BEVOR ES LOSGEHT . . .

NPM & Node.js installiert?

Code Editor mit Syntax-Highlighting?

Laptop mit aktuellem Browser?

Projekt herunterladen

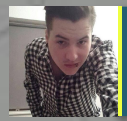
npm install

npm start

localhost:3000

localhost:3000/voyager

GraphQL Grundlagen



Dennis Dubbert
@ddubbert

Vimal Darius Seetohul
@vseetohu

01

GRAPHQL GRUNDLAGEN

GraphQL Grundpfeiler
Vergleich zu REST

02

GRAPHQL CLIENT

GraphQL Konzepte
Voyager
Playground
Queries

03

GRAPHQL SERVER

Serverelemente
GraphQL-Yoga
Typdefinitionen
Resolver

04

FRAGEN

Fragen und Diskussion

01 02 03 04

GRAPHQL GRUNDLAGEN

GraphQL Grundpfeiler
Vergleich zu REST

GRAPHQL CLIENT

GraphQL Konzepte
Voyager
Playground
Queries

GRAPHQL SERVER

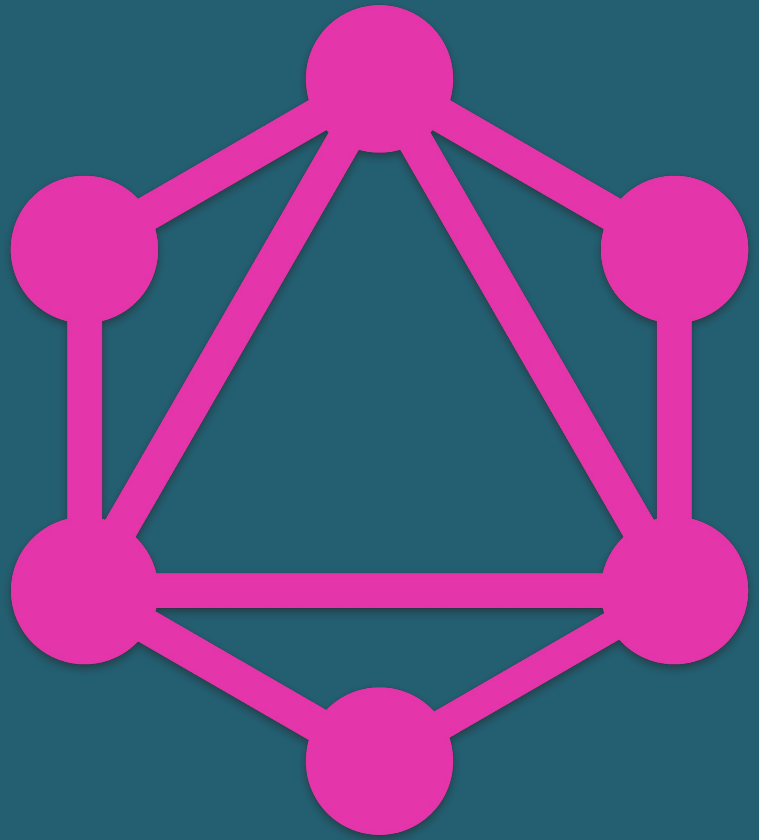
Serverelemente
GraphQL-Yoga
Typdefinitionen
Resolver

FRAGEN

Fragen und Diskussion

GraphQL

ist die Spezifikation einer **Abfragesprache** für APIs und einer serverseitigen **Laufzeitumgebung** für die Ausführung von Abfragen unter Verwendung eines **Typsystems**, welches für die abgefragten Daten definiert ist

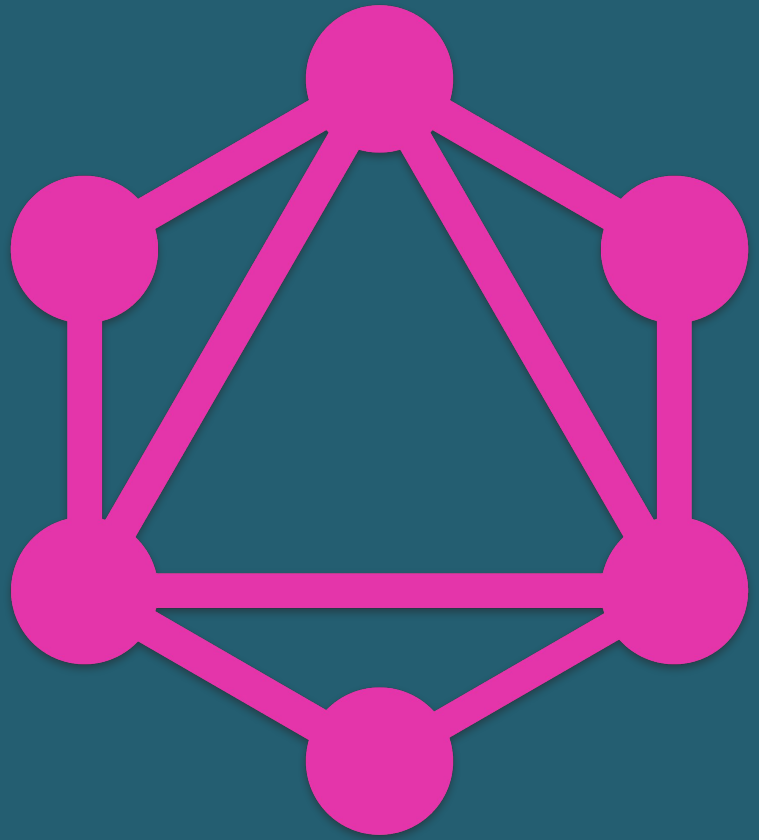


GraphQL

wurde von **Facebook** im Jahr 2012 aufgrund der Schwächen der bestehenden Architekturstile, wie REST und SOAP, und den derzeitigen Anforderungen entwickelt

