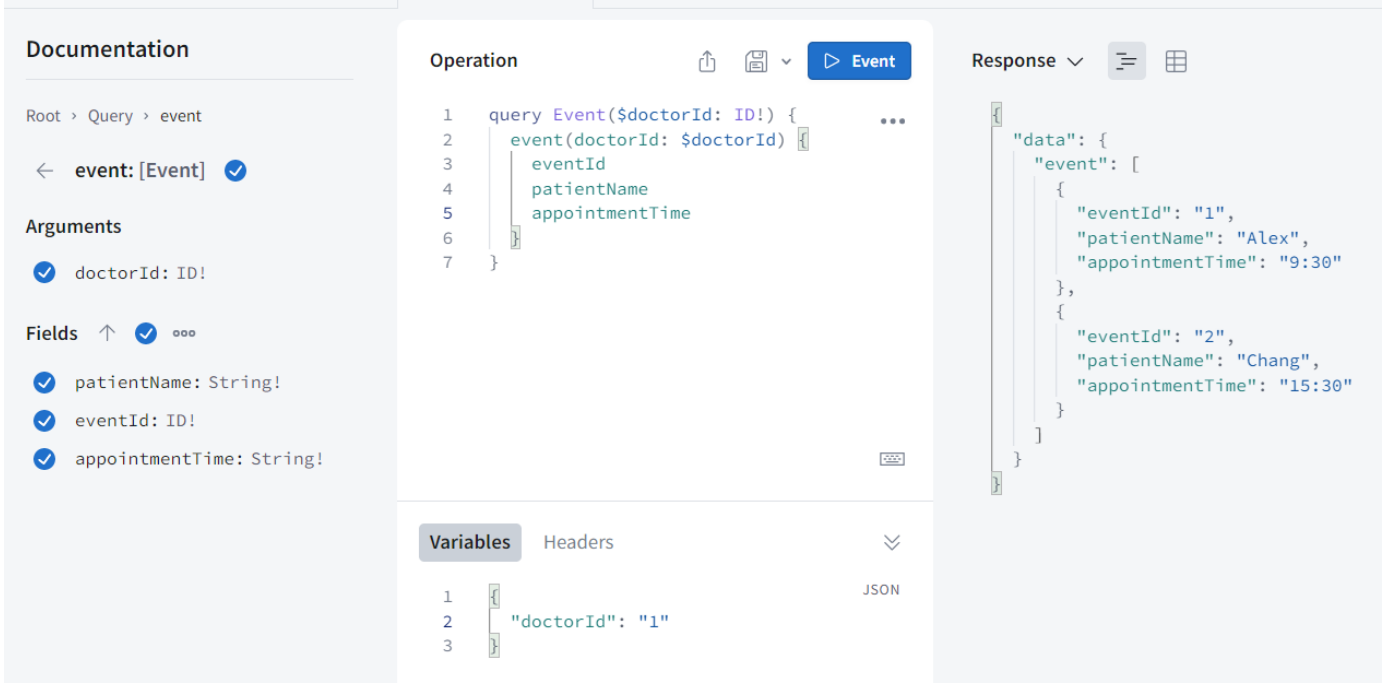
2.2 Testing subtask:

