

---

# GraphQL

*A query language for your API*

## GraphQL?

1996

LUMBERJACK

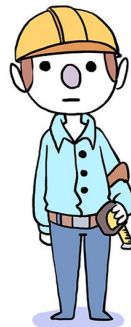


WEB DEVELOPER



2016

LUMBERJACK



WEB DEVELOPER

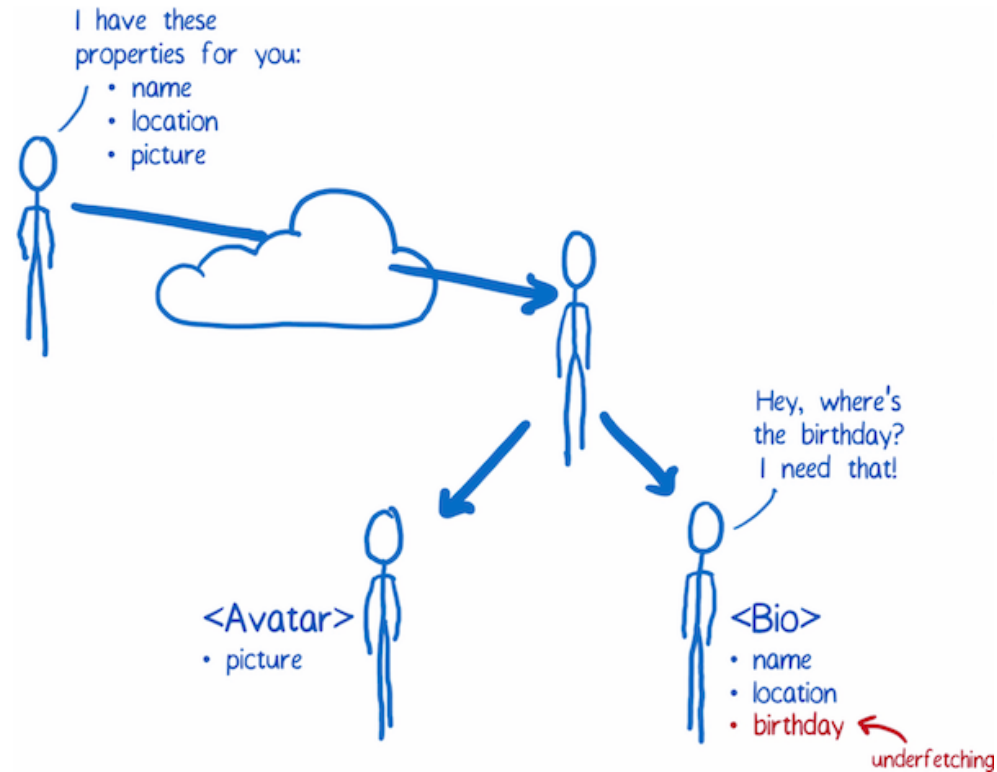


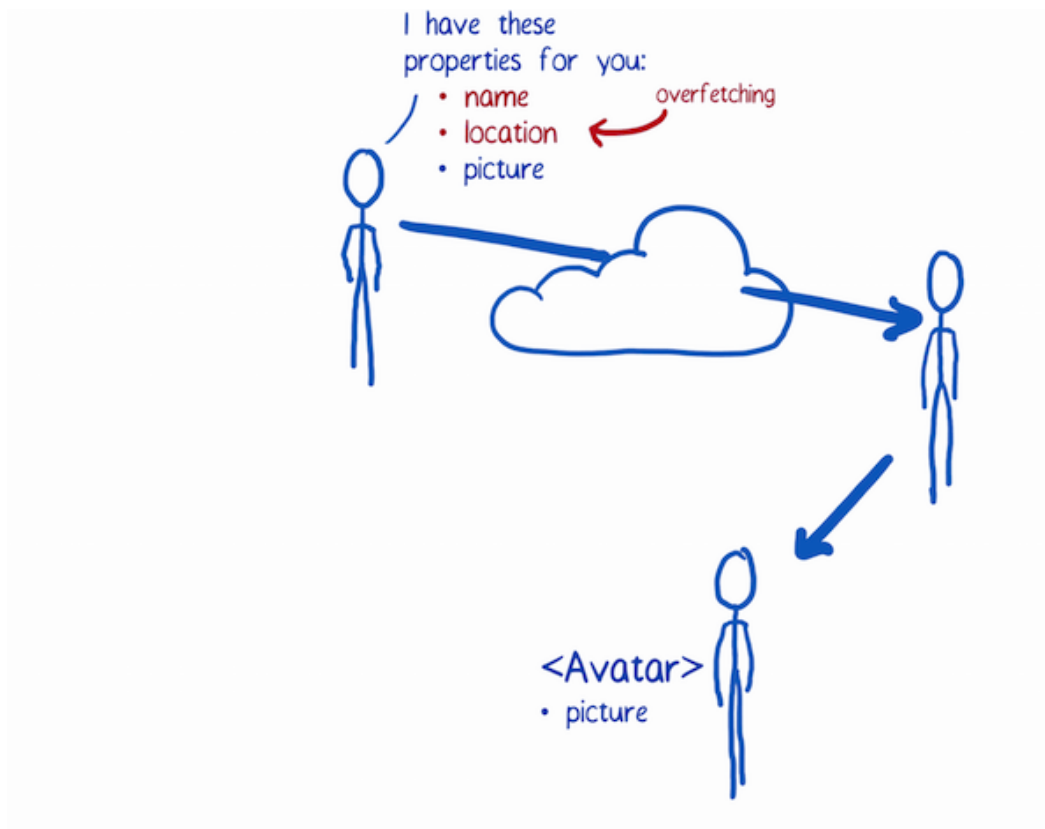
OWLTURD.COM 

## Anbieter/Benutzer

- Facebook
- GitHub
- Shopify
- Yelp
- Sky
- Pinterest
- Autoscout24
- Xing
- ....

