

| Titel der Übung Spring Petclinic GraphQL | |
|---|---|
| Datum / Zeit 24.06.2024 | Abgabedatum 27.06.2024 |
| Professor / Lehrer Dr. Dipl-Ing. Geherad Gabe | Ort / Raum |
| Teammitglieder Muhammed Güzel Melih Sahin Kenan Ergüven | |
| Klasse / Gruppe FTB_INF_VZ_4 | Protokollführung Muhammed Güzel Melih Sahin Kenan Ergüven |

INHALTSVERZEICHNIS

| | |
|---|----|
| Enterprise Applikationen..... | 1 |
| 1. Einleitung | 3 |
| 1.1. Technologien | 3 |
| 1.1.1. Spring Boot | 3 |
| 1.1.2. GraphQL | 3 |
| 1.2. Projektstruktur..... | 4 |
| 1.2.1. Backend | 4 |
| 1.2.2. Frontend..... | 4 |
| 2. Installation und Konfiguration | 5 |
| 3. Architektur..... | 6 |
| 3.1. Architekturdiagramm..... | 6 |
| 3.1.1. Client - Komponentenbeschreibung | 7 |
| 3.1.2. Api-Gateway - Komponent | 7 |
| 3.1.3. Service-Discovery (Eureka)..... | 7 |
| 3.1.4. Services | 7 |
| 4. Analyse von Testdaten..... | 8 |
| 4.1. Frontend | 8 |
| 4.2. Backend..... | 12 |

| | | |
|--------|---|----|
| 4.2.1. | Befehl 1:..... | 13 |
| 4.2.2. | Befehl 2..... | 13 |
| 4.2.3. | Befehl 3..... | 14 |
| 4.2.4. | Befehl 4..... | 14 |
| 4.2.5. | Befehl 5..... | 15 |
| 5. | Architekturenerweiterung..... | 15 |
| 5.1. | Erweiterung – Property Erweiterung SQL..... | 15 |
| 5.2. | Ergebnis..... | 20 |
| 6. | Fazit | 21 |

1. EINLEITUNG

Das Projekt Spring Petclinic GraphQL ist eine Anwendung auf Basis von Spring Boot, die von der Spring-Community entwickelt wurde, um eine Tierarztpraxis zu simulieren. Es verwendet Java Spring Boot für die Backend-Entwicklung, React für die Frontend-Entwicklung und GraphQL als API-Schnittstelle.

Das Projekt erlaubt es den Benutzer, Daten über Haustiere, ihre Benutzer und Tierarztbesuche zu verwalten. Durch die Verwendung von GraphQL bietet das Projekt eine erweiterte flexible Datenabfragen und Kontrolle für Entwickler.

Das Projekt bietet die folgenden Funktionalitäten an:

- Hinzufügen, Bearbeiten und Löschen von Haustieren
- Informationsabfrage und Informationsverwaltung von Besitzern
- Dokumentation von Tierarztbesuchen
- Verwaltung von Tierarztbesuchen
- Datenabfrage (GraphQL)

Unser Ziel ist es, das bestehende Petclinic GraphQL zu erweitern, beziehungsweise Anpassungen durchzuführen. Für die entsprechenden Erweiterungen werden wie Qualitätssicherungen mittels Tests durchführen und letztendlich eine erweiterte Version des Projekts für den Endbenutzer bereitstellen. Letztendlich werden wir eine Dokumentation für Personen erstellen, die an den verschiedenen Funktionen und technischen Details am Projekt interessiert sind.

1.1. TECHNOLOGIEN

In diesem Projekt wurde eine Reihe von Technologien verwendet, um eine optimale Lösung zur Verwaltung des Tierarztpraxis zu realisieren. In dieser Dokumentation werden wir uns mit nur einem spezifischen Teil des Projekts beschäftigen, und aufgrund dessen werden wir nur die für uns relevanten Technologien erwähnen.

1.1.1. SPRING BOOT

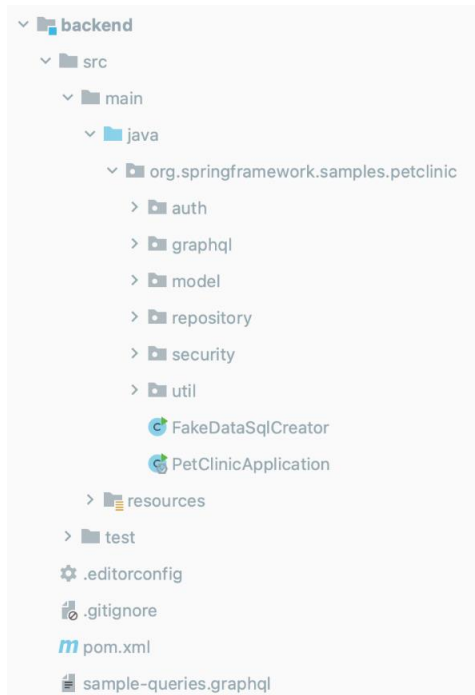
Die gesamte backend Entwicklung basiert auf Java Spring Boot, um eine einfache, flexible Anwendung zu erstellen und zu konfigurieren.

1.1.2. GRAPHQL

Für die API-Schnittstelle dieses Projekts wird GraphQL verwendet, um effiziente und flexible Datenabfragen zu ermöglichen.

