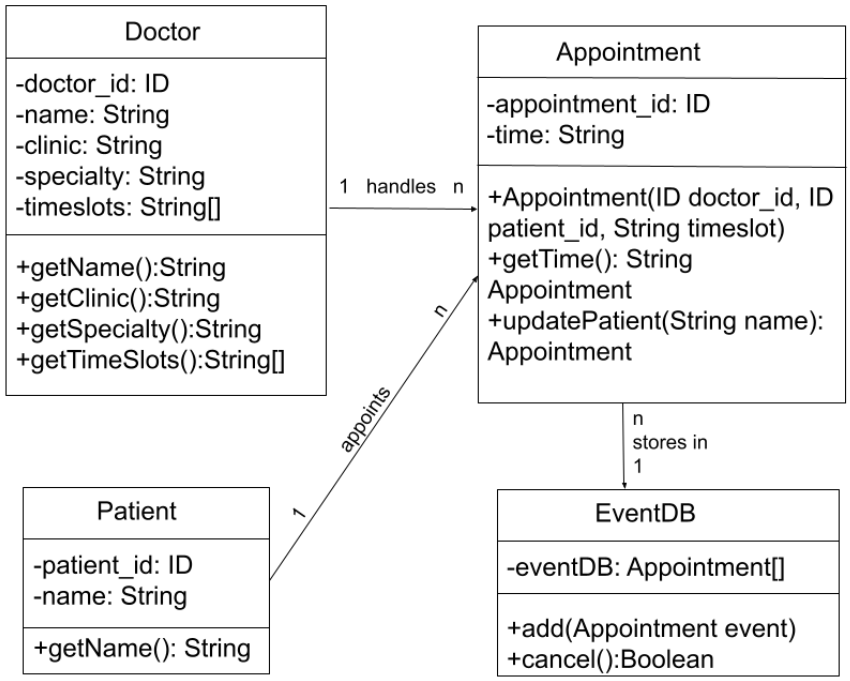
A2.1 GraphQL Design

Jasmine Chu (xiyinc)

1.1 UML Diagram



1.2 Schema (Types)

```
type Doctor {
                doctor_id: ID!
                name: String
                clinic: String
                Specialty: String
                timeslots: [String]
                appointments: [Appointment] @relation
}


type Patient {
                patient_id: ID!
                name: String
                appointments: [Appointment] @relation
}
```

```
type Appointment {
		appointment_id: ID!
		time: String!
		doctor_id: ID!
		patient_id: ID!
}

type Query {
		doctorName(doctor_id: ID):String
		doctorClinic(doctor_id: ID):String
		doctorSpecialty(doctor_id: ID):String
		doctorTimeslots(doctor_id: ID):[String]
}

type Mutation {
		addAppointment(doctor_id: ID, patient_ID: ID, time: String): ID
		cancelAppointment(appointment_id: ID): ID
		updatePatient(appointment_id: ID, name: String): ID
}
```

2. Queries

Query Name	**Input**	**Output**
doctorName	doctor_id: ID	String
doctorClinic	doctor_id: ID	String
doctorSpecialty	doctor_id: ID	String
doctorTimeslots	doctor_id: ID	[String]

3. Mutations

Query Name	**Input**	**Output**
addAppointment	doctor_id: ID, patient_ID: ID, time: String	ID

cancelAppointment	appointment_id: ID	ID
updatePatient	appointment_id: ID, name: String	ID

4. Endpoint (URL)

HTTP Request
Method: POST
URL: http://localhost:4000/
Content-Type: application/json

Query
Operation:

```
query doctorById($doctor_id: ID) {
        doctorNameById(doctor_id: $doctor_id),
        doctorClinicById(doctor_id: $doctor_id),
        doctorSpecialtyById(doctor_id: $doctor_id),
        doctorTimeslotsById(doctor_id: $doctor_id),
}
```

