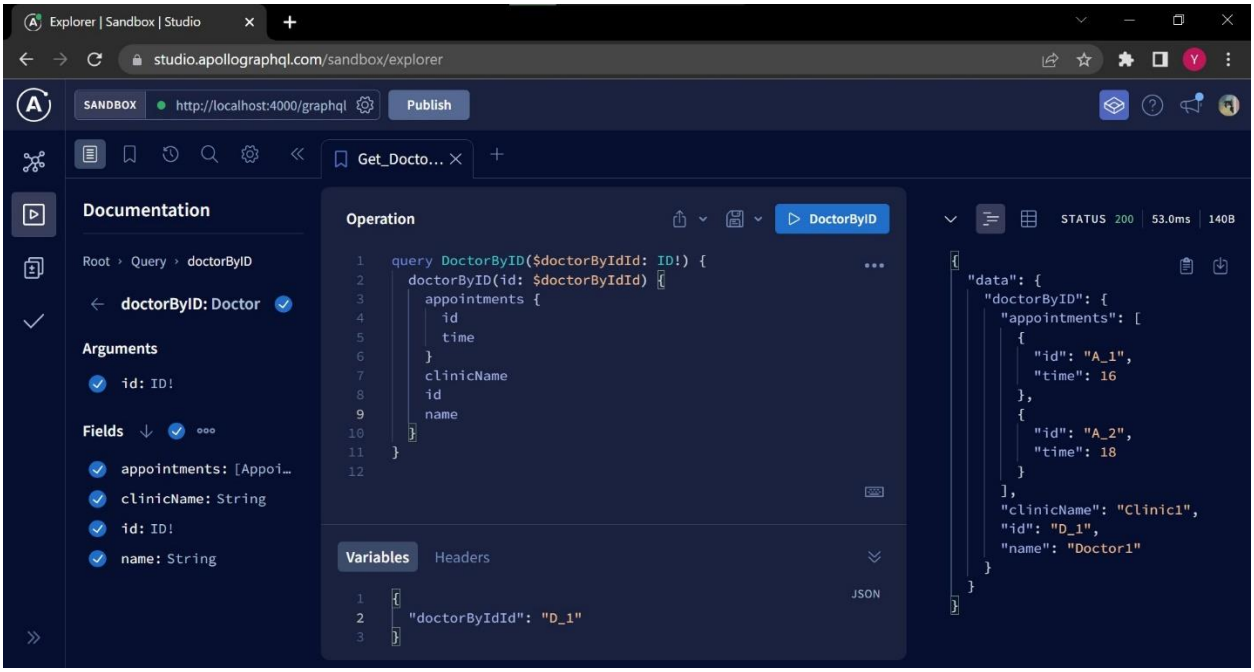
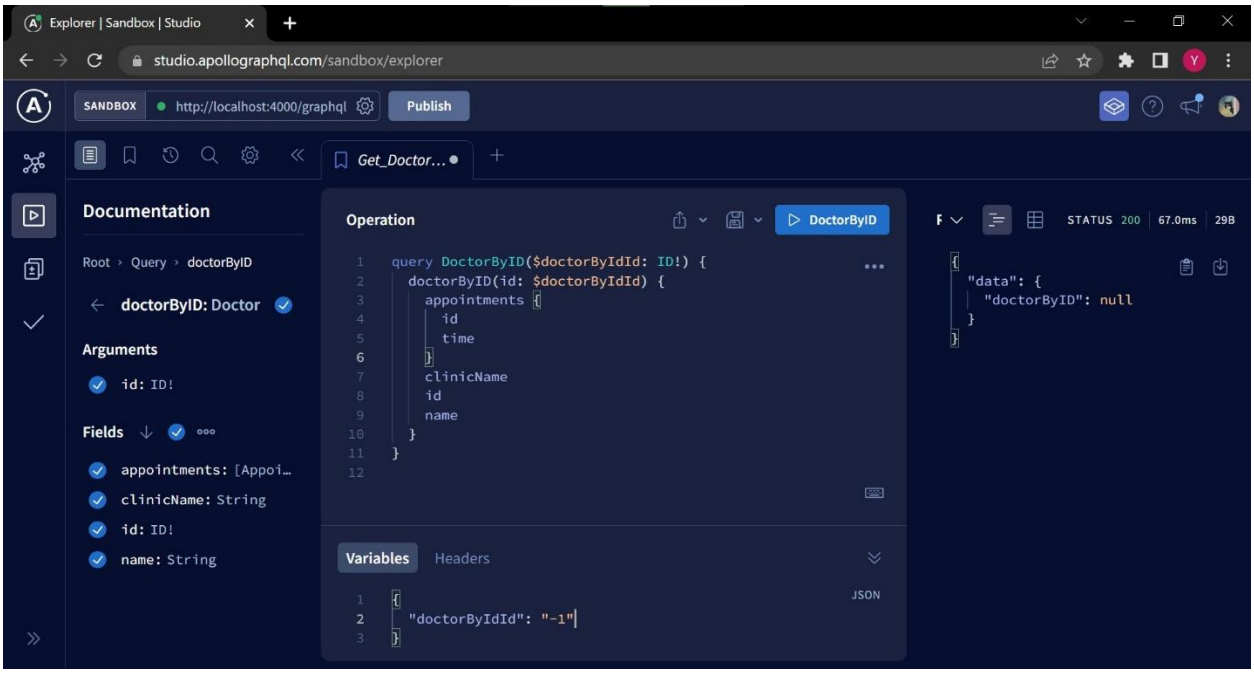
## Testing