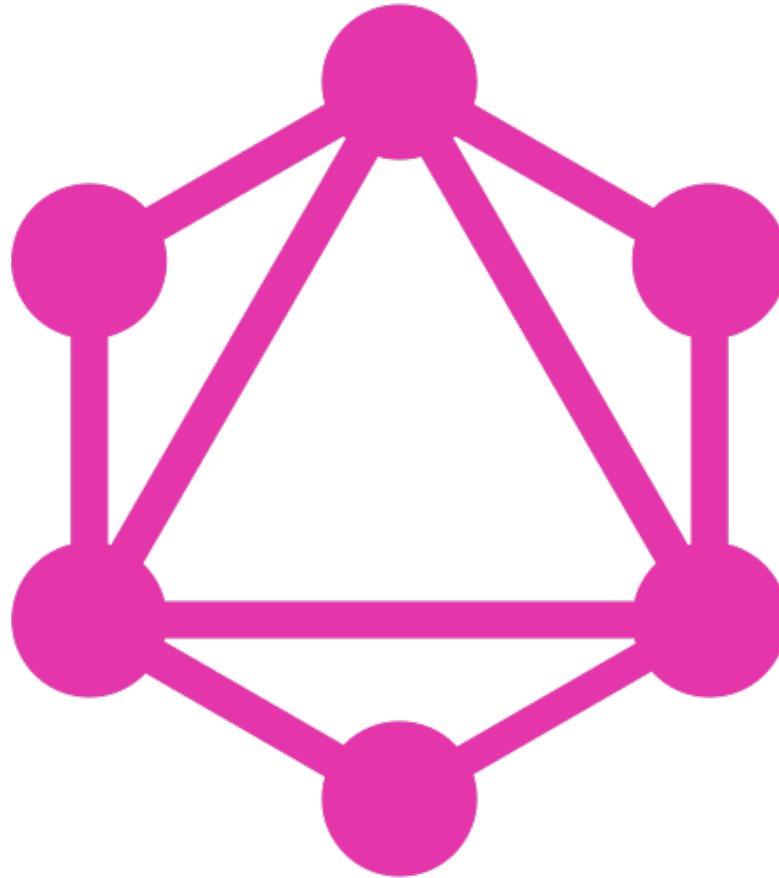
<https://olegilyenko.github.io/presentation-graphql-introduction>





Einführung

## GraphQL?



# *Agenda*

- > Einführung
- > Hintergrund
- > Demo
- > Einbindung
- > Vorteile + Probleme
- > Links

## Previously on API Design - SOAP

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 299
SOAPAction: "http://www.w3.org/2003/05/soap-envelope"

