

NILS HARTMANN

<https://nilshartmann.net>

Heilsbringer oder Teufelszeug?

GraphQL

Eine Einführung

Slides (PDF): <https://react.schule/bettercode2021-graphql>

NILS HARTMANN

nils@nilshartmann.net

Freiberuflicher Entwickler, Architekt, Trainer aus Hamburg

Java
JavaScript, TypeScript
React
GraphQL

Trainings & Workshops



<https://reactbuch.de>

HTTPS://NILSHARTMANN.NET

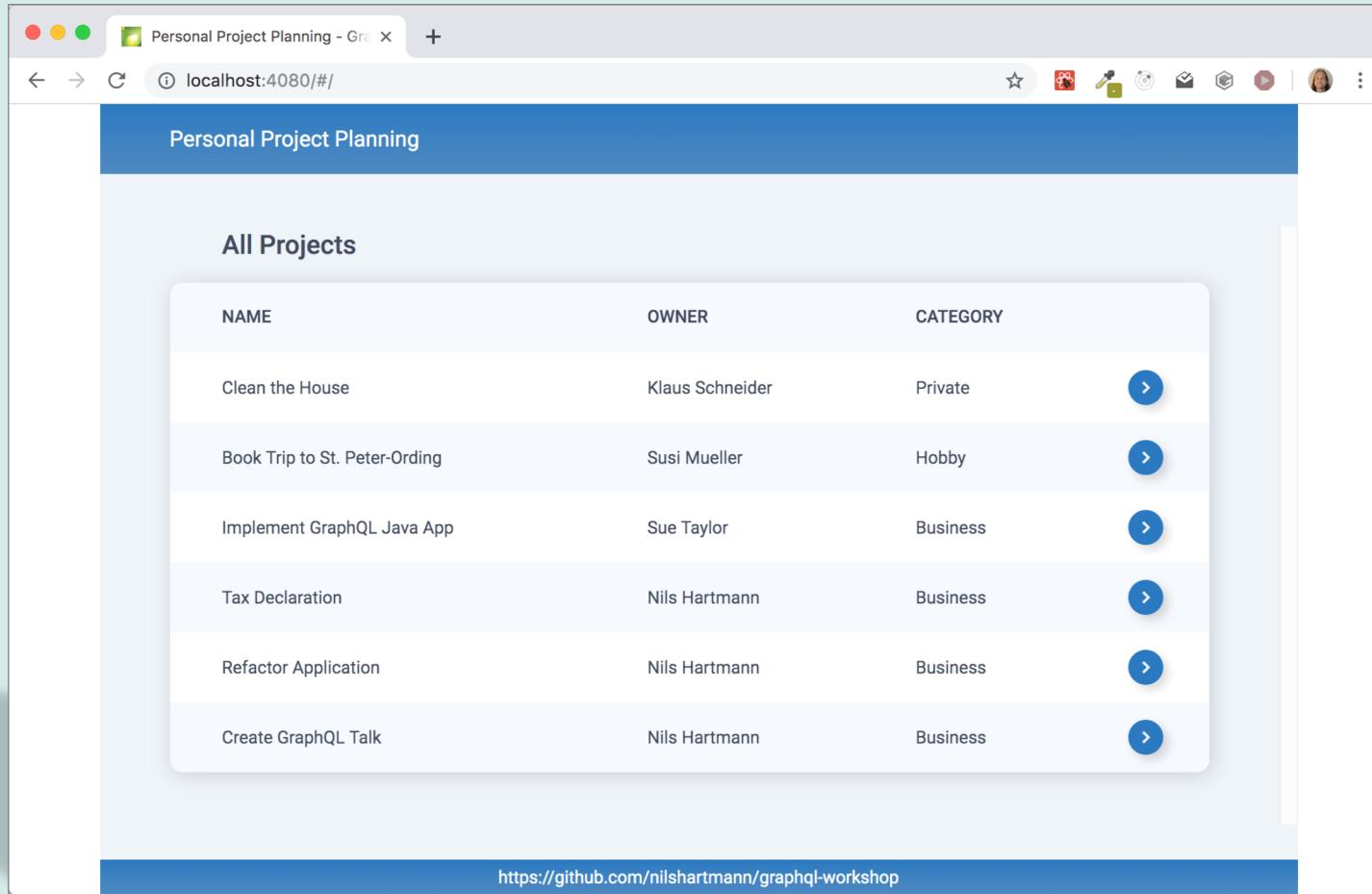
*"GraphQL is a **query language for APIs** and a **runtime for fulfilling those queries** with your existing data"*

