

NILS HARTMANN

<https://nilshartmann.net>

# GraphQL

unter der Lupe

## Eine **kritische** Betrachtung

# NILS HARTMANN

nils@nilshartmann.net

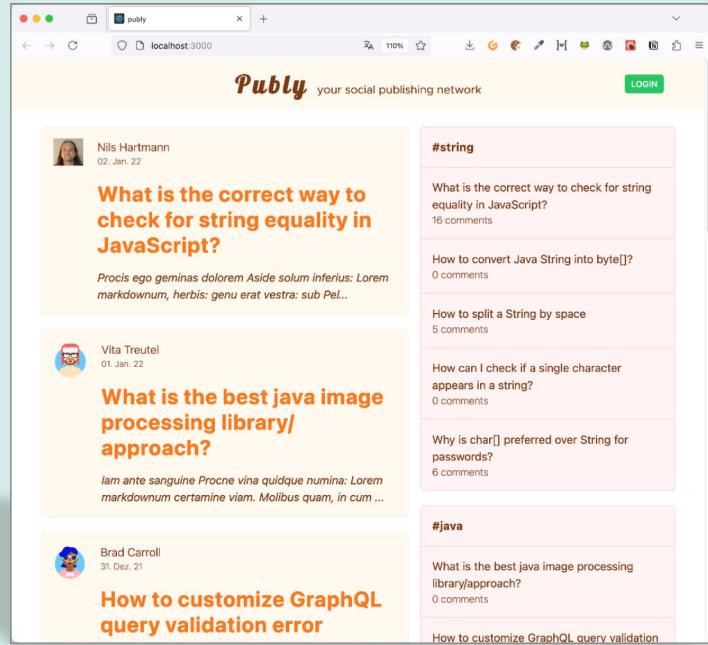
**Freiberuflicher Entwickler, Architekt, Trainer aus Hamburg**  
**Java, Spring, GraphQL, React, TypeScript**



<https://graphql.schule/video-kurs>

<https://reactbuch.de>

[HTTPS://NILSHARTMANN.NET](https://nilshartmann.net)



# Eine Anwendung, drei Ansichten...

## User

- username -

## Story

- title
- excerpt

The screenshot shows a web browser window for the 'Publy' platform at localhost:3000. The interface includes a header with the Publy logo and a 'LOGIN' button. On the left, a sidebar lists user information: 'User' and 'username'. Below that, under 'Story', it lists 'title' and 'excerpt'. The main content area displays three stories in cards:

- Story 1:** By Nils Hartmann on 02. Jan. 22. Title: "What is the correct way to check for string equality in JavaScript?". Excerpt: "Procis ego geminas dolorem Aside solum inferius: Lorem markdownum, herbis: genu erat vestra: sub Pel...". A dashed blue box highlights this story.
- Story 2:** By Vita Treutel on 01. Jan. 22. Title: "What is the best java image processing library/approach?". Excerpt: "Iam ante sanguine Procne vina quidque numina: Lorem markdownum certamine viam. Molibus quam, in cum ...".
- Story 3:** By Brad Carroll on 31. Dez. 21. Title: "How to customize GraphQL query validation error". Excerpt: (partially visible).

To the right of the stories is a sidebar with two sections:

- #string**:
  - What is the correct way to check for string equality in JavaScript?  
16 comments
  - How to convert Java String into byte[]?  
0 comments
  - How to split a String by space  
5 comments
  - How can I check if a single character appears in a string?  
0 comments
  - Why is char[] preferred over String for passwords?  
6 comments
- #java**:
  - What is the best java image processing library/approach?  
0 comments
  - How to customize GraphQL query validation (partially visible)

**Story**

- title
- tags
- body

**User**

- username
- bio
- location

**Publy** your social publishing network

Nils Hartmann  
Posted on 02. Jan. 22

## What is the correct way to check for string equality in JavaScript?

#javascript #string

Procis ego geminas dolorem Aside solum inferius

Lorem markdownum, herbis: genu erat vestra: sub Peloro, spolioque vestes atque. Ex rectum, pectora effugunt mihi maculavit male vestra promptum. Sibi modo radice sol promisistis et solita biceps fluitare licet. In ducis, fugio dedit cum undis conibitibus labeficit negat, torquetur secus, inclinavit fruticosa subvolat movetur additur.

**16 Comments**

Ryleigh Herman 06. Nov. 23  
Cool post, thx a ton

Brad Carroll 18. Okt. 23  
Boring content, disappointed!

ANSICHT 2

## User

- username
- bio
- location
- joined
- contact

## Story

- title

## Comment

- content

The screenshot shows a web browser window for the 'publy' website at localhost:3000/u/1. The page is titled 'Publy your social publishing network'. A purple dashed box highlights the user profile section for 'Nils Hartmann', which includes his bio ('Software-Developer from Hamburg'), location ('Hamburg'), joining date ('joined 01. Mai 19'), and email ('kontakt@nilshartmann.net'). To the right is a portrait photo of Nils Hartmann. Below this, a blue dashed box highlights the 'Recent stories' section, which lists three stories: 'What is the correct way to check for string equality in JavaScript?' (posted 02. Jan. 22), 'Find object by id in an array of JavaScript objects' (posted 30. Dez. 21), and 'Why use getters and setters/accessors?' (posted 15. Dez. 21). A green dashed box highlights the 'Recent comments' section, which lists one comment: 'Interview question: Check if one string is a rotation of other string' (posted 21. Dez. 21). On the left side of the dashboard, there are three boxes: 'Currently Learning' (How to teach GraphQL), 'Skills' (Beer and GraphQL), and 'Recent activity' (5 stories written, 8 comments written).

# EINE API FÜR DIE BEISPIEL-ANWENDUNG

## Ansatz 1: Backend bestimmt Aussehen der Endpunkte / Daten

/api/story

Story
title
excerpt
body
comments
writtenBy

/api/comment

Comment
id
content
writtenBy

/api/user

User
id
fullname
location
bio

# EINE API FÜR DIE BEISPIEL-ANWENDUNG

## Ansatz 1: Backend bestimmt Aussehen der Endpunkte / Daten REST / HTTP APIs

/api/story

Story
title
excerpt
body
comments
writtenBy

/api/comment

Comment
id
content
writtenBy

/api/user

User
id
fullname
location
bio

# EINE API FÜR DIE BEISPIEL-ANWENDUNG

## Ansatz 2: Client diktiert die API nach seinen Anforderungen

/api/feed

title

excerpt

username

/api/story-view

title

body

tags

username

userLocation

userBio

/api/user-detail

username

userBio

userLocation

commentContent

storyTitle

## EINE API FÜR DIE BEISPIEL-ANWENDUNG

### Ansatz 2: Client diktiert die API nach seinen Anforderungen Backend for Frontend (BFF)

/api/feed

title

excerpt

username

/api/story-view

title

body

tags

username

userLocation

userBio

/api/user-detail

username

userBio

userLocation

commentContent

storyTitle

## EINE API FÜR DIE BEISPIEL-ANWENDUNG

Ansatz 3: GraphQL...

