



# GraphQL

als Alternative zu REST

Manuel Mauky

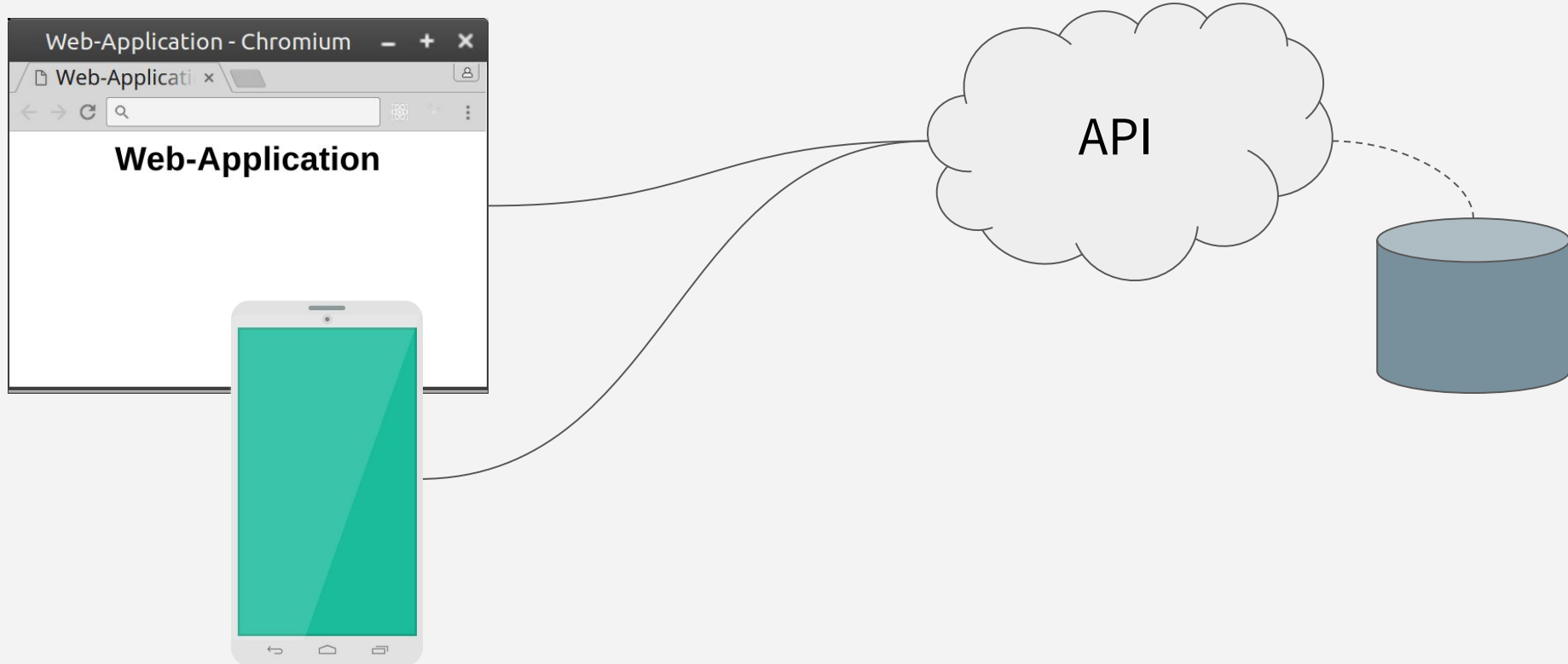
 @manuel\_mauky

JUG  
Görlitz 

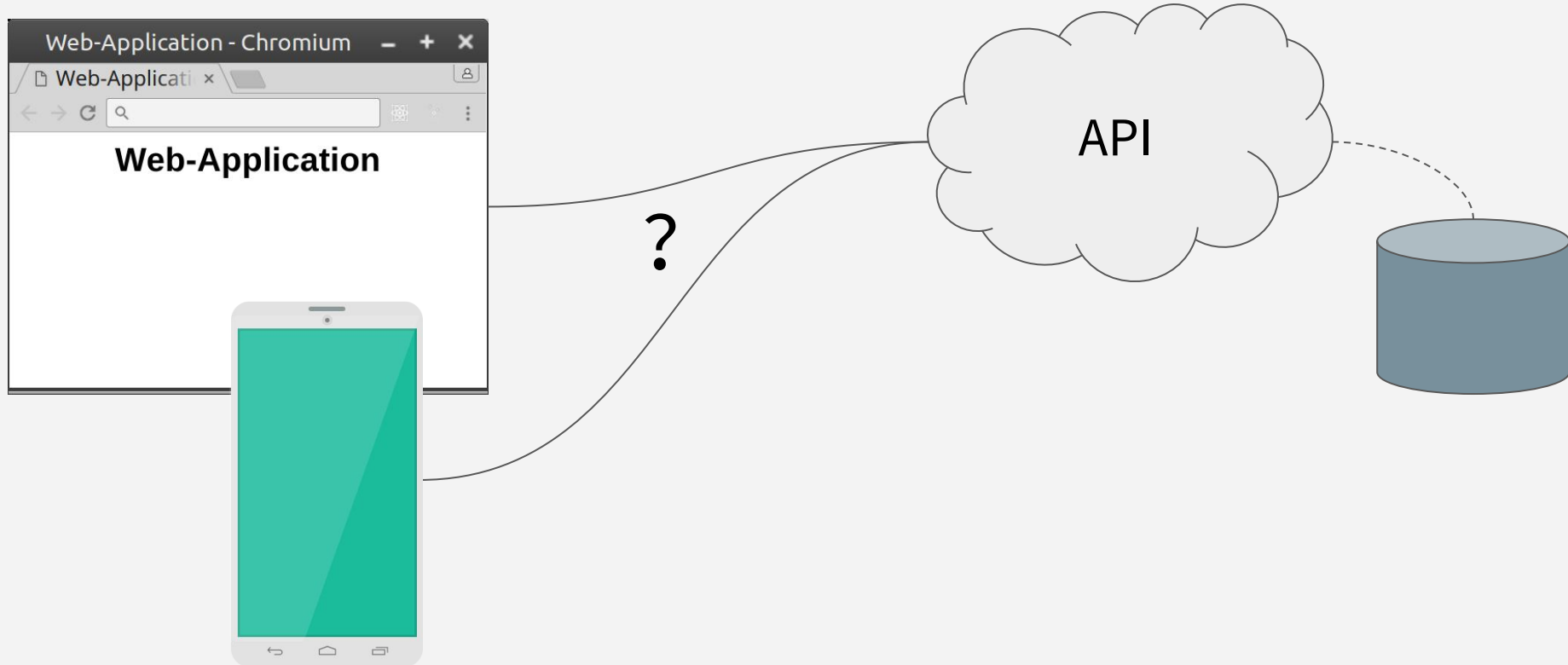


Saxonia Systems  
So geht Software.

# Single-Page-Apps und mobile Apps



# Wie kommen die Daten zum Client?



# Klassisch: REST/Hypermedia

- zahlreiche Ressourcen, jeweils über URIs erreichbar
- Hyperlinks zwischen Ressourcen
- HTTP als Protokoll
- verschiedene Repräsentationen, z.B. JSON

