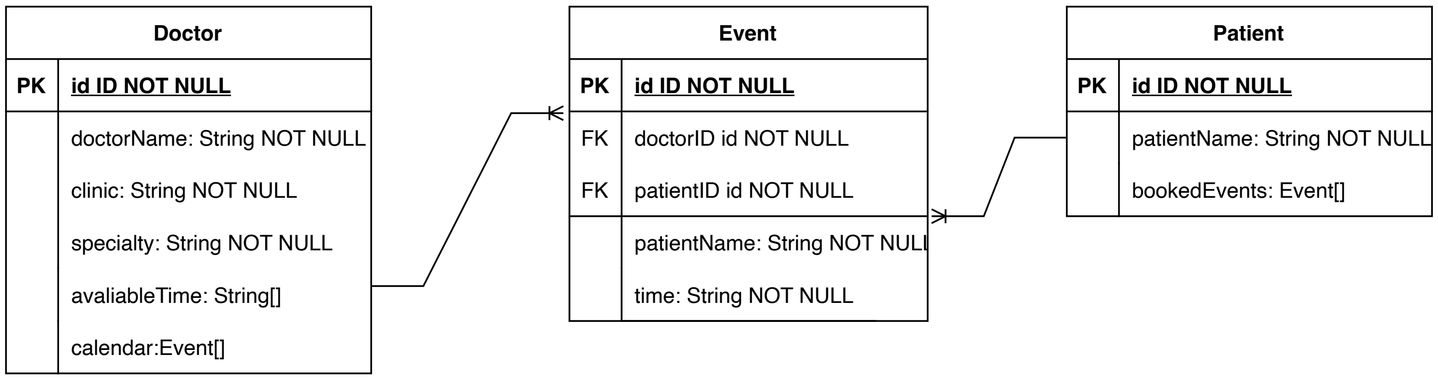
ER digram



There are there basic data models here: 1. Doctor 2. Event 3. Patient. The relationship between Doctor and Event is ***one-to-many***. Since a doctor can have multiple events(appointments) in calendar while an event can only involve one doctor. The relationship between patient and event is also ***one-to-many***, since a patient can book multiple events but an event(appointment) can only involve one patient.

Date Model Schema

Doctor schema which indicates the doctor entity

```
type Doctor{
    id: ID!
    doctorName: String!
    clinic: String!
    specialty: SpecialtyType!
    avaliableTime: [String]
    calender: [Event]
}
```

Event schema which indicates the event entity

```
type Event{
    id: ID!
    doctor: Doctor!
    patient: Patient!
    patientName: String!
    time: String!
}
```

Patient schema which indictaes the patient entity

```
type Patient{
   id: ID!
   patientName: String!
   bookedEvents: [Event]
}
```

Enumerate the possible values of specialty type

```
enum SpecialtyType {
  general physician
  gynecologist
  orthopedic
}
```

Query

- ***Name: doctorDetailById()***: Get doctor details (name, clinic name, specialty)
    - Query schema:

    ```
    type Query {
       doctorDetailById(id:ID!):DoctorDetail
    }
    ```

    - Input: ID type, cannot be null.
    - Output: DoctorDetail, and the schema is defined as below:

    ```
    type DoctorDetail{
      doctorID: ID!
      doctorName: String!
      clinic: String!
      specialty: SpecialtyType!
    }
    ```

- ***Name: doctorAvailableTimeById()***: Get doctor's available timeslots for today
    - Query schema:

```
type Query {
   doctorAvailableTimeById(id:ID!):DoctorAvailableTime
}
```

- Input: ID type, cannot be null.
- Output: DoctorAvailableTime type, whose schema is define as below:

```
type DoctorAvailableTime{
  doctorID: ID!
  avaliableTime: [String]
}
```

Mutation

- ***Name: bookAppointment()***: Book an appointment with a doctor for today
  - Mutation schema:

```
type Mutation{
    bookAppointment(bookAppointmentInput: BookAppointmentInput!): Event!
}
```

  - Input: bookAppointmentInput type, the schema is defined as below:

```
type BookAppointmentInput{
  doctor: Doctor!
  patient: Patient!
  time: String!
}
```

  - Output: the output type of this mutation is Event, which has already been defined in the ***Event Schema***.
- ***Name: cancelAppointmentById()***: Cancel an appointment by id
  - Mutation schema:

```
type Mutation{
  cancelAppointmentById(id:ID!): Event!
}
```

  - Input: ID type which cannot be null
  - Output: the output type of this mutation is Event, which has already been defined in the ***Event***

Schema.