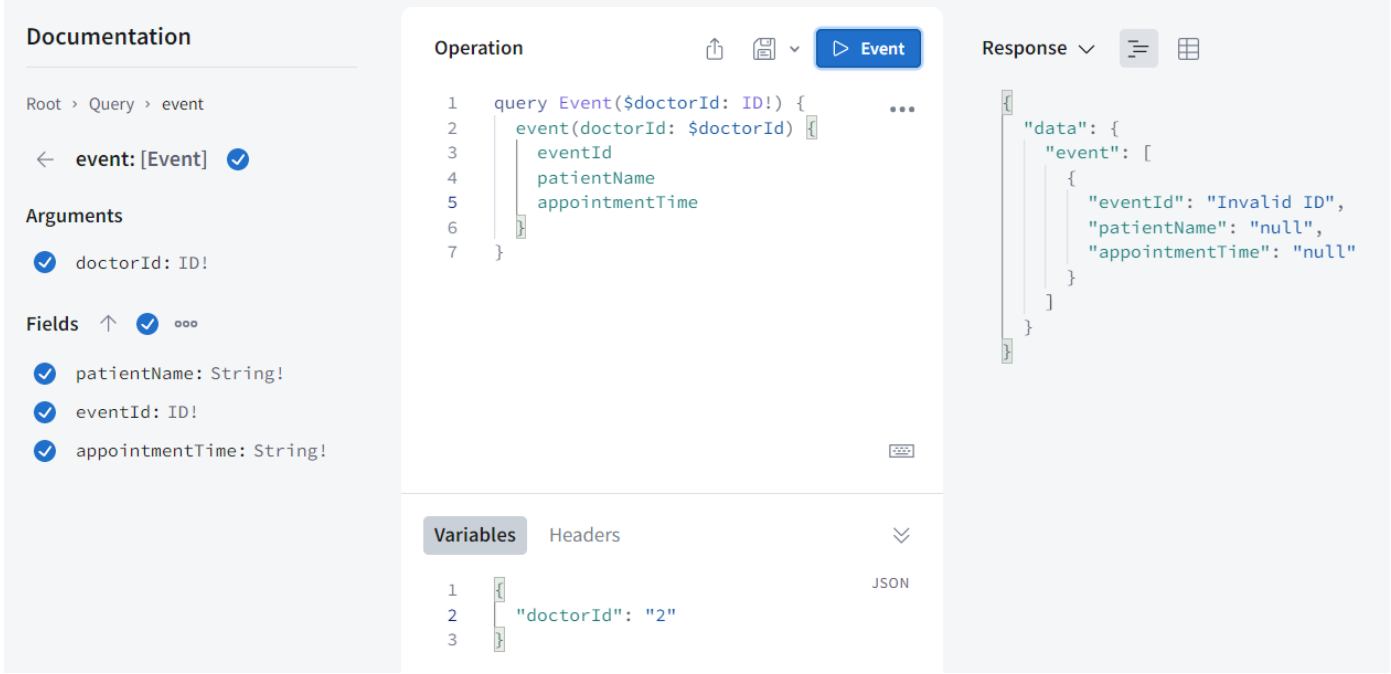
Testcase Id	Testcase Description	Inputs	Expected Output	Remarks	
Q1-H (happy path)	Check if for a given doctor ID, the right name, clinic he/she is at, and the specialty are well-displayed	Doctor ID	Doctor ID, doctor name, clinic name, and specialty, as Strings	Successfully return the right pieces of information. Consistent doctor ID for input and output.	

Q1-E (error condition)	Try to get info given a non-existing doctor ID	Doctor ID	Error message	Returns an error message	

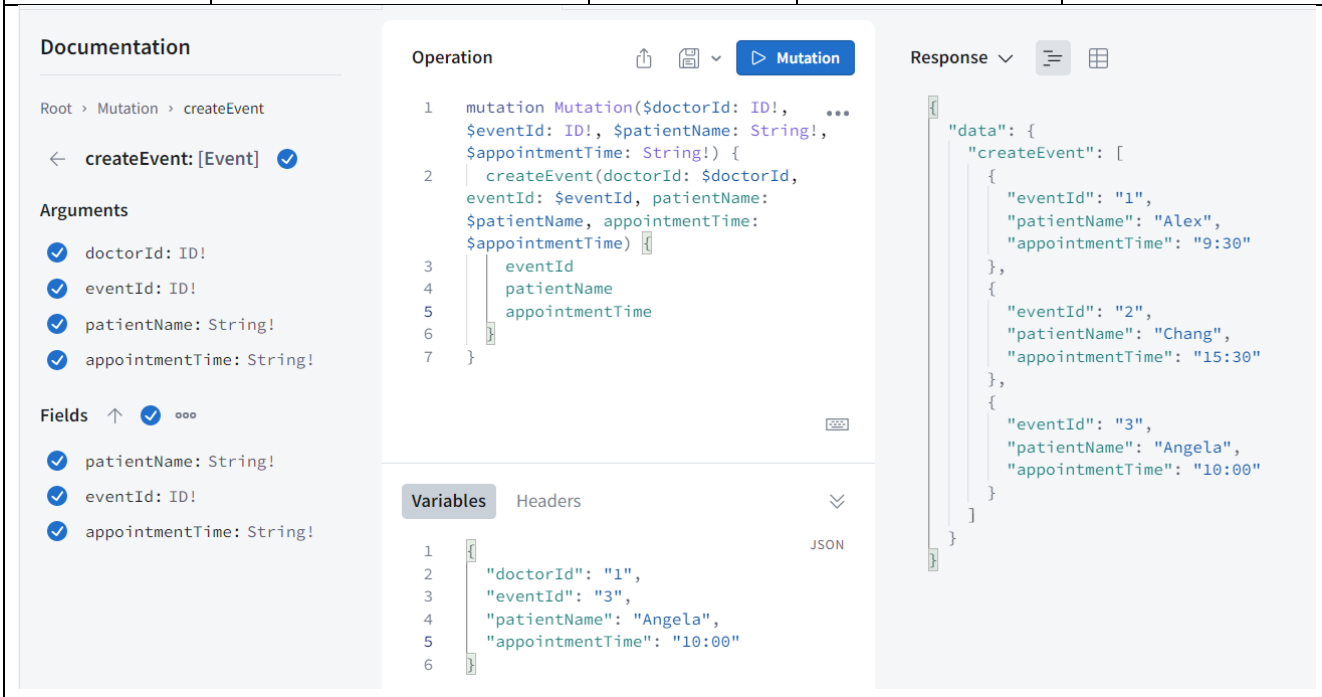
Q2-H	Check if for a given doctor ID, can return all the appointments related to this doctor	Doctor ID	Doctor name and a list of appointments linked to him/her	The available slots will be anytime from 9am to 5pm minus the stated appointments. Time should be distinct.