1.2. PROJEKTSTRUKTUR

1.2.1. BACKEND



auth/: Authentifizierungslogik

graphql/: GraphQL-Resolver und Schema-Definitionen.

model/: Entitäten und Domänenmodelle

repository/: Schnittstellen für den Datenzugriff

security/: Sicherheitskonfigurationen

util/: Hilfsklassen und Utilities

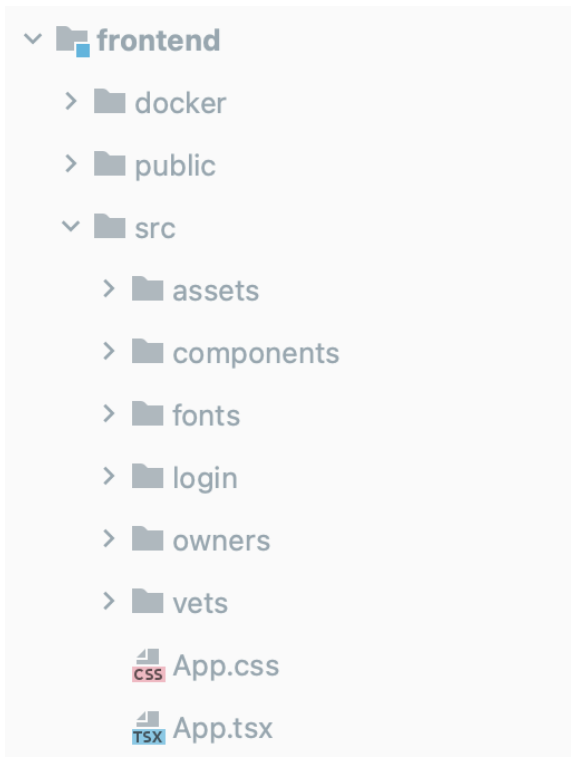
resources/: Statische Ressourcen und Konfigurationsdateien.

test/: Testfälle

pom.xml: Maven-Projektdatei

sample-queries.graphql: Beispiel-GraphQL-Abfragen.

1.2.2. FRONTEND



docker/: Docker-Konfigurationsdateien.

public/: Öffentliche Ressourcen und statische Dateien.

src/: Quellcode des Frontends.

assets/: Statische Ressourcen wie Bilder und Icons.

components/: Wiederverwendbare React-Komponenten.

fonts/: Schriftarten.

login/: Komponenten für das Login-Modul.

owners/: Komponenten für das Besitzer-Modul.

vets/: Komponenten für das Tierarzt-Modul.

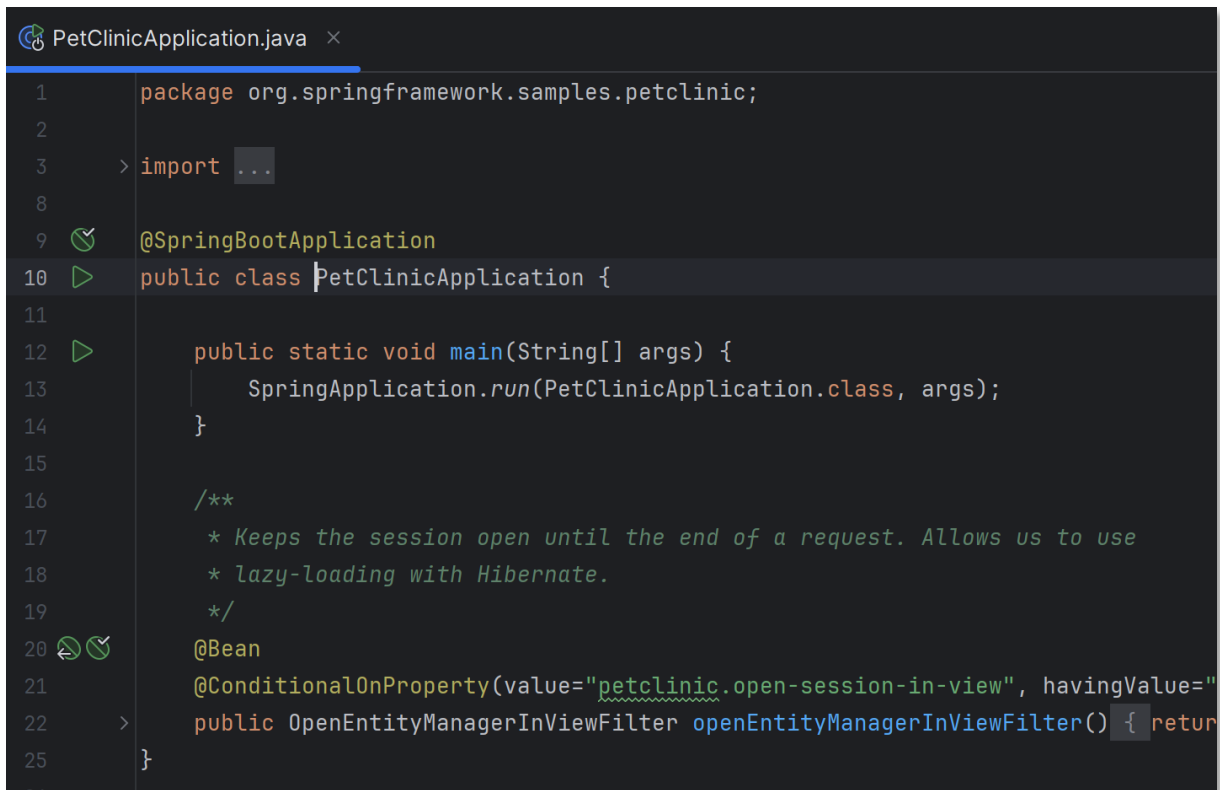
App.css: Hauptstylesheet für die Anwendung.

App.tsx: Hauptkomponente der Anwendung.

2. INSTALLATION UND KONFIGURATION

Bevor die Anwendung gestartet wird, muss Docker ausgeführt werden. Dies ist notwendig, damit die Anwendung beim Start eine Datenbank innerhalb des Docker-Containers erstellen kann.