- <https://graphql.org>

GraphQL

Spezifikation: <https://graphql.org/>

- 2015 von Facebook erstmals veröffentlicht
- Weitere Entwicklung seit 2018 in GraphQL Foundation
- Umfasst:
 - Query Sprache und -Ausführung
 - Schema Definition Language
 - Nicht: Implementierung
 - Referenz-Implementierung: graphql-js



GraphQL praktisch

Source-Code: <https://github.com/graphql-workshop>

GraphQL macht keine Aussage, wie unsere API aussieht

- 👉 Welche Daten wir zur Verfügung stellen, ist unsere Entscheidung
- 👉 Wir entscheiden, wie die API unserer Anwendung aussieht
(GraphQL ist "nur" die Technologie)

GraphQL macht keine Aussage, wie unsere API aussieht

- 👉 Welche Daten wir zur Verfügung stellen, ist unsere Entscheidung
- 👉 Wir entscheiden, wie die API unserer Anwendung aussieht
(GraphQL ist "nur" die Technologie)
- 🚫 ~~"for people who can't make up their minds"~~

API beschreiben: Das Schema

- Eine GraphQL API *muss* mit einem Schema beschrieben werden
- Schema legt fest, welche *Types* und *Fields* es gibt
- **Schema Definition Language** (SDL)

GRAPHQL APIs

Schema Definition

Fachliches Objekt mit Feldern

```
type Project {  
    id: ID!  
    title: String!
```

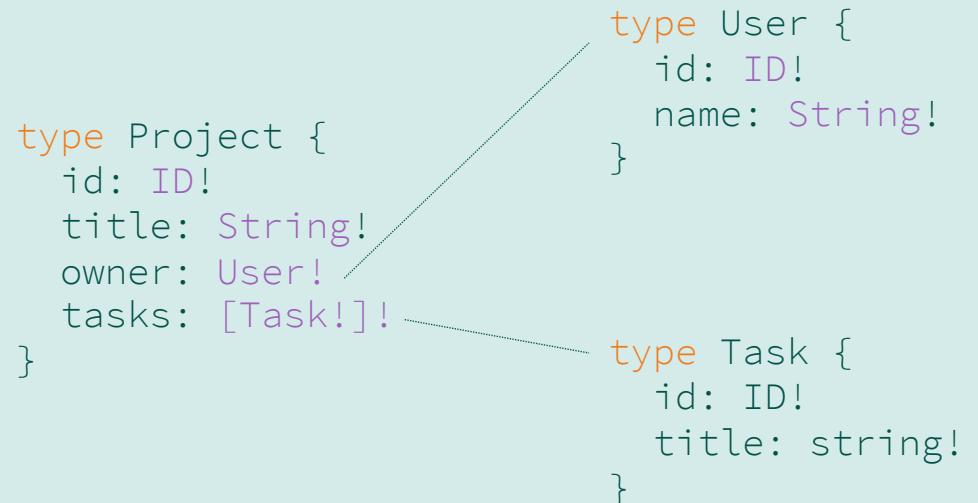
```
}
```

GRAPHQL APIs

Schema Definition

Objekt Graph

```
type User {  
  id: ID!  
  name: String!  
}  
  
type Project {  
  id: ID!  
  title: String!  
  owner: User!  
  tasks: [Task!]!  
}  
  
type Task {  
  id: ID!  
  title: string!  
}
```



The diagram illustrates the GraphQL schema definition. It shows three types: User, Project, and Task. The User type has fields for id (ID!) and name (String!). The Project type has fields for id (ID!), title (String!), owner (User!), and tasks ([Task!]!). The Task type has fields for id (ID!) and title (string!). Relationships are indicated by dotted lines: one line connects the owner field in Project to the User type, and another line connects the tasks field in Project to the Task type.