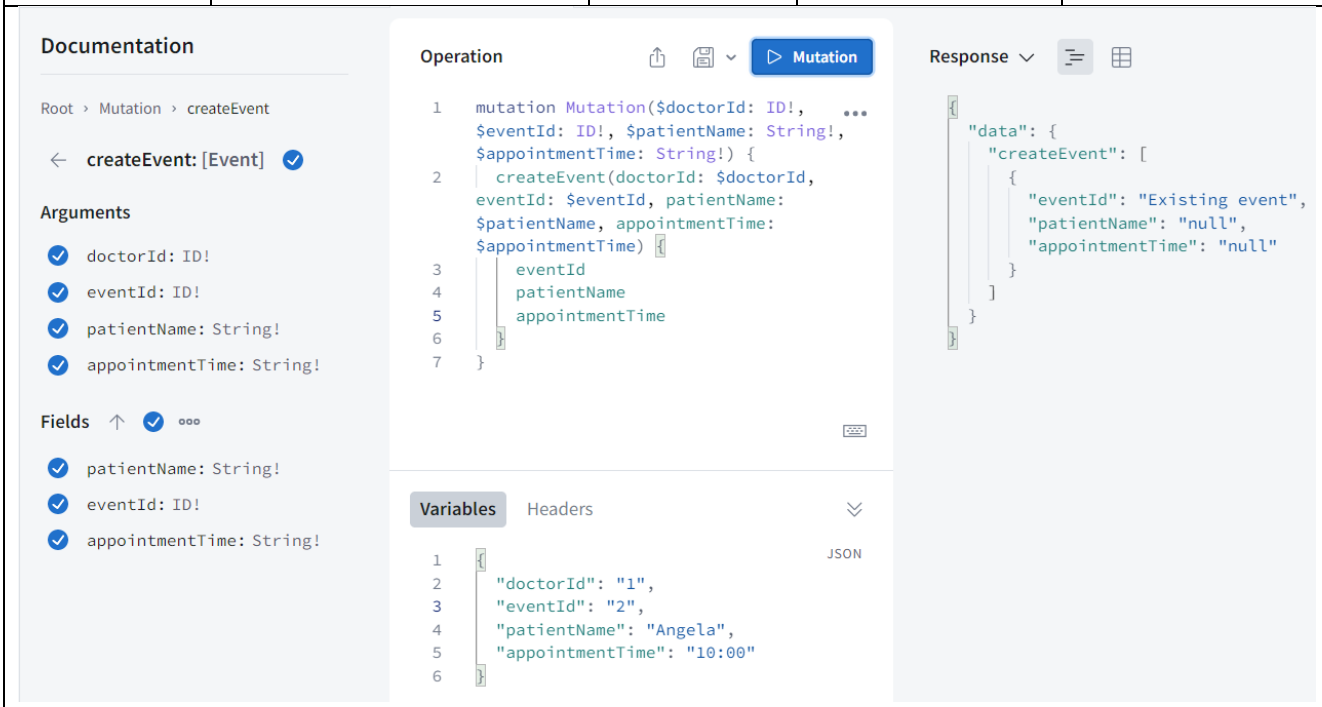
Q2-E	Try to get info given a non-existing doctor ID	Doctor ID	Error message	Returns an error message