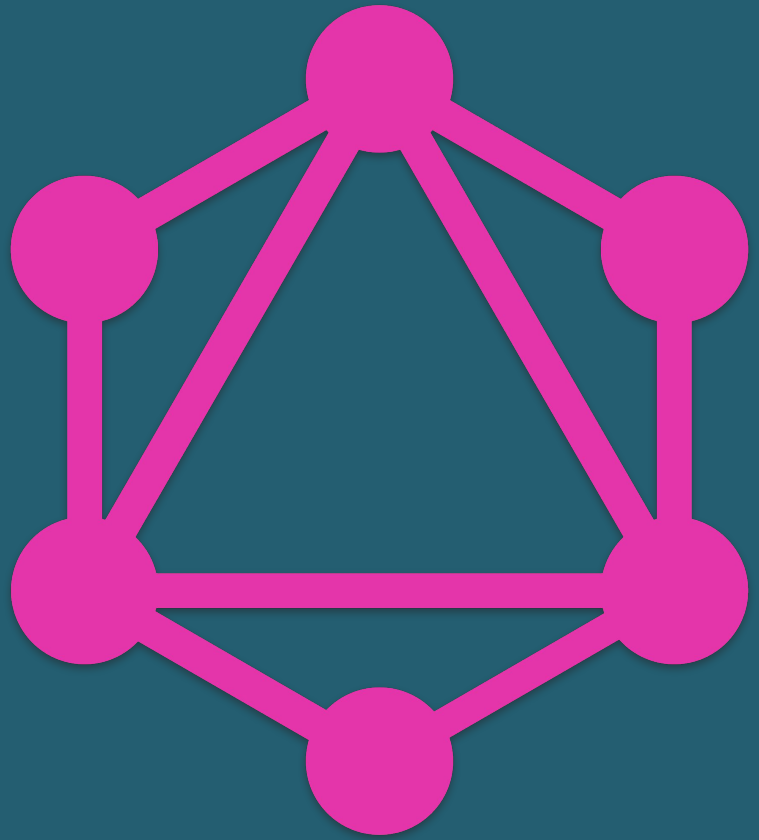
Open Source Spezifikation seit 2015

Gehostet von **Linux Foundation** 2018



GraphQL Grundpfeiler

- Stark typisierte Schnittstelle mit hierarchischem Aufbau
- Produktzentrierte Entwicklung (Anforderungen des Front-Ends zuerst / gegen Schnittstelle entwickeln)
- Client definiert bei Anfrage, welche Daten er haben möchte
- Introspektive Schnittstelle



REST GraphQL

REST	GraphQL
Endpunkte	Schema- und Typsystem (nur 1 Endpunkt)
CRUD-Operationen	Query, Mutation, Subscription
Server-driven	Client-driven
Langsam / Mehrere Anfragen	Schnell / eine Anfrage und spezifischer Aufruf
Ressourcenbasierte Anwendungen	Mehrere Microservices, mobile Anwendungen
Manuelle Dokumentation	Automatische Dokumentation


PayPal
sky
dailymotion
KLM

01

GRAPHQL GRUNDLAGEN

GraphQL Grundpfeiler
Vergleich zu REST

02

GRAPHQL CLIENT

GraphQL Konzepte
Voyager
Playground
Queries

03

GRAPHQL SERVER

Serverelemente
GraphQL-Yoga
Typdefinitionen
Resolver

04

FRAGEN

Fragen und Diskussion

Scalar Types

INT

Vorzeichenbehafteter 32-Bit Integer

FLOAT

Vorzeichenbehafteter Float mit doppelter Genauigkeit

STRING

UTF-8 Zeichenfolge

BOOLEAN

true oder false

ID

Eindeutige Kennung als String serialisiert

```
Address {
  street_name: String
  street_number: String
  city: String
  zip_code: Int
  country: String
}
```

Nullability

garantiert, dass non-nullable Attribute präsent (nicht null) sind bei einer GraphQL-Anfrage

```
name: String
name: String!

names: [String]
names: [String!]
names: [String]!
names: [String!]!
```

Object & Enum Types

OBJECT

Repräsentiert ein Objekt, dessen Attribute und ggf. dessen Beziehung zu anderen Objekten

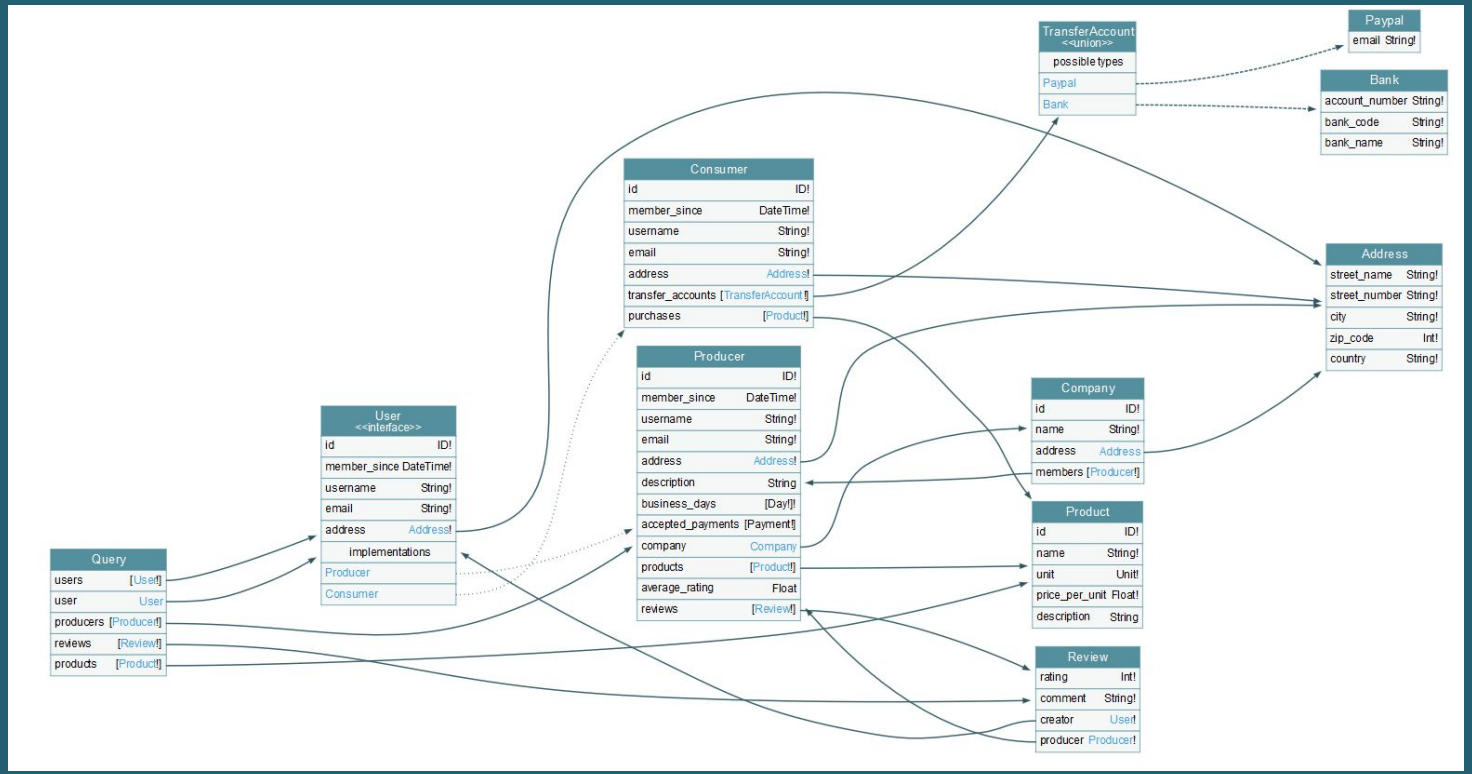
```
Company {
  id: ID!
  name: String!
  address: Address
  members: [Producer!]
}
```