Laut Anleitung sollte das Backend mit dem Befehl `./mvnw spring-boot:run -pl backend` ein Verzeichnis höher gestartet werden können. Jedoch kann dies zu Problemen führen, insbesondere die Fehlermeldung `'JAVA_HOME Variable not set'`, selbst wenn die Umgebungsvariable korrekt gesetzt ist.



```
1 package org.springframework.samples.petclinic;
2
3 > import ...
4
5
6
7
8
9 @SpringBootApplication
10 public class PetClinicApplication {
11
12     public static void main(String[] args) {
13         SpringApplication.run(PetClinicApplication.class, args);
14     }
15
16     /**
17      * Keeps the session open until the end of a request. Allows us to use
18      * lazy-loading with Hibernate.
19      */
20     @Bean
21     @ConditionalOnProperty(value="petclinic.open-session-in-view", havingValue="
22     > public OpenEntityManagerInViewFilter openEntityManagerInViewFilter() { retur
23
24
25 }
```

Das Starten über die Main-Methode hat diesen Fehler nicht erzeugt, aber die Datenbank konnte nicht erstellt werden, obwohl Docker ausgeführt wurde. Um das zu beheben, müssen die yaml-Dateien, die die Docker-Einstellungen enthalten, aus dem übergeordneten Verzeichnis in den Backend-Ordner kopiert werden.

Es handelt sich hierbei um die folgenden zwei Files:

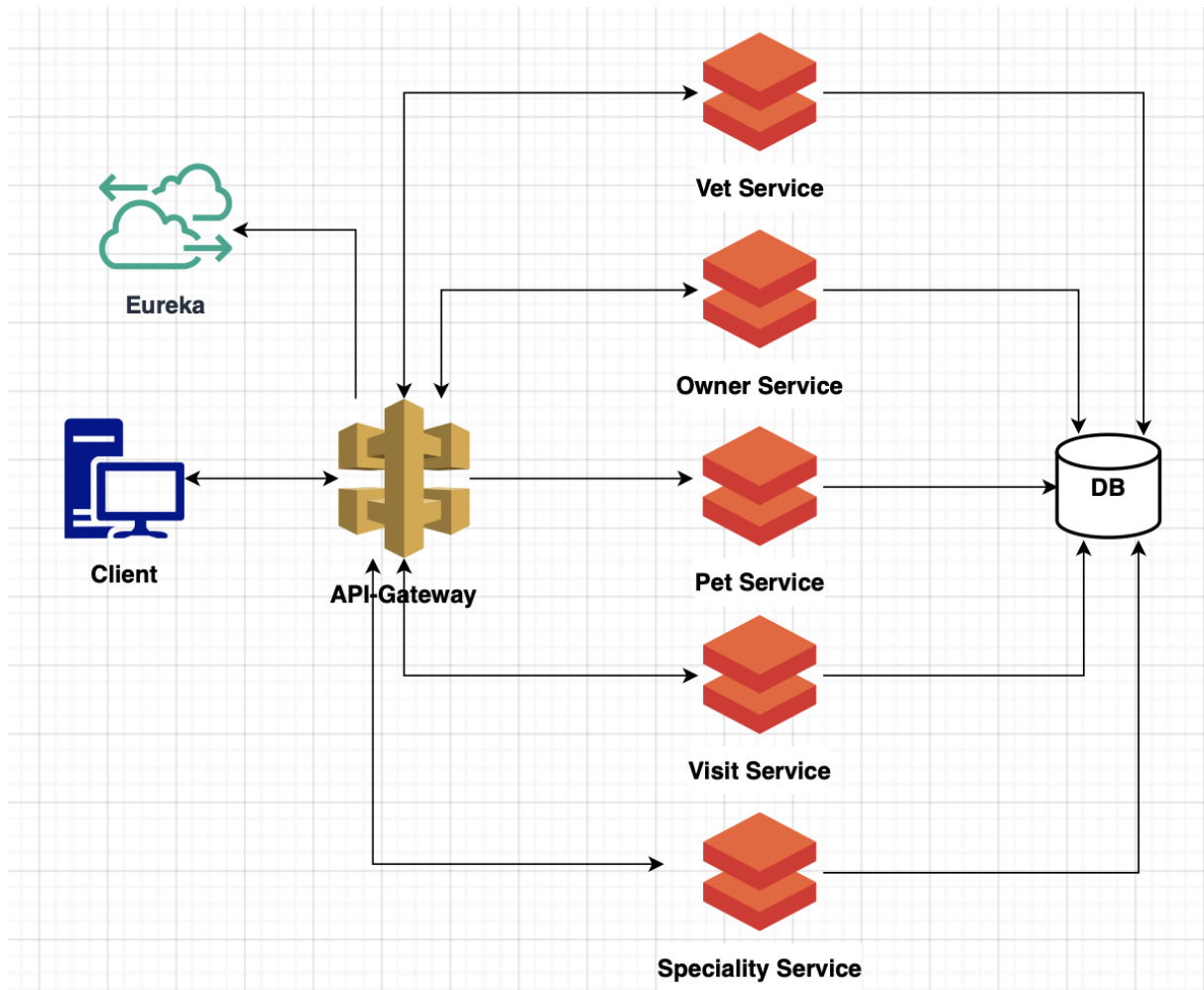
| | | | |
|--|------------------|-----------------|------|
|  docker-compose.yml | 24.06.2024 10:51 | Yaml-Quelldatei | 1 KB |
|  docker-compose-petclinic.yml | 24.06.2024 10:51 | Yaml-Quelldatei | 1 KB |

Das Frontend war im Vergleich einfach. Man öffnet ein Terminal im Frontend-Verzeichnis und gibt diese Befehle der Reihe nach ein: `npm install`, `npm run codegen` und zum Schluss `npm run dev`, um es zu starten. Wenn man einmal `npm install` gemacht hat, muss man es nicht nochmal eingeben, wenn man das Frontend ein andersmal starten möchte.

3. ARCHITEKTUR

Die Spring Petclinic GraphQL Projekt verwendet die Microservice-Architektur, die verschiedene Services bereitstellt. Jeder Service ist für eine bestimmte Funktionalität verantwortlich und kommunizieren über einen zentralen API-Gateway miteinander.