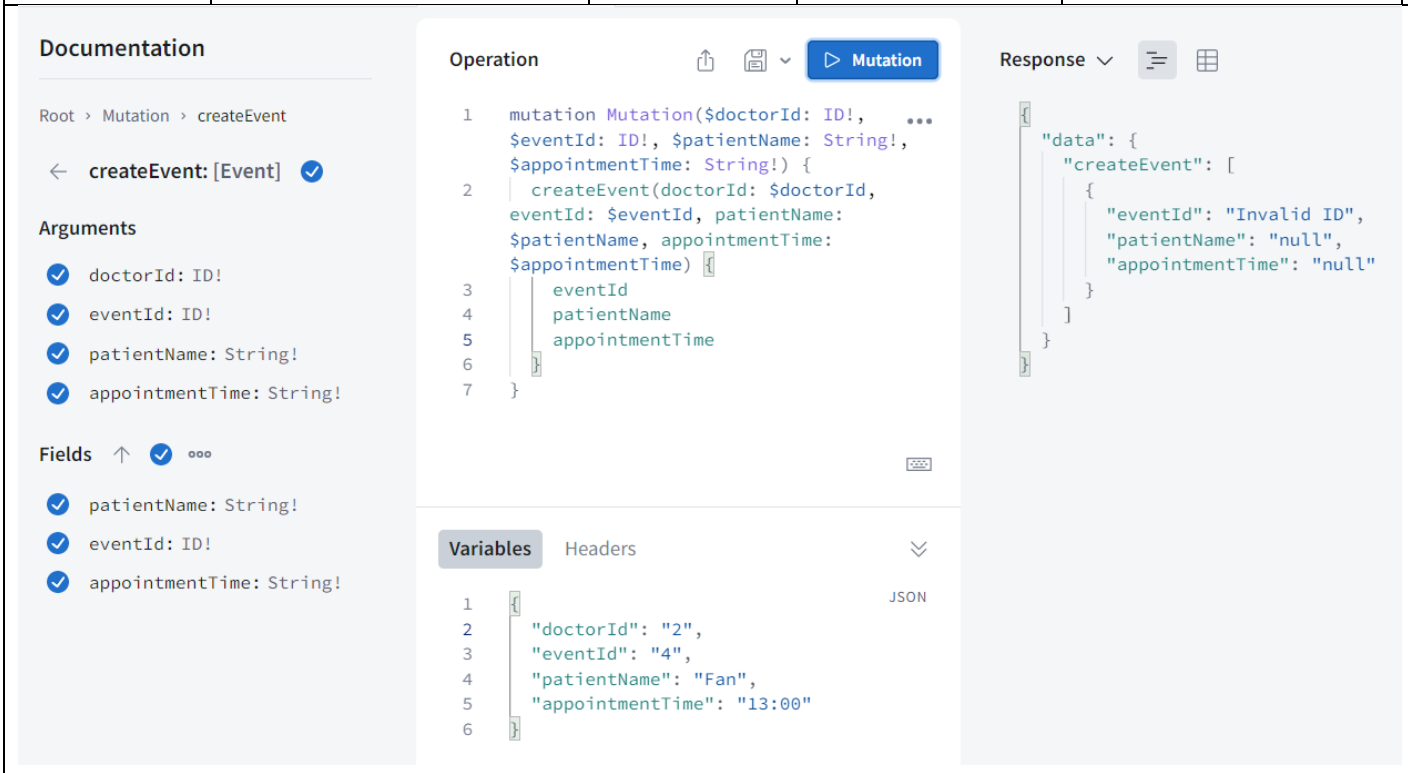
M3-H (mutation)	Create a new event and add to the list of appointments for a given doctor (via the doctor ID of the input)	Doctor ID, event ID, patient name, appointment time	List of appointments of the doctor	Returns the updated list of appointments. Make sure it adds to an available timeslot.	