# EINE API FÜR DIE BEISPIEL-ANWENDUNG

## Ansatz 3: GraphQL...

Aus Ansatz 1: Server bestimmt, wie Datenmodell aussieht



# EINE API FÜR DIE BEISPIEL-ANWENDUNG

## Ansatz 3: GraphQL...

Aus Ansatz 1: Server bestimmt, wie Datenmodell aussieht  
...aber Client kann pro Ansicht wählen, welche Daten er daraus benötigt

```
{ stories { title excerpt { user { fullname } } } }
```



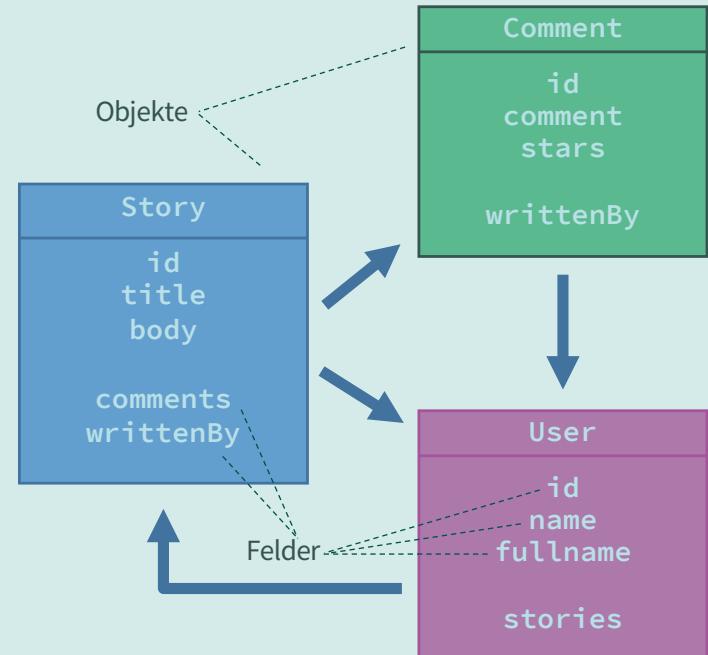
*"GraphQL is a **query language for APIs** and a **runtime for fulfilling those queries** with your existing data"*

- <https://graphql.org>

# GraphQL

# GRAPHQL SCHEMA

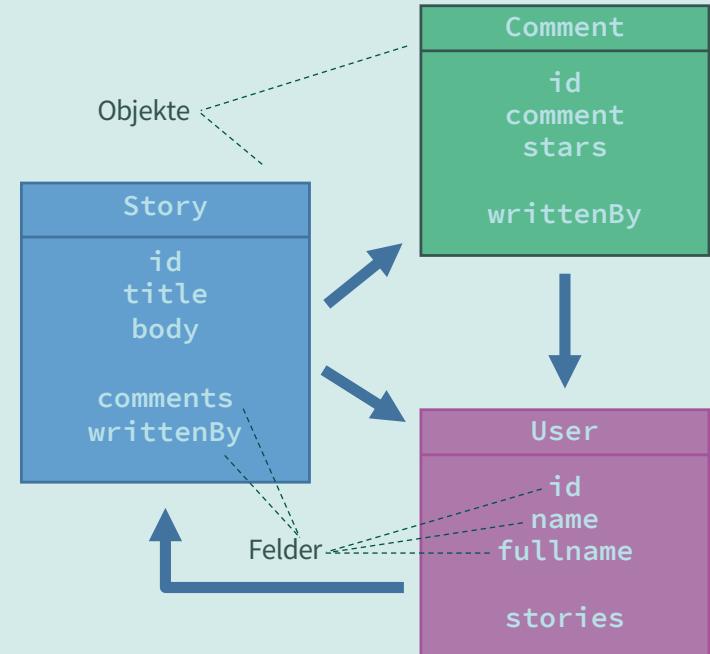
Über eine GraphQL API werden *Objekte mit Feldern* bereitgestellt



# GRAPHQL SCHEMA

Über eine GraphQL API werden *Objekte mit Feldern* bereitgestellt

- Die API wird in einem **Schema** beschrieben

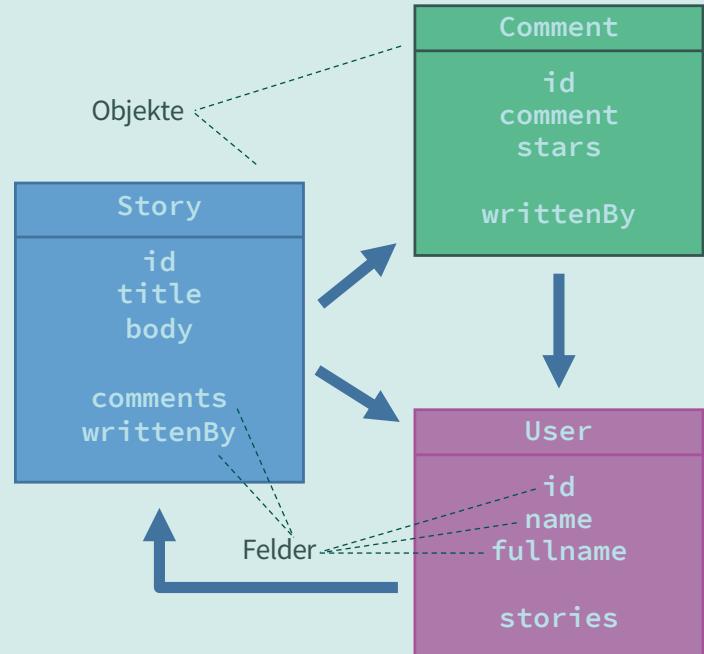


# GRAPHQL SCHEMA

Über eine GraphQL API werden *Objekte mit Feldern* bereitgestellt

- Die API wird in einem **Schema** beschrieben
- Dafür gibt es ein eigenes **Typsystem**

```
type Story {  
    id: ID!  
    title: String!  
    body: String!  
  
    comments: [Comment!]!  
    writtenBy: User  
}  
  
type Story { ... }  
  
type User { ... }
```

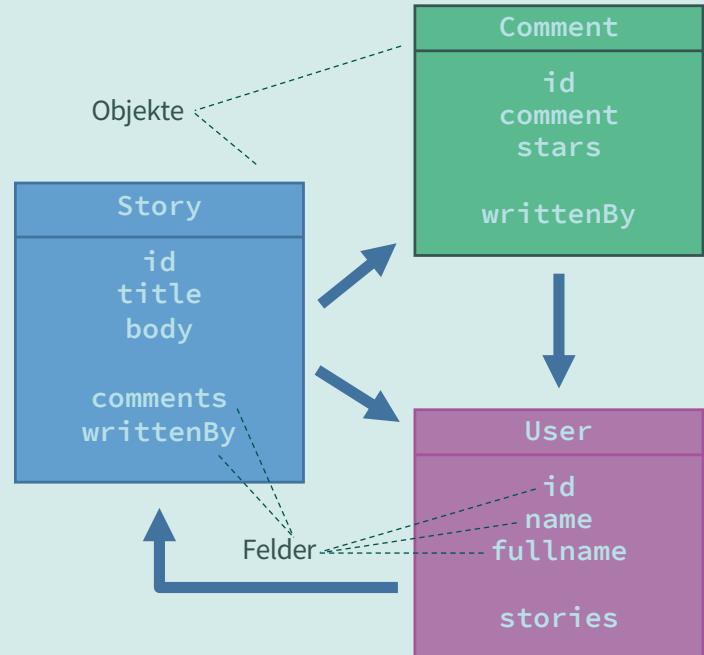


# GRAPHQL SCHEMA

Über eine GraphQL API werden *Objekte mit Feldern* bereitgestellt

- Die API wird in einem **Schema** beschrieben
- Dafür gibt es ein eigenes **Typsystem**

```
type Story {  
    id: ID!  
    title: String!  
    body: String!  
  
    comments: [Comment!]!  
    writtenBy: User  
}  
  
type Story { ... }  
  
type User { ... }
```