<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
xmlns:m="http://www.example.org/stock/Reddy">
  <soap:Header>
  </soap:Header>
  <soap:Body>
    <m:GetCharacterInfo>
      <m:CharacterName>Luke Skywalker</m:CharacterName>
    </m:GetCharacterInfo>
  </soap:Body>
</soap:Envelope>
```



## Previously on API Design – Rest

<u>URL</u>	<u>GET</u>	<u>PUT</u>	<u>PATCH</u>	<u>POST</u>	<u>DELETE</u>
<b>example.com/characters/</b>	Auflisten	Ersetzen	-	Erstelle neuen Charakter	Lösche alle Charaktere
<b>example.com/characters/12</b>	Alle Infos zu Charakter "123"	Ersetze / erstelle Charakter	Aktualisiere Charakter	-	Lösche einzelnen Charakter

## REST - parametrisiert

- Client 1: /characters
- Client 2: /characters?with\_lastnames
- Client 3: /characters?with\_creation\_time
- Client 4: /characters?with\_creation\_time\_and\_lastnames


## Previously on API Design

```
<?xml version="1.0" encoding="UTF-8" ?>
  <firstName>Luke</firstName>
  <lastName>Skywalker</lastName>
  <isAlive>true</isAlive>
  <address>
    <streetAddress>2nd Street</streetAddress>