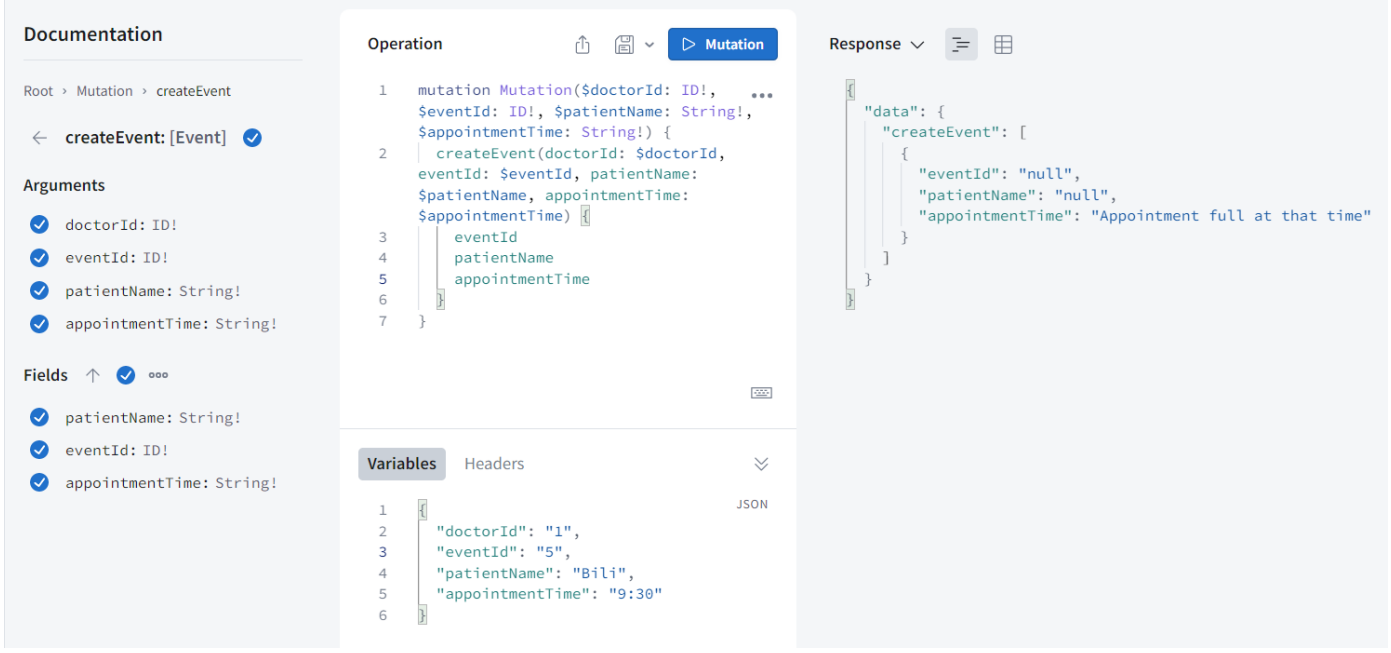
M3-E	Add an existing appointment to the list	Same as above	List of appointments of the doctor	Returns with error message with “Existing event”	



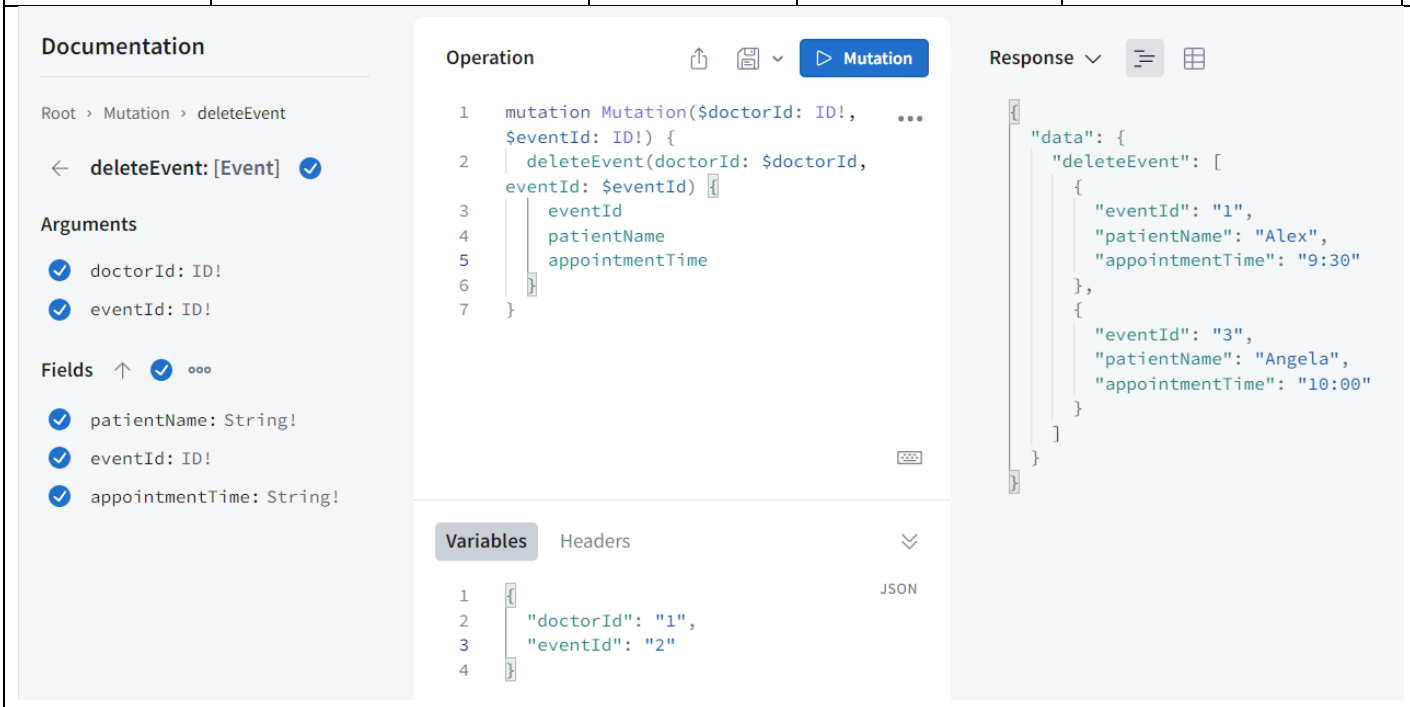
M3-E(2)	Add an appointment to a non-existing doctor	Same as above	List of appointments of the doctor	Returns with error message with “Invalid ID” for the doctor