# Motivation

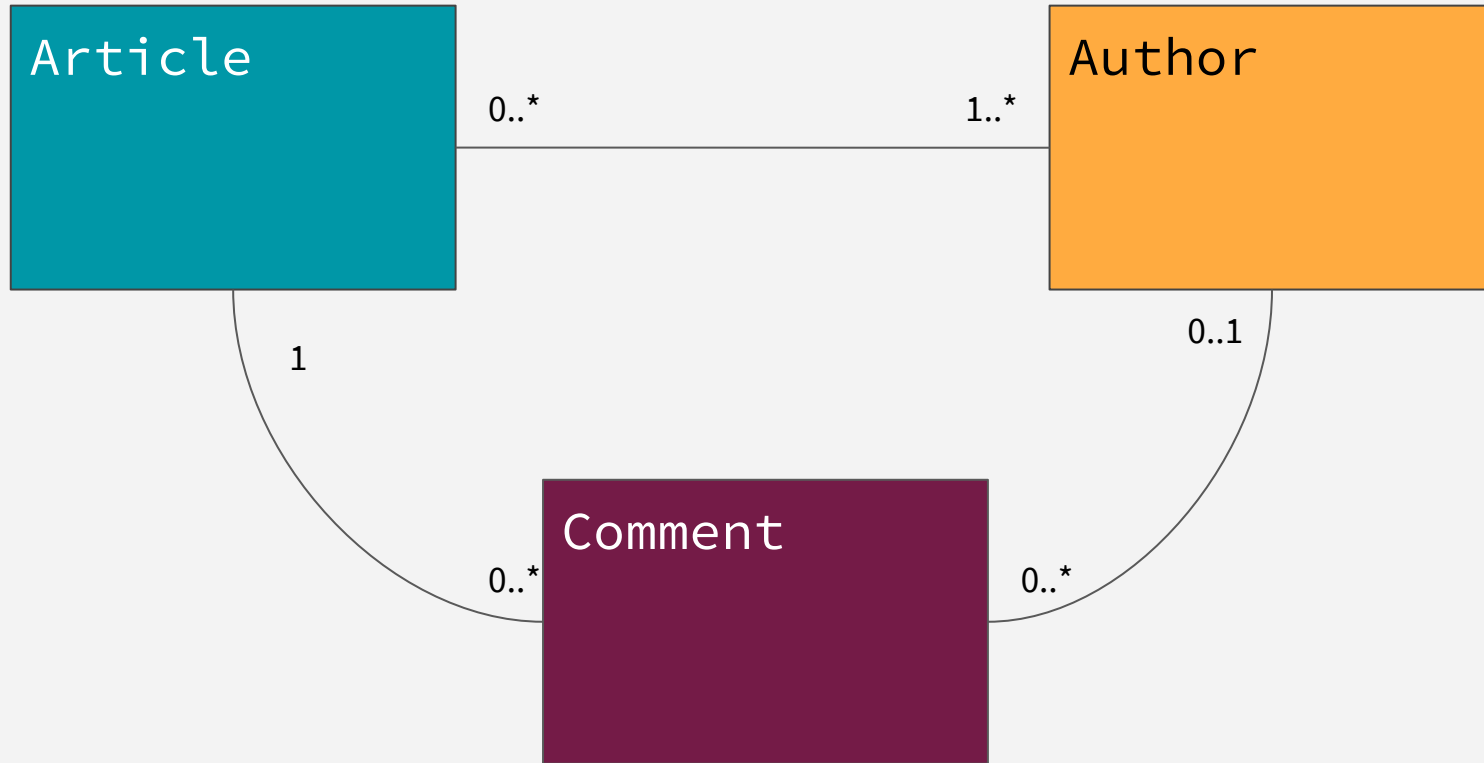
Warum lohnt es sich, sich mit GraphQL zu beschäftigen?

# Gründe, sich mit GraphQL zu beschäftigen

- Löst einige Schwierigkeiten mit REST-APIs
- Netzwerk-Performance
- Ermöglicht neue Ideen im Frontend

Beispiel

# Beispiel: Blog-System





# Article

id: String

permalink: String

teaser: String

text: String

releaseDate: Datetime

authors: [Author]

comments: [Comment]

# Author

id: String

name: String

comments: [Comment]

# Comment

id: String

text: String

author: Author?

guestAuthor: String?

REST+Hypermedia

# Use-Case: Artikel-Übersicht (10 Artikel)

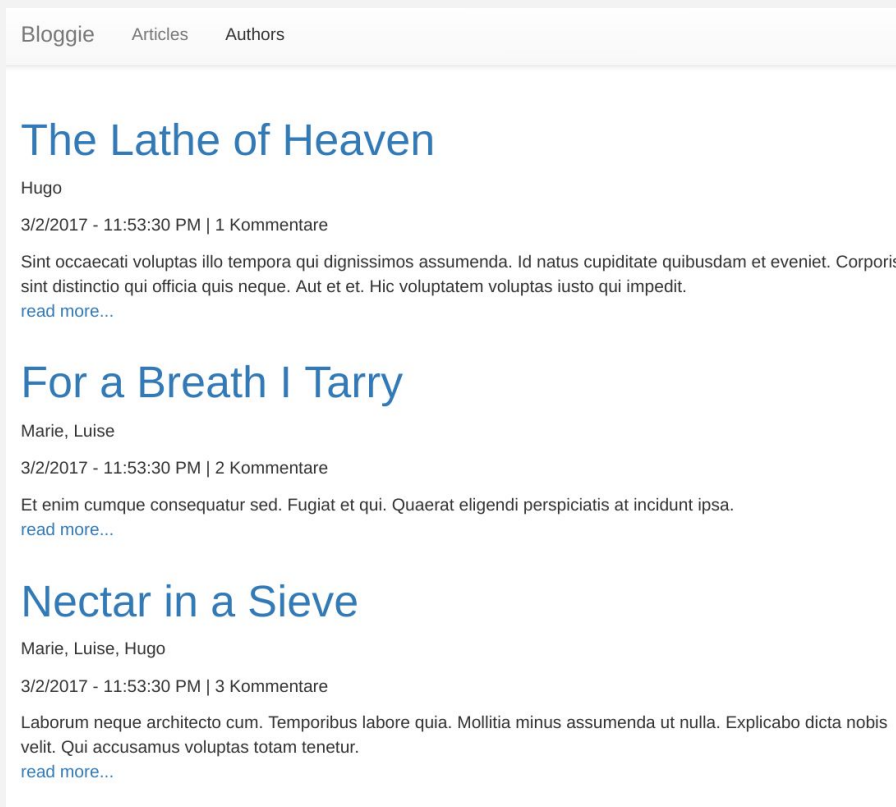
Benötigte Daten:

Je Artikel:

Titel, Permalink, Teasertext,  
releasedate

Autoren: Name

Anzahl Kommentare



Bloggie Articles Authors

## The Lathe of Heaven

Hugo  
3/2/2017 - 11:53:30 PM | 1 Kommentare

Sint occaecati voluptas illo tempora qui dignissimos assumenda. Id natus cupiditate quibusdam et eveniet. Corporis sint distinctio qui officia quis neque. Aut et et. Hic voluptatem voluptas iusto qui impedit.  
[read more...](#)

## For a Breath I Tarry

Marie, Luise  
3/2/2017 - 11:53:30 PM | 2 Kommentare

Et enim cumque consequatur sed. Fugiat et qui. Quaerat eligendi perspiciatis at incidunt ipsa.  
[read more...](#)

## Nectar in a Sieve

Marie, Luise, Hugo  
3/2/2017 - 11:53:30 PM | 3 Kommentare

Laborum neque architecto cum. Temporibus labore quia. Mollitia minus assumenda ut nulla. Explicabo dicta nobis velit. Qui accusamus voluptas totam tenetur.  
[read more...](#)

# Use-Case 2: Artikel-Details

Benötigte Daten:

Artikel:

Titel, Permalink, Teasertext,  
releasedate, **Text**

Autoren: Name

Kommentare: **Autorname, Text**

Bloggie Articles Authors

## The Lathe of Heaven

Hugo

**Sint occaecati voluptas illo tempora qui dignissimos assumenda. Id natus cupiditate quibusdam et eveniet. Corporis sint distinctio qui officia quis neque. Aut et et. Hic voluptatem voluptas iusto qui impedit.**

