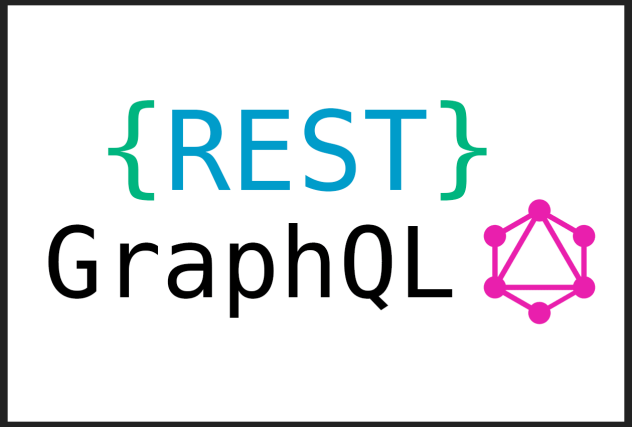
ERSETZEN VON REST DURCH GRAPHQL



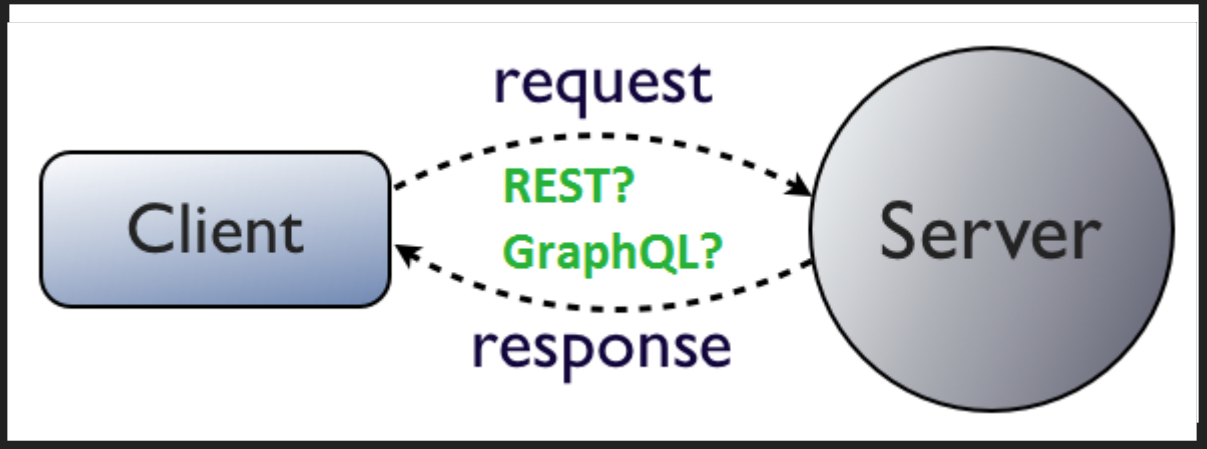
BACHELOR-THESIS ABSCHLUSSVORTRAG

LUKAS KUGLER, 23.03.2017

ÜBERSICHT

- Grundlagen zu REST
- GraphQL: Einführung
- GraphQL: Syntax
- Vergleich der Schnittstellen
- Demo

EINORDNUNG



GRUNDLAGEN ZU REST

- "Representational State Transfer"
- Programmierparadigma
- Existent seit 2000, erst später intensiv eingesetzt

EIGENSCHAFTEN VON REST

- Ziel: Vereinheitlichung der Kommunikation zwischen Webservices
- Einheitliche Schnittstelle
- Ressourcenzentriert
- HTTP-Methoden

HTTP-METHODEN

- **GET:** lesen
- **PUT:** erstellen
- **POST:** aktualisieren
- **DELETE:** löschen

WIE SIEHT EINE REST-SCHNITTSTELLE AUS?