Enum

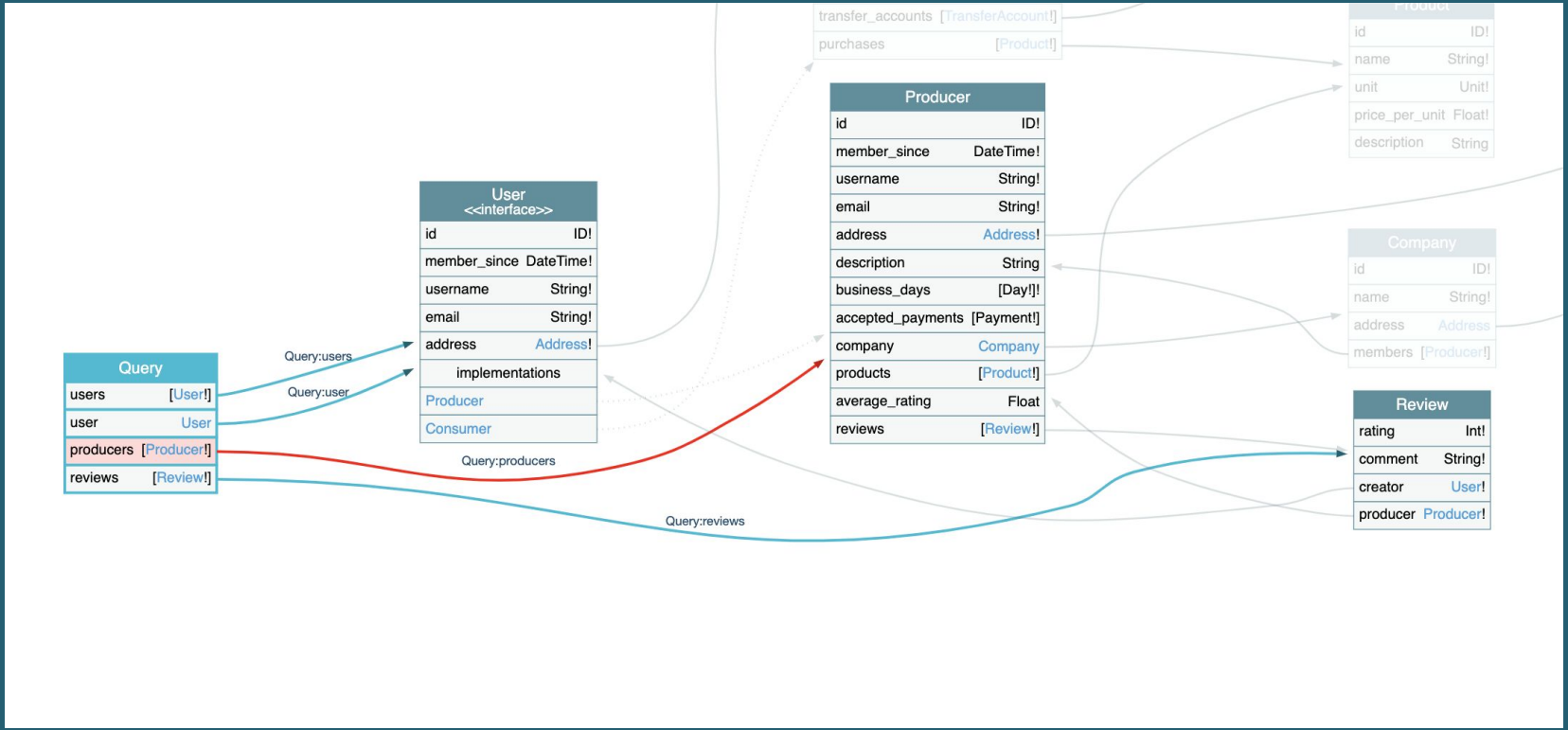
Aufzählungstyp mit endlichen und fest definierten Ausprägungen.

```
WorkingDay {
  MONDAY
  TUESDAY
  WEDNESDAY
  THURSDAY
  FRIDAY
}
```

GraphQL Voyager



GraphQL Query



GraphQL Query

Basic Query

```
{

  producers {
    id
    products {
      id
      ...
    }
    ...
  }

}
```

Named Query

```
query getProducers {

  producers {
    id
    products {
      id
      ...
    }
    ...
  }

}
```

Query mit Argumenten

```
query getProducers {

  producers(name:"Peter") {
    id
    products(name: "Apfel") {
      id
      ...
    }
    ...
  }

}
```

Playground Aufgaben 1+2

10 Minuten

1. Ermitteln sie mithilfe der user-Query die ID des Nutzers mit dem username "klaus-dieter". Rufen sie anschließend alle von diesem Nutzer erstellten Reviews ab (reviews-Query). Fragen sie hier alle Attribute an (bei creator und producer reicht jedoch jeweils der username).