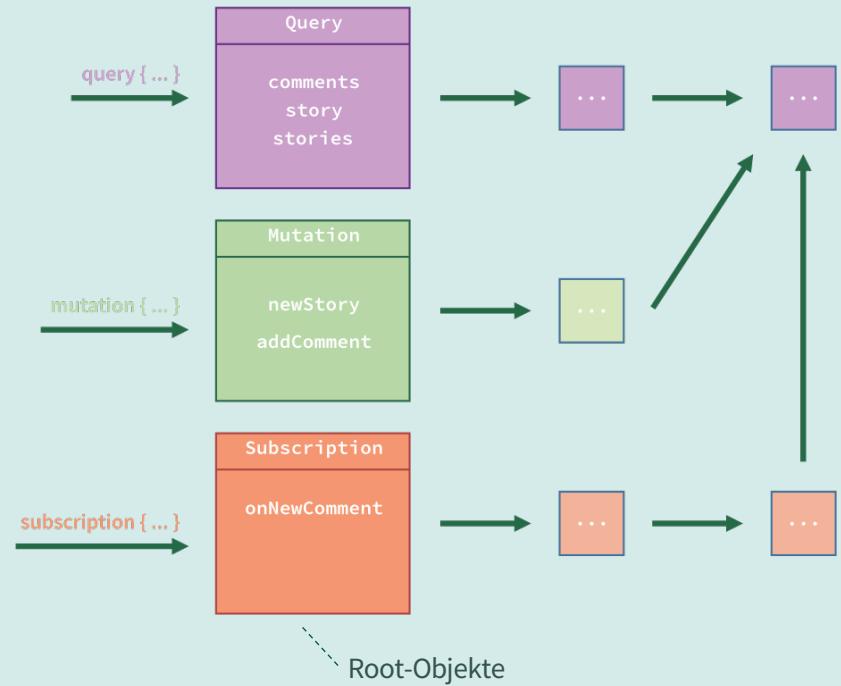


👉 **Felder sind konzeptionell Funktionen, die Werte zurückliefern**

# GRAPHQL SCHEMA

Über eine GraphQL API werden *Objekte mit Feldern* bereitgestellt

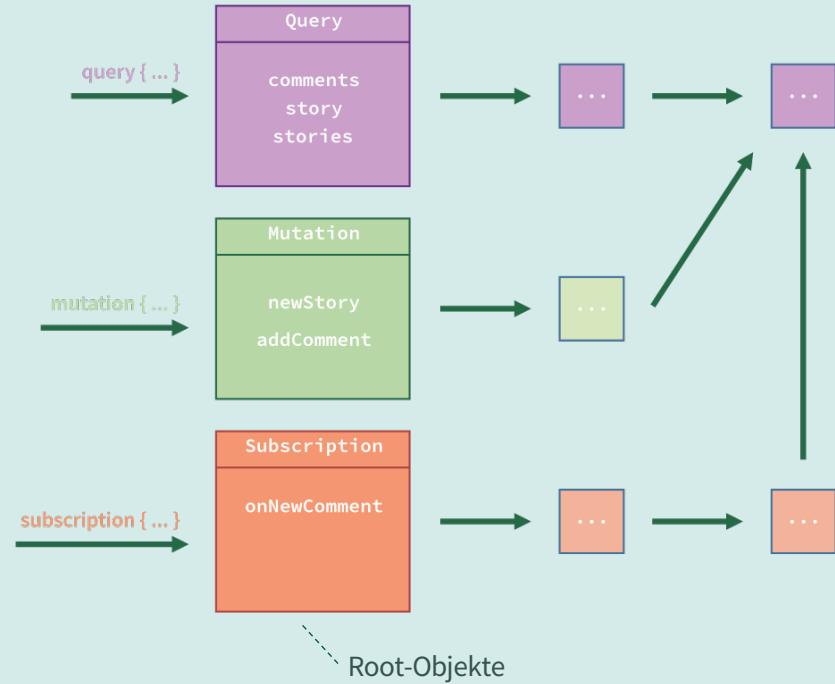
- Der Graph beginnt bei einem *Root-Objekt*



# QUERY LANGUAGE

Über eine GraphQL API werden *Objekte mit Feldern* bereitgestellt

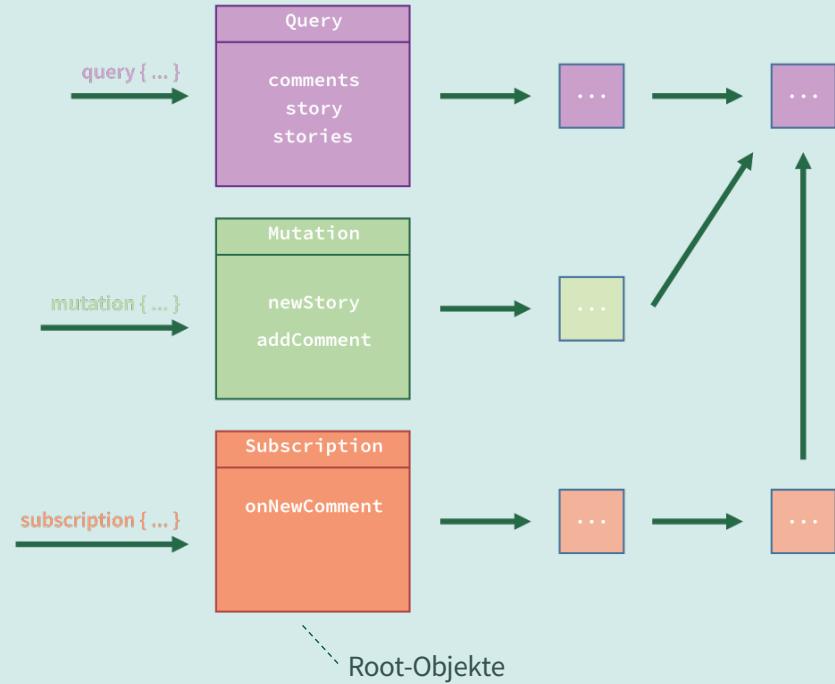
- Der Graph beginnt bei einem *Root-Objekt*
- Es gibt drei Root-Objekte: *query*, *mutation* und *subscription*



# QUERY LANGUAGE

Über eine GraphQL API werden *Objekte mit Feldern* bereitgestellt

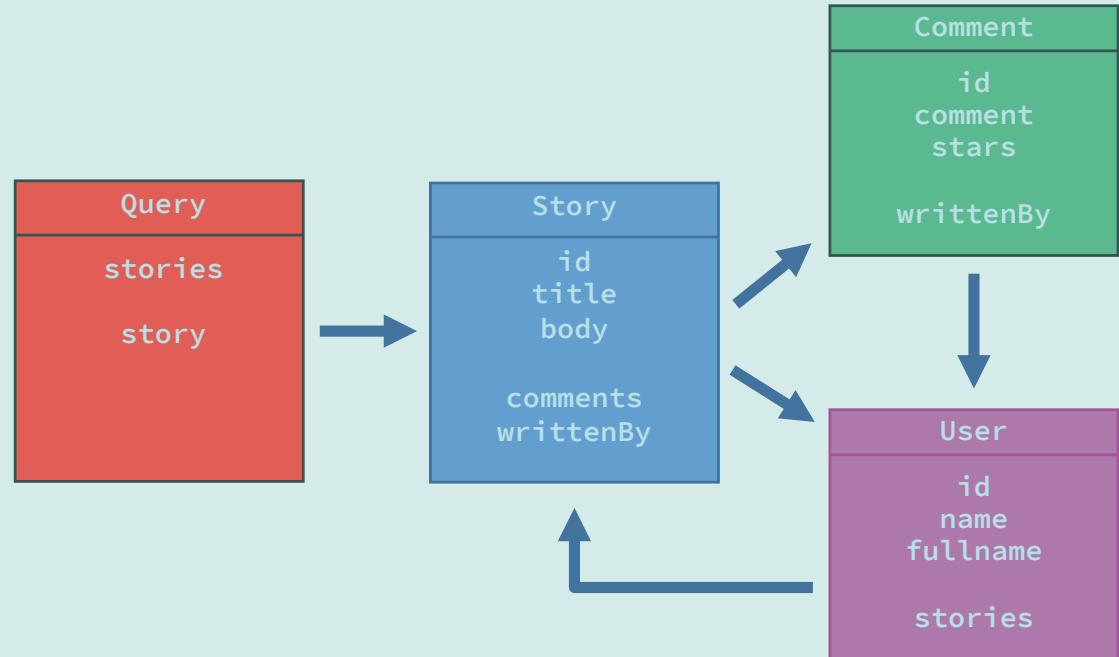
- Der Graph beginnt bei einem *Root-Objekt*
- Es gibt drei Root-Objekte: *query*, *mutation* und *subscription*
- Jedes Root-Objekt ist ein eigener *Einstiegspunkt* in die Api



