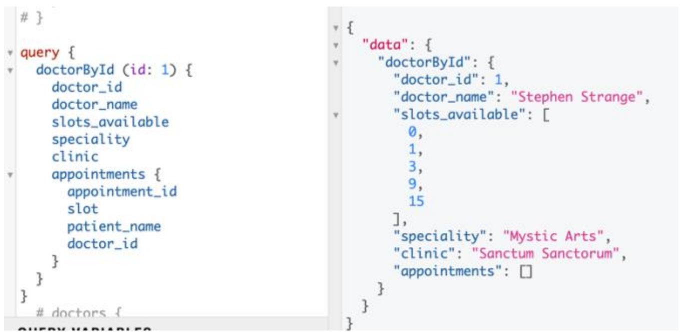
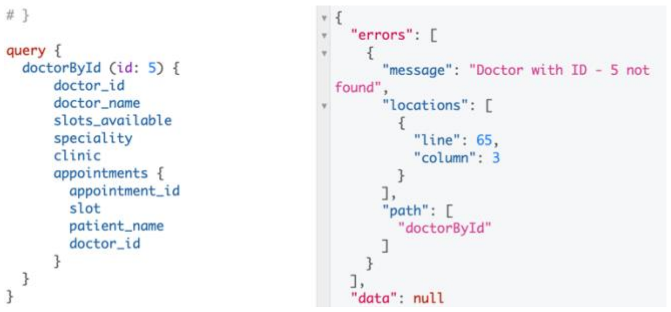
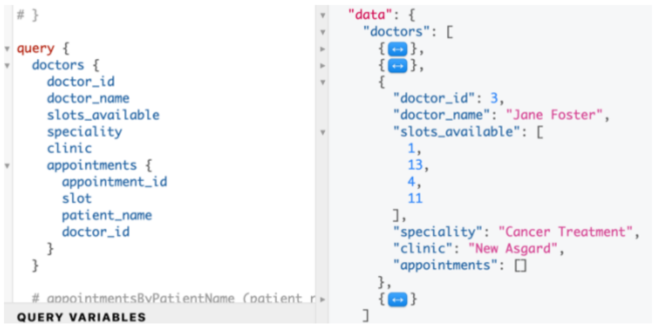
# 17-625: API Design
Assignment 2.2 - GraphQL Implementation

## Testing the GQL Implementation

For each query and mutation, there is a happy path and an error path.

| Id | Testcase Description | Test in GQL Playground |
|---|---|---|
| **1** 🟩 | Get doctor by id, with the id existing valid |  |
| **2** 🟥 | Get doctor by id, with the id not existing |  |
| **3** 🟩 | Get a list of doctors with valid fields queried |  |

```
# }

query {
  doctorById (id: 1) {
    doctor_id
    doctor_name
    slots_available
    speciality
    clinic
    appointments {
      appointment_id
      slot
      patient_name
      doctor_id
    }
  }
}
  # doctors {
QUERY VARIABLES
```

```
{
  "data": {
    "doctorById": {
      "doctor_id": 1,
      "doctor_name": "Stephen Strange",
      "slots_available": [
        0,
        1,
        3,
        9,
        15
      ],
      "speciality": "Mystic Arts",
      "clinic": "Sanctum Sanctorum",
      "appointments": []
    }
  }
}
```

```
# }

query {
  doctorById (id: 5) {
    doctor_id
    doctor_name
    slots_available
    speciality
    clinic
    appointments {
      appointment_id
      slot
      patient_name
      doctor_id
    }
  }
}
```