3.1. ARCHITEKTURDIAGRAMM



3.1.1. CLIENT - KOMPONENTENBESCHREIBUNG

Der Client ist die Benutzeroberfläche der Anwendung. Es ermöglicht den Benutzern Anfragen an das API-Gateway zu senden. Zudem ist der Client verantwortlich für die Darstellung der Daten und für die Interaktionen mit dem Benutzer.

3.1.2. API-GATEWAY - KOMPONENT

Der API-Gateway ist der zentrale Punkt für alle eingehenden Anfragen des Clients. Die Anfragen, die am API-Gateway ankommen, werden an die entsprechenden Services weitergeleitet und die Antwort, die man eventuell vom jeweiligen Service erhält, bearbeitet. Zusätzlich dient diese Schnittstelle als eine Art Sicherheitsfunktion, welche Authentifizierungen und Lastverteilungen übernimmt.

3.1.3. SERVICE-DISCOVERY (EUREKA)

Der Service Discovery oder auch Eureka genannt, ermöglicht die automatische Erkennung und Registrierung der Microservices. Alle Services in der Anwendung melden sich bei der Eureka an, und können andere Services über die Eureka finden.

3.1.4. SERVICES

3.1.4.1. VET-SERVICE

Der VET-Service in der Anwendung ist für die Verwaltung von den Tierärzten zuständig. Dieser Service ermöglicht die Abfrage, Erstellen, Aktualisieren und Löschen von den Informationen der Tierärzte.

3.1.4.2. OWNER-SERVICE

Der OWNER-Service in der Anwendung ist für die Verwaltung von den Haustierbesitzern zuständig. Dieser Service ermöglicht es Daten wie Name, Adresse und andere Details des Besitzers zu verwalten.

3.1.4.3. PET-SERVICE

Der PET-Service in der Anwendung ist für die Verwaltung von den Haustieren zuständig. Dieser Service ermöglicht die Verwendung von Funktionalitäten wie das Abfragen, Erstellen, Aktualisieren und Löschen von den Daten der Haustiere.

3.1.4.4. VISIT-SERVICE

Der VISIT-Service in der Anwendung ist für die Verwaltung von Besucherinformationen zuständig. Dieser Service ermöglicht das Abfragen von Tierarztbesuchen, einschließlich Datum, Uhrzeit und den Zweck des Besuchs.

3.1.4.5. SPECIALITY-SERVICE

Der SPECIALITY-Service in der Anwendung ist für die Verwaltung der Spezialgebiete von den Tierärzten zuständig. Dieser Service ermöglicht das Hinzufügen, Aktualisieren und Abrufen von Informationen zu den Fachgebieten, in denen die Tierärzte tätig sind.

4. ANALYSE VON TESTDATEN

4.1. FRONTEND

Für die Anmeldung in der Anwendung stehen zwei User zur Verfügung. Der User mit dem Username 'susi' besitzt Admin-Rechte.

Users

Choose one of the following users for login:

| Username | Password | Role |
|----------|----------|--------------|
| susi | susi | ROLE_MANAGER |
| joe | joe | ROLE_USER |

Auf der Homepage der Website befindet sich nur ein Bild. Andere Funktionalitäten werden hier nicht Angeboten.





Welcome to PetClinic!

Welcome



Unter der Kategorie “Owner“ können detaillierte Kundeninformationen eingesehen werden. ASC und DESC sortieren die Reihenfolge der Kundeninformationen entsprechend dem Nachnamen aufsteigend oder absteigend.



HOMEOWNERSVETERINARIANS

Search Owner

Last Name

FIND


Owners


ASCDESC

| Last name | First name | Address | City | Telephone | Pets |
|-----------|------------|-----------------------------|-----------|-------------------|-------|
| Li | Jing | 101 Forbidden Palace Avenue | Beijing | +86 136 4567 8901 | Loki |
| Lindgren | Oskar | 909 Fika Lane | Stockholm | +46 70 456 78 90 | Toby |
| Liu | Mei | 202 Great Wall Lane | Beijing | +86 136 5678 9012 | Scout |

LOAD MORE

Unter der Kategorie “Veterinarians“ findet man eine detaillierte Auflistung aller Tierärzte.



HOMEOWNERSVETERINARIANS

Manage Veterinarians

All Veterinarians

| Name | Specialities |
|-----------------|--------------------|
| Carter, James | |
| Douglas, Linda | dentistry, surgery |
| Dubois, Sophie | |
| Jenkins, Sharon | |
| Leary, Helen | radiology |
| Ortega, Rafael | surgery |
| Petrova, Elena | |
| Sahin, Melih | radiology |

Um die Liste der Patienten zu sehen, die von einem bestimmten Tierarzt behandelt wurden, wählt man den entsprechenden Arzt aus der obenstehenden Liste aus. Anschließend gelangt man zu den Details der behandelten Tiere.

Treatments of John Smith

| Date | Pet | Owner |
|------------|---------|----------------------------------|
| 2022/01/15 | Bella | John Smith |
| 2022/09/05 | Cooper | Anita Müller |
| 2023/05/26 | Sadie | Xi Chen |
| 2024/02/01 | Stella | Isabella Gomez |
| 2024/10/19 | Duke | Elsa Larsson |
| 2022/06/15 | Bentley | Takashi Yamamoto |
| 2023/02/22 | Rocky | Amit Kumar |
| 2023/11/06 | Jack | Emma Williams |
| 2024/07/27 | Zeus | Lotte Schneider |

Hier besteht auch die Möglichkeit einen neuen Tierarzt einzufügen, jedoch kann es in diesem Fall nur der User Susi, weil sie Admin rechte besitzt.

You can add a new veterinary here. Note that this is only allowed for users with role `ROLE_MANAGER`

ADD VETERINARY

Nach einer erfolgreichen Erstellung, wird man weitergeleitet in die Vorherige Seite.
Notiz: Das Age wurde von uns zu einem späteren Zeitpunkt hinzugefügt, eine Erklärung dazu ist unter dem Kapitel „Architekturen Erweiterung“ zu finden.