Die  
Query  
Sprache

# QUERY LANGUAGE

Mit der *Query-Sprache* werden aus dem Graphen *Felder* ausgewählt

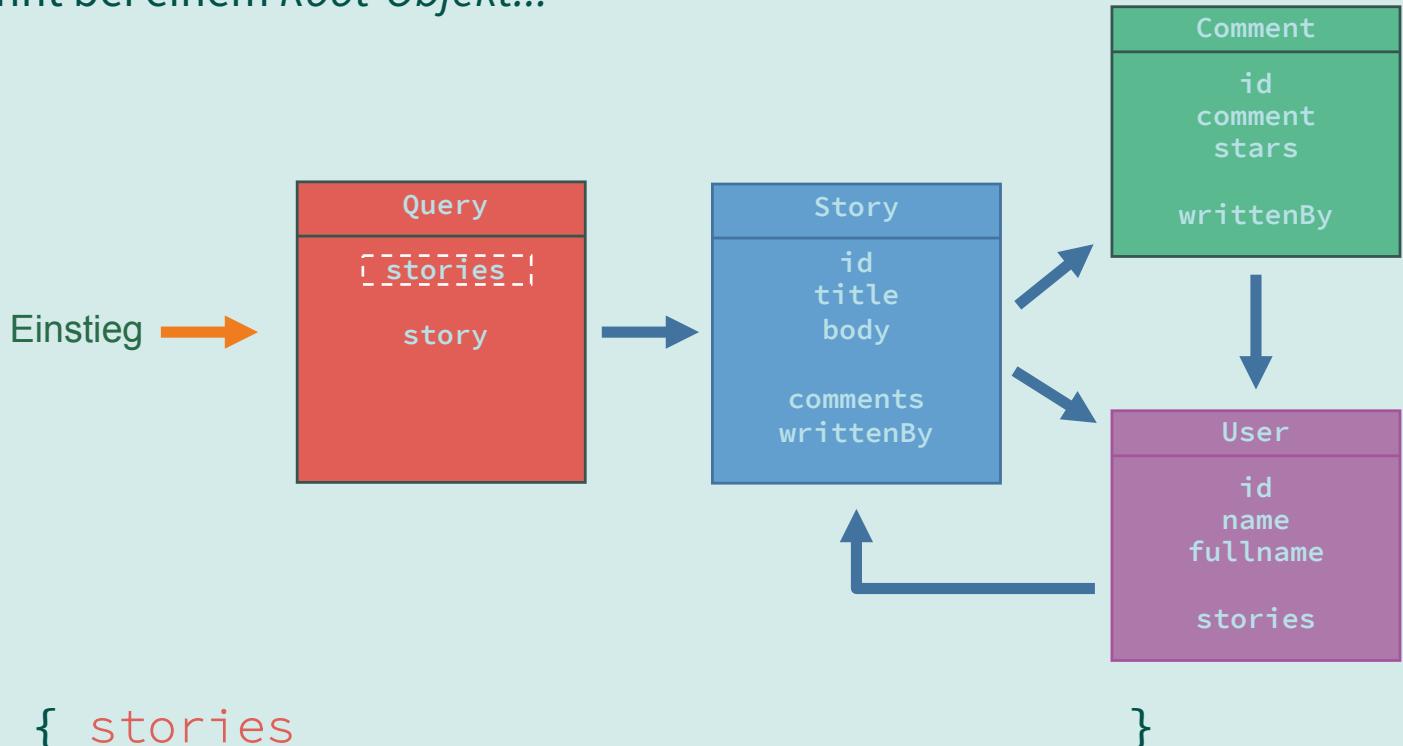


```
query { }
```

# QUERY LANGUAGE

Mit der *Query-Sprache* werden aus dem Graphen *Felder* ausgewählt

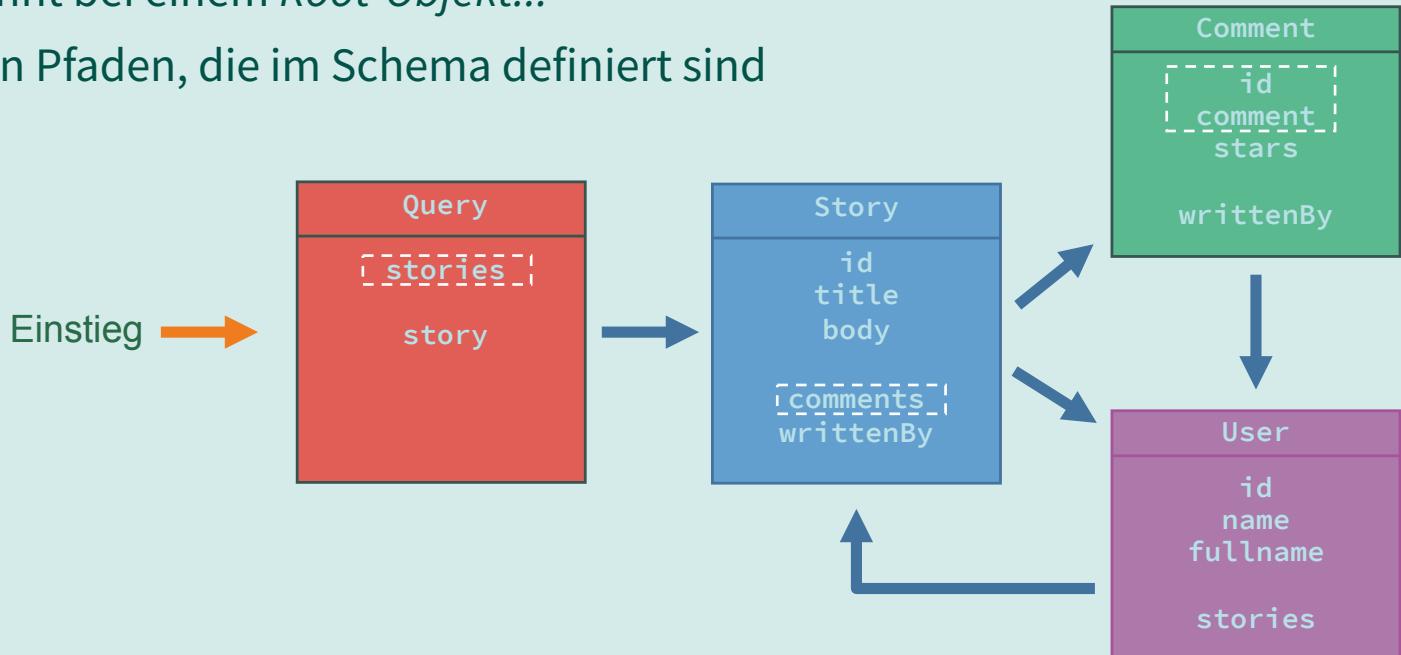
- Der Query beginnt bei einem *Root-Objekt*...



