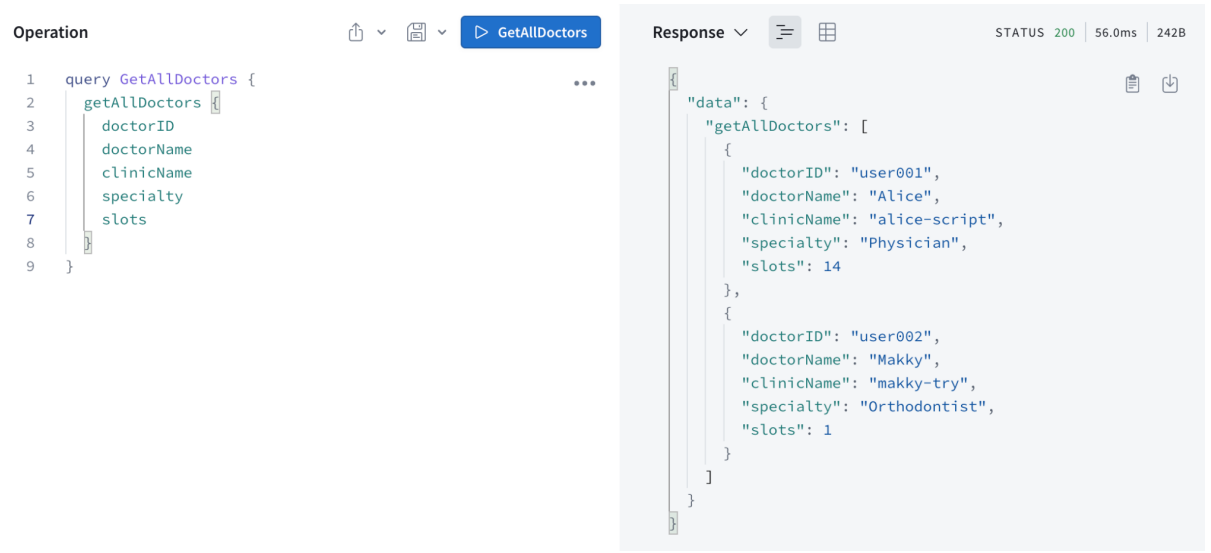
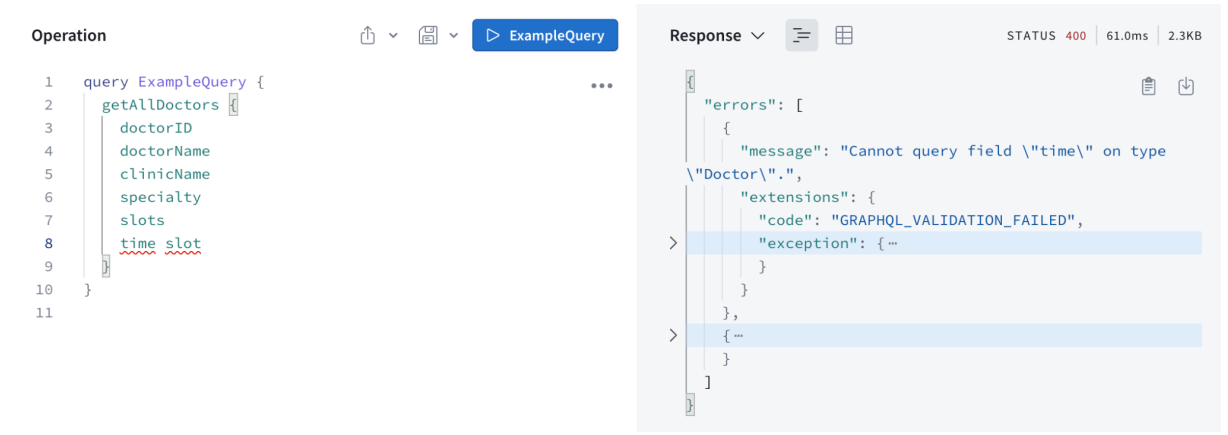
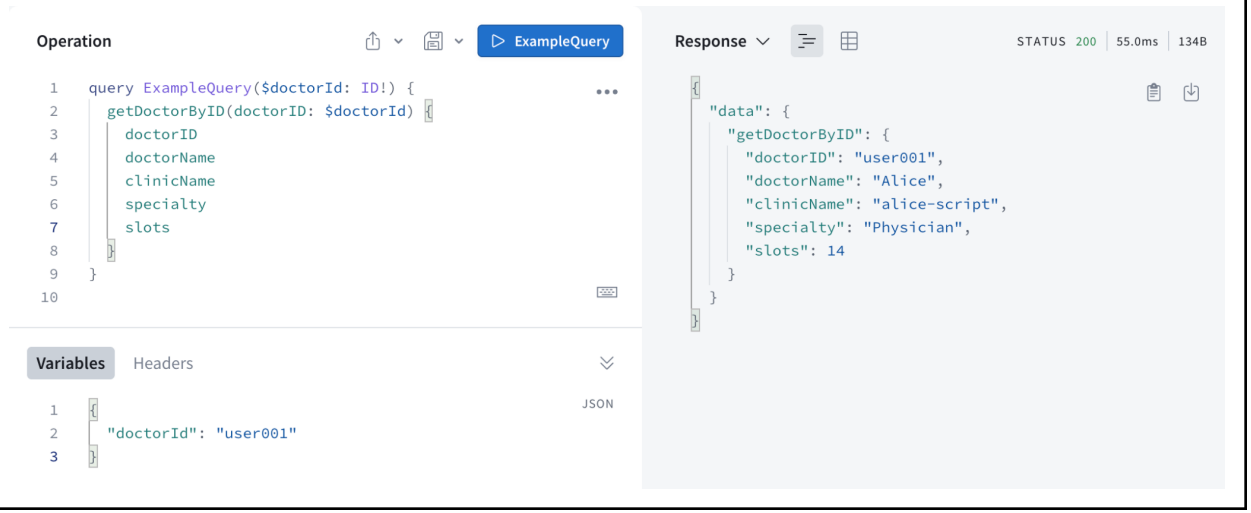