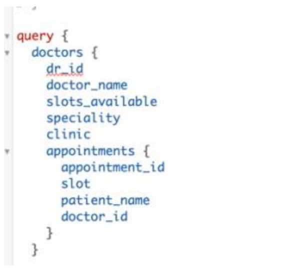
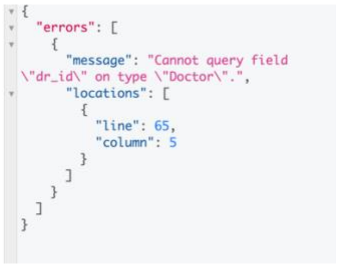
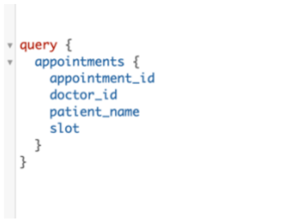
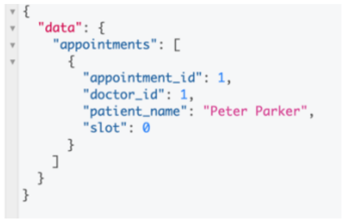
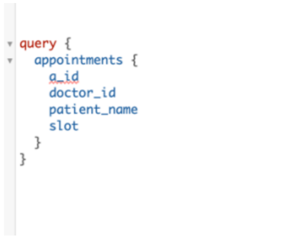
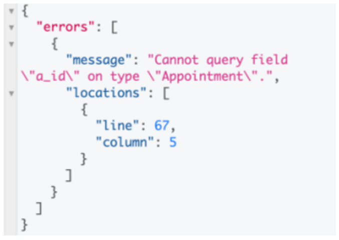
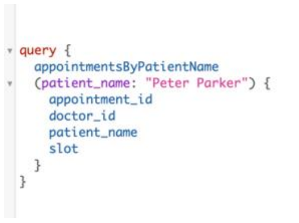
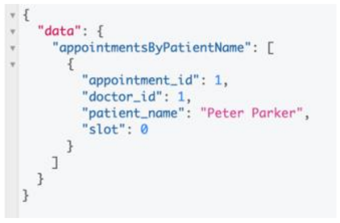
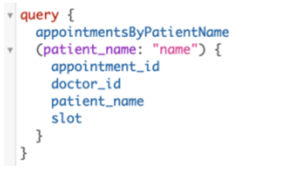
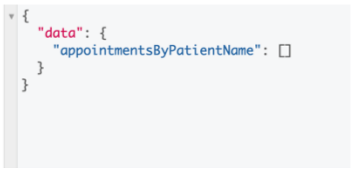
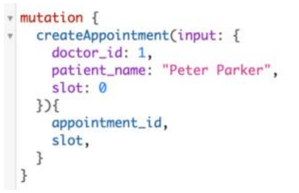
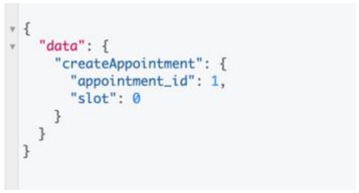
```
{
  "errors": [
    {
      "message": "Doctor with ID - 5 not
found",
      "locations": [
        {
          "line": 65,
          "column": 3
        }
      ],
      "path": [
        "doctorById"
      ]
    }
  ],
  "data": null
```

```
# }

query {
  doctors {
    doctor_id
    doctor_name
    slots_available
    speciality
    clinic
    appointments {
      appointment_id
      slot
      patient_name
      doctor_id
    }
  }

  # appointmentsByPatientName (patient_r
QUERY VARIABLES
```

```
  "data": {
    "doctors": [
      {↔},
      {↔},
      {
        "doctor_id": 3,
        "doctor_name": "Jane Foster",
        "slots_available": [
          1,
          13,
          4,
          11
        ],
        "speciality": "Cancer Treatment",
        "clinic": "New Asgard",
        "appointments": []
      },
      {↔}
    ]
```

| | | | |
|---|---|---|---|
| 🟥 **4** | Get a list of doctors with an invalid field queried |  |  |
| 🟩 **5** | Get a list of appointments with valid fields queried |  |  |
| 🟥 **6** | Get a list of appointments with invalid fields queried |  |  |
| 🟩 **7** | Get appointments for an existing *patient_name* |  |  |
| 🟥 **8** | Get appointments for a non-existing *patient_name.* **Note** - The result here is different than what I documented previously. Here, instead of an error I get an empty list, which is more logical. |  |  |
| 🟩 **9** | Create appointment with all fields valid |  |  |

```
query {
  doctors {
    dr_id
    doctor_name
    slots_available
    speciality
    clinic
    appointments {
      appointment_id
      slot
      patient_name
      doctor_id
    }
  }
```

```
{
  "errors": [
    {
      "message": "Cannot query field
\"dr_id\" on type \"Doctor\".",
      "locations": [
        {
          "line": 65,
          "column": 5
        }
      ]
    }
  ]
}
```

```
query {
  appointments {
    appointment_id
    doctor_id
    patient_name
    slot
  }
}
```

```
{
  "data": {
    "appointments": [
      {
        "appointment_id": 1,
        "doctor_id": 1,
        "patient_name": "Peter Parker",
        "slot": 0
      }
    ]
  }
}
```

```
query {
  appointments {
    a_id
    doctor_id
    patient_name
    slot
  }
}
```

```
{
  "errors": [
    {
      "message": "Cannot query field
\"a_id\" on type \"Appointment\".",
      "locations": [
        {
          "line": 67,
          "column": 5
        }
      ]
    }
  ]
}
```

```
query {
  appointmentsByPatientName
  (patient_name: "Peter Parker") {
    appointment_id
    doctor_id
    patient_name
    slot
  }
}
```