Benutzer abfragen:

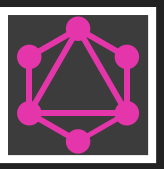
```
GET /api/users/1
```

Bestellungen eines Benutzers abfragen:

```
GET /api/users/1/orders
```

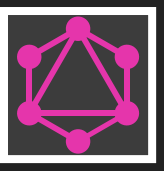
OVERHEAD

- Zurückgeben aller Artikeldetails mit der User-Anfrage?
- Stellen einer zusätzlichen Anfrage?



GRAPHQL: EINFÜHRUNG

- Abfragesprache ("Query Language")
- Entwickelt von Facebook
- Verließ "Technical Preview" im September 2016



GRAPHQL: EINFÜHRUNG

DESIGN-PRINZIPIEN

Hierarchisch

Eine GraphQL-Anfrage an sich ist eine hierarchische Anordnung von Feldern. Die Anfrage hat dieselbe Form, wie die Daten, die sie zurückgibt.

Produktorientiert

GraphQL ist speziell für die Anforderungen von Views und für Frontend-Entwickler gedacht.

Client-spezifizierte Anfragen

Die Spezifikation für Anfragen ist sehr granular im Client definiert, nicht im Server.

Abwärtskompatibel

Durch die Clientspezifizierten Anfragen ist eine Abwärtskompatibilität der API einfacher zu realisieren.

Strukturierter, beliebiger Code

GraphQL stellt dem Server eine Struktur zur Verfügung und öffnet einzelne Felder, die mit beliebigem Code hinterlegt sind.

Protokoll der Applikationsschicht

GraphQL ist ein Protokoll der Applikationsschicht und benötigt kein bestimmtes Transportprotokoll.

Streng Typisiert

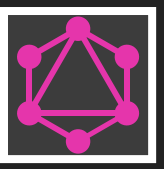
Anfragen können vor ihrer Ausführung auf syntaktische und und typbezogene Korrektheit überprüft werden.

Introspektiv

Clients und Tools können das Typsystem mit GraphQL-Syntax selbst abfragen.

GRAPHQL: UMSETZUNG

- Gesamte Schnittstelle über einen Endpunkt erreichbar
- Server öffnet bestimmte Einstiegspunkte
- GraphQL-String als Payload in der Anfrage
- String als Antwort (JSON)
- Alle Ressourcen und Anfragen streng typisiert



GRAPHQL: SYNTAX

DAS SCHEMA & RESOLVER

- Legt fest, welche Anfragen an den Server gestellt werden können
- Legt die Form der Daten fest
- Legt fest, was der Server an den Client als Antwort zurückgibt

DAS SCHEMA: TYPEN

```
type User {
  firstName: String!
  lastName: String!
  orders: [Order]
}
```

```
type Query {
  user(id: id!): User
}
```

Resolver

- Werden immer aufgerufen, wenn ein Typ abgefragt wird
- Beliebiger Code
- Für Skalare üblicherweise Zugriff auf Datenbank

Resolver-Funktion für user-Anfrage

```
user: async (obj, args, context) => {
  return await Users.findById(args.id);
}
```

Resolver für bestimmten Typ

```
User: {
        friends: async (user) => {
            return await Users.getUsersById(user.friends);
        }
    }
```

Eine einfache Anfrage...

```
user {
  firstName
}
```

... und die Antwort:

```
{
"data": {
  "user": {
    "firstName": "Lukas"
    }
  }
}
```

Mit Subselektionen:

```
user(id: "1") {
  firstName,
  lastName,
  orders(year: "2016") {
    sum
    date
    status
    articles {
      name
      price
      category
    }
  }
}
```

