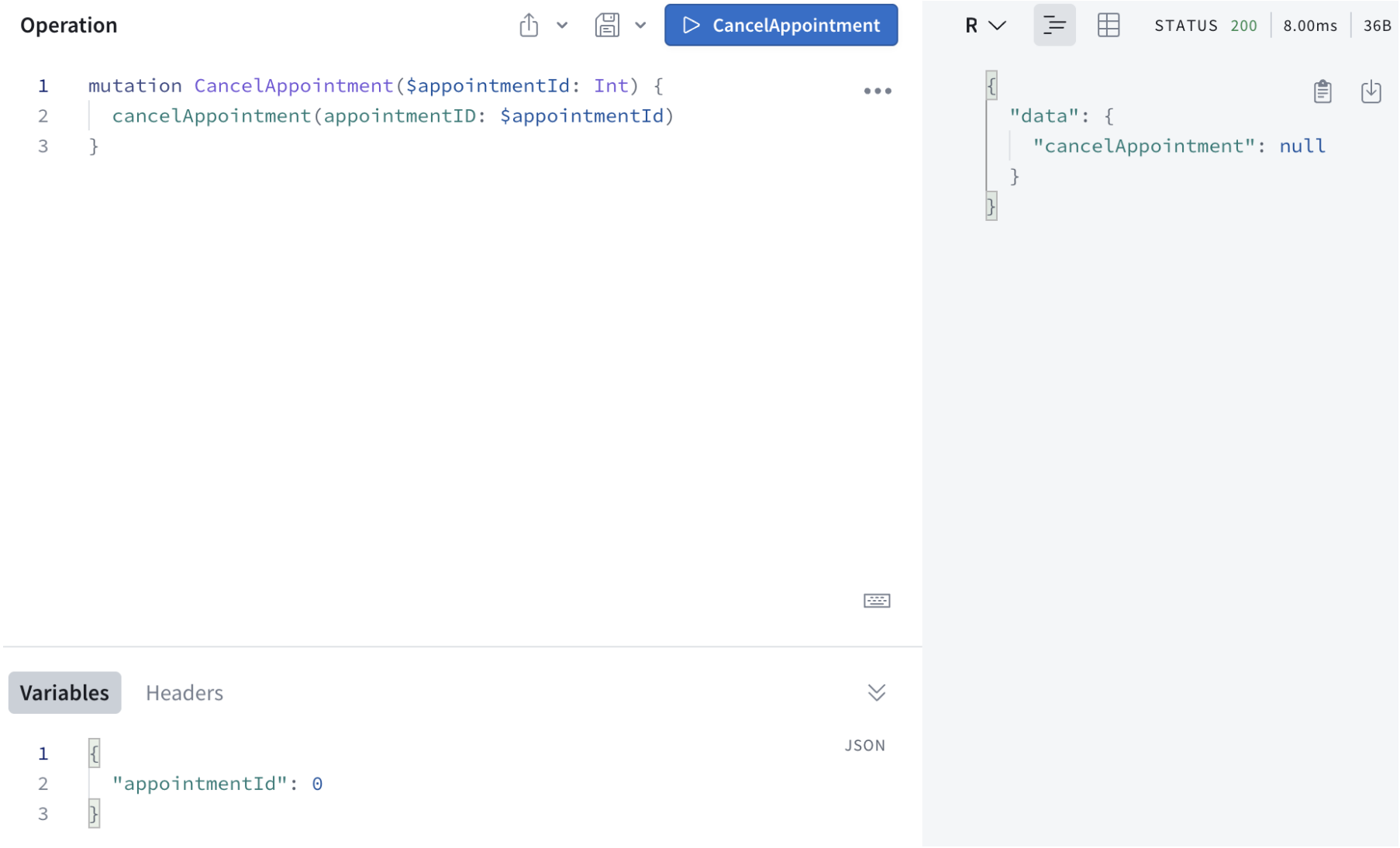
JSON

```
{
  "doctorId": 1,
  "time": "15:00 - 15:30",
  "patient": "Kate"
}
```