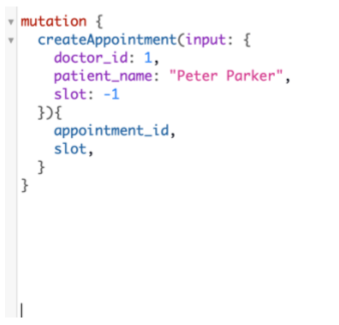
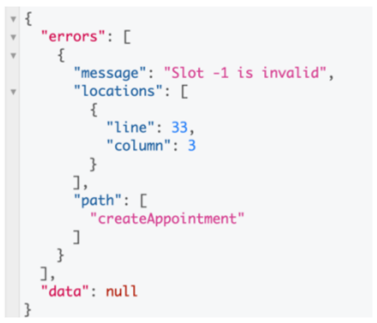
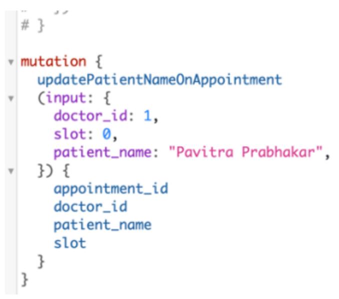
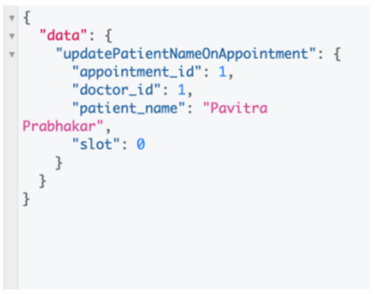
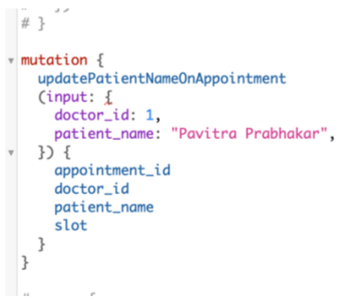
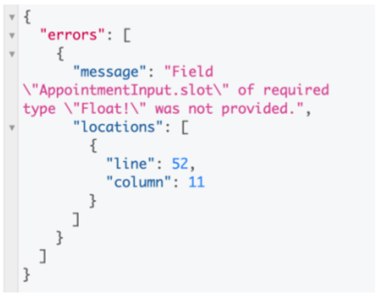
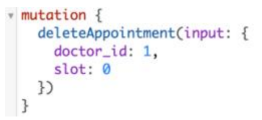
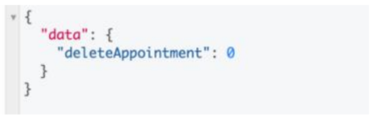
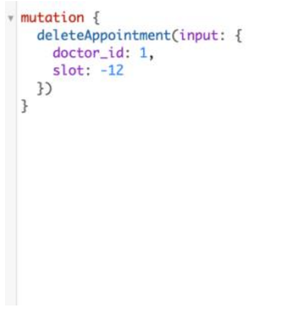
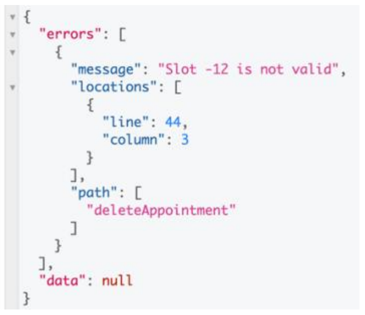
```
{
  "data": {
    "appointmentsByPatientName": [
      {
        "appointment_id": 1,
        "doctor_id": 1,
        "patient_name": "Peter Parker",
        "slot": 0
      }
    ]
  }
}
```

```
query {
  appointmentsByPatientName
  (patient_name: "name") {
    appointment_id
    doctor_id
    patient_name
    slot
  }
}
```

```
{
  "data": {
    "appointmentsByPatientName": []
  }
}
```

```
mutation {
  createAppointment(input: {
    doctor_id: 1,
    patient_name: "Peter Parker",
    slot: 0
  }){
    appointment_id,
    slot,
  }
}
```

```
{
  "data": {
    "createAppointment": {
      "appointment_id": 1,
      "slot": 0
    }
  }
}
```