    <state>Dusty</state>
    <planet>10021-3100</planet>
  </address>
  <spouse />
```

```
{
  "firstName": "Luke",
  "lastName": "Skywalker",
  "isAlive": true,
  "address": {
    "streetAddress": "21 2nd Street",
    "state": "Dusty",
    "planet": "10021-3100"
  },
  "spouse": null
}
```

## API-Design ist schwer

### How to Design a Good API and Why it Matters

**Joshua Bloch**  
Principal Software Engineer  


1 —How to Design a Good API and Why it Matters

### The Little Manual of API Design

Jasmin Blanchette  
Trolltech, a Nokia company

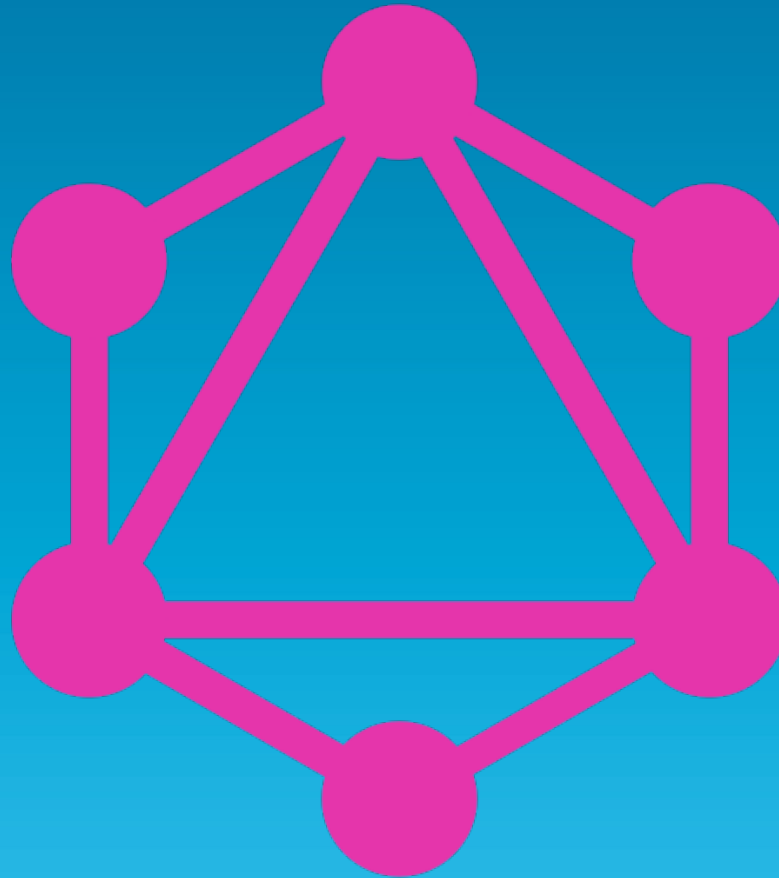
## GraphQL?

- Why I'm Meh About JSON API

- „My frustrations with JSON API are ultimately because it doesn't solve the problems I have as an API consumer [...]"
- „Perhaps JSON API's audience is specifically API producers, not consumers, and that's why I don't find it addressing my needs."

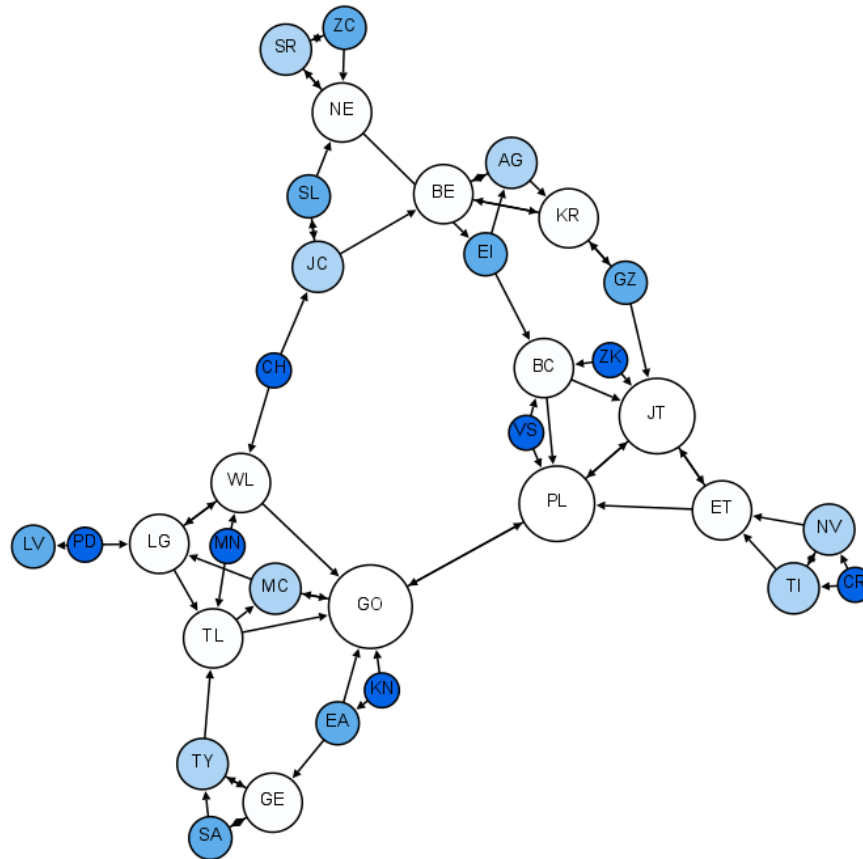
<https://jeremywsherman.com/blog/2016/07/23/why-im-meh-about-json-api/>

# GraphQL!



Hintergrund

# Graph-



Hintergrund

## -QL (Query Language)