2. Der Nutzer "klaus-dieter" hat einige Reviews erstellt, in welchen er dem Nutzer "peter-lustig" unbegründet schaden möchte. Nun ist Rache angesagt! Schreiben sie als "peter-lustig" ein Hate-Review gegen "klaus-dieter". Nutzen sie hierfür die createReview-Mutation. Benötigte ID's können mit der zuvor erstellten user-Query ermittelt werden.

Variables

VARIABLE

dient der Parametrisierung von Queries

```
mutation Review($producer:ID!, $creator:ID!){
  createReview(
    producer: $producer
    creator: $creator
  ){
    ...
  }
}
```

Query variables

```
{
    "producer": "da8ab4c0",
    "creator": "d467f50a"
}
```

Fragments

FRAGMENTS

ermöglichen es eine Menge von Attributen in verschiedenen Queries einzusetzen

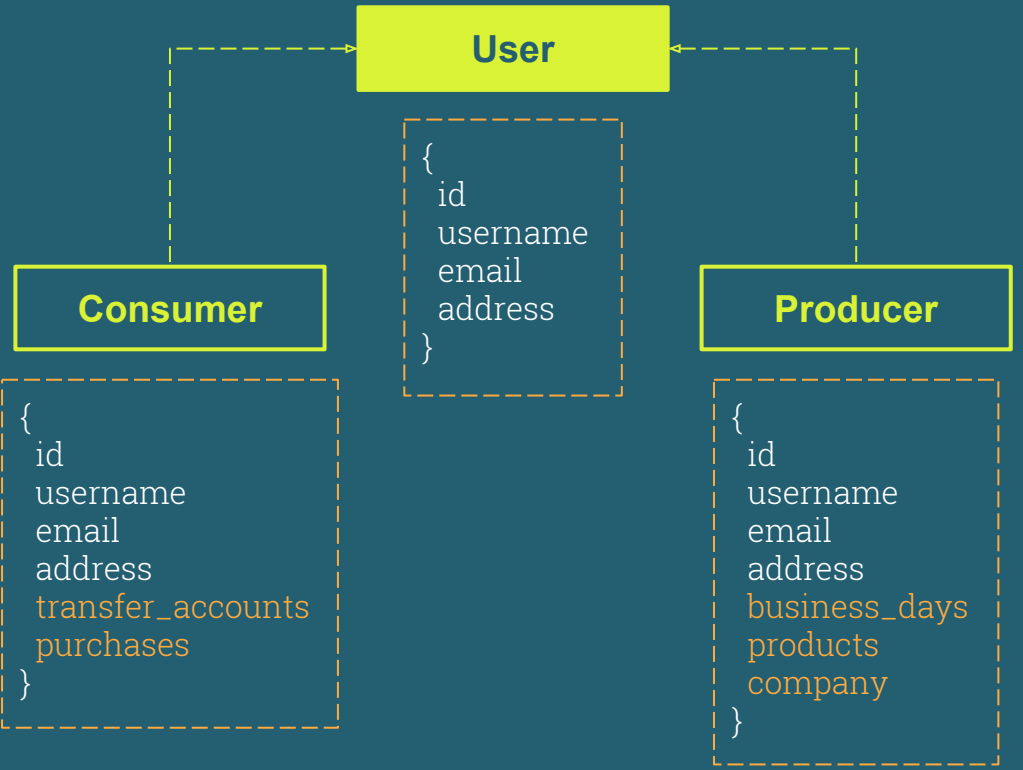
```
fragment UserFragment on User {
  id
  username
  address{
    street_name
    city
  }
}

query Users{
  users{
    ...UserFragment
  }
}
```

Interfaces

INTERFACE

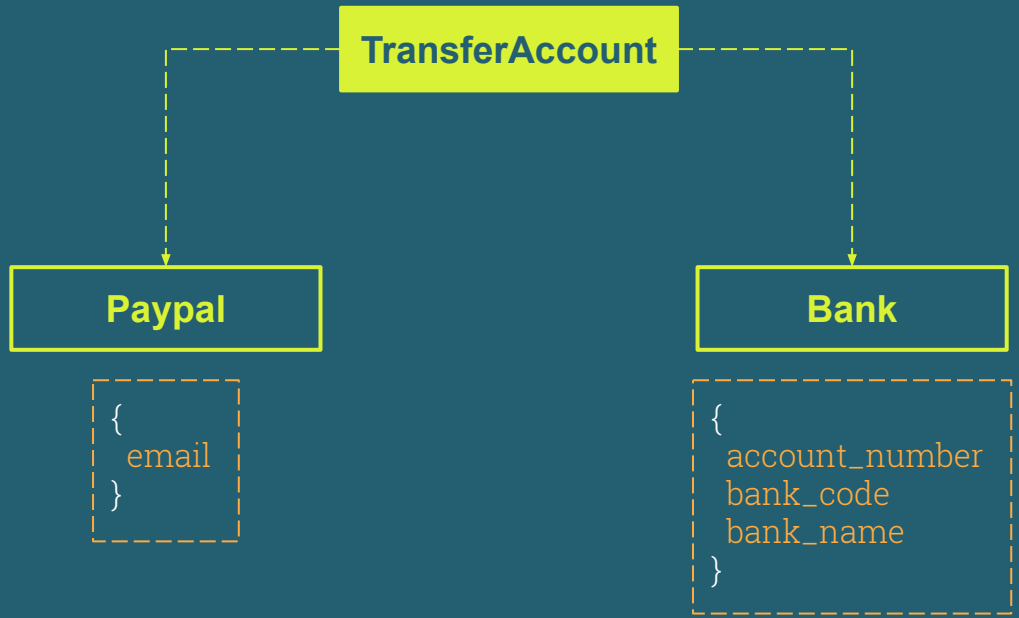
ist ein abstrakter Typ, welcher eine Kategorie von Typen darstellt. Definiert Attribute, die für alle Sub-Typen identisch sind.