Qui ut est. Quidem minima omnis. Expedita pariatur accusamus et iusto. Nihil ut eaque assumenda alias atque dolor. Est eligendi soluta ullam non. Doloribus in itaque qui consequatur et quia. Sed unde quaerat libero tenetur quia. Quo eos corporis et alias. Deserunt quia eum. Molestiae distinctio ut labore facere. Suscipit qui et error et placeat est. Dolorem deleniti rerum similique sunt distinctio excepturi. Qui dolore sed. Laborum ratione commodi et dicta. Et consequuntur hic nobis rerum eaque voluptas molestias. Sint nobis dolore. Nihil ea labore. Eaque vel dolores quia nihil aut. Laboriosam vero et sapiente iste. Temporibus quia blanditiis asperiores ut et omnis id. Itaque corporis praesentium corrupti delectus et possimus similique. Enim vel aut voluptatibus non. Et unde fuga est rem repellendus nihil corrupti. A id numquam laboriosam et et. Suscipit autem officia voluptatem qui blanditiis deleniti est. Quibusdam mollitia sit quisquam. Sed saepe eos eligendi. Quod ducimus ut aut suscipit consectetur rem officia. Veniam cumque quos inventore commodi consequatur qui quia. Nemo sed quos provident maxime tenetur. Consequatur in et unde sapiente. Molestiae quia enim corrupti. Commodi possimus cum mollitia perspiciatis quis autem eos. Consequatur et temporibus molestias voluptates numquam. Et ea doloribus eum consequatur. Quia quia omnis necessitatibus doloribus. Neque odit nisi voluptas. Id tempora amet ratione. Voluptatem voluptas praesentium. Rerum cumque explicabo non aut sint fugiat. Sit nisi aut ab officia. Quidem quaerat sint optio. Ea minima numquam. Aspernatur nihil nesciunt debitis consectetur. Explicabo nisi quia iste ea. Ea fuga necessitatibus consequatur commodi. Nulla iusto nesciunt voluptates praesentium. Ea dolorum quia repellat iusto quasi. Necessitatibus odio et aut qui. Soluta voluptate et incidunt reiciendis sunt saepe eos. Omnis hic excepturi quia et recusandae. Non animi velit. Doloremque fuga eum at. Harum inventore ipsa saepe qui quisquam nulla. Accusamus

# REST+Hypermedia

GET /api/ → Liste von verfügbaren Ressourcen

GET /api/article → Liste von Artikeln mit Titel, Permalink, Teaser, ReleaseDate

# REST+Hypermedia

Je Artikel:

GET /api/article/:id/authors → Autoren des Artikels

GET /api/article/:id/comments → Kommentare



# REST+Hypermedia

Detail-Ansicht:

GET /api/article/:id → Text

# REST+Hypermedia

- viele Requests (im Beispiel 22 Requests für Startseite)
- Underfetching (zu wenig Daten geladen)
- Overfetching (überflüssige Daten geladen)

Naive "Lösung":

~~Hypermedia~~ → Alle Daten "embedden"



# Lösung: Alle Daten "embedden"

Idee: Artikel-Ressource enthält direkt alle Daten

Nur noch 1 Request

# Lösung: Alle Daten "embedden"

Idee: Artikel-Ressource enthält direkt alle Daten

Nur noch 1 Request

- Probleme:
- Unnötige Daten werden übertragen
  - Vorteile von Hypermedia gehen verloren



# Query-Parameter / Projections

# Query-Parameter / Projections

`/api/article?withAuthor=true&withComments=true`

# Query-Parameter / Projections

`/api/article?projection=short-article`



# View-Spezifische Ressourcen

# View-Spezifische Ressourcen

/api/article\_overview

/api/article\_details/:id

/api/author\_list

/api/authors\_comments\_list/:id

Problem: Kopplung Server und Client

# Problem: Kopplung Server und Client

Neue Ansichten im Client benötigen neue Ressourcen auf dem Server

Herausforderung: Fehlende Typisierung

# Herausforderung: Fehlende Typisierung

`/api/article/:id`

Welche Daten kommen eigentlich als Antwort?

Herausforderung: Versionierung

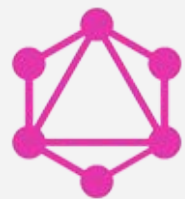
# Herausforderung: Versionierung

/api/article/:id

Welche Daten kommen eigentlich als Antwort?

Welche Daten kommen in 2 Monaten als Antwort?





GraphQL

# GraphQL

- Abfragesprache für (Web-)APIs
- Entwickelt von Facebook
- Seit 2012 bei Facebook Mobile App im Einsatz
- Seit Ende 2015 OpenSource

# GraphQL - Technisches

- ein einziger Endpunkt
- statisch typisiertes Schema
- Schema erlaubt Introspection
- Queries per HTTP POST oder GET
- Trennung von Query und Mutation → Ansätze von CQRS

# Beispiel: Blog-System

# Use-Case: Artikel-Übersicht (10 Artikel)

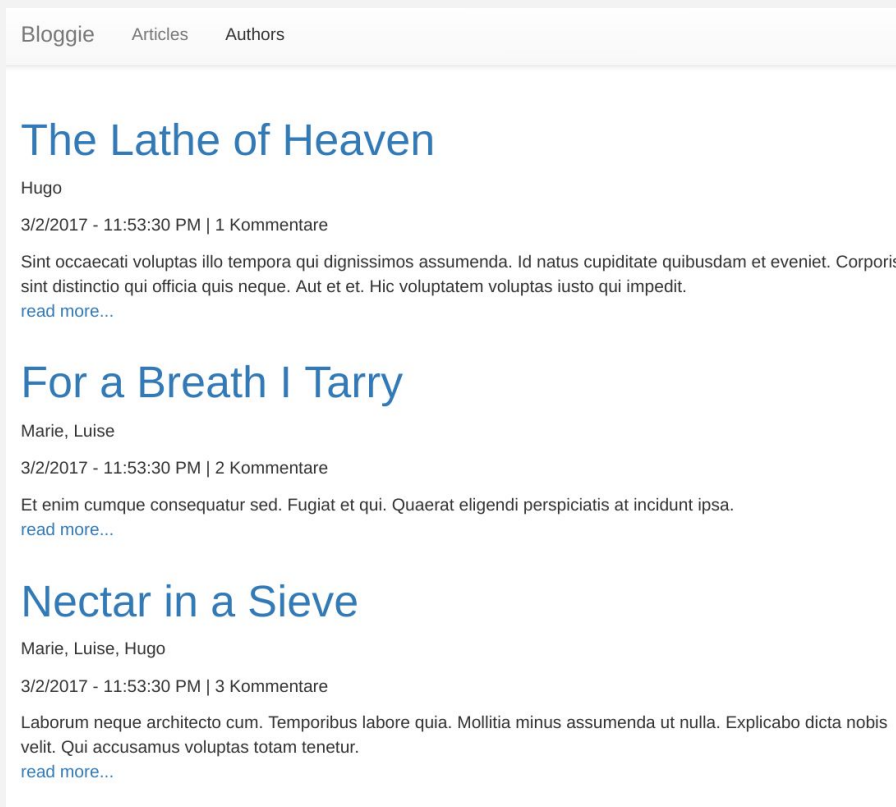
Benötigte Daten:

Je Artikel:

Titel, Permalink, Teasertext,  
releasedate

Autoren: Name

Anzahl Kommentare



Bloggie Articles Authors

## The Lathe of Heaven

Hugo  
3/2/2017 - 11:53:30 PM | 1 Kommentare

Sint occaecati voluptas illo tempora qui dignissimos assumenda. Id natus cupiditate quibusdam et eveniet. Corporis sint distinctio qui officia quis neque. Aut et et. Hic voluptatem voluptas iusto qui impedit.  
[read more...](#)

## For a Breath I Tarry

Marie, Luise  
3/2/2017 - 11:53:30 PM | 2 Kommentare

Et enim cumque consequatur sed. Fugiat et qui. Quaerat eligendi perspiciatis at incidunt ipsa.  
[read more...](#)

## Nectar in a Sieve

Marie, Luise, Hugo  
3/2/2017 - 11:53:30 PM | 3 Kommentare

Laborum neque architecto cum. Temporibus labore quia. Mollitia minus assumenda ut nulla. Explicabo dicta nobis velit. Qui accusamus voluptas totam tenetur.  
[read more...](#)

Query:

```
{  
  Article {  
    title  
  }  
}
```

Response:

```
{  
  "data": {  
    "Article": [  
      {  
        "title": "Some Title"  
      },  
      {  
        "title": "Other.."  
      }  
    ]  
  }  
}
```

```
{
  Article {
    id
    title
    permalink
    teaser
    releaseDate
  }
}

{
  "data": {
    "Article": [
      {
        "id": "sd3423s32",
        "title": "Some Title",
        "permalink": "some_title",
        "teaser": "Lorem Ipsum..",
        "releaseDate": "2017-02.."
      },
      {
        "id": "sdöf234ds2",
        "title": "Other..",
        ...
      }
    ]
  }
}
```

```
{
  Article {
    id
    title
    permalink
    teaser
    releaseDate
    authors {
      name
    }
    comments {
      id
    }
  }
}
```

```
{
  "data": {
    "Article": [
      {
        "id": "sd3423s32",
        "title": "Some Title",
        "permalink": "some_title",
        "teaser": "Lorem Ipsum..",
        "releaseDate": "2017-02..",
        "authors": [
          {
            "name": "Hugo",
          }
        ],
        "comments": [
          {
            "id": "lslldsö2fö2",
          }
        ]
      },
      {
        "id": "sdöf234ds2",
        "title": "Other "
      }
    ]
  }
}
```