- ***Name: updatePatientNameByAppointmentId()***: Update name of the patient for an appointment
  - Mutation schema:

    ```
    type Mutation{
      updatePatientNameByAppointmentId(id:ID!, name: String!): Event!
    }
    ```

  - Input: ID type which indicates the event id and the new patient name of string type. Both of the inputs cannot be null.
  - Output: the output type of this mutation is Event, which has already been defined in the ***Event Schema***.

API Endpoint

the default endpoint of in Apollo Explorer is (http://localhost:4000/), I choose that as my API endpoint as this time

Test cases

Testcase Identifier	Testcase Description	Inputs	Expected Outputs	Remarks

Getting doctor details by id succeeds	Successfully get the detailed information of a doctor by giving the valid doctor id	Query{
		doctorDetailById(id: "1")
		}
			DoctorDetail{
			"id": "1",
			"doctorName": "hyy"
			"clinic": "hyy"
			"specialty":"orthopedic"
			}
				If the given doctor id is find and an DoctorDetail type entity will be returned which contains the detail information of the doctor

Get doctor details by id fails	Fail to get the detailed information of a doctor by not giving the required id	Query{
		doctorDetailById(id:)
		}
			"message": "Field "doctorDetailById" argument "id" of type "ID!" is required, but it was not provided."	If the required field "id" is not given, the error message will throw

Get doctor available time by id succeeds	Successfully get the avaliable time of doctor by giving a valid doctor id	Query{
		doctorAvailableTimeById(id: "1")
		}
			DoctorAvailableTime{
			"id": "1"
			"avaliableTime": ["4:00"]
			}
				If the given doctor id is find and an DoctorAvailableTime type entity will be returned, which contains the doctor id and the corresponding available time

Getting doctor available time by id fails	Fail to get the avaliable time of doctor by not giving the required id	Query{
		doctorAvailableTimeById(id:)
		}
			"message": "Field "doctorAvailableTimeById" argument "id" of type "ID!" is required, but it was not provided."	If the required field "id" is not given, the error message will throw

Booking an		Mutation{	Event{	If all the required information is given

appointment with a doctor for today succeeds	Successfully book an appointment with a doctor by provding the corrent "BookAppointmentInput"	bookAppointment(doctor: Doctor1 patient: Patient1 time: "4:00") }	"id": "1" "doctor": Doctor1 "patient": Patient1 "patientName": "sss" "time": "4:00" }	for the mutation, it will return an Event type entity with corresponding information
Booking an appointment with a doctor for today fails	Fail to book an appointment with a doctor by missing some of the required fields in "BookAppointmentInput"	Mutation{ bookAppointment(doctor: Doctor1 patient: Patient1 time:) }	"message": "Field "bookAppointment" argument "time" of type "String!" is required, but it was not provided."	If the required field "time" is not given, the error message will throw
Canceling event by id succeeds	Successfully cancel an event(appointment) by providing the valid event id	Mutation{ cancelAppointmentById(id: "1") }	Event{ "id": "1" "doctor": Doctor1 "patient": Patient1 "patientName": "sss" "time": "4:00" }	If the required field is given and is valid, then an Event type entity response will be returned, which is the event being cancelled
Canceling event by id fails	Fail to cancel an event(appointment) by not providing the valid event id	Mutation{ cancelAppointmentById(id:) }	"message": "Field "cancelAppointmentById" argument "id" of type "ID!" is required, but it was not provided."	If the required field "id" is not given, the error message will throw
Updating patient name by appointment id succeeds	Successfully update the name of the patient of an appointment by providing the valid event id and patient name	Mutation{ updatePatientNameByAppointmentId(id: "1", name:"new name") }	Event{ "id": "1" "doctor": Doctor1 "patient": Patient1 "patientName": "new name" "time": "4:00" }	If the required field is given and is valid, then an Event type entity response will be returned, which contains the updated event information
	Fail to update the name of the patient of an appointment by not providing the valid event id and patient name	Mutation{ updatePatientNameByAppointmentId(id: "1", name:) }	"message": "Field "updatePatientNameByAppointmentId" argument "name" of type "String!" is required, but it was not provided."	If the required field "name" is not given, the error message will throw