Manage Veterinarians

Add Veterinary

First name

Last name

Age

Specialties

234

- radiology
- surgery
- dentistry
- invalid (use to see errors in response)

SAVE

CANCEL

Für die Eingaben existieren Validierung und generische Fehlermeldungen.

First name

Please enter a valid first name

Last name

Please enter a valid last name

Variable 'input' has an invalid value: Erwartet wurde ein Wert, der in den Typ 'Int' konvertiert werden kann, aber es war ein 'String'

Ein Beispiel dafür, dass User wie Joe, die keine Admin-Rechte haben, auch keine neuen Ärzte erstellen können.

Add Veterinary

First name

Last name

Age

Specialties

- radiology
- surgery
- dentistry
- invalid (use to see errors in response)

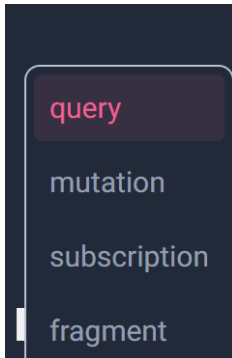
SAVE

CANCEL

Forbidden

4.2. BACKEND

Über den folgenden Link ist das Interface erreichbar, um direkte Anfragen an die Datenbank zu stellen: <http://localhost:9977/?path=/graphql&wsPath=/graphqlws>



Mit den gelisteten 4 Optionen wird der Start eines neuen Befehls angegeben. Mit der Option Query, kann man die Daten fetchen, mit Mutation die Daten ändern und mit Subskription bleiben die Daten bestehen, jedoch kann man sie zur Laufzeit ändern.

Mit der Option Fragment kann die Logik erstellt werden, um mit anderen drei Optionen teilen. Danach kann irgendein Name für den Befehl eingegeben werden.

```
query Name|
```

Anschließend folgt immer dieselbe Vorgehensweise: Mit geschweiften Klammern {} kannst du eine Tabelle aufrufen. Innerhalb dieser kannst du dann mit weiteren geschweiften Klammern {} zum Beispiel ein Attribut der Tabelle verwenden. Mit runden Klammern () kannst du deine Ausgaben sortieren oder andere Filter anwenden.

```
query Name {owners (order: [{field: id}]) {edges {node {firstName}}}}
```

4.2.1. BEFEHL 1:

Hier wird angezeigt, dass die ersten beiden Besitzer ausgegeben wurden, deren Name mit "D" beginnt.

```
query find {
  owners(
    first: 2
    filter: { lastName: "d" }
    order: [{ field: lastName }, { field: firstName, direction: DESC }]
  ) {
    edges {
      cursor
      node {
        id
        firstName
        lastName
        pets {
          id
          name
        }
      }
    }
    pageInfo {
      hasNextPage
      endCursor
    }
  }
}
```

```
"data": {
  "owners": {
    "edges": [
      {
        "cursor": "T18x",
        "node": {
          "id": 17,
          "firstName": "Maria",
          "lastName": "da Silva",
          "pets": [
            {
              "id": 20,
              "name": "Bailey"
            },
            {
              "id": 67,
              "name": "Marley"
            }
          ]
        }
      },
      {
        "cursor": "T18y",
        "node": {
          "id": 4,
          "firstName": "Harold",
          "lastName": "Davis",
          "pets": [
            {
              "id": 5,
              "name": "Iggy"
            }
          ]
        }
      }
    ],
    "pageInfo": {
      "hasNextPage": true,
      "endCursor": "T18y"
    }
  }
}
```

4.2.2. BEFEHL 2

Beim ersten Befehl war der Cursor auf "T18y" gesetzt, der dem zweiten Kunden gehörte. Mit "after" sage ich in diesem Fall: Gib mir die nächsten zwei Kunden nach ihm.

```
query findAfter {
  owners(
    first: 2
    after: "T18y"
    filter: { lastName: "d" }
    order: [{ field: lastName }, { field: firstName, direction: DESC }]
  ) {
    edges {
      cursor
      node {
        id
        firstName
        lastName
        pets {
          id
          name
        }
      }
    }
    pageInfo {
      hasNextPage
      endCursor
    }
  }
}
```

```
"data": {
  "owners": {
    "edges": [
      {
        "cursor": "T18z",
        "node": {
          "id": 2,
          "firstName": "Betty",
          "lastName": "Davis",
          "pets": [
            {
              "id": 2,
              "name": "Basil"
            }
          ]
        }
      },
      {
        "cursor": "T180",
        "node": {
          "id": 12,
          "firstName": "Sophie",
          "lastName": "Dubois",
          "pets": [
            {
              "id": 15,
              "name": "Luna"
            },
            {
              "id": 62,
              "name": "Ollie"
            }
          ]
        }
      }
    ],
    "pageInfo": {
      "hasNextPage": true,
      "endCursor": "T180"
    }
  }
}
```

4.2.3. BEFEHL 3

Nachdem ein Besuch erfolgreich hinzugefügt wurde, werden die eingegebenen Daten erneut ausgegeben.

```
mutation addVisit {
  addVisit(input:{
    petId:3,
    description:"Check teeth",
    date:"2022/03/30",
    vetId:1
  }) {
    newVisit:visit {
      id
      pet {
        id
        name
        birthDate
      }
    }
  }
}
```

```
{
  "data": {
    "addVisit": {
      "newVisit": {
        "id": 200,
        "pet": {
          "id": 3,
          "name": "Rosy",
          "birthDate": "2011/04/17"
        }
      }
    }
  }
}
```