# Use-Case 2: Artikel-Details

Benötigte Daten:

Artikel:

Titel, Permalink, Teasertext,  
releasedate, **Text**

Autoren: Name

Kommentare: **Autorname, Text**

Bloggie Articles Authors

## The Lathe of Heaven

Hugo

**Sint occaecati voluptas illo tempora qui dignissimos assumenda. Id natus cupiditate quibusdam et eveniet. Corporis sint distinctio qui officia quis neque. Aut et et. Hic voluptatem voluptas iusto qui impedit.**

Qui ut est. Quidem minima omnis. Expedita pariatur accusamus et iusto. Nihil ut eaque assumenda alias atque dolor. Est eligendi soluta ullam non. Doloribus in itaque qui consequatur et quia. Sed unde quaerat libero tenetur quia. Quo eos corporis et alias. Deserunt quia eum. Molestiae distinctio ut labore facere. Suscipit qui et error et placeat est. Dolorem deleniti rerum similique sunt distinctio excepturi. Qui dolore sed. Laborum ratione commodi et dicta. Et consequuntur hic nobis rerum eaque voluptas molestias. Sint nobis dolore. Nihil ea labore. Eaque vel dolores quia nihil aut. Laboriosam vero et sapiente iste. Temporibus quia blanditiis asperiores ut et omnis id. Itaque corporis praesentium corrupti delectus et possimus similique. Enim vel aut voluptatibus non. Et unde fuga est rem repellendus nihil corrupti. A id numquam laboriosam et et. Suscipit autem officia voluptatem qui blanditiis deleniti est. Quibusdam mollitia sit quisquam. Sed saepe eos eligendi. Quod ducimus ut aut suscipit consectetur rem officia. Veniam cumque quos inventore commodi consequatur qui quia. Nemo sed quos provident maxime tenetur. Consequatur in et unde sapiente. Molestiae quia enim corrupti. Commodi possimus cum mollitia perspiciatis quis autem eos. Consequatur et temporibus molestias voluptates numquam. Et ea doloribus eum consequatur. Quia quia omnis necessitatibus doloribus. Neque odit nisi voluptas. Id tempora amet ratione. Voluptatem voluptas praesentium. Rerum cumque explicabo non aut sint fugiat. Sit nisi aut ab officia. Quidem quaerat sint optio. Ea minima numquam. Aspernatur nihil nesciunt debitis consectetur. Explicabo nisi quia iste ea. Ea fuga necessitatibus consequatur commodi. Nulla iusto nesciunt voluptates praesentium. Ea dolorum quia repellat iusto quasi. Necessitatibus odio et aut qui. Soluta voluptate et incidunt reiciendis sunt saepe eos. Omnis hic excepturi quia et recusandae. Non animi velit. Doloremque fuga eum at. Harum inventore ipsa saepe qui quisquam nulla. Accusamus

```
{
  Article(id: "sd342") {
    id
    text
    ...
    authors {
      name
    }
    comments {
      id
      text
      author {
        name
      }
    }
  }
}

{
  "data": {
    "Article": [
      {
        "id": "sd342",
        ...
        "authors": [
          {
            "name": "Hugo",
          }
        ],
        "comments": [
          {
            "id": "lsldsö2fö2",
            "text": "Erster!!111!",
            "author": {
              "name": "Horsti"
            }
          }
        ]
      }
    ]
  },
}
```

# Mutations

```
mutation {
  addAuthor(username: "Horsti") {
    id
    name
  }
}
```

```
{
  "data": {
    "addAuthor": {
      "id": "l̂sdö34",
      "name": "Horsti"
    }
  }
}
```

Neue Ideen im Frontend

# Query-Optimierung + Caching

## Schritt 1: Artikel-Übersicht

```
{
  Article {
    id
    title
    permalink
    teaser
    releaseDate
  }
}

{
  "data": {
    "Article": [
      {
        "id": "sd3423s32",
        "title": "Some Title",
        "permalink": "some_title",
        "teaser": "Lorem Ipsum..",
        "releaseDate": "2017-02.."
      },
      {
        "id": "sdöf234ds2",
        "title": "Other..",
        ...
      }
    ]
  }
}
```

## Schritt 2: Artikel-Details

```
{
  Article (id: "sd34..") {
    id
    title
    text
    permalink
    teaser
    releaseDate
  }
}

{
  "data": {
    "Article": [
      {
        "id": "sd3423s32",
        "title": "Some Title",
        "text": "some long text..",
        "permalink": "some_title",
        "teaser": "Lorem Ipsum..",
        "releaseDate": "2017-02.."
      }
    ]
  }
}
```



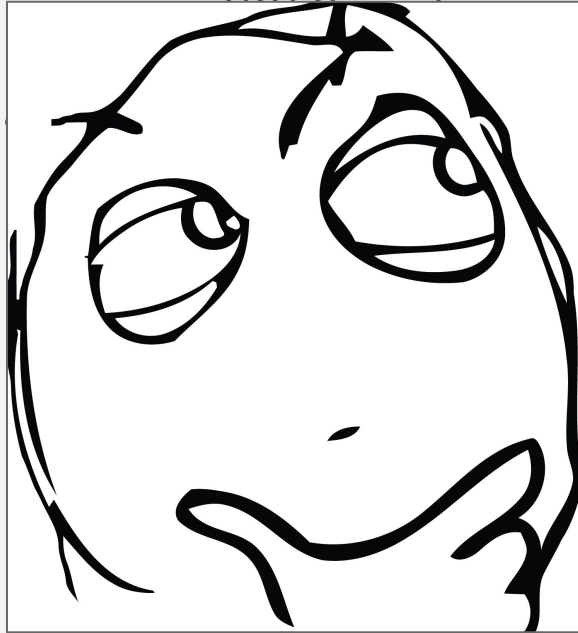
## Schritt 2: Artikel-Details

```
{
  Article (id: "sd34..") {
    id
    title
    text
    permalink
    teaser
    releaseDate
  }
}

{
  "data": {
    "Article": [
      {
        "id": "sd3423s32",
        "title": "Some Title",
        "text": "some long text...",
        "permalink": "some_title",
        "teaser": "Lorem Ipsum..",
        "releaseDate": "2017-02.."
      }
    ]
  }
}
```

## Schritt 2: Artikel-Details

```
{  
  Article (id: "sd34  
    id  
    title  
    text  
    permalink  
    teaser  
    releaseDate  
  }  
}
```



```
{  
  "data": {  
    "id": "sd3423s32",  
    "title": "Some Title",  
    "text": "some long text..",  
    "permalink": "some_title",  
    "teaser": "Lorem Ipsum..",  
    "releaseDate": "2017-02.."  
  }  
}
```

## Schritt 2: Artikel-Details

```
{
  Article (id: "sd34..") {
    id
    title
    text
    permalink
    teaser
    releaseDate
  }
}

{
  "data": {
    "Article": [
      {
        "id": "sd3423s32",
        "title": "Some Title",
        "text": "some long text..",
        "permalink": "some_title",
        "teaser": "Lorem Ipsum..",
        "releaseDate": "2017-02.."
      }
    ]
  }
}
```

## Schritt 2: Artikel-Details

```
{
  Article (id: "sd34..") {
    id
    text
  }
}

{
  "data": {
    "Article": [
      {
        "id": "sd3423s32",
        "text": "some long text...",
      }
    ]
  }
}
```

## Schritt 2: Artikel-Details

```
{  
  Article (id:  
    id  
    text  
  }  
}
```



```
3s32",  
e long text..",
```

## Schritt 3: Zurück zur Übersicht {

```
{
  Article {
    id
    title
    permalink
    teaser
    releaseDate
  }
}

"data": {
  "Article": [
    {
      "id": "sd3423s32",
      "title": "Some Title",
      "permalink": "some_title",
      "teaser": "Lorem Ipsum..",
      "releaseDate": "2017-02.."
    },
    {
      "id": "sdöf234ds2",
      "title": "Other..",
      ...
    }
  ]
}
```

## Schritt 3: Zurück zur Übersicht {

```
{
  Article {
    id
    title
    permalink
    teaser
    releaseDate
  }
}

"data": {
  "Article": [
    {
      "id": "sd3423s32",
      "title": "Some Title",
      "permalink": "some_title",
      "teaser": "Lorem Ipsum..",
      "releaseDate": "2017-02.."
    },
    {
      "id": "sdöf234ds2",
      "title": "Other..",
      ...
    }
  ]
}
```

# Query-Optimierung und Caching

- Entwickler beschreibt Daten-Anforderungen je Ansicht
- Framework optimiert Queries und übernimmt Caching



# Query co-location

Nächstes Treffen:

## Angular 2 für Java Entwickler

von Sven Hoffmann | 22 March 2017

Mit Angular bietet Google ein komponenten-basiertes Webframework an, welches den Anspruch erhebt, die Anforderungen an zeitgemäße Softwareentwicklung ganzheitlich zu unterstützen.