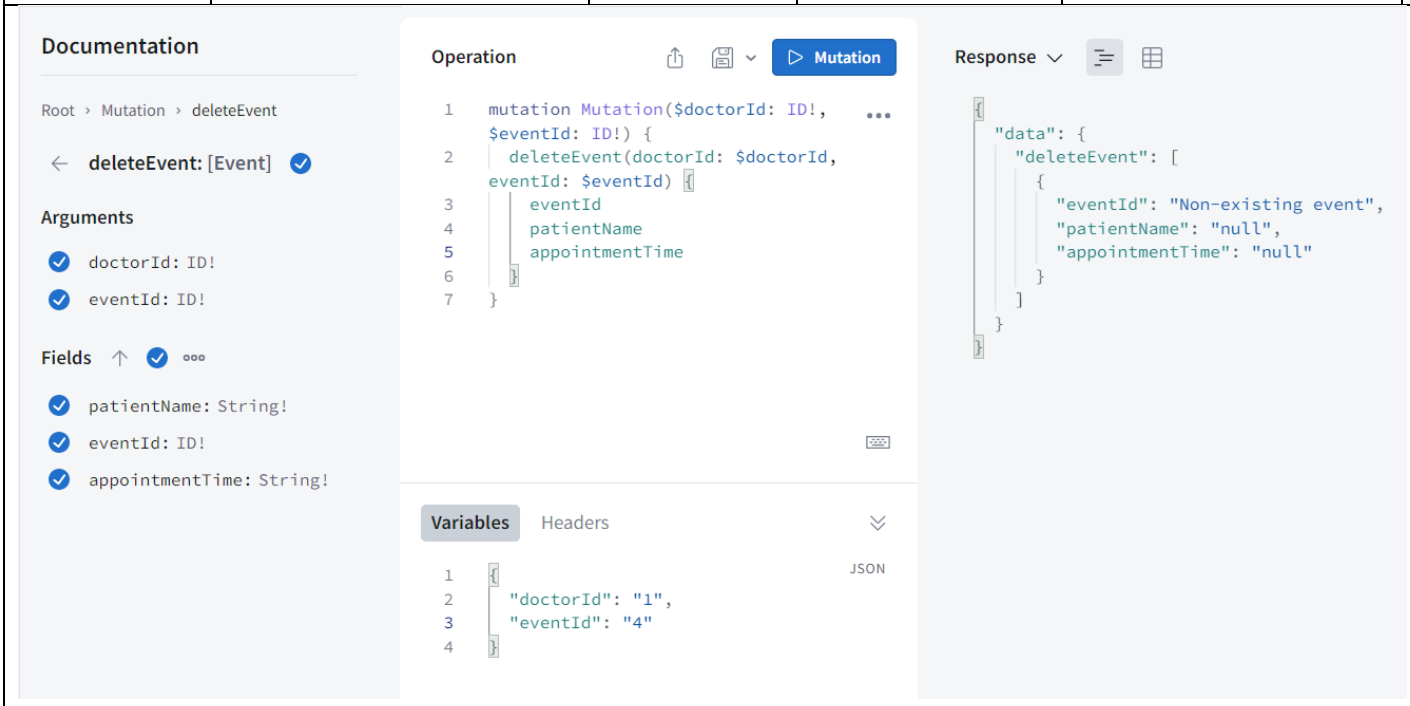
M3-E(3)	Add an appointment to an unavailable timeslot	Same as above	List of appointments of the doctor	Returns with error message with “appointment full” for that specific event time