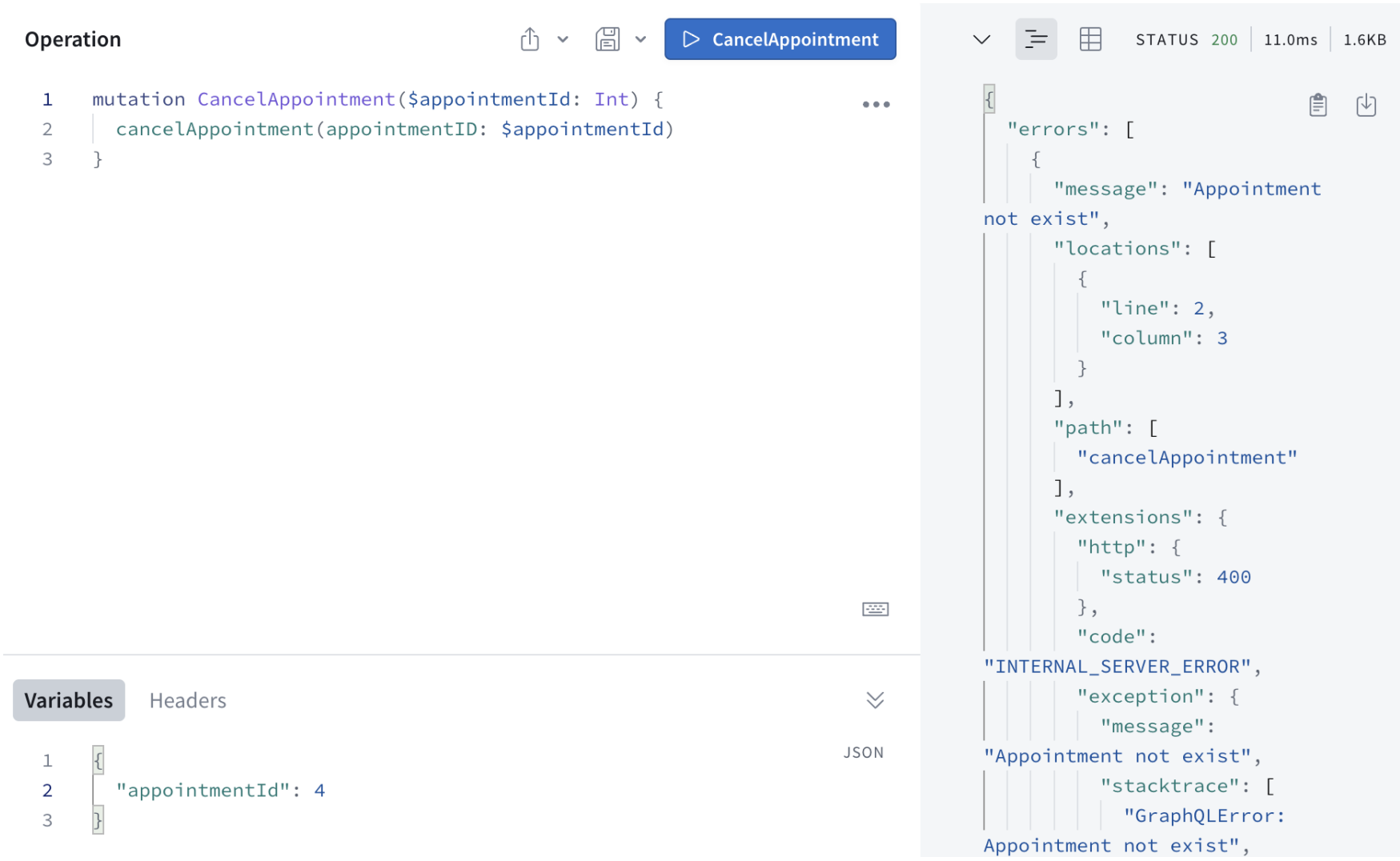
STATUS 200 | 8.00ms | 1.6KB


```
{
  "errors": [
    {
      "message": "Unavailable time slot",
      "locations": [
        {
          "line": 2,
          "column": 3
        }
      ],
      "path": [
        "bookAppointment"
      ],
      "extensions": {
        "http": {
          "status": 400
        },
        "code": "INTERNAL_SERVER_ERROR",
        "exception": {
          "message": "Unavailable time slot",
          "stacktrace": [
            "GraphQLError: Unavailable time slot",
            "    at ErrorMsg (/
```