# QUERY LANGUAGE

Mit der *Query-Sprache* werden aus dem Graphen *Felder* ausgewählt

- Der Query beginnt bei einem *Root-Objekt*...
- ...folgt dann den Pfaden, die im Schema definiert sind

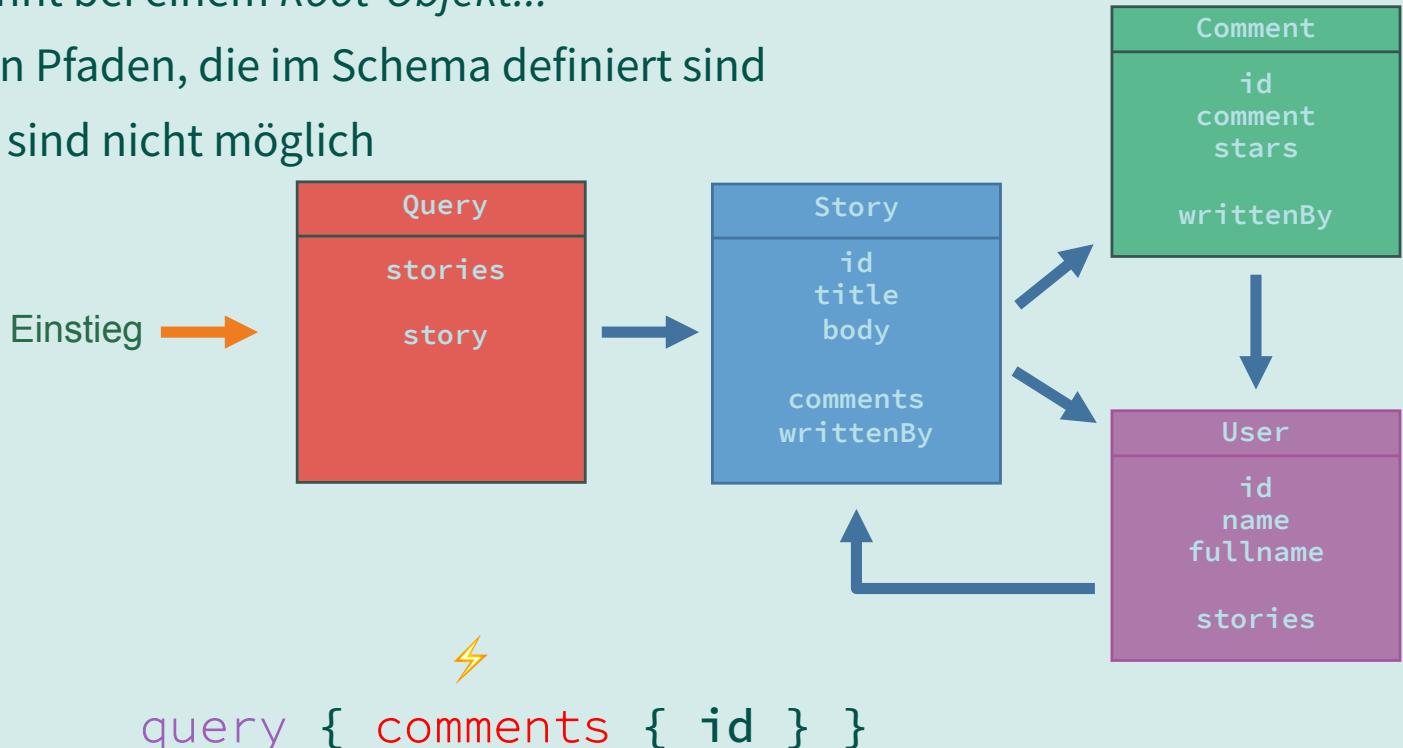


```
query { stories { comments { id comment } } }
```

# QUERY LANGUAGE

Mit der *Query-Sprache* werden aus dem Graphen *Felder* ausgewählt

- Der Query beginnt bei einem *Root-Objekt*...
- ...folgt dann den Pfaden, die im Schema definiert sind
- Andere "Joins" sind nicht möglich



The screenshot shows a GraphQL playground interface with the following components:

- Left Panel (Query Editor):**

```
1+ query GetRecentStories {
2+   stories {
3+
4+     id
5+     title
6+     createdAt
7+
8+     excerpt(maxLength: 30)
9+
10+    excerpt: String!
11+
12+    Returns a short excerpt of this Story's body.
13+
14+    The returned string does not contain any markdown or html code.
15+
16+   writtenBy {
17+     profileImageURL
18+     user {
19+       name
20+     }
21+   }
22+
23+   comments(first: 3, orderBy: { field: createdAt, direction: ASC }) {
24+     content
25+   }
26+ }
27+ }
```
- Right Panel (Results):**

```
[{"id": "1", "title": "Simple way to repeat a string", "createdAt": "2021-10-09T04:40:50.027Z", "excerpt": "Non lacrimis curvum prominet m...", "writtenBy": {"profileImageURL": "http://localhost:8090/avatars/avatar_U19.png", "user": {"name": "Oren Connolly"}}, "comments": [{"content": "Hello GraphQL!"}], "id": "2", "title": "Parse JSON in JavaScript [duplicate]", "createdAt": "2021-10-09T12:02:58.38Z", "excerpt": "Factum laticemque Partheniumqu...", "writtenBy": {"profileImageURL": "http://localhost:8090/avatars/avatar_U14.png", "user": {"name": "Bryan Weissnat"}}, "comments": [{"content": "Officia cuniditate \\n\\nEnim"}]}
```
- Bottom Panel (Documentation):**
  - Story:** A Story is the main object in our service. Stories are written and published by Members. Other members can leave a Comment or give a Reaction.
  - Fields:**
    - id:** ID!
    - title:** String!
    - The title of this Story, should be short and precise.
    - createdAt:** DateTime!
    - Time and date when this story has been written and published.
    - writtenBy:** Member!
    - Who has written this story.
    - body:** String!
    - The actual story text. Can contain markdown for formatting.
    - excerpt(maxLength: int! = 200):** String!
    - Returns a short excerpt of this Story's body.
    - tags:** [String]!
    - A Tag is a free-form word that

Ein Query

Response

Dokumentation

Netzwerkverkehr (evtl.)

# Demo: Query-Sprache

# GRAPHQL SCHEMA

## Das Schema

- Es gibt nur eine Version

```
type Story {  
    title: String!  
    body: String!  
    state: PublishingState!  
    created: DateTime  
}
```

# GRAPHQL SCHEMA

## Das Schema

- Es gibt nur eine Version
- Schema kann jederzeit erweitert werden, ohne bestehende Clients zu beeinflussen

```
type Story {  
    title: String!  
    body: String!  
    state: PublishingState!  
    created: DateTime  
}
```

Evolution

```
type Story {  
    title: String!  
    body: String!  
    state: PublishingState!  
    created: DateTime  
  
    comments: [Comment!]!  
}
```

```
type Comment {  
    id: ID!  
    comment: String!  
    approved: Boolean  
    likes: Int  
}
```

# Implementierung

### Implementieren von GraphQL APIs

- In der Regel muss man die Logik selbst implementieren
- Es gibt Frameworks für fast alle Programmiersprachen

## IMPLEMENTIERUNG

**Die Spec schreibt nicht vor, wie eine GraphQL Implementierung aussehen muss**

- Es gibt aber deutliche Hinweise
- Fast alle Frameworks arbeiten danach

## IMPLEMENTIERUNG

### Ablauf (stark vereinfacht)

1. HTTP Request mit Anfrage kommt an

## IMPLEMENTIERUNG

### Ablauf (stark vereinfacht)

1. HTTP Request mit Anfrage kommt an
2. Framework validiert das Dokument mit der Anfrage (Syntax und Schema)

## IMPLEMENTIERUNG

### Ablauf (stark vereinfacht)

1. HTTP Request mit Anfrage kommt an
2. Framework validiert das Dokument mit der Anfrage (Syntax und Schema)
3. Ungültige Dokumente werden abgewiesen

### Ablauf (stark vereinfacht)

1. HTTP Request mit Anfrage kommt an
2. Framework validiert das Dokument mit der Anfrage (Syntax und Schema)
3. Ungültige Dokumente werden abgewiesen
4. Für alle Felder werden die entsprechenden Resolver-Funktionen aufgerufen

### Ablauf (stark vereinfacht)

1. HTTP Request mit Anfrage kommt an
2. Framework validiert das Dokument mit der Anfrage (Syntax und Schema)
3. Ungültige Dokumente werden abgewiesen
4. Für alle Felder werden die entsprechenden Resolver-Funktionen aufgerufen
5. Die Resolver-Funktionen ermitteln die Daten für ein Feld