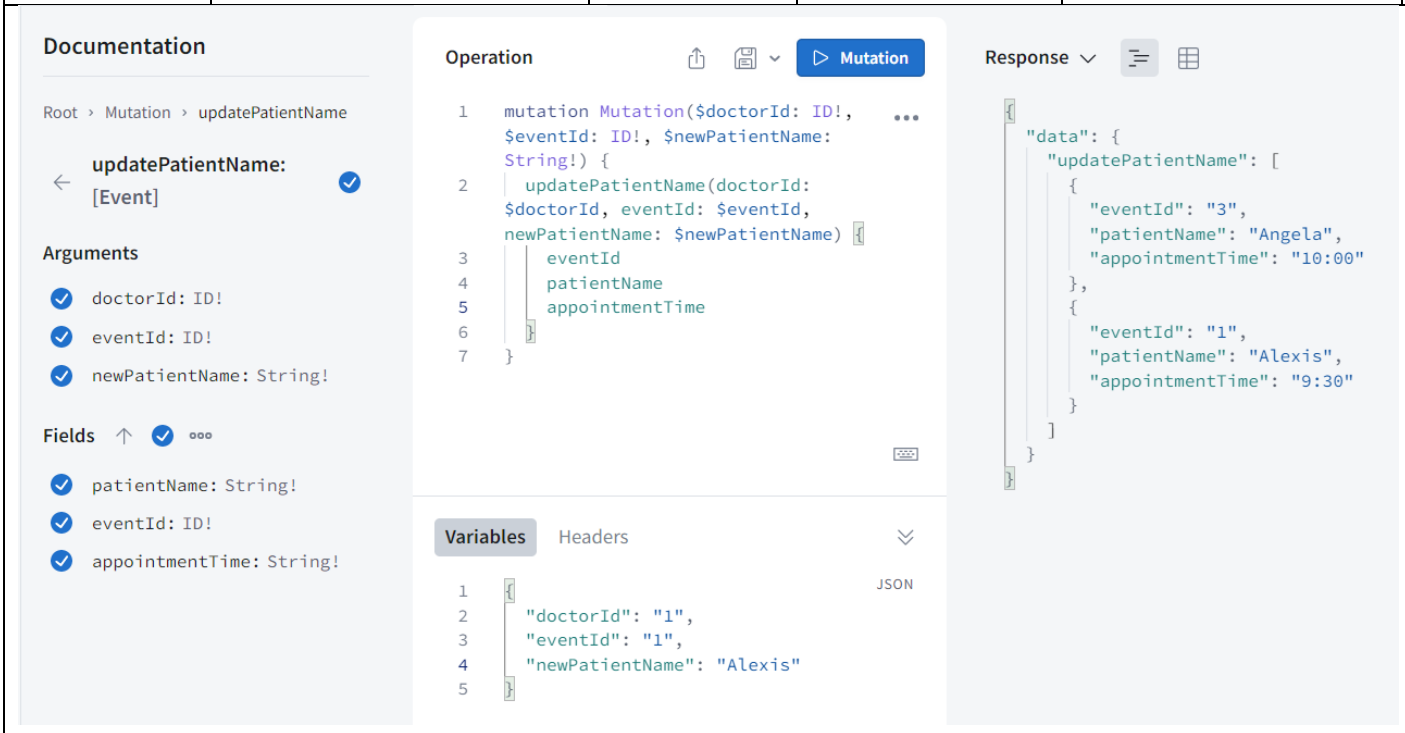
M4-H	Delete an existing event from the list of appointments	Event ID	List of appointments of the doctor	Returns the updated list of appointments for successfully deleted the event from the list