GRAPHQL APIs

Schema Definition

Einstiegspunkt in die API:
Root-Typen

```
type Query {  
    projects: [Project!]!  
    project(projectId: ID!): Project  
}  
  
type Mutation {  
    addTask(newTask: NewTask): Task!  
}
```

```
type Project {  
    id: ID!  
    title: String!  
    owner: User!  
    tasks: [Task!]!  
}
```

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

```
type Task {  
    id: ID!  
    title: string!  
}
```

Schema: Instrospection

- Root-Felder "__schema" und "__type" (Beispiel)

```
query {  
  __type(name: "Project") {  
    name  
    kind  
    description  
    fields {  
      name description  
      type { ofType { name } }  
    }  
  }  
}
```

```
{  
  "data": {  
    "__type": {  
      "name": "Project",  
      "kind": "OBJECT",  
      "description": "Representation of a Project",  
      "fields": [ {  
        "name": "id", "description": "Id for this Project",  
        "type": { "ofType": { "name": "ID" }  
      }  
    },  
    {  
      "name": "title", "description": "Title of the project",  
      "type": { "ofType": { "name": "Int" }  
    }  
    ],  
    "..."  
  }  
}
```

Schema: Instrospection

- Root-Felder "__schema" und "__type" (Beispiel)

```
query {  
  __type(name: "Project") {  
    name  
    kind  
    description  
    fields {  
      name description  
      type { ofType { name } }  
    }  
  }  
}
```

```
{  
  "data": {  
    "__type": {  
      "name": "Project",  
      "kind": "OBJECT",  
      "description": "Representation of a Project",  
      "fields": [ {  
        "name": "id", "description": "Id for this Project",  
        "type": { "ofType": { "name": "ID" }  
      }  
    },  
  },  
}
```

Sehr gutes Tooling in IDE und anderen Tools! 😍

- "Web-IDEs": GraphiQL, Playground
- Plug-in für IntelliJ
- Code Generatoren

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

- <https://graphql.org>

GraphQL

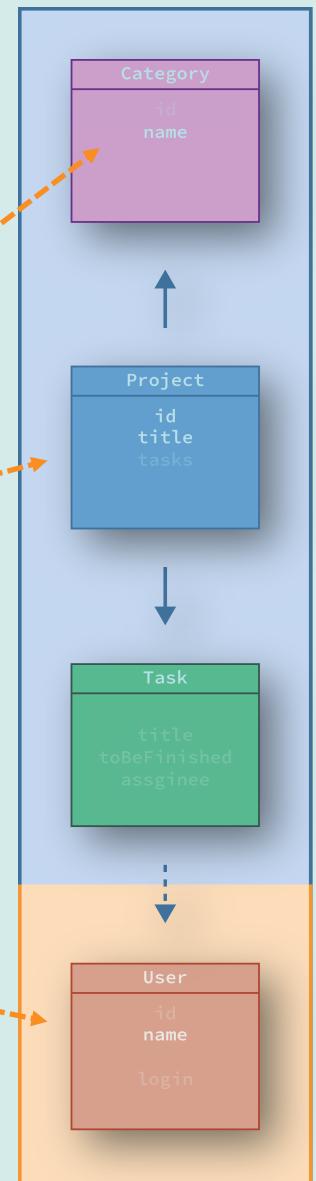
TEIL 1: ABFRAGEN UND SCHEMA

GRAPHQL EINSATZSzenariEN

Use-Case spezifische Abfragen – 1

```
{ projects {  
  id  
  title  
  owner { name }  
  category { name }  
}
```

NAME	OWNER	CATEGORY
Create GraphQL Talk	Nils Hartmann	Business
Book Trip to St. Peter-Ording	Susi Mueller	Hobby
Clean the House	Klaus Schneider	Private
Refactor Application	Nils Hartmann	Business
Tax Declaration	Nils Hartmann	Business
Implement GraphQL Java App	Sue Taylor	Business