| | | |
|---|---|---|
| 🟥 **10** | Create appointment with invalid slot |  mutation {<br>  createAppointment(input: {<br>    doctor_id: 1,<br>    patient_name: "Peter Parker",<br>    slot: -1<br>  }){<br>    appointment_id,<br>    slot,<br>  }<br>}  {<br>  "errors": [<br>    {<br>      "message": "Slot -1 is invalid",<br>      "locations": [<br>        {<br>          "line": 33,<br>          "column": 3<br>        }<br>      ],<br>      "path": [<br>        "createAppointment"<br>      ]<br>    }<br>  ],<br>  "data": null<br>} |
| 🟩 **11** | Update patient name on an existing appointment slot with and existing doctor |  # }<br>mutation {<br>  updatePatientNameOnAppointment<br>  (input: {<br>    doctor_id: 1,<br>    slot: 0,<br>    patient_name: "Pavitra Prabhakar",<br>  }) {<br>    appointment_id<br>    doctor_id<br>    patient_name<br>    slot<br>  }<br>}  {<br>  "data": {<br>    "updatePatientNameOnAppointment": {<br>      "appointment_id": 1,<br>      "doctor_id": 1,<br>      "patient_name": "Pavitra<br>Prabhakar",<br>      "slot": 0<br>    }<br>  }<br>} |
| 🟥 **12** | Update patient name on an existing doctor's appointment, but with slot missing |  # }<br>mutation {<br>  updatePatientNameOnAppointment<br>  (input: {<br>    doctor_id: 1,<br>    patient_name: "Pavitra Prabhakar",<br>  }) {<br>    appointment_id<br>    doctor_id<br>    patient_name<br>    slot<br>  }<br>}  {<br>  "errors": [<br>    {<br>      "message": "Field<br>\\"AppointmentInput.slot\\" of required<br>type \\"Float!\\" was not provided.",<br>      "locations": [<br>        {<br>          "line": 52,<br>          "column": 11<br>        }<br>      ]<br>    }<br>  ]<br>} |
| 🟩 **13** | Delete appointment for a valid doctor at a valid slot |  mutation {<br>  deleteAppointment(input: {<br>    doctor_id: 1,<br>    slot: 0<br>  })<br>}  {<br>  "data": {<br>    "deleteAppointment": 0<br>  }<br>} |
| 🟥 **14** | Delete appointment for a valid doctor at an invalid slot |  mutation {<br>  deleteAppointment(input: {<br>    doctor_id: 1,<br>    slot: -12<br>  })<br>}  {<br>  "errors": [<br>    {<br>      "message": "Slot -12 is not valid",<br>      "locations": [<br>        {<br>          "line": 44,<br>          "column": 3<br>        }<br>      ],<br>      "path": [<br>        "deleteAppointment"<br>      ]<br>    }<br>  ],<br>  "data": null<br>} |

## Reflections

### Alternative Design Options

The schema currently just consists of an Appointment and a Doctor entity. I also considered having a patient entity, but I did not include it in the current design due to the unnecessary complexity it could add, especially in the case of changing a patient's name on an appointment. This case could have introduced inconsistency issues, something which I avoided for the scope of this assignment.

For getting the details of a particular doctor, I considered having the doctor's name as a parameter, instead of their ID (what I have currently). This made more sense from a user's perspective, but not so much from a development perspective, as names are never unique, and cannot be used to find unique doctors. Hence, I created this query keeping in mind that in order to call the query, the client would have to know the ID of the doctor in advance, something they can do using other methods. One example on the UI could be that clicking on a doctor's name would give access to that doctor's object which contains the ID, the UI can then make the *getDoctorById* query to get more information about the doctor.

Lastly, I should ideally have all the Id's for all entities of type ID provided out of the box for GQL, but instead I went with having simple integers. This helped made my manual tests easier and faster, but if this code was to be productionized, these types should definitely be ID.

### Enhancing the "Appointment" Structure

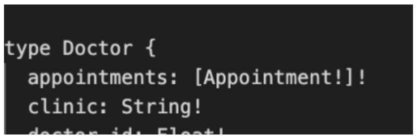
If we were to add new fields to the currently existing "Appointment" type, it would not break the API, due GQL being backwards compatible by design. However, there would be some changes by the client in their queries to include these fields, if they want to query them. For example, if we add a "firstTime" Boolean to the model, the query will also need to have this mentioned, if they want to get the value of this field.

For the current design, adding such fields is straightforward. It would just require a change in the *doctors.schema.ts* file to add the field in the Appointment class, the rest would be automatically taken care of, thanks to the type coupling that the query resolvers have with the classes in the file.

### GQL Best Practices

Two of the best practices that I followed for the project are -

1. Having objects instead of objectIds nested in other objects. For the Doctor, I have the appointments array, instead of the appointmentIds array. This removes the need for another query to get a doctor's appointments.



```
type Doctor {
  appointments: [Appointment!]!
  clinic: String!
```

2. Naming queries like "doctorById", instead of "getDoctor". Queries are always used to get something, so this naming scheme is redundant.