Antwort:

```
"user": {
  "firstName": "Lukas",
  "lastName": "Kugler",
  "orders": [
    {
      "sum": "3",
      "date": "07-01-2016"
      "status" "versandt"
      "articles": [
        {"name" : "Bleistift",
        "price" : "2",
        "category" : "Schreibwaren"
        },
        {"name" : "Block",
        "price" : "1",
        "category" : "Schreibwaren"
        }
      ]
    },
    {
      "sum": "15",
      "date": "07-01-2016"
      "status" "in Bearbeitung"
      "articles": [
        {"name" : "USB-Stick 32GB",
        "price" : "10",
        "category" : "Elektronik"
        },
        {"name" : "AAA-Batterien 6er Pack",
        "price" : "5",
        "category" : "Elektronik"
```

Mit Variable:

```
query userById($id: Int) {
  user(id: $id) {
    firstName
    lastName
  }
}
```

Antwort:

```
{
  "data": {
    "user": {
      "firstName": "Lukas",
      "lastName": "Kugler"
    }
  }
}
```

Daten anlegen:

```
mutation addNewUser($firstName: String, $lastName: String!) {
    addUser(firstName: $firstName, lastName: $lastName) {
        firstName
        lastName
    }
}
```

```
{
  "firstName": "Max",
  "lastName": "Mustermann"
}
```

VERGLEICH

TYPISCHES BENUTZER-OBJEKT

- id (String)
- username (String)
- firstName (String)
- lastName (String)
- statusMessage (String)
- password (String)
- friends ([User])
- admin (Boolean)

Auslesen aller Benutzer	GET /api/users
Auslesen eines Benutzers anhand seiner ID	GET /api/users/ID
Hinzufügen eines Benutzers	POST /api/users
Löschen eines Benutzers	DELETE /api/users/ID
Ändern des Passworts	POST /api/users/ID/password
Ändern der Statusnachricht	POST /api/users/ID/statusMessage
Einen Freund hinzufügen	POST /api/users/ID/friends

GET /api/users	Eine Query, welche ein Array aller User-Typen zurückgibt
GET /api/users/ID	Eine Query, welche ein User-Objekt anhand seiner ID zurückgibt
POST /api/users	Eine Mutation, welche ein User-Objekt erzeugt
DELETE /api/users/ID	Eine Mutation, welche ein User-Objekt anhand seiner ID löscht
POST /api/users/ID/password	Eine Mutation, welche das Feld Password eines Users ändert
POST /api/users/ID/statusMessage	Eine Mutation, welche das Feld statusMessage eines Users ändert
POST /api/users/ID/friends	Eine Mutation, welche dem Feld friends eines Users einen Eintrag hinzufügt

AUTHORISIERUNG (EXPRESS)

- Über Middleware mit JSON Web Tokens
- Endpunkt offen, geschützt wird in den Resolvern
- Query- und Field-Level Auth möglich dank Custom Resolvern
- Verschiedene Rechte-Ebenen möglich (Admin, User)
- Login: JWT kann über GraphQL abgefragt werden

DATENLAST

```
user(id: ID) {
  firstName,
  lastName,
  statusMessage,
  friends {
        firstName
        lastName
  }
}
```

DATENLAST

REST	GraphQL
Anfrage mit User-ID	Anfrage mit GraphQL-Query-String
5 Strings	3 Strings
1 Bool	
Array aus IDs	
Pro Freund:	Pro Freund:
Anfrage mit ID	2 Strings
5 Strings	
1 Bool	
Array aus IDs	

API-Versionierung

- Versionierung üblicherweise, da Client Daten erhält mit denen er nichts anfangen kann
- In GraphQL merkt der Client nichts von neuen Feldern
- Best Practice: Versionslose API
- Felder können @deprecated markiert werden
- Änderungen in der API können in hoher Frequenz vorgenommen werden

SUBSCRIPTIONS

- Push von Live-Daten vom Server an Clients
- PubSub-System
- Client sendet Anfrage mit Schlüsselwort
- Server reagiert auf Events (Queue, Timer, Änderungen)
- GraphQL-Syntax mit allen Vorteilen (Selektion)

FÜR ENTWICKLER

- Einfaches Mocking
- GraphiQL

Fragen?
Demo!