Seit dem Release von Angular 2 war es für gestandene Java Entwickler noch nie so einfach in die Webentwicklung einzusteigen. Mit Klassen, Dependency Injection, Annotations und Data-Binding bietet Angular dem versierten Desktop Entwickler ein vertrautes Umfeld, in dem man sich schnell zurecht findet.

In dieser Veranstaltung möchte ich am lebenden Beispiel zeigen, wie es sich heutzutage anfühlt eine Webanwendung, konkret mit Angular 2, zu erstellen.



**Sven Hoffmann**

Sven Hoffmann ist seit über 8 Jahren für die Saxonia Systems AG tätig. Er beschäftigte sich in dieser Zeit mit einer Vielzahl von Backend und Frontend-Technologien. Zu seinen technologischen Hauptschwerpunkten zählen Webanwendungen und JavaEE.

Datum: 22.03.2017, 19:00 Uhr

Ort: Wird noch festgelegt.

Startseite - Chromium

Startseite x

www.jug-gr.de

Java User Group Görlitz   Aktuell   Vorträge   Unterstützer   About   Mitmachen

## Nächstes Treffen:

### Angular 2 für Java Entwickler

von Sven Hoffmann | 22 March 2017

Mit Angular bietet Google ein komponenten-basiertes Webframework an, welches den Anspruch erhebt, die Anforderungen an zeitgemäße Softwareentwicklung ganzheitlich zu unterstützen.

Seit dem Release von Angular 2 war es für gestandene Java Entwickler noch nie so einfach in die Webentwicklung einzusteigen. Mit Klassen, Dependency Injection, Annotations und Data-Binding bietet Angular dem versierten Desktop Entwickler ein vertrautes Umfeld, in dem man sich schnell zurecht findet.

In dieser Veranstaltung möchte ich am lebenden Beispiel zeigen, wie es sich heutzutage anfühlt eine Webanwendung, konkret mit Angular 2, zu erstellen.

---



Sven Hoffmann ist seit über 8 Jahren für die Saxonia Systems AG tätig. Er beschäftigte sich in dieser Zeit mit einer Vielzahl von Backend und Frontend-Technologien. Zu seinen technologischen Hauptschwerpunkten zählen Webanwendungen und JavaEE.

**Sven Hoffmann**

---

Datum: 22.03.2017, 19:00 Uhr  
Ort: Wird noch festgelegt.

## Header-Komponente

```
{
  Article(id: "123") {
    title
    authors {
      name
    }
    releaseDate
  }
}
```

Startseite - Chromium

Startseite x

www.jug-gr.de

Java User Group Görlitz   Aktuell   Vorträge   Unterstützer   About   Mitmachen

## Nächstes Treffen:

### Angular 2 für Java Entwickler

von Sven Hoffmann | 22 March 2017

Mit Angular bietet Google ein komponenten-basiertes Webframework an, welches den Anspruch erhebt, die Anforderungen an zeitgemäße Softwareentwicklung ganzheitlich zu unterstützen.

Seit dem Release von Angular 2 war es für gestandene Java Entwickler noch nie so einfach in die Webentwicklung einzusteigen. Mit Klassen, Dependency Injection, Annotations und Data-Binding bietet Angular dem versierten Desktop Entwickler ein vertrautes Umfeld, in dem man sich schnell zurecht findet.

In dieser Veranstaltung möchte ich am lebenden Beispiel zeigen, wie es sich heutzutage anfühlt eine Webanwendung, konkret mit Angular 2, zu erstellen.



Sven Hoffmann ist seit über 8 Jahren für die Saxonia Systems AG tätig. Er beschäftigte sich in dieser Zeit mit einer Vielzahl von Backend und Frontend-Technologien. Zu seinen technologischen Hauptschwerpunkten zählen Webanwendungen und JavaEE.

**Sven Hoffmann**

Datum: 22.03.2017, 19:00 Uhr  
Ort: Wird noch festgelegt.

## Autor-Komponente

```
{  
  Article(id: "123") {  
    authors {  
      name  
      pictureUrl  
      description  
    }  
  }  
}
```

Startseite - Chromium

Startseite x

www.jug-gr.de

Java User Group Görlitz   Aktuell   Vorträge   Unterstützer   About   Mitmachen

## Nächstes Treffen:

### Angular 2 für Java Entwickler

von Sven Hoffmann | 22 March 2017

Mit Angular bietet Google ein komponenten-basiertes Webframework an, welches den Anspruch erhebt, die Anforderungen an zeitgemäße Softwareentwicklung ganzheitlich zu unterstützen.

Seit dem Release von Angular 2 war es für gestandene Java Entwickler noch nie so einfach in die Webentwicklung einzusteigen. Mit Klassen, Dependency Injection, Annotations und Data-Binding bietet Angular dem versierten Desktop Entwickler ein vertrautes Umfeld, in dem man sich schnell zurecht findet.

In dieser Veranstaltung möchte ich am lebenden Beispiel zeigen, wie es sich heutzutage anfühlt eine Webanwendung, konkret mit Angular 2, zu erstellen.



Sven Hoffmann ist seit über 8 Jahren für die Saxonia Systems AG tätig. Er beschäftigte sich in dieser Zeit mit einer Vielzahl von Backend und Frontend-Technologien. Zu seinen technologischen Hauptschwerpunkten zählen Webanwendungen und JavaEE.

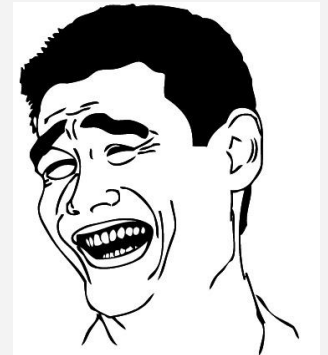
**Sven Hoffmann**

Datum: 22.03.2017, 19:00 Uhr  
Ort: Wird noch festgelegt.

## Content-Komponente

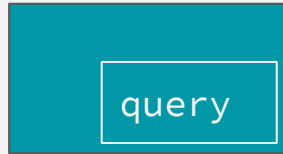
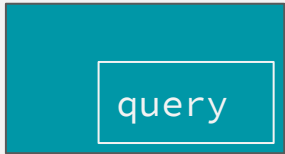
```
{  
  Article(id: "123") {  
    text  
  }  
}
```