4.2.4. BEFEHL 4

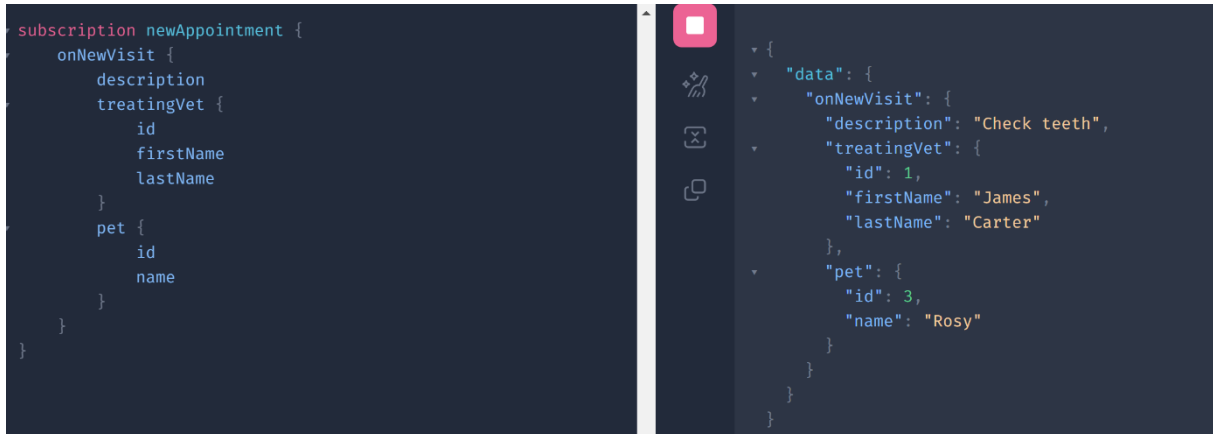
Nachdem ein neuer Tierarzt erfolgreich hinzugefügt wurde, werden die eingegebenen Daten erneut ausgegeben

```
mutation addVeterarian {
  addVet(input: {
    firstName: "Dagmar",
    lastName: "Smith",
    specialtyIds: [1, 3],
    age: 12}) {
    ... on AddVetSuccessPayload {
      newVet: vet {
        id
        specialties {
          id
          name
        }
      }
    }
    ... on AddVetErrorPayload {
      error
    }
  }
}
```

```
{
  "data": {
    "addVet": {
      "newVet": {
        "id": 202,
        "specialties": [
          {
            "id": 3,
            "name": "dentistry"
          },
          {
            "id": 1,
            "name": "radiology"
          }
        ]
      }
    }
  }
}
```

4.2.5. BEFEHL 5

Das funktioniert im Wesentlichen genauso wie die Abfrage (Query), mit dem Unterschied, dass die Verbindung bestehen bleibt. Das bedeutet, wenn sich etwas bei "onNewVisit" ändert, wird diese Änderung sofort in Echtzeit hier sichtbar.



```
subscription newAppointment {
  onNewVisit {
    description
    treatingVet {
      id
      firstName
      lastName
    }
    pet {
      id
      name
    }
  }
}
```

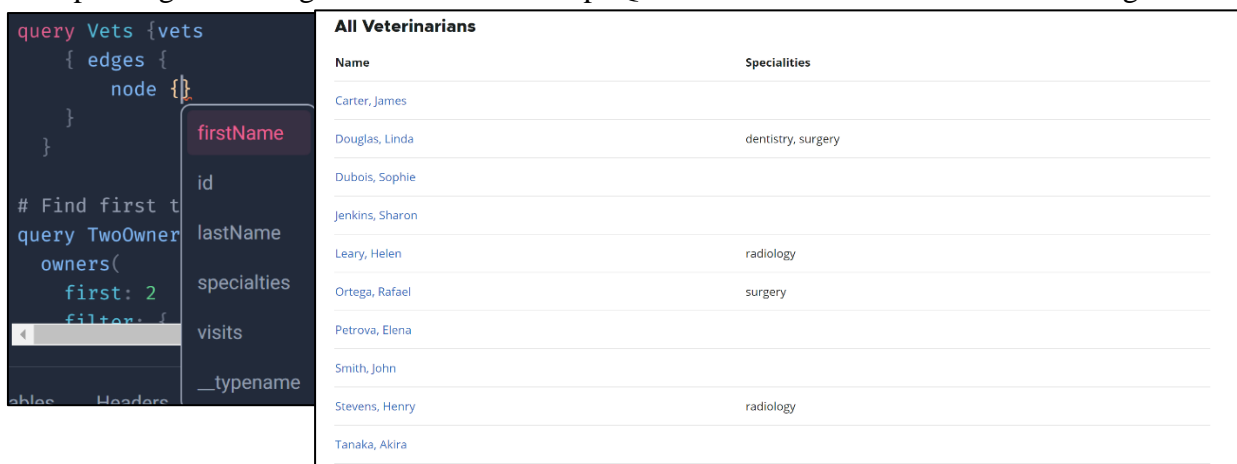
```
{
  "data": {
    "onNewVisit": {
      "description": "Check teeth",
      "treatingVet": {
        "id": 1,
        "firstName": "James",
        "lastName": "Carter"
      },
      "pet": {
        "id": 3,
        "name": "Rosy"
      }
    }
  }
}
```

5. ARCHITEKTURERWEITERUNG

5.1. ERWEITERUNG – PROPERTY ERWEITERUNG SQL

Nachdem wir uns mit den Daten vertraut gemacht haben und auch mehr über GraphQL erfahren haben, ist es an der Zeit, einen Teil des Codes zu erweitern. Nach Überlegung haben wir beschlossen, die Funktionalität zum Hinzufügen von Tierärzten anzupassen.