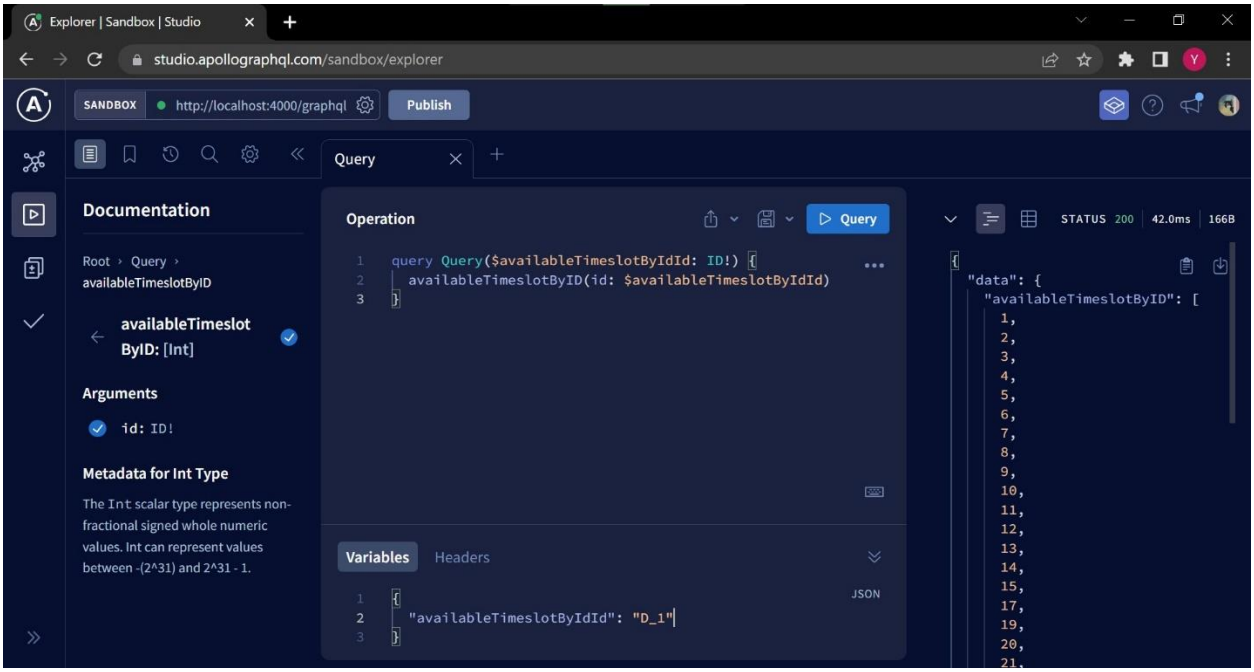
Get_Doctor_Success



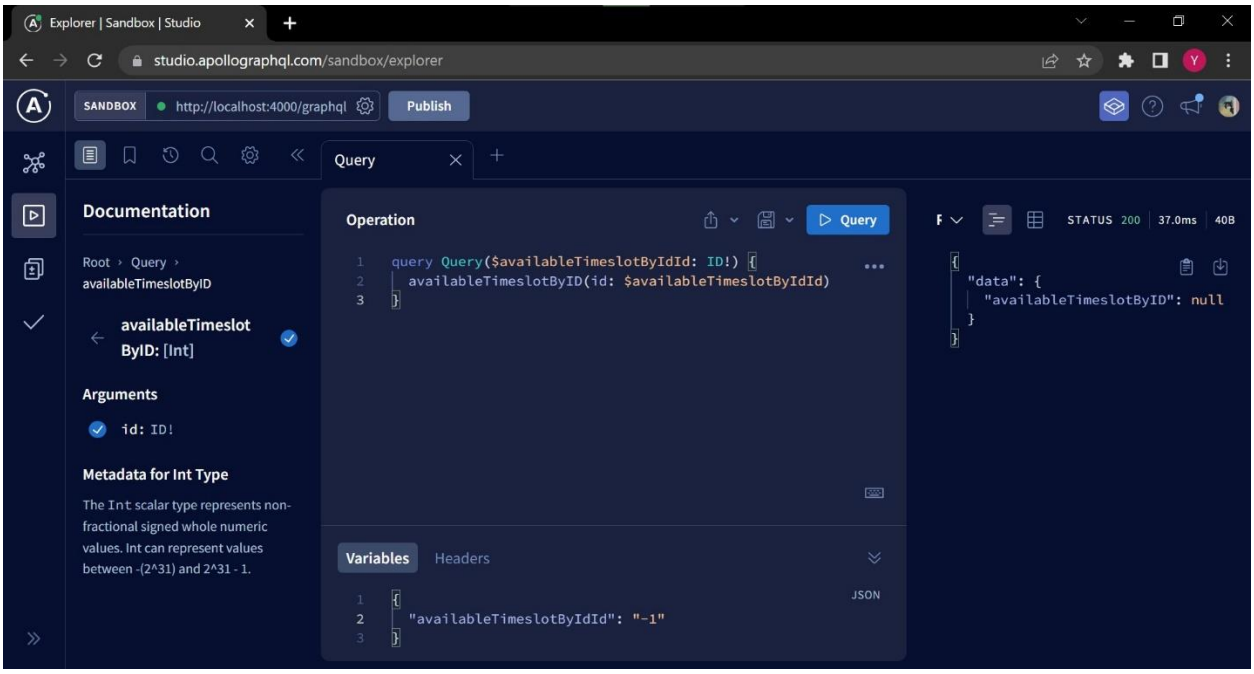

Get_Doctor_Fail



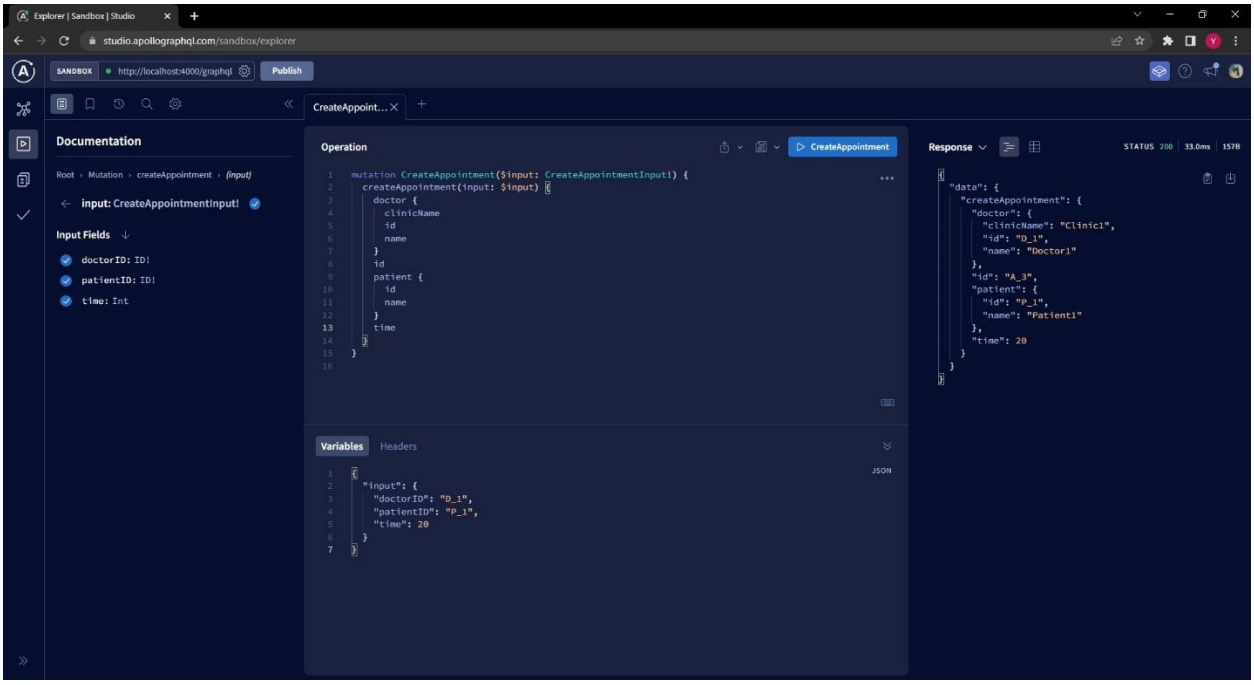
Get_Time_Success



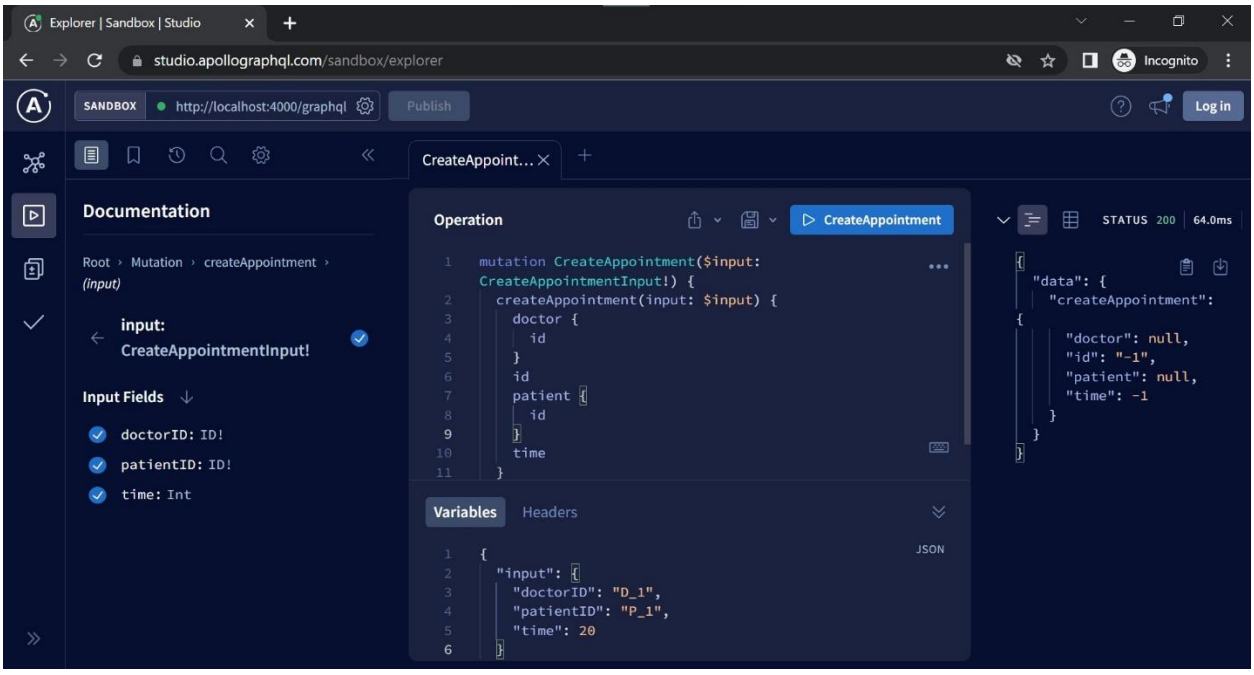

Get_Time_Fail



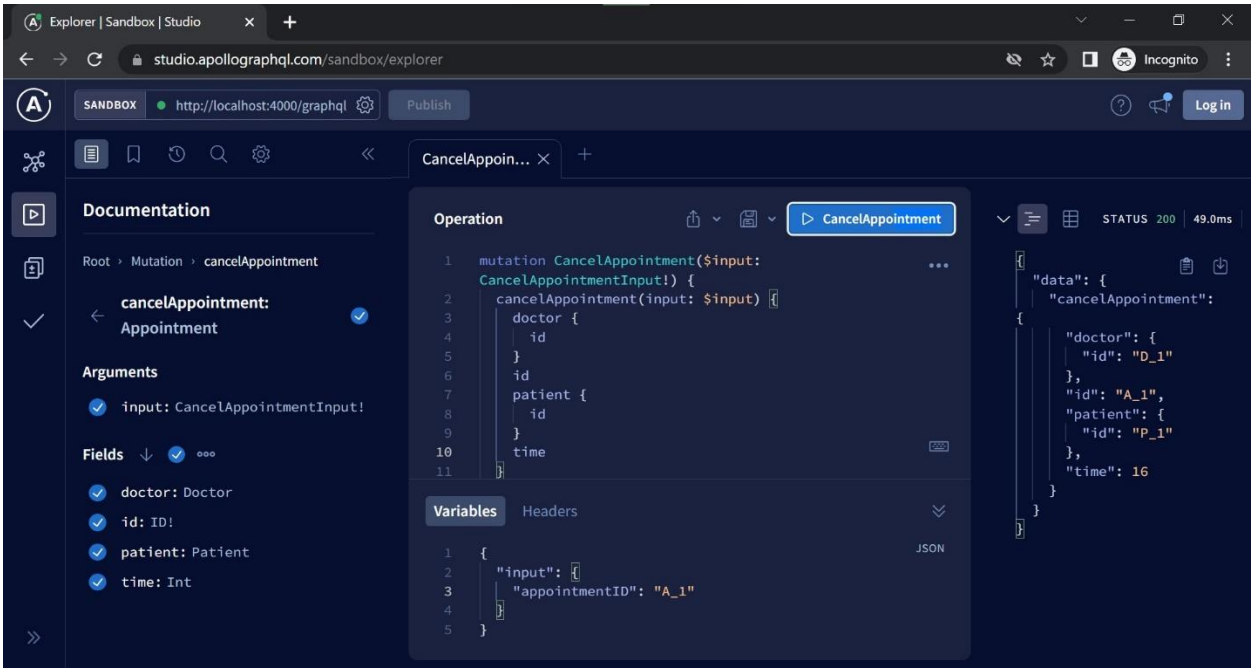
Add_Appointment_Success