Schlechte Idee:  
Jede Komponente führt eigene Requests aus



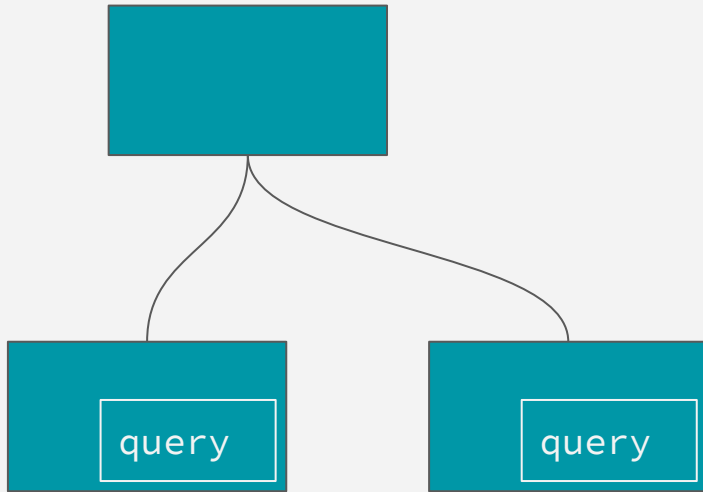
# Idee

- Jede UI-Komponente definiert ihre Daten-Anforderungen



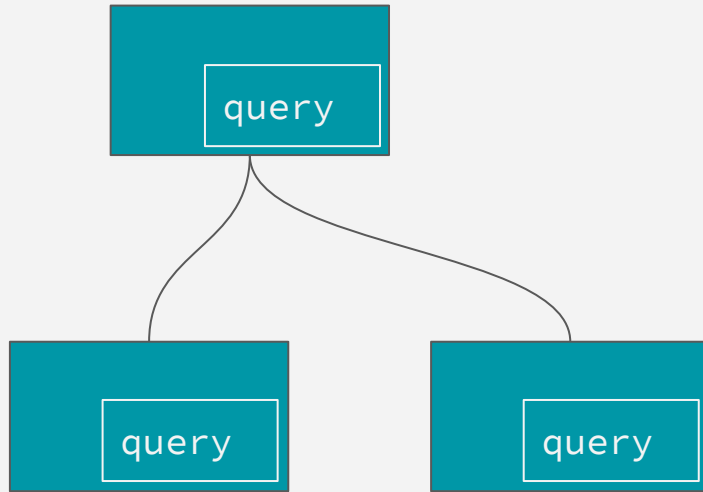
# Idee

- Jede UI-Komponente definiert ihre Daten-Anforderungen
- Parents sammeln Queries von Childs



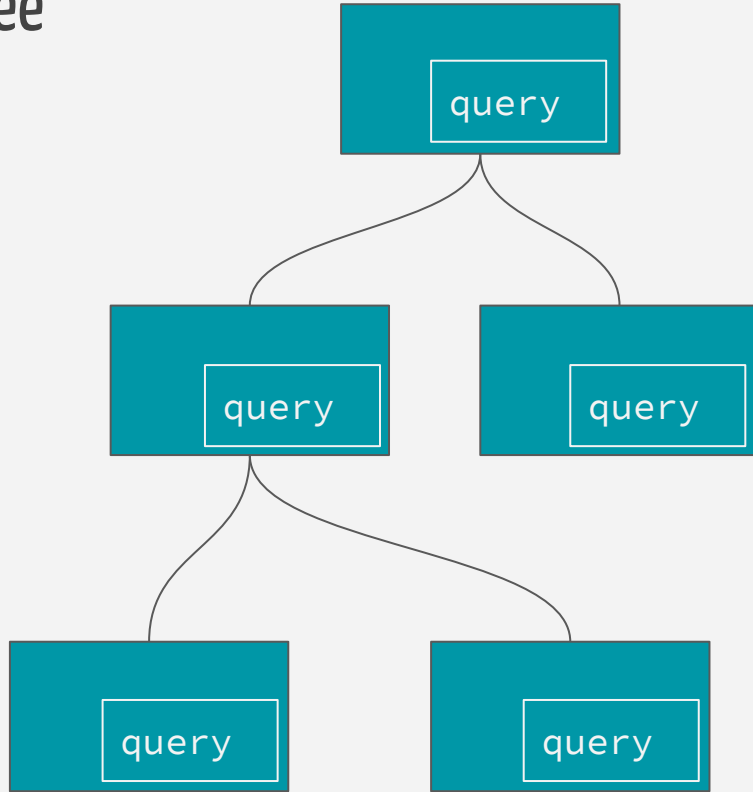


# Idee



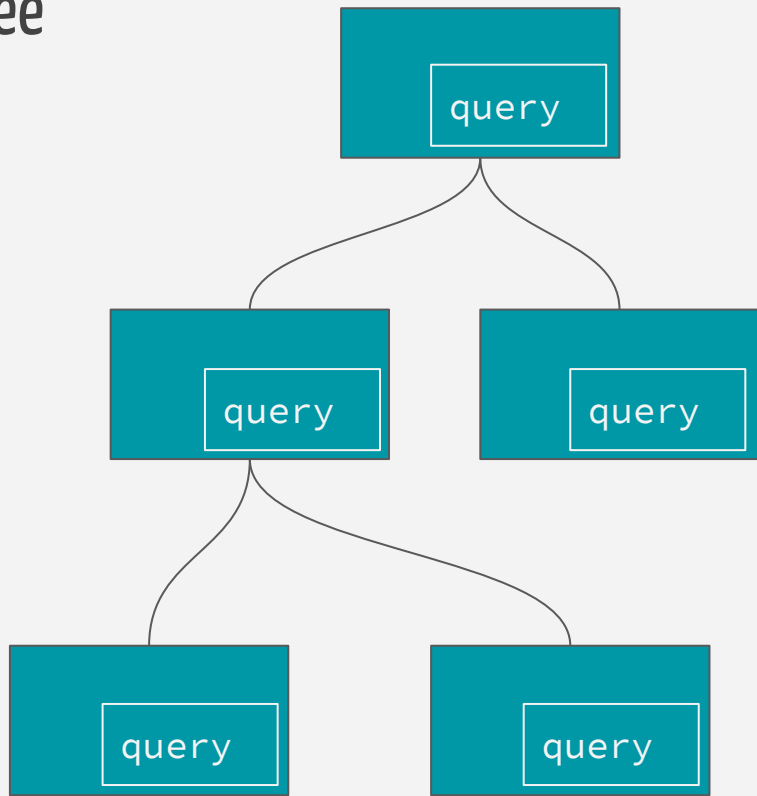
- Jede UI-Komponente definiert ihre Daten-Anforderungen
- Parents sammeln Queries von Childs
- Parents "mergen" Queries

# Idee



- Jede UI-Komponente definiert ihre Daten-Anforderungen
- Parents sammeln Queries von Childs
- Parents "mergen" Queries
- Root-Komponente führt Gesamt-Query aus

# Idee



- Jede UI-Komponente definiert ihre Daten-Anforderungen
- Parents sammeln Queries von Childs
- Parents "mergen" Queries
- Root-Komponente führt Gesamt-Query aus
- Root reicht Daten an Kinder durch

# Vorteile

- Childs sind unabhängig von ihren Parents
- Parents sind Unabhängig von Daten-Bedürfnissen der Childs
  - Neues Feature (Daten) in Child → keine Änderung in Parent
- Queries können optimiert werden
- Caching

# GraphQL Schema

Wie sieht der Server aus?

```
type Author {  
  id: ID!  
  name: String!  
}
```

```
type Author {  
  id: ID!  
  name: String!  
  articles: [Article]  
}
```

```
type Comment {  
  id: ID!  
  text: String!  
  author: Author  
}
```

```
type Article {  
  id: ID!  
  releaseDate: String  
  teaser: String  
  text: String  
  permalink: String  
  authors: [Author]  
  comments: [Comment]  
}
```

```
type QueryType {
  Author(id: String): [Authors]
  Article(id: String, permalink: String): [Article]
  Comment: [Comment]
}
```

```
schema {
  query: QueryType
  mutation: MutationType
}
```



Wo kommen die Daten her?