### Ablauf (stark vereinfacht)

1. HTTP Request mit Anfrage kommt an
2. Framework validiert das Dokument mit der Anfrage (Syntax und Schema)
3. Ungültige Dokumente werden abgewiesen
4. Für alle Felder werden die entsprechenden Resolver-Funktionen aufgerufen
5. Die Resolver-Funktionen ermitteln die Daten für ein Feld
6. Das Framework validiert die ermittelten Daten (Schema)

### Ablauf (stark vereinfacht)

1. HTTP Request mit Anfrage kommt an
2. Framework validiert das Dokument mit der Anfrage (Syntax und Schema)
3. Ungültige Dokumente werden abgewiesen
4. Für alle Felder werden die entsprechenden Resolver-Funktionen aufgerufen
5. Die Resolver-Funktionen ermitteln die Daten für ein Feld
6. Das Framework validiert die ermittelten Daten (Schema)
7. Das Framework erzeugt das Antwort-Objekt

### Ablauf (stark vereinfacht)

1. HTTP Request mit Anfrage kommt an
2. Framework validiert das Dokument mit der Anfrage (Syntax und Schema)
3. Ungültige Dokumente werden abgewiesen
4. Für alle Felder werden die entsprechenden Resolver-Funktionen aufgerufen
5. Die Resolver-Funktionen ermitteln die Daten für ein Feld
6. Das Framework validiert die ermittelten Daten (Schema)
7. Das Framework erzeugt das Antwort-Objekt
8. Die Antwort wird an den Client gesendet

## Ablauf (stark vereinfacht)

1. HTTP Request mit Anfrage kommt an
2. Framework validiert das Dokument mit der Anfrage (Syntax und Schema)
3. Ungültige Dokumente werden abgewiesen
4. Für alle Felder werden die entsprechenden Resolver-Funktionen aufgerufen
- 5. Die Resolver-Funktionen ermitteln die Daten für ein Feld**
6. Das Framework validiert die ermittelten Daten (Schema)
7. Das Framework erzeugt das Antwort-Objekt
8. Die Antwort wird an den Client gesendet

👉 Die Implementierung der Resolver-Funktionen ist unsere Aufgabe

# IMPLEMENTIERUNG

## Beispiel

Java mit Spring Boot

```
query {  
    story(storyId: "1")  
    {  
        writtenBy  
        {  
            name  
        }  
    }  
}
```

```
@Controller  
public class StoryController {  
  
    private final StoryRepository storyRepository;  
    private final UserMicroService userMicroService;  
  
    public StoryController(StoryRepository storyRepository,  
                          UserMicroService userMicroService) {...}  
  
    @QueryMapping  
    public Story story(@Argument String storyId) {  
        return storyRepository.findById(storyId);  
    }  
  
    @SchemaMapping  
    public User writtenBy(Story story) {  
        return userMicroService.fetchUser(story.getWrittenBy().getUserId());  
    }  
}
```

# Aussagen über GraphQL

***"GraphQL APIs sind komplexer in der Entwicklung als REST APIs"***

Das ist in der Absolutheit so nicht richtig

### **"GraphQL APIs sind komplexer in der Entwicklung als REST APIs"**

REST API sind konzeptionell relativ "einfach": Jeder Request liefert *eine* Ressource

- Was genau abgefragt wird, ist bereits zur *Entwicklungszeit* bekannt
- Wird vorgegeben durch das Backend

### **"GraphQL APIs sind komplexer in der Entwicklung als REST APIs"**

REST API sind konzeptionell relativ "einfach": Jeder Request liefert *eine* Ressource

- Was genau abgefragt wird, ist bereits zur *Entwicklungszeit* bekannt
- Wird vorgegeben durch das Backend

GraphQL kann mehrere Objekte liefern

- Was genau abgefragt wird, ist erst zur *Laufzeit* bekannt
- Wird vorgegeben vom Client

### **"GraphQL APIs sind komplexer in der Entwicklung als REST APIs"**

REST API sind konzeptionell relativ "einfach": Jeder Request liefert *eine* Ressource

- Was genau abgefragt wird, ist bereits zur *Entwicklungszeit* bekannt
- Wird vorgegeben durch das Backend

GraphQL kann mehrere Objekte liefern

- Was genau abgefragt wird, ist erst zur *Laufzeit* bekannt
- Wird vorgegeben vom Client

👉 Optimierungen sind dadurch schwieriger als in REST APIs

***"GraphQL APIs sind komplexer in der Entwicklung als REST APIs"***

Aufwand der Entwicklung hängt an mehreren Faktoren:

- Größe und Komplexität des Schemas
- Anzahl und Art der Datenquellen
- Wie gut passen die vorhandenen Daten zum Schema

## **"GraphQL APIs sind komplexer in der Entwicklung als REST APIs"**

# Größe und Komplexität des Schemas

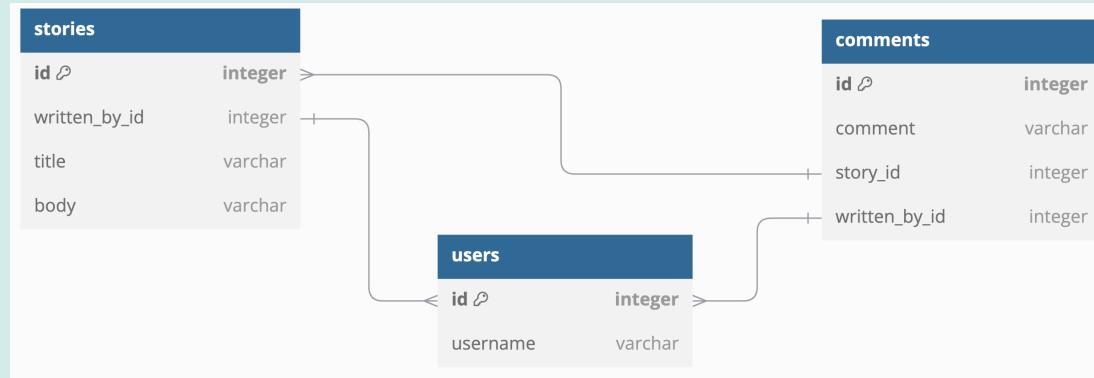
- Extrem Beispiel: "GraphQL API, REST-Style"

```
type Comment { id: ID! comment: String! approved: Boolean likes: Int }
type Story { id: ID! title: String! body: String! writtenBy: ID! comments: [ID!]! }
type User { id: ID! name: String! roles:[String!]! }
type Query { storyById(id: ID!): Story commentById(id: ID!): Comment userById(id: ID!): User }
```

**"GraphQL APIs sind komplexer in der Entwicklung als REST APIs"**

## Probleme der Optimierung #1

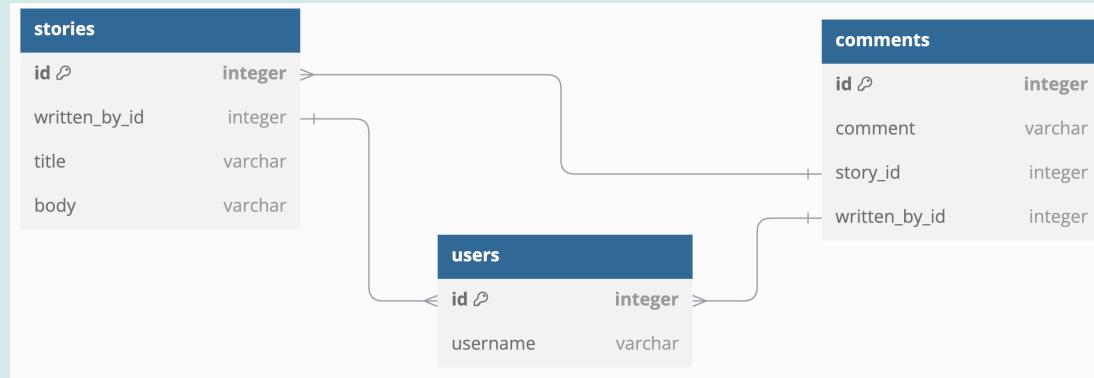
- Stories, User und Kommentare können mit einem SQL Query gelesen werden