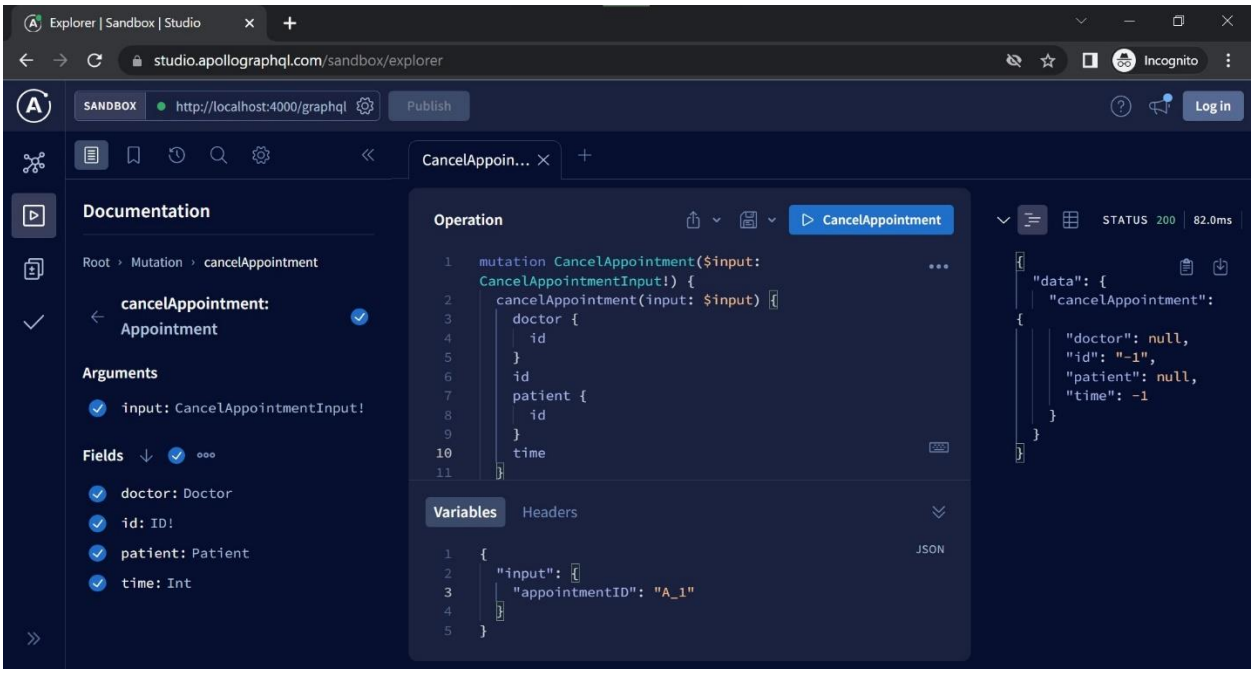

Add_Appointment_Fail



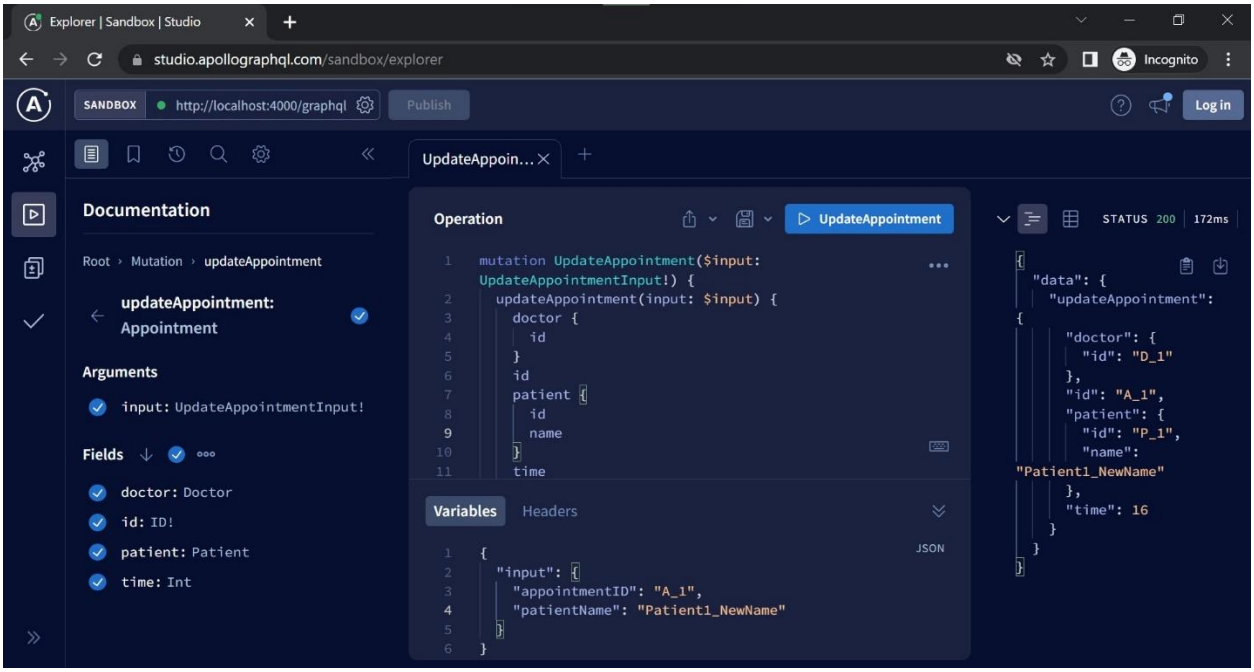
Cancel_Appointment_Success



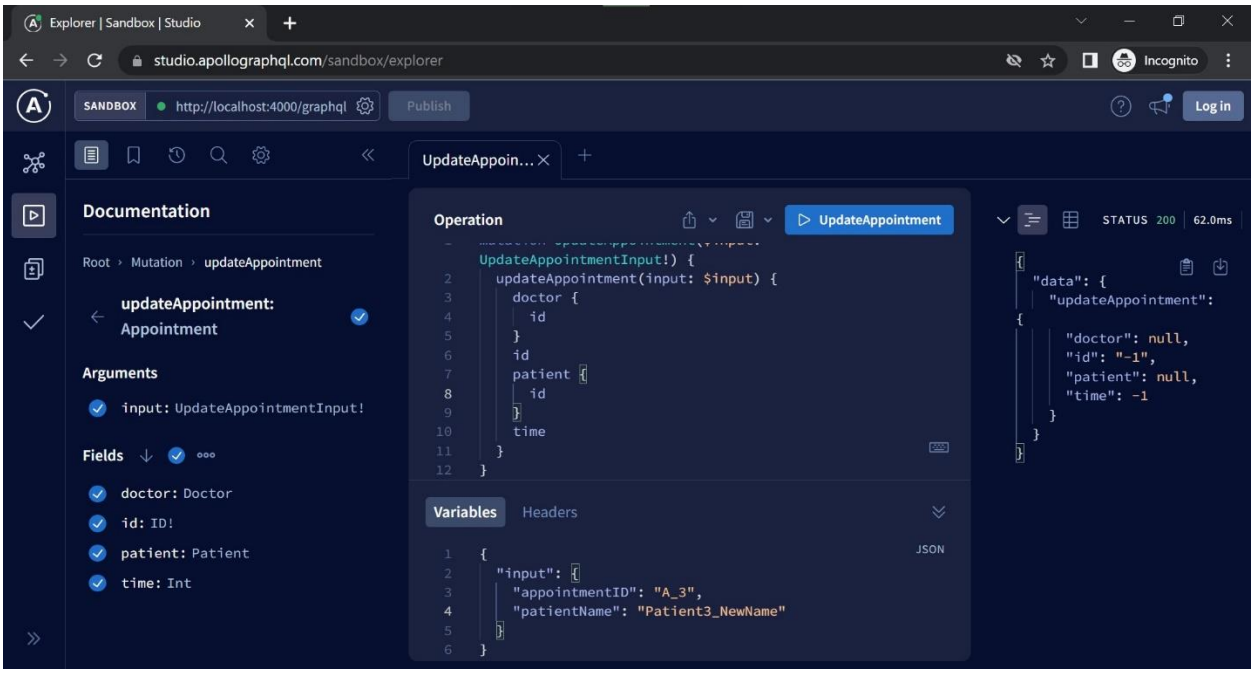

Cancel_Appointment_Fail



Update_Appointment_Success




Update_Appointment_Fail



**Reflection**

- **What were some of the alternative schema and query design options you considered? Why did you choose the selected options?**
  An alternative schema is to store available timeslots as a field for each doctor instead of calculating the available timeslots upon query.
  The alternative schema is not chosen because every time an appointment is cancel, we'll have to update the available timeslots, which will take more time and is redundant.

- **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).**
    The clients won't need to change the old code for existing queries since the API will be backward compatible.
  - **How will you accommodate the changes in your existing Schema and Query types?**
    New fields (consultation type) need to be added to existing schema. Mutation for creating event will need to be changed to accommodate the new fields.

- **Describe two GraphQL best practices that you have incorporated in your API design.**
  1. While designing queries, use object types instead of object Ids.

     Instead of IDs, I used Patient and Doctor objects in the appointment type, so that client can get all information in one query.

  2. Naming conventions
     All mutations are named as verbs (CreateAppointment, CancelAppointment and UpdateAppointment), and all the queries are self-explanatory (doctorByID, availableTimeslotByID).