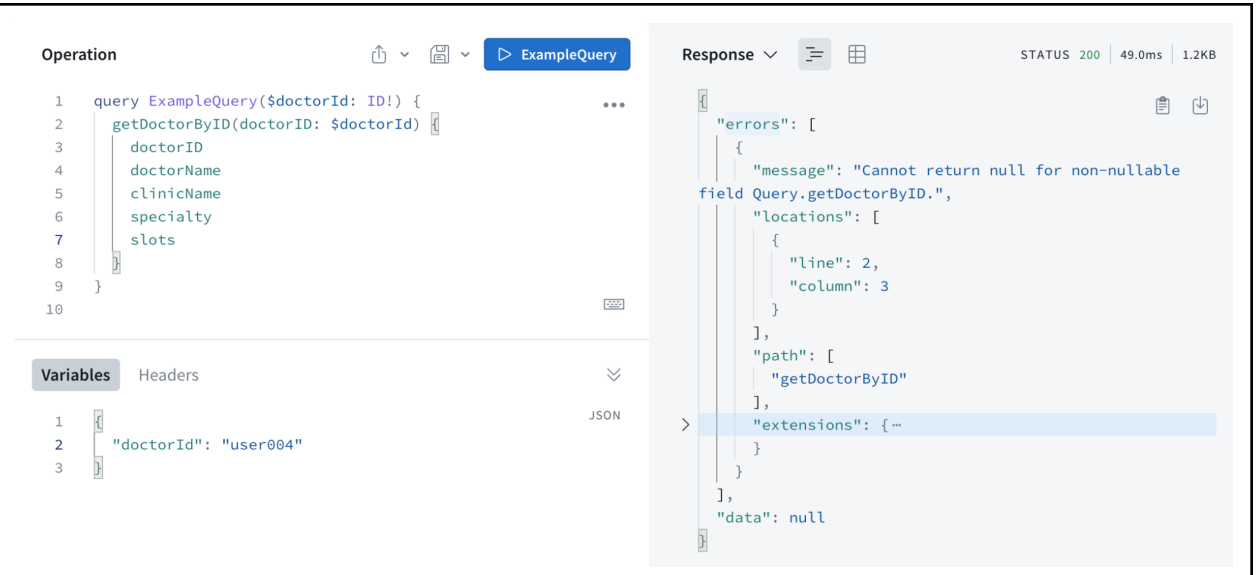
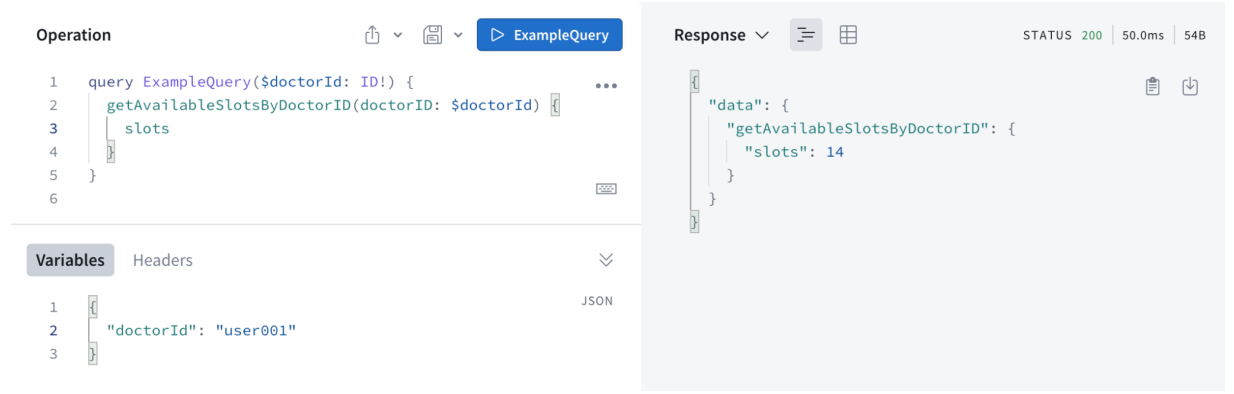