# Resolver

```
{  
  Articles {  
    id  
    text  
    authors {  
      name  
    }  
  }  
}
```

# Resolver

Articles()

```
{  
  Articles {  
    id  
    text  
    authors {  
      name  
    }  
  }  
}
```

# Resolver

```
{  
  Articles {  
    id  
    text  
    authors {  
      name  
    }  
  }  
}
```

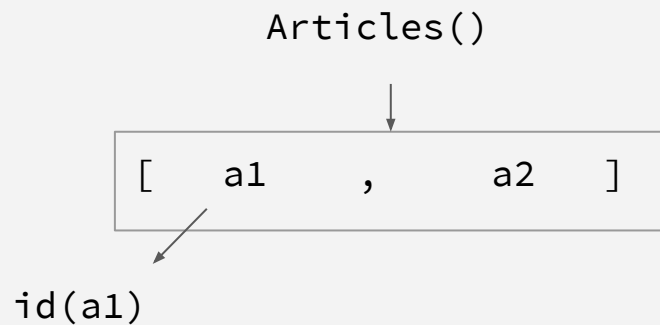
Articles()



```
[ a1 , a2 ]
```

# Resolver

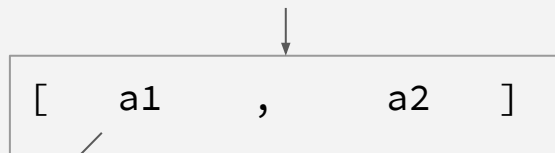
```
{  
  Articles {  
    id  
    text  
    authors {  
      name  
    }  
  }  
}
```



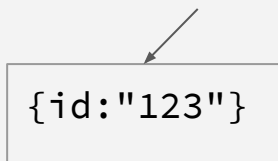
# Resolver

```
{  
  Articles {  
    id  
    text  
    authors {  
      name  
    }  
  }  
}
```

Articles()

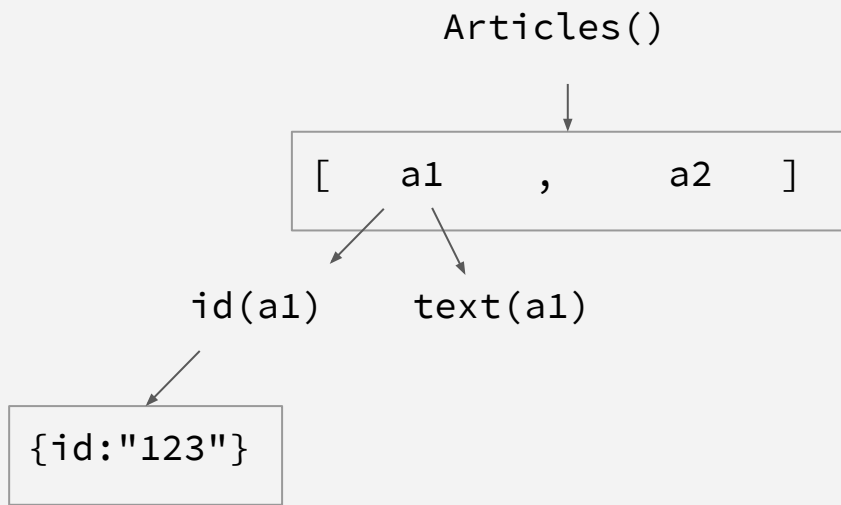


id(a1)



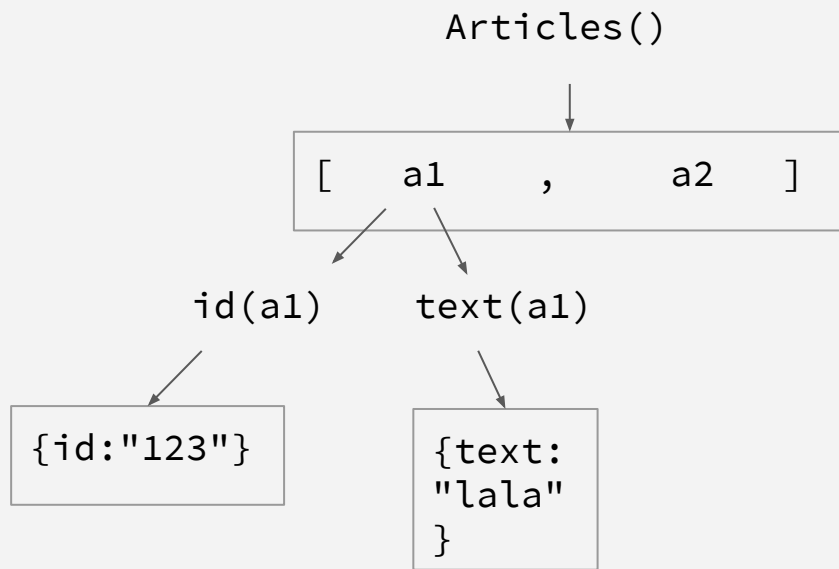
# Resolver

```
{  
  Articles {  
    id  
    text  
    authors {  
      name  
    }  
  }  
}
```



# Resolver

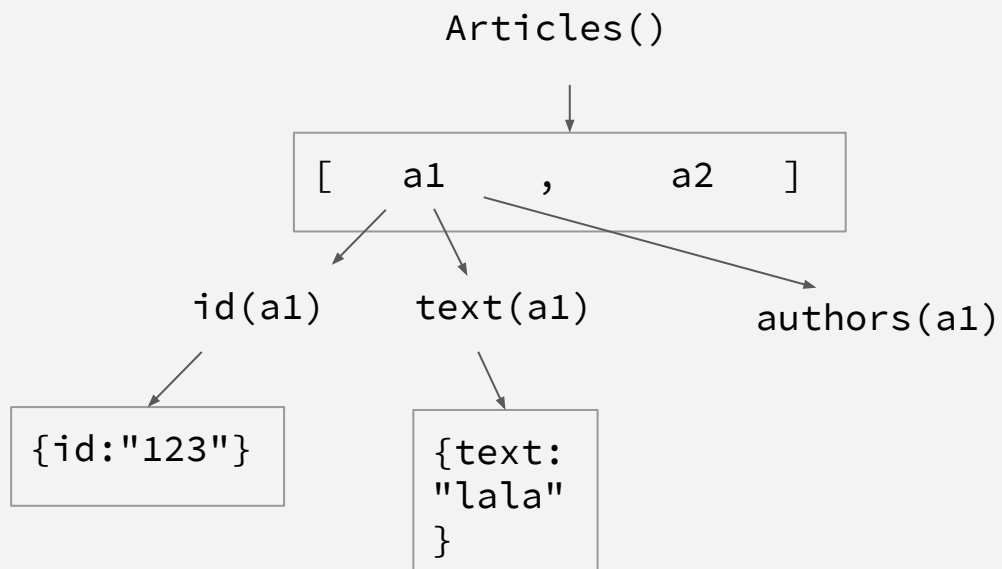
```
{  
  Articles {  
    id  
    text  
    authors {  
      name  
    }  
  }  
}
```





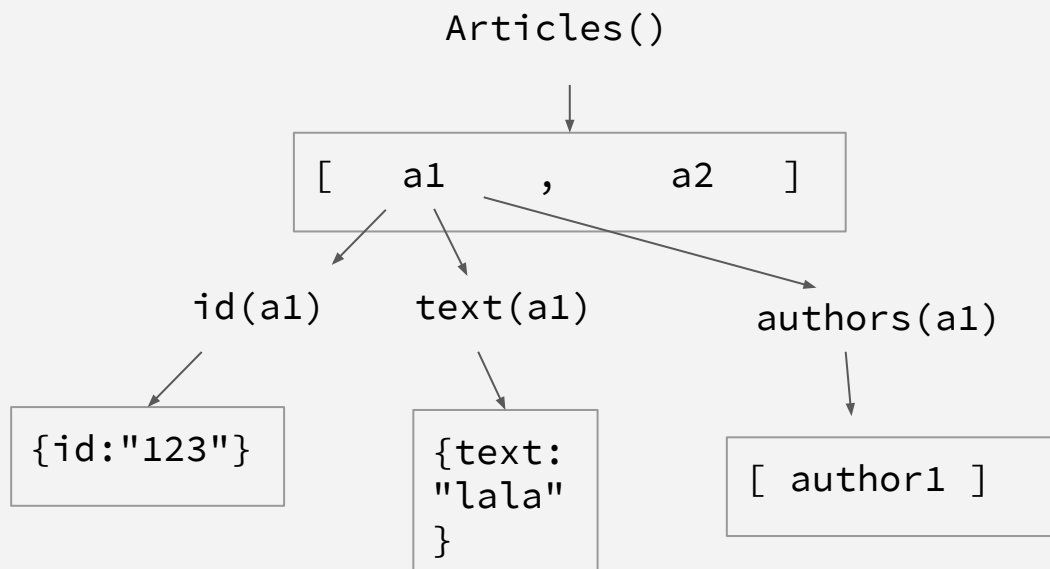
# Resolver

```
{  
  Articles {  
    id  
    text  
    authors {  
      name  
    }  
  }  
}
```