Union Types

UNION TYPES

ähneln den Interfaces.
Sub-Typen besitzen jedoch keine gemeinsamen Attribute und werden lediglich für denselben Zweck verwendet.



Inline Fragments

Inline Fragments

Unterscheidung von Sub-Typen
und direkter Zugriff auf
Typ-spezifische Attribute.

Identisch für Interfaces und Union
Types.

```
query ... {
  transfer_accounts {
    ... on Paypal {
      email
    }

    ... on Bank {
      account_number
      bank_code
      bank_name
    }
  }
}
```

Playground Aufgaben 3+4

10 Minuten

1. Erstellen sie Fragments für die Typen User, Producer und Consumer und nutzen sie diese in den zuvor erstellten Queries und Mutations. (Optional: Nutzen sie auch Variablen)
   a. Das User-Fragment sollte alle Attribute eines Users beinhalten.
   b. Beim Producer-Fragment reichen die Attribute business_days, accepted_payments und products.
   c. Beim Consumer-Fragment müssen die Attribute transfer_accounts und purchases angegeben werden. Achten sie darauf, dass das Attribut transfer_accounts ein Union-Type ist (Inline-Fragments).

2. Nutzen sie nun die users-Query, um alle Nutzer mit den zu ihrer Rolle passenden Attributen auszugeben (Inline-Fragments). Welche Produkte bieten "peter-lustig" und "klaus-dieter" an? Welche Produkte wurden vom Herrn Paschulke bereits gekauft?

GraphQL Request-Struktur

```
POST /graphql HTTP/1.2
Host: localhost:3000/graphql
Content-Type: application/json
Body: {
   "query": "query User{ user(name:  "klaus-dieter"){ id username } }",
   "operationName": "User",
   "variables": {}
}
```

```
Response Body: {
   "data": {
      "user": {
         "id": "da8ab4c0",
         "username": "klaus-dieter"
      }
   }
}
```

01 GRAPHQL GRUNDLAGEN

GraphQL Grundpfeiler
Vergleich zu REST

02 GRAPHQL CLIENT

GraphQL Konzepte
Voyager
Playground
Queries

03 GRAPHQL SERVER

Serverelemente
GraphQL-Yoga
Typdefinitionen
Resolver

04 FRAGEN

Fragen und Diskussion

Aufgaben eines Servers

1

Schema

Datenstrukturen festlegen und Schnittstelle bereitstellen

2

Request interpretieren

Anfragen dekonstruieren und benötigte Datenstrukturen / Attribute ermitteln

3

Resolver

Funktionen implementieren, welche angeforderte Daten bereitstellen

4

Response zusammenbauen

Angeforderte Daten zu einem Response zusammensetzen und übermitteln

GraphQL-Yoga

ist eine GraphQL-Server Library für Node.js von Prisma (release v1.0: 01.2018). Sie baut auf verschiedenen erprobten Bibliotheken wie express, apollo-server und graphql.js auf und erleichtert durch eine Abstraktion der Vorgänge den Einstieg in GraphQL, sowie den Aufbau eines GraphQL-Servers.