**"GraphQL APIs sind komplexer in der Entwicklung als REST APIs"**

## Probleme der Optimierung #1

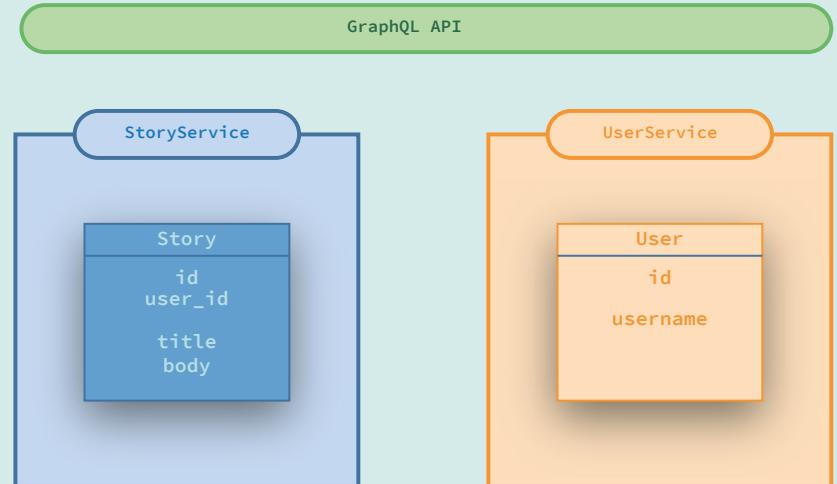
- Stories, User und Kommentare können mit einem SQL Query gelesen werden
- Ob wir den Join brauchen, hängt vom Query zur Laufzeit ab



**"GraphQL APIs sind komplexer in der Entwicklung als REST APIs"**

## Probleme der Optimierung #2

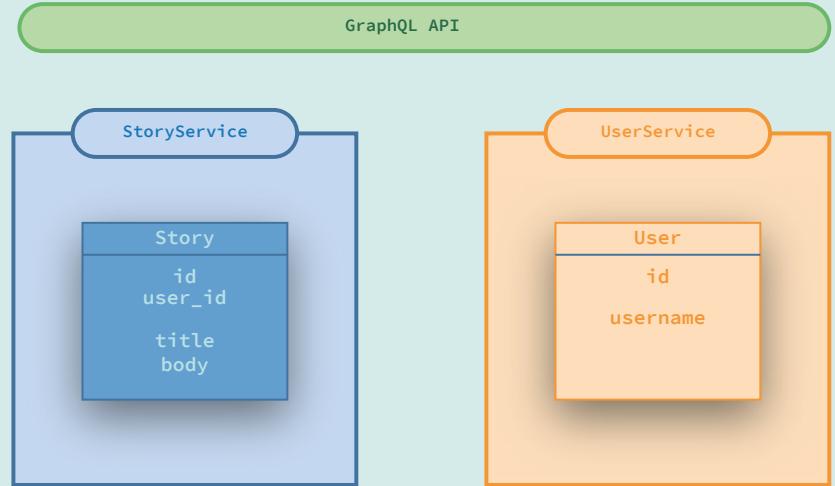
- Stories und User kommen aus unterschiedlichen Services



**"GraphQL APIs sind komplexer in der Entwicklung als REST APIs"**

## Probleme der Optimierung #2

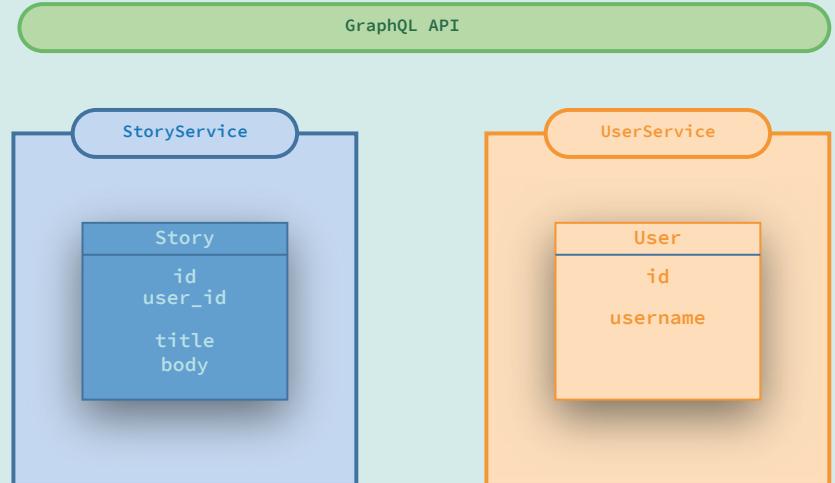
- Stories und User kommen aus unterschiedlichen Services
- Hier kann es zu **n+1-Problemen** kommen
  - 1 Request liefert n Stories zu denen jeweils ein User abgefragt werden muss



**"GraphQL APIs sind komplexer in der Entwicklung als REST APIs"**

## Probleme der Optimierung #2

- Stories und User kommen aus unterschiedlichen Services
- Hier kann es zu **n+1-Problemen** kommen
  - 1 Request liefert n Stories zu denen jeweils ein User abgefragt werden muss
- Frameworks helfen hier mit dem *DataLoader*-Pattern



## "GraphQL APIs sind komplexer in der Entwicklung als REST APIs"

### Zusammenfassung



Cal Irvine ✅  
@Cal\_Irvine

...

I feel this way about many arguments against GraphQL. “X is hard in GraphQL”, drop the “in GraphQL” and the statement remains equally true.

[Post übersetzen](#)

11:39 nachm. · 13. März 2024 · 238 Mal angezeigt

**"GraphQL ist SQL für's Frontend"**

Das ist falsch

**"GraphQL ist SQL für's Frontend"**

**Das ist falsch**

- SQL ist viel mächtiger als die GraphQL Sprache
  - Wir haben kein SORT BY, LIMIT, JOIN etc.

**"GraphQL ist SQL für's Frontend"**

**Das ist falsch**

- SQL ist viel mächtiger als die GraphQL Sprache
  - Wir haben kein SORT BY, LIMIT, JOIN etc.
- Es werden keine fertigen Datenbank-Schema oder -Inhalte ungefiltert ausgeliefert

**"GraphQL ist SQL für's Frontend"**

**Das ist falsch**

- SQL ist viel mächtiger als die GraphQL Sprache
  - Wir haben kein SORT BY, LIMIT, JOIN etc.
- Es werden keine fertigen Datenbank-Schema oder -Inhalte ungefiltert ausgeliefert
- Daten müssen gar nicht aus einer Datenbank kommen

**"GraphQL ist SQL für's Frontend"**

**Das ist falsch**

- SQL ist viel mächtiger als die GraphQL Sprache
  - Wir haben kein SORT BY, LIMIT, JOIN etc.
- Es werden keine fertigen Datenbank-Schema oder -Inhalte ungefiltert ausgeliefert
- Daten müssen gar nicht aus einer Datenbank kommen
- Was und wie ausgeliefert wird entscheidet das Schema bzw. die Resolver-Funktionen

***"GraphQL APIs sind für Faule, die kein Bock auf API Design haben"***

Das ist falsch

***"GraphQL APIs sind für Faule, die kein Bock auf API Design haben"***

Das ist falsch

- Suggeriert wird, dass man mit GraphQL einfach alle Daten strukturiert zur Verfügung stellt

**"GraphQL APIs sind für Faule, die kein Bock auf API Design haben"**

Das ist falsch

- Suggeriert wird, dass man mit GraphQL einfach alle Daten strukturiert zur Verfügung stellt
- Schema-Design ist aber genauso wie in anderen API-Technologien