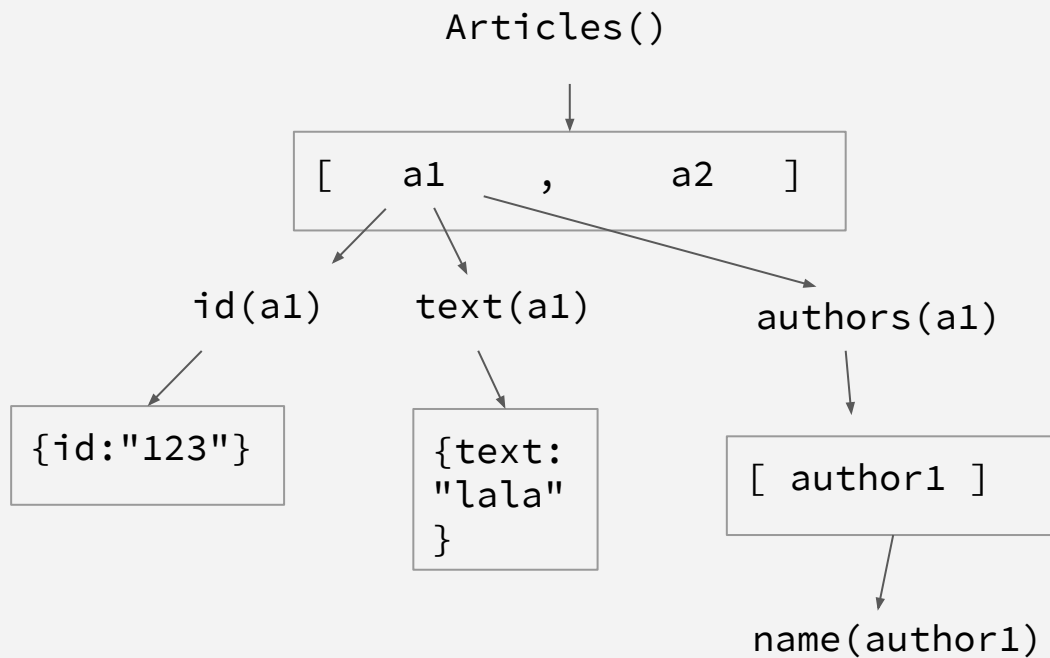
# Resolver

```
{  
  Articles {  
    id  
    text  
    authors {  
      name  
    }  
  }  
}
```



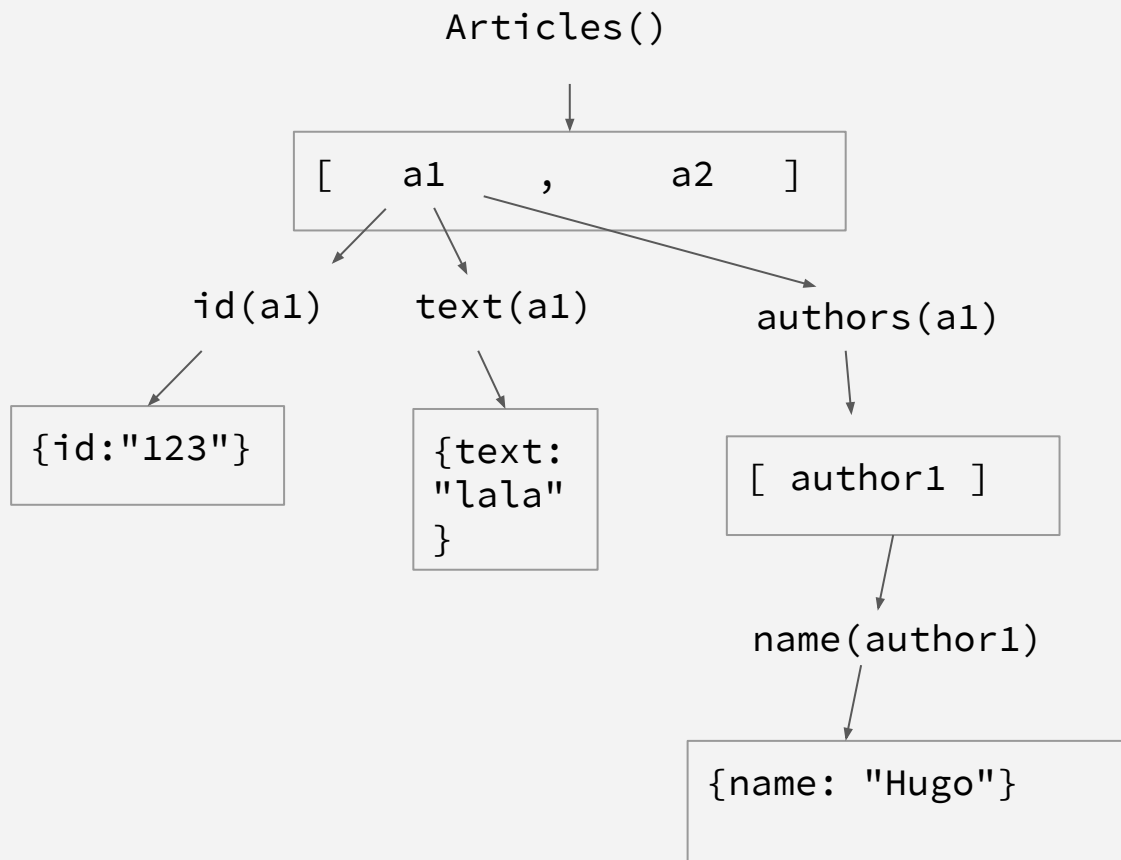
# Resolver

```
{  
  Articles {  
    id  
    text  
    authors {  
      name  
    }  
  }  
}
```



# Resolver

```
{  
  Articles {  
    id  
    text  
    authors {  
      name  
    }  
  }  
}
```



# Wo kommen die Daten her?

- für jedes Feld existiert eine Resolver-Funktion
  - skalare Typen → "Getter" by default
  - komplexe Typen → selbst implementieren
- Dadurch beliebige Daten-Quellen anbindbar
- Automatisierung durch Libraries

# GraphQL JPA

- <https://github.com/jcrygier/graphql-jpa>
- Bibliothek generiert GraphQL-Schema aus JPA Entities inkl. Resolver
- Beta-Status
- noch kein offizieller Mutation-Support

Fazit

# Aktueller Stand

- GraphQL-Spec existiert
- Mehrere PAAS-Anbieter existieren
- Ein paar Firmen migrieren (z.B. Github)
- Referenz-Implementierung in JavaScript wird aktiv entwickelt
- Java-Implementierungen existieren, aber weniger aktiv
- REST-Wrapper für sanfte Migration



# Vorteile

- Minimierung von Netzwerk-Requests
- Minimierung von übertragenen Daten
- Interessante Möglichkeiten für Entwickler
  - Statische Typisierung des Schemas
  - API-Explorer
- Neue Frontend-Patterns ermöglichen saubere Komponenten-Trennung
- Nur notwendige Felder werden resolved → Vorteil bei teurer Berechnung

# Nachteile / Offene Fragen

- Framework-Support außerhalb von JavaScript
- Microservices? API-Gateway?
- Requests sind größer
- Kein Caching auf Netzwerk-Ebene
- N+1 Problem zum Server verschoben
  - Lösung: DataFetcher
- Rekursive Queries / Nested-Queries

# Rekursive Queries


```
{  
  Article {  
    authors {  
      articles {  
        authors {  
          articles {  
            authors {  
              ...  
            }  
          }  
        }  
      }  
    }  
  }  
}
```

# Nachteile / Offene Fragen

- Framework-Support außerhalb von JavaScript
- Microservices? API-Gateway?
- Requests sind größer
- Kein Caching auf Netzwerk-Ebene
- N+1 Problem zum Server verschoben
  - Lösung: DataFetcher
- Rekursive Queries
  - potentiell Sicherheits/Performance-Risiko
  - Lösung: Nesting-Level begrenzen

# Vorteile von REST+Hypermedia

- Etablierte Technologie
- Breite Tool- und Framework-Unterstützung
- Caching auf Netzwerkebene
- Server kann Client steuern → Update auf Server ändert Client-Verhalten
- Dynamische Änderungen

"REST" ohne Hypermedia? 

# Fragen?

 @manuel\_mauky

 [github.com/lestard](https://github.com/lestard)

[www.lestard.eu](http://www.lestard.eu)

**JUG**  
**Görlitz** 



**Saxonia Systems**  
So geht Software.