Aufgaben bei der Verwendung von GraphQL-Yoga

1

Schema

Datenstrukturen festlegen und Schnittstelle bereitstellen

Request

Anfrage

Daten

ermitteln

3

Resolver

Funktionen implementieren, welche angeforderte Daten bereitstellen

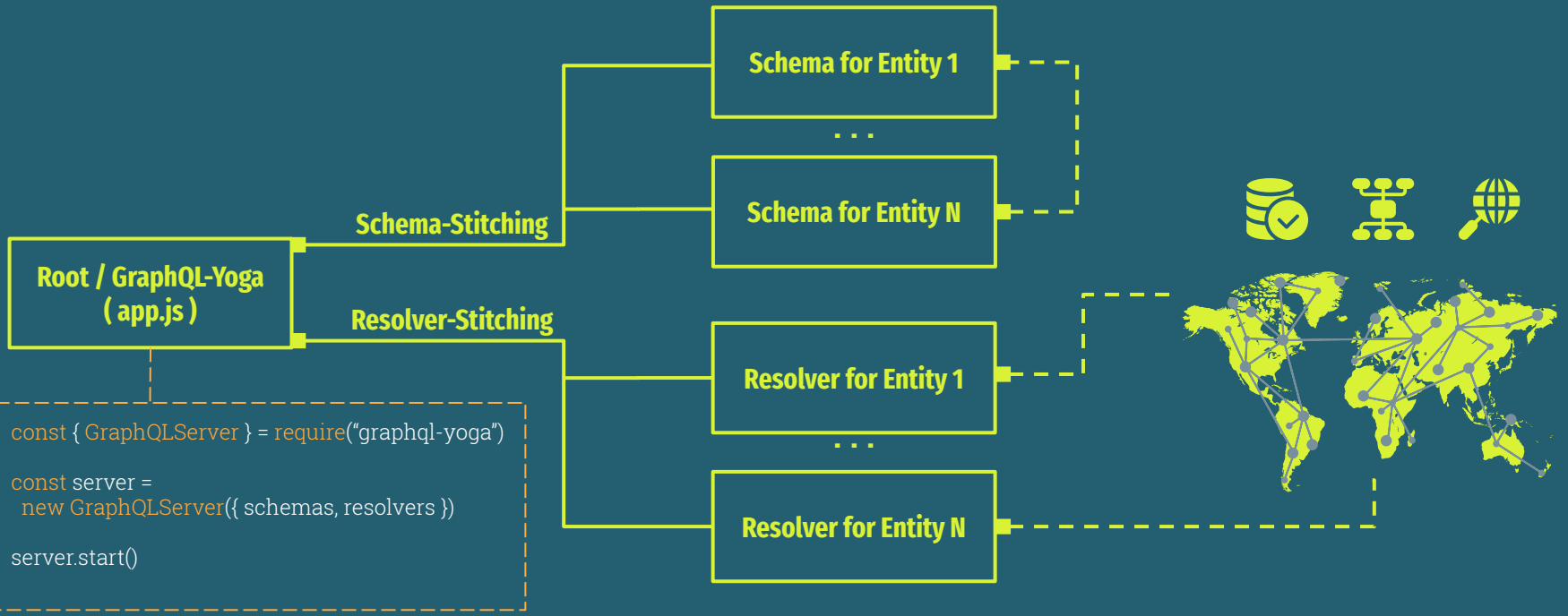
Response

Angef

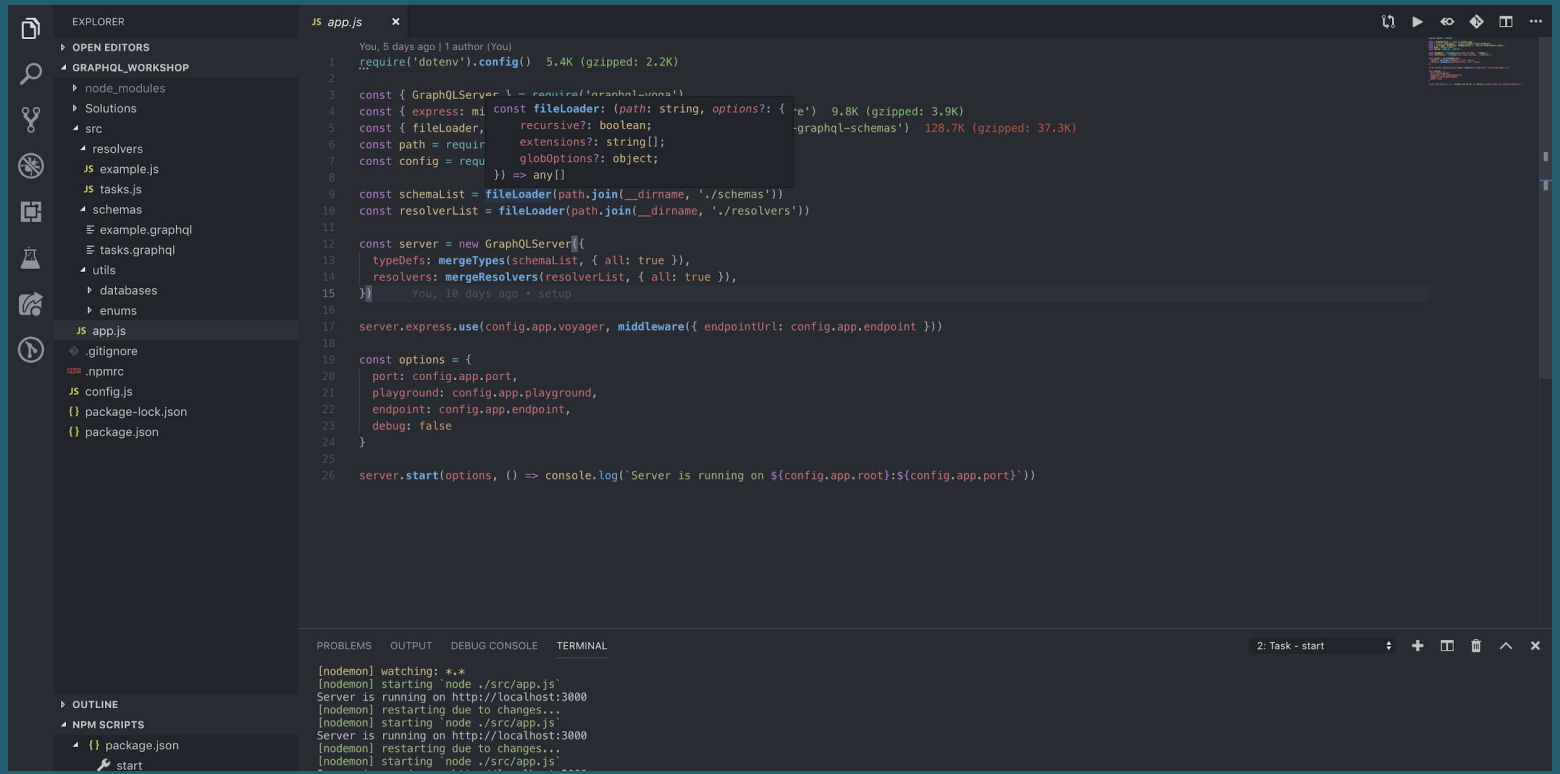
zus

übermitteln

Typischer Aufbau eines Servers mit GraphQL-Yoga



Unsere Projektstruktur



Object Types definieren

SIMPLE OBJECT

Definition einer einfachen Entität, welche lediglich aus Skalartypen besteht.

```
type User {
  id: ID!
  username: String!
  email: String!
}
```

Query Types definieren

Queries

definieren den Zugriff auf Daten. Besitzen einen Namen und einen Rückgabewert. Können zudem Argumente entgegennehmen.

```
type Query {

    users : [User!]

    user(id: ID!) : User

}
```

Mutation Types definieren

Mutations

definieren die Erstellung und Bearbeitung von Daten. Besitzen einen Namen und einen Rückgabewert. Nehmen immer Argumente entgegen.

```
type Mutation {

        updateUser(id: ID!, name: String, email: String) : User

}
```

Input Types definieren

Input Types

ermöglichen die Auslagerung von Argumenten in einen eigenen Typen.

```
input UserUpdateInput {
        name: String
        email: String
}
```

```
type Mutation {
        updateUser(id: ID!, input: UserUpdateInput) : User
}
```

Query / Mutation Resolver definieren

```
{
    Query: {
        QueryName: (parent, args, context, info) => {
            const data = // angeforderte Daten besorgen
            return data
        }
    }
}
```

parent

Beinhaltet das Objekt, was von einem vorherigen Resolver zurückgegeben wurde.