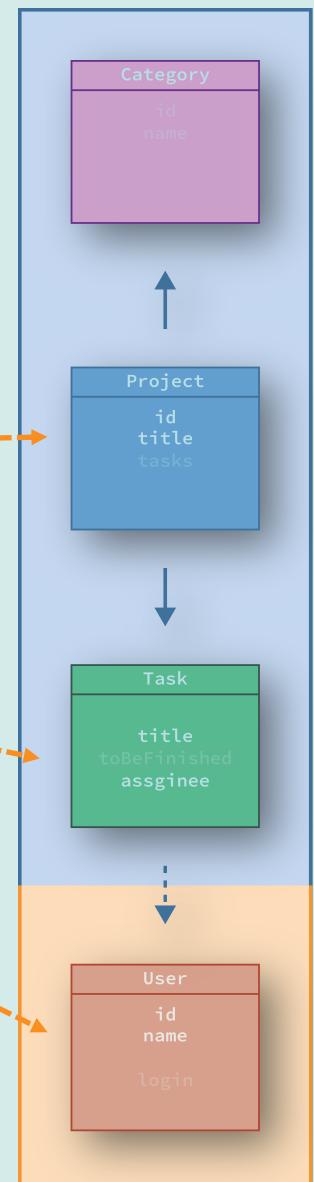
GRAPHQL EINSATZSzenariEN

Use-Case spezifische Abfragen – 2

```
{ project(...) {  
  title  
  tasks {  
    name  
    assignee { name }  
    state  
  }  
}
```

NAME	ASSIGNEE	STATE
Create a draft story	Nils Hartmann	In Progress
Finish Example App	Susi Mueller	In Progress
Design Slides	Nils Hartmann	New

Add Task >



GRAPHQL EINSATZSzenariEN

Use-Case spezifische Abfragen – 2

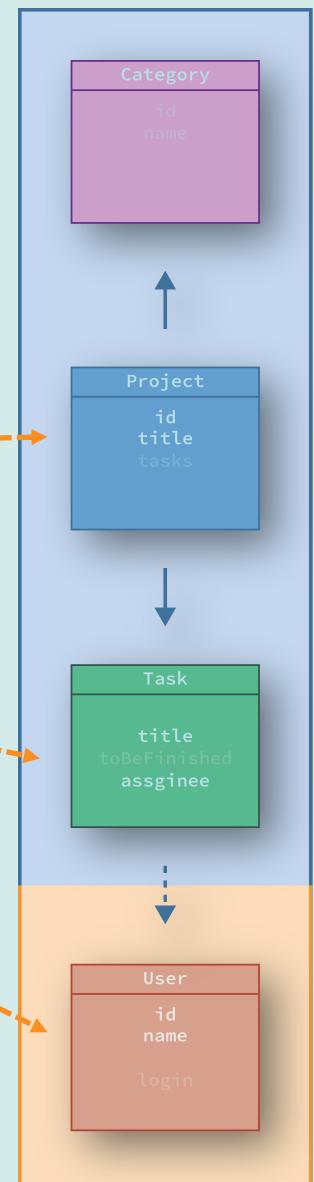
```
{ project(...) {  
  title  
  tasks {  
    name  
    assignee { name }  
    state  
  }  
}
```

Personal Project Planning

All Projects > Create GraphQL Talk Tasks

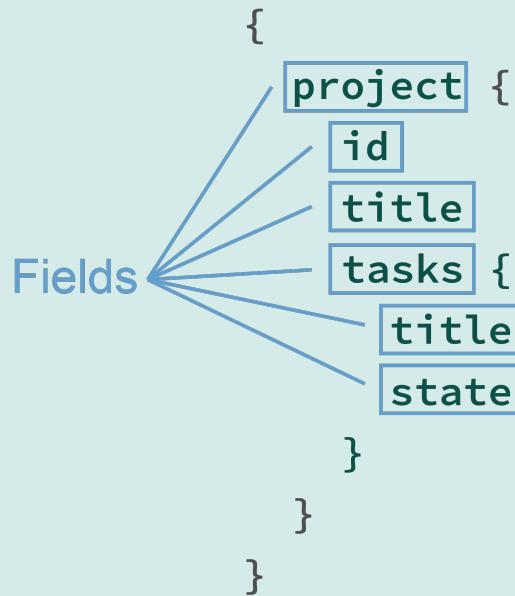
NAME	ASSIGNEE	STATE
Create a draft story	Nils Hartmann	In Progress
Finish Example App	Susi Mueller	In Progress
Design Slides	Nils Hartmann	New