Cancel Appointment Valid



Cancel Appointment Invalid



Update Appointment Valid

Operation

UpdateAppointment

```
mutation UpdateAppointment($appointmentId: Int, $patient: String) {
  updateAppointment(appointmentID: $appointmentId, patient: $patient) {
    id
    doctorID
    patient
    time
  }
}
```

R

STATUS 200 | 6.00ms | 95B


```
{
  "data": {
    "updateAppointment": {
      "id": 1,
      "doctorID": 1,
      "patient": "Taylor",
      "time": "15:00 - 15:30"
    }
  }
}
```


Variables Headers

JSON

```
{
  "appointmentId": 1,
  "patient": "Taylor"
}
```

Update Appointment Invalid

Operation

UpdateAppointment

```
1  mutation UpdateAppointment($appointmentId: Int, $patient: String) {
2    updateAppointment(appointmentID: $appointmentId, patient:
   $patient) {
3      id
4      doctorID
5      patient
6      time
7    }
8  }
```

Variables Headers

JSON

```
1  {
2    "appointmentId": 2,
3    "patient": "David"
4  }
```

STATUS 200 | 19.0ms | 1.6KB


```
{
  "errors": [
    {
      "message": "Patient not
updated",
      "locations": [
        {
          "line": 2,
          "column": 3
        }
      ],
      "path": [
        "updateAppointment"
      ],
      "extensions": {
        "http": {
          "status": 400
        },
        "code":
"INTERNAL_SERVER_ERROR",
        "exception": {
          "message": "Patient
not updated",
          "stacktrace": [
            "GraphQLError:
Patient not updated",
            "    at ErrorMsg (/
```

Reflection:

1. What were some of the alternative schema and query design options you considered? Why did you choose the selected options?
   One alternative schema is to remove the available slots field from Doctor. However, in that case, the query available slot needs to go through all of the appointments to filter out the time slots already taken for that doctor, which would be time consuming if there are a bunch of appointments.
   Another alternative schema is for the time slots, which could also use the datetime data type. However, since the time slots are fixed intervals of half an hour, using datetime type also costs trouble in expressing an interval. Thus, String is used to express time slots.

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.
   1. What changes will the clients (API consumer) need to make to their existing queries (if any).
      Clients need to include more fields to acquire more information, such as adding consultation type in their query when doing an appointment by ID. Besides, the clients should also select which fields of patient, namely patient details they want to know by adding it in query as below:

      ```
      appointmentById(1){
          doctorID
          patient {
              name
              height
              weight
          }
          time
      }
      ```

   2. How will you accommodate the changes in your existing Schema and Query types?
      Need to include more details in the fields such as consultation type (first time/follow-up etc.) and also to add reference to patient details, we can change the type of field patient from String to Patient as below:

      ```
      type Appointment {
          id: Int!
      ```

```
    doctorID: Int!
    patient: Patient
    time: String
    type: String
    ...
}
```

The query return types need not to be modified because the return type is just Appointment which itself will contain changes made to ‘Event’. However, the type of their input parameters, namely patient, should be changed from String to PatientID to incorporate the change of schema.

3. Describe two GraphQL best practices that you have incorporated in your API design.
    1. Queries doctorbyID and appointmentbyID are added though not explicitly required by spec. In real life practice, searching information of objects by index would be convenient and useful. In addition, when more elements are added to the schema and queries, these two functions could also be used by other queries to reduce redundant code.
    2. When there is an error occurring, the resolver would throw an error with a message and a return status code included in the extensions to indicate the problem of input. To reduce code redundancy, this part is also wrapped as a function to be used multiple times as below:

```
const ErrorMsg = (message) => {
    throw new GraphQLError(
        message,
        { extensions: { http: { status: 400 }}});
}
```