(Wird auf Folien 39 + 40 genauer erklärt)

args

Beinhaltet alle, von dem anfragenden Client, übergebenen Argumente.

context

Beinhaltet geteilten internen state, welcher in jedem Resolver erreichbar ist.

info

Beinhaltet genaue Informationen über die Anfrage und dessen momentanen Status.

Query / Mutation Resolver definieren

TypeDef

```
type Query {

    user(id: ID!) : User

}
```

Resolvers

```
const userDB = require("../database/user.db")

module.exports = {
    Query: {
        user: (_parent, args, _context, _info) => {
            const { id } = args // const id = args.id
            return userDB.getUserById(id)
        }
    },
    Mutation: {
        ...
    }
}
```

Server Aufgabe 1

15 Minuten

a. Erstellen sie in der Datei "src/schemas/tasks.graphql" den Object-Type Order. Dieser stellt eine einzelne Bestellung eines Produktes dar. Er benötigt folgende Attribute, wovon keines nullable ist:

id : Die id der Bestellung. Typ ID

product : Das Produkt was gekauft wurde. Typ ID (Definition in "examples.graphql")

amount : Die Anzahl von Einheiten des Produktes, welche gekauft wurden. Typ Int.

producer : Der Anbieter des Produktes. Typ ID (Definition in examples.graphql")

customer : Der Käufer des Produktes. Typ ID.

b. Erstellen sie in der Datei "src/schemas/tasks.graphql" eine Query namens orders. Diese Query soll ein Array aller Bestellungen für einen Produzenten ausgeben oder null, wenn keine passende Bestellung gefunden wurde. Hierzu muss der Query die ID des Produzenten als Argument übergeben werden. Nennen sie dieses Argument producer.

c. Erstellen sie in der Datei "src/resolver/tasks.js" einen Resolver für die soeben erstellte Query. In diesem Resolver müssen sie zunächst das Argument entgegen nehmen. Rufen sie anschließend die getOrdersForProducer(producerId)-Funktion des orderDB-Objektes auf, um alle Bestellungen eines Produzenten zu erhalten. Nutzen sie dieses Array als Rückgabewert der Query. Sollte dieses Array jedoch leer sein, so geben sie stattdessen null zurück.

d. Die Bestellungen sollen nun zusätzlich nach den Kriterien product und customer, beide vom Typ ID, gefiltert werden können. Erstellen sie hierzu einen Input-Type OrderFilter, welcher beide optionalen Attribute trägt. Ergänzen sie die Query nun um ein optionales Argument filter vom Typ OrderFilter.

e. Auch der Resolver muss das neue Argument filter nun abgreifen und als zweites Argument an die getOrdersForProducer-Funktion übergeben. Nun sollte die Query vollständig nutzbar sein. Testen sie diese im Playground mit den Werten producer = "d467f50a" und filter = { product: "b4867cbd" }.

Object Types verschachteln

EXTENDED OBJECT

Definition einer Entität, welche neben einfachen Skalartypen auch andere Objekttypen beinhaltet.

```
type User {
    id: ID!
    username: String!
    email: String!
    company: Company
}

type Company {
    id: ID!
    name: String!
    members: User!
}
```

Object Types Resolver definieren

```
{
    TypeName: {
        AttributeName: (parent, args, context, info) => {
            const { ParentAttribute } = parent
            const data = // angeforderte Daten besorgen
            return data
        }
    }
}
```

Benötigt wenn

Daten aus dem vorherigen Resolver nicht der Form der Schnittstelle entsprechen und weitere Berechnungen notwendig sind.

(Vorheriger Resolver = Query- oder Objekttyp-Resolver)

Nicht benötigt wenn

Daten aus dem vorherigen Resolver exakt der Schnittstelle entsprechen.

parent

Ausgabe des vorherigen Resolvers.

Wiederverwendung

Bei jedem Zugriff auf ein Attribut eines Typen wird dessen Resolver-Funktion aufgerufen. Durch die Verschachtelung und Wiederverwendung der Typen werden auch deren Resolver wiederverwendet.

Object Types Resolver definieren

TypeDef

```
type User {
        id: ID!
        username: String!
        email: String!
        company: Company
}
```

Database User

```
{
        id: "d467f50a",
        username: "peter-lustig",
        email: "peter@lustig.com",
        company: "371299b7"
}
```

Resolvers

```
module.exports = {
        Query: {
                user: (_parent, args, _context, _info) => {
                        const { id } = args
                        return userDB.getUserById(id)
                }
        },
        User: {
                company: (parent, _args, _context, _info) => {
                        const { company } = parent
                        return companyDB.getCompanyById(company)
                }
        }
}
```

Server Aufgabe 2

10 Minuten

a. Da eine GraphQL-Schnittstelle von der Verschachtelung und Wiederverwendung der Objekt-Typen profitiert, sollte auch der Order-Type dies tun. Ändern sie also im Schema den Typen des product-Attributes zu Product!, den Typen des producer-Attributes zu Producer! und den Typen des customer-Attributes zu User!.

b. Die Datenbankstruktur der Bestellungen bietet lediglich ID's für diese geänderten Attribute, sodass die zuvor erstellte Query zu Fehlern führt. Schreiben sie also für die Eigenschaften product, producer und customer des Order-Typen eigene Attribut-Resolver. Greifen sie in jedem Resolver auf die entsprechenden Attribute des parent-Objektes zu und nutzen sie diese, um die vollständigen Entitäten anzufragen. An Produkte gelangen sie mithilfe der getProductById(productId)-Funktion des productDB-Objektes. Producer und Customer können beide mithilfe der getUserById(userId)-Funktion des userDB-Objektes angefragt werden.

c. Testen sie nun die neue Datenstruktur mithilfe des Playgrounds.