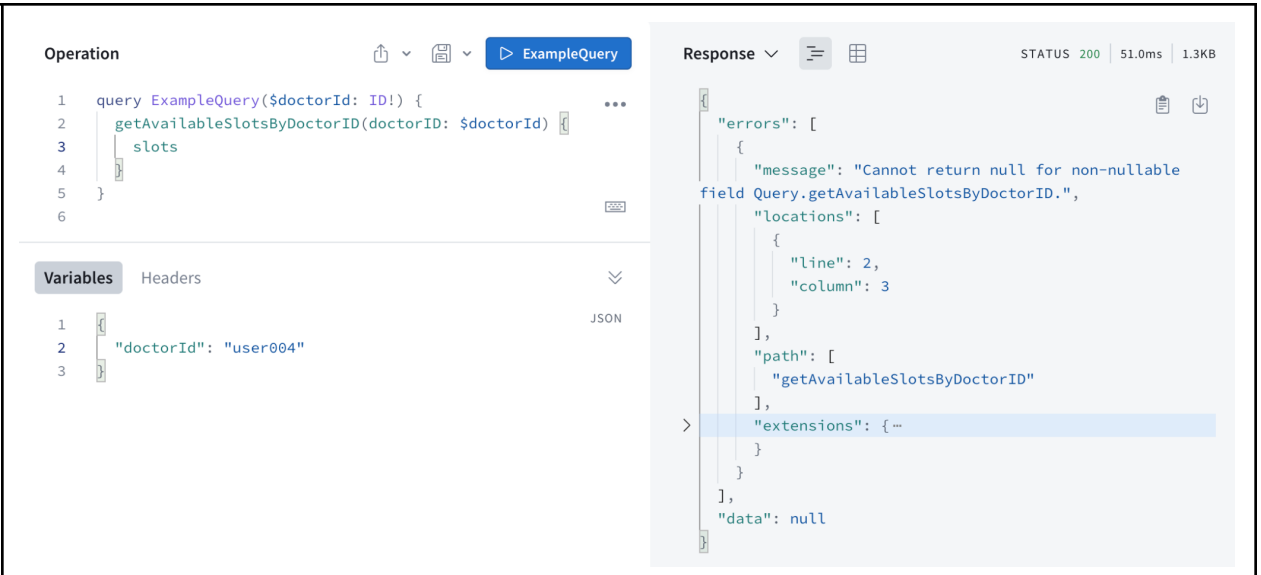
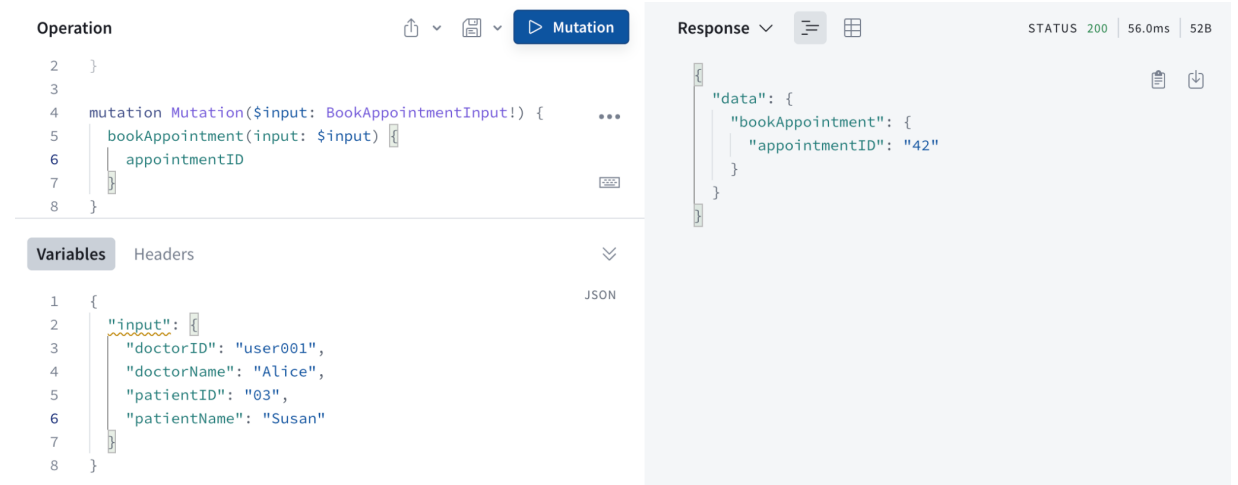
Assignment A2.2
GraphQL API