Wir haben entschieden, den Tierärzten ein Alter hinzuzufügen. Es soll lediglich als Eingabeparameter sichtbar sein, jedoch nicht auf der Website angezeigt werden. Zur Überprüfung des Erfolgs werden wir die GraphQL-API verwenden und das Alter abfragen.



```
query Vets {
  vets {
    edges {
      node {
        firstName
        id
        lastName
        specialties
        visits
        __typename
      }
    }
  }
}
```

Find first two owners

```
query TwoOwners {
  owners(
    first: 2
    filter: {
      # ...
    }
  ) {
    edges {
      node {
        # ...
      }
    }
  }
}
```

| Name | Specialities |
|-----------------|--------------------|
| Carter, James | |
| Douglas, Linda | dentistry, surgery |
| Dubois, Sophie | |
| Jenkins, Sharon | |
| Leary, Helen | radiology |
| Ortega, Rafael | surgery |
| Petrova, Elena | |
| Smith, John | |
| Stevens, Henry | radiology |
| Tanaka, Akira | |

Im Frontend wurde zunächst ein Eingabefeld erstellt, um das Alter einzugeben. Es wurde bewusst einfach gehalten

```
<Input
  type="text"
  {...register( name: "age", options: {required: true})}
  label="Age"
  error={errors.age && "Please enter a valid age"}
/>
```

AddVetForm.tsx

Anschließend wurde das Formular angepasst, um das Alter einzubeziehen.

```
type VetFormData = {
  firstName: string;
  lastName: string;
  specialtyIds: number[];
  age: number;
};
```

AddVetForm.tsx

Da wir das Formular geändert haben, wird hier klargestellt, dass das Alter mitgeschickt wird, wenn man auf "Speichern" klickt.

```
async function handleAddClick({ Show usages
  firstName,
  lastName,
  specialtyIds,
  age,
}: VetFormData) : Promise<void> {
  const result :... = await addVet( options: {
    variables: {
      input: {
        firstName: firstName,
        lastName: lastName,
        specialtyIds: specialtyIds || [],
        age: age,
      },
    },
  });
}
```

AddVetForm.tsx

Auch im Interface, das mit dem Hinzufügen von Tierärzten zusammenhängt, müssen wir nun das Feld "age" hinzufügen, damit die Datenstruktur korrekt ist. Dabei ist "age" in unserem Fall vom Typ Zahl und "input" bedeutet, dass der Benutzer etwas eingegeben hat.

```
export interface AddVetInput {  Show usages
  age: Scalars["Int"]["input"];
  firstName: Scalars["String"]["input"];
  lastName: Scalars["String"]["input"];
  specialtyIds: Array<Scalars["Int"]["input"]>;
}
```

graphql-types.tsx

Im Backend müssen wir die Datenbankstruktur abändern, um den Wert zu speichern.

```
CREATE TABLE vets
(
  id          integer generated_by default as identity,
  first_name  VARCHAR(30),
  last_name   VARCHAR(30),
  age         INTEGER default 0,
  primary key (id)
)
CREATE INDEX vets_last_name ON vets (last_name);
```

create-schema.sql

Was lange übersehen wurde, war diese Datei. Wenn hier das "age" nicht hinzugefügt wird, erscheint die Meldung, dass "age" nicht gefunden wurde. Diese Datei dient wahrscheinlich als Schnittstelle zwischen Java und der Datenbank.

```
" A Veterenarian"
type Vet implements Person {
  " The Veterenarian's first name"
  firstName: String!
  id: Int!
  " The Veterenarian's last name"
  lastName: String!
  " What is this Vet specialized in?"
  specialties: [Specialty!]!
  " All of this Vet's visits"
  visits: VisitConnection!
  " The age of the Vets"
  age: Int!
}
```

petclinic.graph.qls

Im Frontend wurde das Interface für "AddVetInput" angepasst, daher muss dies auch hier geschehen. Beachte: Normalerweise sollte nichts in "petclinic.graphqls" geändert werden, da es automatisch generiert wird. Das Problem war jedoch, dass wir es nicht geschafft haben, die automatische Generierung zu aktivieren, wenn Änderungen an unserer Struktur vorgenommen wurden.

```
input AddVetInput {  
  firstName: String!  
  lastName: String!  
  specialtyIds: [Int!]!  
  age: Int!  
}
```

petclinic.graph.qls

Hier passen wir unser POJO an. Natürlich wurden auch entsprechende Getter und Setter erzeugt.

```
@Entity  
@Table(name = "vets")  
public class Vet extends Person {  
  
    @ManyToMany(fetch = FetchType.EAGER) 5 usages  
    @JoinTable(name = "vet_specialties", joinColumns = @JoinColumn(name = "vet_id"),  
        inverseJoinColumns = @JoinColumn(name = "specialty_id"))  
    private Set<Specialty> specialties;  
  
    @Column(name = "age") 3 usages  
    @NotNull  
    private Integer age;
```

Vet.java

Wenn die Daten vom Frontend geschickt werden, gelangen sie hier. Wenn wir beim Service einen neuen Vet erstellen, muss hier nur noch das age hinzugeschrieben werden.

```
@MutationMapping
public AddVetPayload addVet(@Argument AddVetInput input) {
    try {
        Vet newVet = vetService.createVet(
            input.getFirstName(),
            input.getLastName(),
            input.getAge(),
            input.getSpecialtyIds());

        return new AddVetSuccessPayload(newVet);
    } catch (InvalidVetDataException ex) {
        return new AddVetErrorPayload(ex.getLocalizedMessage());
    }
}
```

VetController.java

Hier fügen wir nur den Attribute age hinzu. Im Grunde genommen wird diese Klasse vom Controller aufgerufen und fügt dann die Daten in die Datenbank ein.

```
@Transactional 1 usage
@PreAuthorize("hasAuthority('SCOPE_MANAGER')")
public Vet createVet(String firstName, String lastName, Integer age, List<Integer> specialtyIds) throws InvalidVetDataException {
    Vet vet = new Vet();
    vet.setFirstName(firstName);
    vet.setLastName(lastName);
    vet.setAge(age);
    for (Integer specialtyId : specialtyIds) {
        log.info("Specialty Id '{}'", specialtyId);
        Specialty specialty = specialtyRepository.findById(specialtyId)
            .orElseThrow(() -> new InvalidVetDataException("Specialty with Id '%s' not found", specialtyId));
        log.info("Specialty '{}'", specialty);
        vet.addSpecialty(specialty);
    }

    vetRepository.save(vet);

    log.info("VET {}", vet);

    return vet;
}
```

VetService.java

5.2. ERGEBNIS

Im Frontend haben wir nun das age. Ein Limit haben wir nicht implementiert, da es hier um das Konzept vom GraphQL geht.