Variables:

```
{
        "doctor_id": "doctor1"
}
```

Response:

```
{
  "data": {
        "doctorNameById": "Lucy",
        "doctorClinicById": "Nice Care Clinic",
        "doctorSpecialtyById": "General",
        "doctorTimeslotsById": [
          "0930",
          "1130"
    ]
  }
}
```

Mutation
Operation

```
mutation mutate($doctor_id:ID, $patient_id:ID, $time:String, $appointment_id: ID,
$name: String) {
        addAppointment(doctor_id: $doctor_id, patient_id: $patient_id, time:
      $time),
        updatePatient(appointment_id: $appointment_id, name: $name),
        cancelAppointment(appointment_id: $appointment_id)
}
```

Variables

```
{
        "doctor_id": "doctor1",
        "patient_id": "patient1",
        "time": "1000",
        "appointment_id": "appointment1",
        "name": "Lily"
}
```

Response

```
{
        "data": {
          "addAppointment": "17ce5689-bd08-40ce-9d78-6e9b4a48c722",
          "updatePatient": "appointment1",
          "cancelAppointment": "appointment1"
        }
}
```

Test Cases Design

Test Case Identifier	**Test Case Description**	**Inputs**	**Expected Output**	**Remarks**
testDoctorName_success	Test passed because it can query a doctor’s name by their id correctly.	query { doctor (id: 1) { id name } }	{name:“Lucy”}	query
testDoctorName_fail	Test failed because the doctor didn't exist.	query { doctor (id: 100) { id name } }	500 Internal Server Error: The doctor doesn’t exist!	query

testDoctorClinic_success	Test passed because it can query a doctor’s clinic name by their id correctly.	query { doctor (id: 1) { id clinic } }	{clinic:“Lucy’s Clinic”}	query
testDoctorClinic_fail	Test failed because the doctor didn't exist.	query { doctor (id: 100) { id clinic } }	500 Internal Server Error: The doctor doesn’t exist!	query
testDoctorSpecialty_success	Test passed because it can query a doctor’s specialty by their id correctly.	query { doctor (id: 1) { id specialty } }	{specialty:“General Physician”}	query
testDoctorSpecialty_fail	Test failed because the doctor didn't exist.	query { doctor (id: 100) { id specialty } }	500 Internal Server Error: The doctor doesn’t exist!	query
testDoctorTimeSlots_success	Test passed because it can query a doctor’s timeslots by their id correctly.	query { doctor (id: 1) { id timeslots } }	{timeslots:[“9:30”, “10:30”, “20:30”]}	query
testDoctorTimeSlots_fail	Test failed because the doctor didn't exist.	query { doctor (id: 100) { id timeslots } }	500 Internal Server Error: The doctor doesn’t exist!	query
testAddAppointment_success	Test passed because it can add a new appointment successfully.	mutation { addAppointment (doctor_id: 1, patient_id: 1, time: “9:30”) }	{appointmentID : 102}	mutation
testAddAppointment_fail	Test failed because the user didn’t input the appointment time.	mutation { addAppointment (doctor_id: 1, patient_id: 1) }	500 Internal Server Error: Lack of variable time!	mutation

testCancelAppointment_success	Test passed because it can cancel the appointment successfully.	mutation { cancelAppointment (appointment_id: 101) }	{appointmentID : 101}	mutation
testCancelAppointment_fail	Test failed because the appointment didn't exist.	mutation { cancelAppointment (appointment_id: 1000) }	500 Internal Server Error: The appointment doesn’t exist!	mutation
testUpdatePatient_success	Test passed because it can update the patient’s name of the given appointment successfully.	mutation { updateAppointment (appointment_id: 102, patientName:” John”) }	{appointmentID : 102}	mutation
testUpdatePatient_fail	Test failed because the appointment didn't exist.	mutation { updateAppointment (appointment_id: 1000, patientName:” John”) }	500 Internal Server Error: The appointment doesn’t exist!	mutation