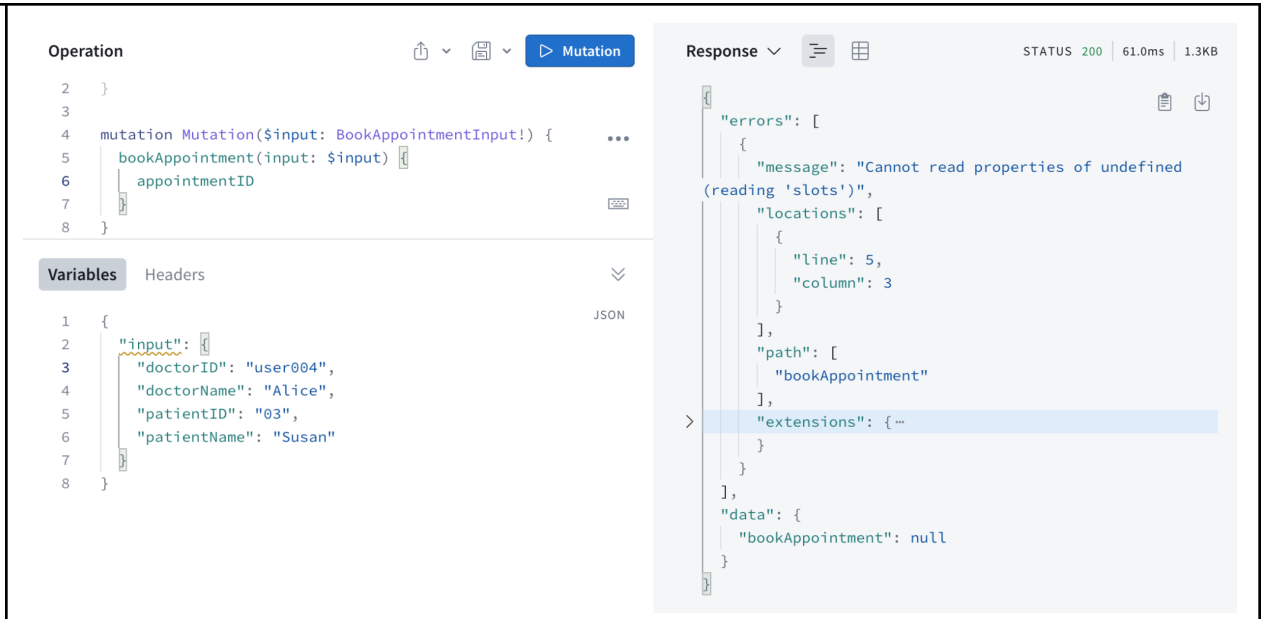
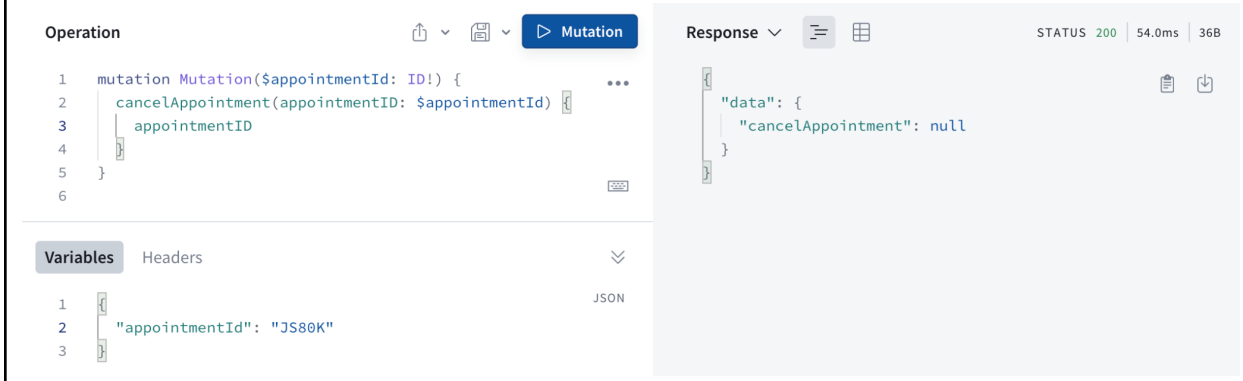
1. Testing