First name

Last name

Age

Specialties

radiology

surgery

dentistry

invalid (use to see errors in response)

SAVE

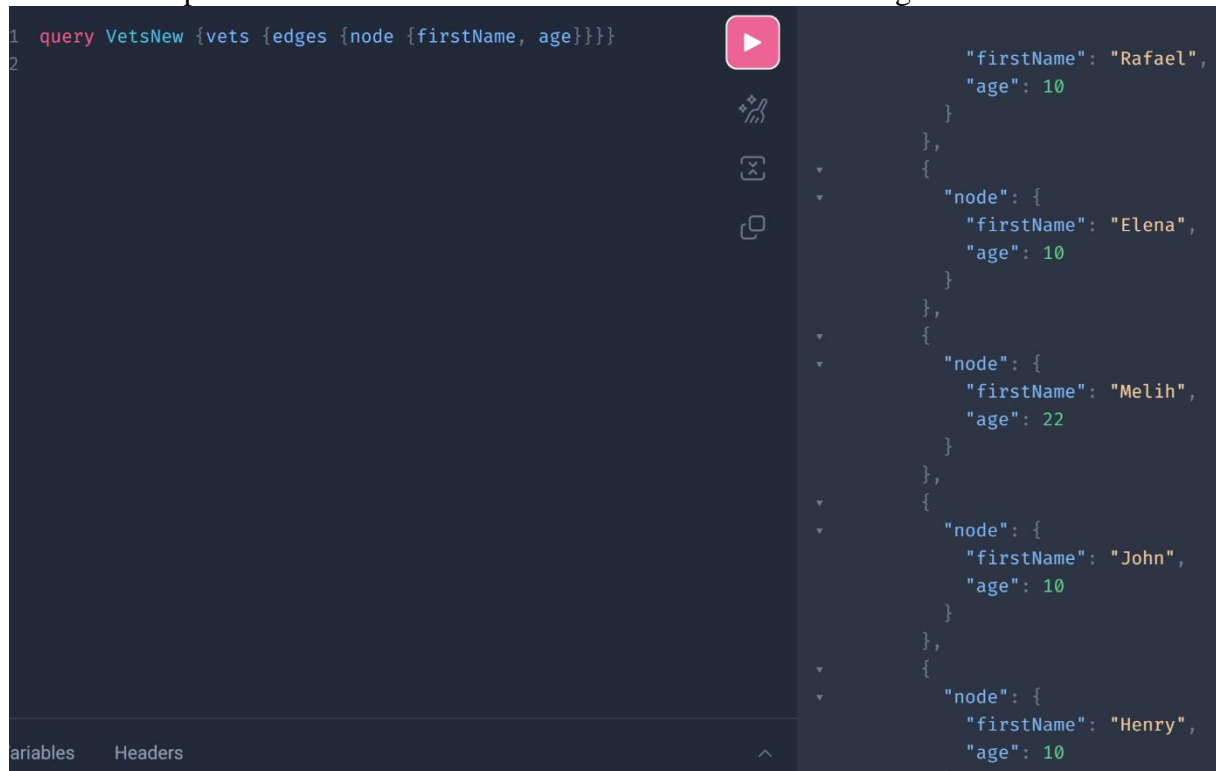
CANCEL

Nach dem Speichern sieht man hier, dass Melih hinzugefügt worden ist.

| Name | Specialities |
|---------------------------------|--------------------|
| Carter, James | |
| Douglas, Linda | dentistry, surgery |
| Dubois, Sophie | |
| Jenkins, Sharon | |
| Leary, Helen | radiology |
| Ortega, Rafael | surgery |
| Petrova, Elena | |
| Sahin, Melih | radiology |
| Smith, John | |
| Stevens, Henry | radiology |
| Tanaka, Akira | |

Wir haben eine einfache Abfrage gemacht. Wie man sieht, erscheint der User Melih mit dem richtigen Alter. Somit war die Erweiterung erfolgreich. Als nächstes könnte man sich ansehen wie zum Beispiel das Alter in der Übersicht von den Tierärzten anzeigen lassen könnte.

```
1 query VetsNew {vets {edges {node {firstName, age}}}}
2
```



```
{
  "vets": {
    "edges": [
      {
        "node": {
          "firstName": "Rafael",
          "age": 10
        }
      },
      {
        "node": {
          "firstName": "Elena",
          "age": 10
        }
      },
      {
        "node": {
          "firstName": "Melih",
          "age": 22
        }
      },
      {
        "node": {
          "firstName": "John",
          "age": 10
        }
      },
      {
        "node": {
          "firstName": "Henry",
          "age": 10
        }
      }
    ]
  }
}
```

6. FAZIT

Die Arbeit an dem Projekt "Petclinic GraphQL" war eine sehr gute Erfahrung für uns. Durch die Arbeit an solch einem großen Projekt konnten wir unseren Horizont und unser Wissen erweitern. Die detaillierte Auseinandersetzung mit den technischen Details und deren Umsetzung hat uns dabei geholfen, dass in den Vorlesungen und Seminaren Gelernte besser verstehen und nachvollziehen zu können.

Vor allem die Erweiterung der bestehenden Funktionen in einem realen Projekt wie in dieser Simulation eines Tierarztpraxis hat uns einen neuen Aspekt der Softwareentwicklung nähergebracht. Wir haben gemerkt, wie wichtig eine gründliche und sorgfältige Planung und Umsetzung ist. Durch diese Erfahrung haben wir uns nicht nur vom technischen Aspekt aus weiterentwickelt, sondern auch unser Verständnis und die Herausforderung, die in der Praxis auftreten können, vertieft.

Insgesamt sind wir zufrieden mit den erzielten Ergebnissen und wir hoffen, dass diese Dokumentation eventuell anderen Entwicklern als Nutzen dienen kann.