```
{  
  language(title: „GraphQL“) {  
    description  
    demo  
    types  
  }  
}
```



*DEMO TIME!*

# *Einbindung*

## Beispiel n+1 query

```
public async pull(): Promise<void> {  
    const characters = await this.characterRestService.getAll();  
  
    const changedCharacters = this.getChangedCharacters(characters);  
    for (const character of changedCharacters) {  
        await this.filesSyncService.pull(character.id);  
        const characterDetails = await this.getDetails(character.id);  
        const characterComments = await this.getComments(character.id);  
    }  
    await this.syncCharacterStatusService.done();  
}
```

## Die Abfrage als GraphQL

```
{  
  allCharacters {  
    character {  
      id  
      detail  
      comments  
      files  
    }  
  }  
}
```

Einbindung

## Inkrementelle Einbindung

Bis jetzt (REST):

- /characters/
- /characters/load-characters
- /characters/save-characters
- /character/details/load
- /character/details/save
- ...

*Einbindung*

## *Inkrementelle Einbindung*

Mit GraphQL (zusätzlich):

- /graphql

## *Vorteile + Probleme*

## Vorteile

- Clients bekommen nur die Daten, die sie brauchen
- Ein Request/Response
- Statische Typisierung
- Nur ein Endpoint
- API ist dynamisch
- API ist einfach gestaltet für den Anwender (harte Arbeit liegt hinter dem GraphQL-Server)
- API kann sich selbst beschreiben
- Keine SQL Injection
- Tools (GraphiQL, Slow Query Analyzer, Query Cache, ...)

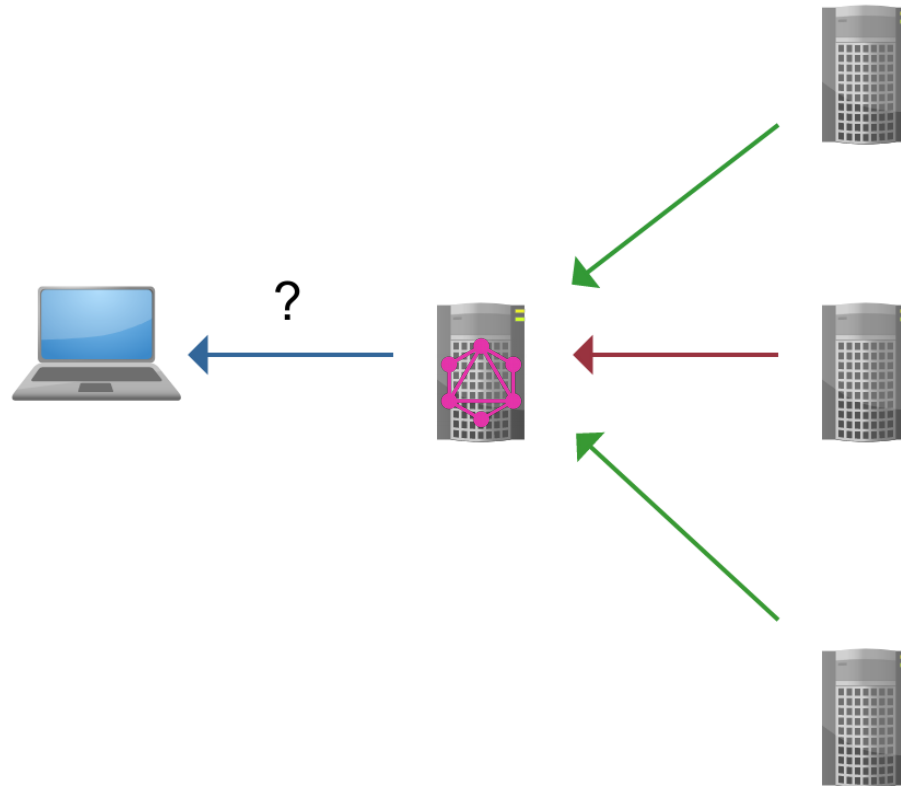


## Probleme

- Relativ neu → wenig Erfahrungswerte
- Server-seitig wenig gut dokumentierte Projekte
- Davon 2 in „unbekannten“ Technologien (Scala + Elixir)
- Autorisierung muss neu überdacht werden
- Falls ein Feld nicht vorhanden ist, werden andere dennoch zurück gegeben

Vorteile + Probleme

## Probleme



## *Links*

## Links

- Allgemein
  - <https://www.graphql.org/>
  - <https://www.howtographql.com/>
  - <http://graphql.org/swapi-graphql/> (Test-API)
- Bücher
  - <https://pragprog.com/book/wwgraphql/craft-graphql-apis-in-elixir-with-absinthe>
  - <https://www.manning.com/books/react-quickly>
- Server
  - <http://absinthe-graphql.org/>
  - <http://sangria-graphql.org/>
  - <https://www.apollographql.com/docs/apollo-server/>
- Weiteres Material: <https://github.com/chentsulin/awesome-graphql>

meteocontrol GmbH

Spicherer Straße 48 | 86157 Augsburg

Telefon +49 (0)821 34666-0 | Fax +49 (0)821 34666-11

E-mail [info@meteocontrol.de](mailto:info@meteocontrol.de) | Web [www.meteocontrol.de](http://www.meteocontrol.de)

Urheberrechte:

Copyright meteocontrol GmbH, Augsburg (Deutschland). Alle Rechte vorbehalten. Text, Bilder, Grafiken sowie deren Anordnung unterliegen dem Schutz des Urheberrechts und anderer Schutzgesetze. Sie dürfen nicht ohne die Zustimmung der meteocontrol GmbH zu kommerziellen Zwecken kopiert, verbreitet, verändert oder Dritten zugänglich gemacht werden. Wir weisen daraufhin, dass die Bilder teilweise dem Urheberrecht Dritter unterliegen.