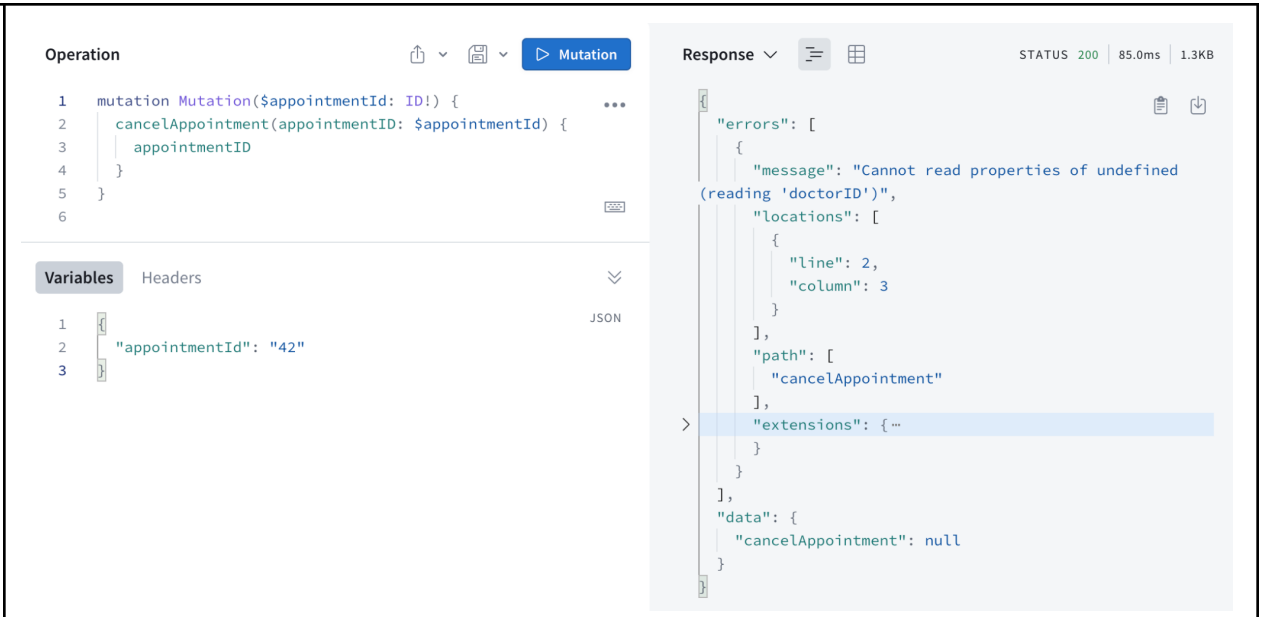
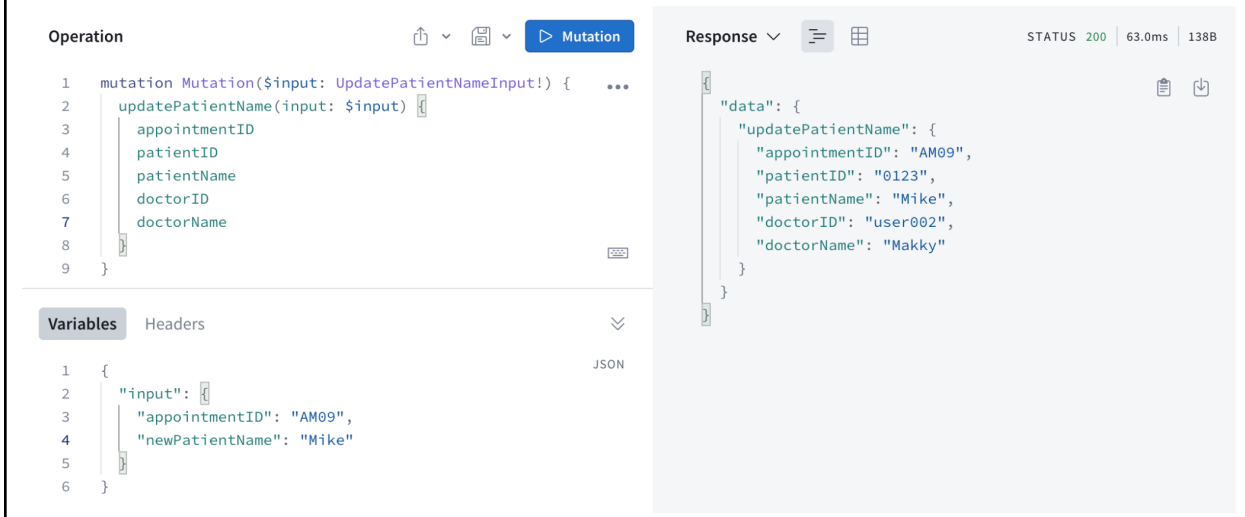
Test Case Identifier	Test Case Description	Screenshot
GetAllDoctors Success	A list of all doctors is requested while fetching valid fields	