Enum Types definieren

ENUM

Aufzählungstyp mit endlichen und fest definierten Ausprägungen.

```
enum Day {
        MONDAY
        TUESDAY
        WEDNESDAY
        THURSDAY
        FRIDAY
        SATURDAY
        SUNDAY
    }
```

Enum Types Resolver definieren

```
{
        TypeName: {
                Key_1: “Value_1”,
                Key_2: “Value_2”,
                Key_N: “Value_N”
        }
}
```

Key

Name der Ausprägung, welche im Schema definiert wurde.

Value

Value vom Typ String, welcher dem jeweiligen Key zugeteilt wird und diesen identifiziert.

Enum Types Resolver definieren

TypeDef

```
type User {
        id: ID!
        username: String!
        email: String!
        business_days: [Day!]
}
```

Database User

```
{
        id: "d467f50a",
        username: "peter-lustig",
        email: "peter@lustig.com",
        business_days:
           [ "monday", "tuesday" ]
}
```

Resolvers

```
module.exports = {
        Query: {
                ...
        },
        User: {
                ...
        },
        Day: {
                MONDAY: "monday",
                TUESDAY: "tuesday",
                WEDNESDAY: "wednesday",
                ...
        },
}
```

Interfaces definieren

```
interface User {
    id: ID!
    username: String!
    email: String!
}
```

```
type Producer  implements User {
    id: ID!
    username: String!
    email: String!
    business_days: [Day!]!
    products: [Product!]
    company: Company
}

type Consumer  implements User {
    id: ID!
    username: String!
    email: String!
    purchases: [Product!]
}
```

Union Types definieren

```
union TransferAccount =
    Paypal | Bank
```

```
type Paypal {
    email: String!
}

type Bank {
    account_number: String!
    bank_code: String!
    bank_name: String!
}
```

__resolveType Resolver definieren

```
{
        InterfaceName oder UnionName: {
                __resolveType: (object) => {
                        const typename = // Objekttyp herausfinden
                        return typeName
                }
        }
}
```

Unterscheidung

kann von unterschiedlichen Aspekten abhängig sein. Beispiele sind: vorhandene Attribute oder Attributausprägungen.

TypeName

ist der Name des Typen, wie er im Schema definiert ist, als String.

__resolveType Resolver definieren

Database User

```
{
        id: “d467f50a”,
        username: “peter-lustig”,
        email: “peter@lustig.com”,
        type: “producer”,
        business_days:
            [ “monday”, “tuesday” ],
        ...
}

{
        id: “d467f50a”,
        username: “peter-lustig”,
        email: “peter@lustig.com”,
        type: “consumer”,
        purchases: []
}
```

Resolvers

```
module.exports = {
        Query: {
                ...
        },
        User: {
                __resolveType: (user) => {
                        switch(user.type) {
                                case “producer”: “Producer”,
                                case “consumer”: “Consumer”,
                                default: throw Error(“Could not identify”)
                        }
                }
        },
}
```

Interfaces Resolver GraphQL-Yoga

Resolver werden bei GraphQL-Yoga noch **NICHT** von einem Interface an dessen Implementierungen vererbt und müssen somit für jede Sub-Entität selbst implementiert werden !

Resolvers

```
module.exports = {
        User: {
                __resolveType: (user) => {
                        ...
                }
        },
        Producer: {
                ...AlleResolverEinesUsers,
                ...AlleResolverEinesProducers
        },
        Consumer: {
                ...AlleResolverEinesUsers,
                ...AlleResolverEinesConsumers
        },
}
```

Server Aufgaben 3 + 4

Open End

Enum-Types & Resolvers:

a. Eine Bestellung kann per Post verschickt, oder beim Produzenten abgeholt werden. Ergänzen sie im Schema den Order-Typen um das Pflichtfeld type vom Typ OrderType, welches Auskunft über die Art der Bestellung liefern soll. Erzeugen sie hierzu den Enum-Type OrderType mit den Ausprägungen MAIL und PICKUP.

b. Erstellen sie einen Resolver für diesen Enum-Typen. Weisen sie der Ausprägung MAIL hierbei den String "mail" und der Ausprägung PICKUP den String "pickup" zu. Erweitern und testen sie im Playground nun ihre orders-Query, sodass sie auch dieses Attribut anfragen.

Interfaces & Resolvers:

a. Da eine Bestellung per Mail andere zusätzliche Attribute benötigt, als jene bei einer Abholung, bietet sich hier ein Interface an. Ändern sie im Schema den Order-Typen zu einem Interface und erstellen sie hierzu die zwei Sub-Typen MailOrder und PickupOrder. Eine MailOrder benötigt das zusätzliche Pflichtfeld shipping_address vom Typ Address. Eine PickupOrder benötigt das zusätzliche Pflichtfeld pickup_date vom Typ DateTime.

b. Erstellen sie nun Resolver für die Typen MailOrder und PickupOrder. Da in GraphQL-Yoga keine Vererbung der Resolver stattfindet, müssen sie die zuvor geschriebenen Resolver des Order-Typen auf diese beiden Sub-Entitäten übertragen (copy-paste). Fügen sie nun dem Order-Typen einen Type-Resolver hinzu (__resolveType: (order) => { // TODO }), in welchem sie definieren, wann eine Bestellung eine MailOrder und wann sie eine PickupOrder ist. Hierzu können sie, wie in den Beispielen, auf das type-Attribut zugreifen. Testen sie anschließend ob ihre Implementierung funktioniert, indem sie im Playground Inline-Fragments nutzen.

01 GRAPHQL GRUNDLAGEN

GraphQL Grundpfeiler
Vergleich zu REST

02 GRAPHQL CLIENT

GraphQL Konzepte
Voyager
Playground
Queries

03 GRAPHQL SERVER

Serverelemente
GraphQL-Yoga
Typdefinitionen
Resolver

04 FRAGEN

Fragen und Diskussion

QUELLEN

- Bilder:
  - https://www.pexels.com/
  - https://www.graphql.com/
  - https://insights.stackoverflow.com/trends
  - https://www.getpostman.com/
  - https://github.com/prisma/graphql-yoga
- https://goodapi.co/blog/rest-vs-graphql
- https://www.apollographql.com/docs/apollo-server/