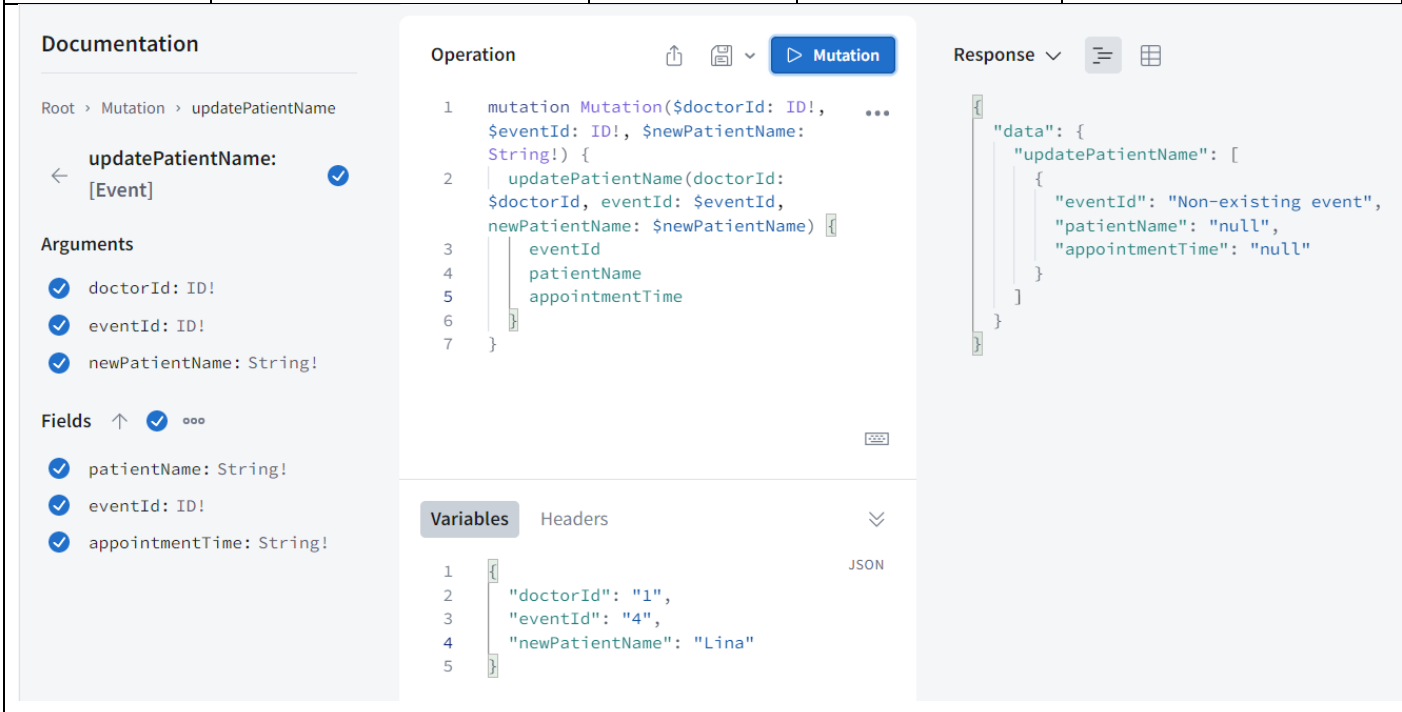
M4-E	Delete a non-existing appointment	Event ID	List of appointments of the doctor	Returns with error message saying the appointment to be deleted does not exist



M5-H	Update the patient's name for an event on the list	Event ID, patient name to be updated	List of appointments of the doctor	Returns true if successfully updated the event on the list



M5-E	Update a patient's name for a non-existing appointment	Event ID, patient name to be updated	List of appointments of the doctor	Returns false with an error message



3. Reflection task:

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

I initially wanted to make a separate schema for the available timeslots, containing tuple sets of available timeslots with beginning and end times for available times for the doctor to take appointments. However, this will make the implementation much more complicated, and also, confusing for the user. Therefore, considering building a user-friendly program, I decided to use a list of appointments instead, called [Event], inside the event schema, and each doctor has such a list. This is more visual, and when creating a new appointment, can simply iterate through the list to see if the doctor is free during that time. Given that the doctor works from 9 to 5, there are only a fixed number of timeslots – more precisely, 16 not considering lunch breaks. Therefore, this will very negligibly affect time complexity, but more visual, easier to use and easier to implement too, which is why I ended up choosing the option of having an [Event] list inside Doctor, containing the appointment time and patient info.

- **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).**

If the client does not need the new info, they can still use the API as in the past, without any further modifications. However, if they want to have more information on the new fields, will simply need to request for more fields inside the 'variables'. However, in terms of the inputs, when doing a query, if we do not request for more inputs, there will be no action needed from the clients' side.

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

In terms of the Schema, I will simply add more fields inside the 'Event' structure, and make sure the types are correctly stated, as well as non-null. For the Query, there will be more fields to be returned, but in my implementation, I made it such that given the doctor ID, it finds all events related to him/her, it will output all fields that I defined in my Schema, so nothing to change in there. However, for the returned error message, I will need to add more fields because it has to have the same structure – same number of fields – as inside the Schema.

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

1. When designing the Schemas, I made sure the mandatory fields take non-null types and when referring to IDs, I used the type 'ID' to make a clear distinction between ids and normal integers.
2. When designing queries, I used self-explanatory queries and avoided using names with 'get' to start with, because Queries are always used to get info on fields. Also, I separated by queries to avoid having queries doing more than one thing. For example, I separated queries to get events and get doctor info even if they all take 'Doctor Id' as input and the events are inside doctor info. I wanted to differentiate the two to make the API cleaner. Also, when naming the mutations, I made sure to use verbs