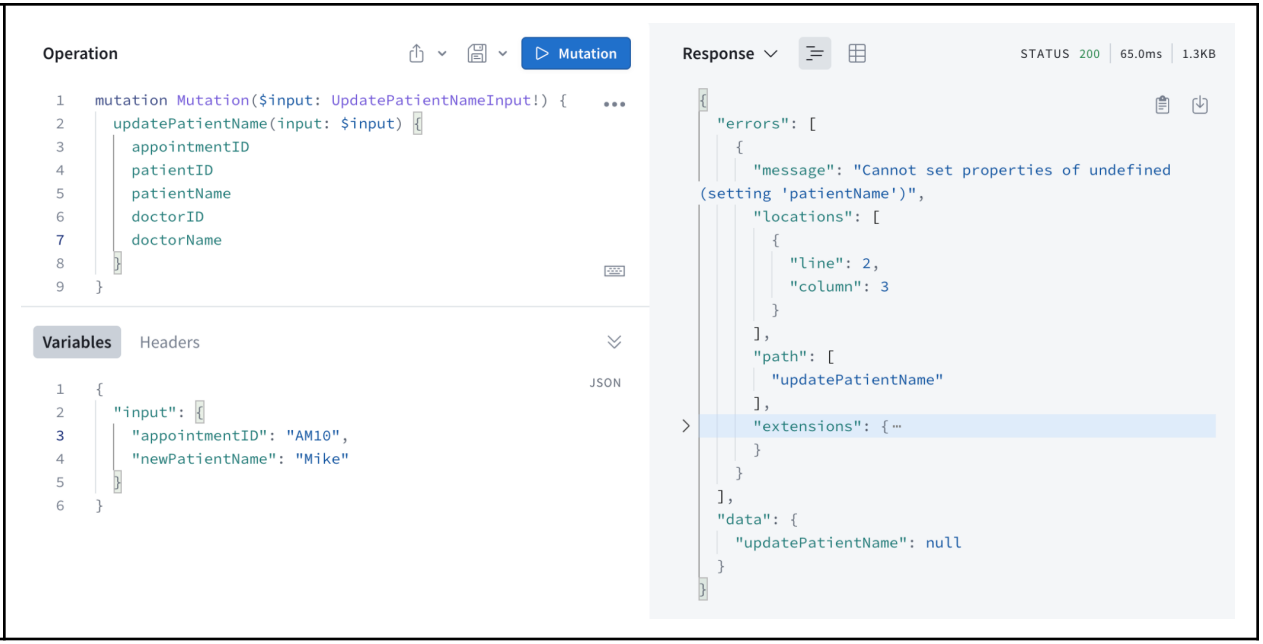
GetAllDoctors Fail	A list of all doctors is requested while fetching an invalid field	
GetDoctorByIDSuccess	A valid doctor ID is requested	