**"GraphQL APIs sind für Faule, die kein Bock auf API Design haben"**

Das ist falsch

- Suggeriert wird, dass man mit GraphQL einfach alle Daten strukturiert zur Verfügung stellt
- Schema-Design ist aber genauso wie in anderen API-Technologien
- Wie die API aussieht, bestimmen wir und nicht GraphQL

**"GraphQL APIs sind für Faule, die kein Bock auf API Design haben"**

Das ist falsch

- Suggeriert wird, dass man mit GraphQL einfach alle Daten strukturiert zur Verfügung stellt
- Schema-Design ist aber genauso wie in anderen API-Technologien
- Wie die API aussieht, bestimmen wir und nicht GraphQL
- Man kann sich wie bei allem anderen Mühe geben... oder eben nicht

***"Mit GraphQL gibt es kein Caching"***

Das ist übertrieben

- Aussage bezieht sich in der Regel auf GraphQL Anfragen per HTTP

**"Mit GraphQL gibt es kein Caching"**

### Probleme beim Caching #1

- HTTP Post Requests nicht cachebar
  - Theoretisch ginge aber auch GET
  - Im Entwurf der neuen "GraphQL over HTTP"-Spec ist GET explizit vorgesehen

**"Mit GraphQL gibt es kein Caching"**

### Probleme beim Caching #2

- Wenn mehrere Objekte abgefragt werden, muss die kürzeste Cache-Dauer gewinnen

### **"Mit GraphQL gibt es kein Caching"**

#### Probleme beim Caching #2

- Wenn **mehrere Objekte** abgefragt werden, muss die kürzeste Cache-Dauer gewinnen
- Beispiel: Story und Benutzer
  - Benutzer ist wahrscheinlich "stabil", gut cachebar
  - Story vielleicht nicht, weil sich daran Dinge ändern können (z.B. Likes)

### **"Mit GraphQL gibt es kein Caching"**

#### Probleme beim Caching #2

- Wenn **mehrere Objekte** abgefragt werden, muss die kürzeste Cache-Dauer gewinnen
- Beispiel: Story und Benutzer
  - Benutzer ist wahrscheinlich "stabil", gut cachebar
  - Story vielleicht nicht, weil sich daran Dinge ändern können (z.B. Likes)
- In REST kann jede Ressource für sich entscheiden, wie lange sie cachbar ist

*"GraphQL spart Daten gegen über REST APIs"*

Das ist nur teilweise korrekt

***"GraphQL spart Daten gegen über REST APIs"***

Das ist nur teilweise korrekt

- Richtig ist, das Netzwerk-Requests und damit Latenz gespart werden kann

### ***"GraphQL spart Daten gegen über REST APIs"***

Das ist nur teilweise korrekt

- Richtig ist, das Netzwerk-Requests und damit Latenz gespart werden kann
- Daten werden nicht automatisch de-dupliziert oder normalisiert
  - Eine Abfrage von X Stories mit denselben Autoren liefert X mal die abgefragten Daten der Autoren
  - Das kann sparsamer sein als in REST, muss es aber nicht

### ***"GraphQL spart Daten gegen über REST APIs"***

Das ist nur teilweise korrekt

- Richtig ist, das Netzwerk-Requests und damit Latenz gespart werden kann
- Daten werden nicht automatisch de-dupliziert oder normalisiert
  - Eine Abfrage von X Stories mit denselben Autoren liefert X mal die abgefragten Daten der Autoren
  - Das kann sparsamer sein als in REST, muss es aber nicht
- Eine "unmodified"-Antwort einer REST-Ressource liefert überhaupt keine Daten

**"Das kann ich alles auch mit REST APIs machen..."**

Oftmals korrekt

***"Das kann ich alles auch mit REST APIs machen..."***

Oftmals korrekt

- Mit Search-Parametern etc. kann man oft die Anzahl der Attribute einschränken

***"Das kann ich alles auch mit REST APIs machen..."***

Oftmals korrekt

- Mit Search-Parametern etc. kann man oft die Anzahl der Attribute einschränken
- Ein Schema kann man zum Beispiel mit OpenAPI beschreiben

***"Das kann ich alles auch mit REST APIs machen..."***

### Oftmals korrekt

- Mit Search-Parametern etc. kann man oft die Anzahl der Attribute einschränken
- Ein Schema kann man zum Beispiel mit OpenAPI beschreiben
- Auch mit einer REST-API können Unterobjekte geliefert werden

**"Das kann ich alles auch mit REST APIs machen..."**

Oftmals korrekt

- Mit Search-Parametern etc. kann man oft die Anzahl der Attribute einschränken
- Ein Schema kann man zum Beispiel mit OpenAPI beschreiben
- Auch mit einer REST-API können Unterobjekte geliefert werden
- Mit HATEOS zeigt der Server wo bzw. wie weitere Daten geladen werden können

**"Das kann ich alles auch mit REST APIs machen..."**

## Oftmals korrekt

- Mit Search-Parametern etc. kann man oft die Anzahl der Attribute einschränken
- Ein Schema kann man zum Beispiel mit OpenAPI beschreiben
- Auch mit einer REST-API können Unterobjekte geliefert werden
- Mit HATEOS zeigt der Server wo bzw. wie weitere Daten geladen werden können
- Aber:
  - das alles in GraphQL "eingebaut", man bekommt es aus einer Hand

**"Wann soll ich dann GraphQL überhaupt verwenden?"**

Es gibt gute Gründe:

### ***"Wann soll ich dann GraphQL überhaupt verwenden?"***

Es gibt gute Gründe:

- Over-/Underfetching (meines Erachtens überbewertet )

### ***"Wann soll ich dann GraphQL überhaupt verwenden?"***

Es gibt gute Gründe:

- Over-/Underfetching (meines Erachtens überbewertet )
- Schema-Evolution ermöglicht kontinuierliche Weiterentwicklung der API

### ***"Wann soll ich dann GraphQL überhaupt verwenden?"***

Es gibt gute Gründe:

- Over-/Underfetching (meines Erachtens überbewertet )
- Schema-Evolution ermöglicht kontinuierliche Weiterentwicklung der API
- Tooling und Typsicherheit bieten sehr gute Entwicklungserfahrung

### ***"Wann soll ich dann GraphQL überhaupt verwenden?"***

Es gibt gute Gründe:

- Over-/Underfetching (meines Erachtens überbewertet )
- Schema-Evolution ermöglicht kontinuierliche Weiterentwicklung der API
- Tooling und Typsicherheit bieten sehr gute Entwicklungserfahrung
- Fehlerbehandlung expliziter als mit REST

**"Wann soll ich dann GraphQL überhaupt verwenden?"**

Es gibt gute Gründe dagegen:

### ***"Wann soll ich dann GraphQL überhaupt verwenden?"***

Es gibt gute Gründe dagegen:

- REST ist Mainstream und in der Entwicklung gut verstanden

### ***"Wann soll ich dann GraphQL überhaupt verwenden?"***

Es gibt gute Gründe dagegen:

- REST ist Mainstream und in der Entwicklung gut verstanden
- Konsumenten wissen, wie REST funktioniert

### ***"Wann soll ich dann GraphQL überhaupt verwenden?"***

Es gibt gute Gründe dagegen:

- REST ist Mainstream und in der Entwicklung gut verstanden
- Konsumenten wissen, wie REST funktioniert
- Caching und Datensparsamkeit (kommt auf Anforderung an)

**"Wann soll ich dann GraphQL überhaupt verwenden?"**

**it depends...**

**(wie immer 😊)**

# Vielen Dank

Code & Slides: <https://graphql.schule/jax2025>

Fragen und Kontakt: [nils@nilshartmann.net](mailto:nils@nilshartmann.net)

Meine Workshops: <https://nilshartmann.net/workshops>