GetDoctorByIDFail	An invalid doctor ID is requested	
GetAvailableSlotsByDoctorIDSuccess	A valid doctor ID is requested	

GetAvailableSlotsByDoctorIDFail	An invalid doctor ID is requested	
BookAppointmentSuccess	A valid doctor ID and patient ID are entered	

BookAppointmentFail	An invalid doctor ID is entered	
CancelAppointmentSuccess	A valid appointment ID is entered	

CancelAppointmentFail	An invalid appointment ID is entered	
UpdatePatientNameSuccess	A valid appointment ID is requested	

UpdatePatientNameFail	An invalid appointment ID is requested	

2. Reflection

- **What were some of the alternative schema and query design options you considered? Why did you choose the selected options?**
  An alternative schema design I considered had only two resources: doctor and appointment. This would have made meeting the requirements easier to implement. However, to make the system more scalable and comprehensive, I decided to add another resource called patient.

  It also allowed me to keep track of all the patients in the record while also allowing me to validate the patient ID when creating new appointments. As the system grows, this new feature will allow it to keep track of patients' medical histories.

  Additionally, while designing the mutation for booking appointments, I considered returning a specific time slot to the patient. But, in order to keep things simple, and because we only have 16 possible time slots, I decided to keep track of available time slots for each doctor and return appointment IDs to patients rather than specific time slots. The number of available time slots is kept between 0 and 16, and if all of the slots for a doctor are taken, no more appointments can be scheduled.

  This can be extended to include specific time slots, but for the time being, it maintains a count of available slots, which is incremented when a new appointment is created for the specific doctor and decremented when the appointment is canceled.

- **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).**
    In order to incorporate these changes into the existing system, minor changes at the server side will allow clients to access this data conveniently. The client's end would only need to request the new information.

  - **How will you accommodate the changes in your existing Query and Schema types?**
    Minor changes, such as the addition of fields representing the new requested information and the nesting of objects in the schema design to access this information, such as accessing patient details from the appointment, will be made at the server end.

    However, no changes to existing queries would be required when retrieving the entire set of objects. If the user wants to request specific information, minor changes to the process of retrieving these details will suffice.

- **Describe two GraphQL best practices that you have incorporated in your API design.**
  The two GraphQL best practices incorporated in my API design are:
  - Use of Non-nullable fields
    I used non-null type variations, which cause the server to return an error if a mandatory field is left blank. This helps to avoid miscalculations or erroneous outputs caused by incorrect field values, and it also makes the frontend application easier to design by eliminating the possibility of receiving invalid data from the backend.

  - Use of ID type
    To make accessing and validating objects easier, the type ID has been used to define unique identifiers. It improves the process of identifying and retrieving data from a large collection of data, reducing the complexity of performing the different mutations defined above.