Add Task >



Abgefragt werden Daten, nicht Endpunkte 😈

QUERY LANGUAGE



- Strukturierte Sprache, um Daten von der API abzufragen
- Abgefragt werden **Felder** von (verschachtelten) Objekten

QUERY LANGUAGE

```
{  
  project(projectId: "P1") {  
    id  
    title  
    tasks {  
      title  
      state  
    }  
  }  
}
```

Fields

Argumente

- Strukturierte Sprache, um Daten von der API abzufragen
- Abgefragt werden **Felder** von (verschachtelten) Objekten
- Felder können **Argumente** haben

QUERY LANGUAGE

Ergebnis

```
{  
  project(projectId: "P1") {  
    id  
    title  
    tasks {  
      title  
      state  
    }  
  }  
}
```



```
"data": {  
  "project": {  
    "id": "P1"  
    "title": "GraphQL Talk"  
    "tasks": [  
      {  
        "state": "IN_PROGRESS",  
        "title": "Create Story"  
      },  
      {  
        "state": "NEW",  
        "title": "Finish Example"  
      }  
    ]  
  }  
}
```

- Identische Struktur wie bei der Abfrage

QUERY LANGUAGE: OPERATIONS

Operation: beschreibt, was getan werden soll

- query, mutation, subscription

Operation type

Operation name (optional)

```
query GetProject {  
  project(projectId: "P1") {  
    id  
    title  
    owner { name }  
  }  
}
```

QUERY LANGUAGE: MUTATIONS

Mutations

- Mutation wird zum Verändern von Daten verwendet
- Entspricht POST, PUT, PATCH, DELETE in REST
- Rückgabe Wert kann frei definiert werden (z.B. neue Entität)

```
Operation type
  | Operation name (optional)  Variable Definition
  |
mutation AddTaskMutation($pid: ID!, $input: AddTaskInput!) {
  addTask(projectId: $pid, input: $input) {
    id
    title
    state
  }
}

"input": {
  title: "Create GraphQL Example",
  description: "Simple example application",
  author: "Nils",
  toBeFinishedAt: "2019-07-04T22:00:00.000Z",
  assgineeId: "U3"
}
```

QUERY LANGUAGE: MUTATIONS

Subscription

- Automatische Benachrichtigung bei neuen Daten

```
Operation type
  |
  | Operation name (optional)
  |
subscription NewTaskSubscription {
  newTask: onNewTask {
    Field alias
    | id
    title
    assignee { id name }
    description
  }
}
```

QUERIES AUSFÜHREN

Queries werden über HTTP ausgeführt

- Üblicherweise per POST
- Ein einzelner Endpoint (üblicherweise /graphql)
- **Eine** Version

```
$ curl -X POST -H "Content-Type: application/json" \
      -d '{"query":"{ projects { title } }}' \
      http://localhost:5000/graphql
```

```
{"data": {
  "projects": [
    {"title": "Create GraphQL Talk"},
    {"title": "Book Trip to St. Peter-Ording"},
    {"title": "Clean the House"},
    {"title": "Refactor Application"},
    {"title": "Tax Declaration"},
    {"title": "Implement GraphQL Java App"}
  ]
}
```

QUERIES AUSFÜHREN

Antwort vom Server

- JSON-Objekt
- HTTP Status Codes spielen keine Rolle

```
{  
  "errors": [  
    { "message": "Could not read User with ID 123",  
      "locations": [ . . . ],  
      "path": [ "project", "task", "assignee" ]  
    }  
  ],  
  "data": {"projects": [ . . . ] },  
  "extensions": { . . . }  
}
```

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

- <https://graphql.org>

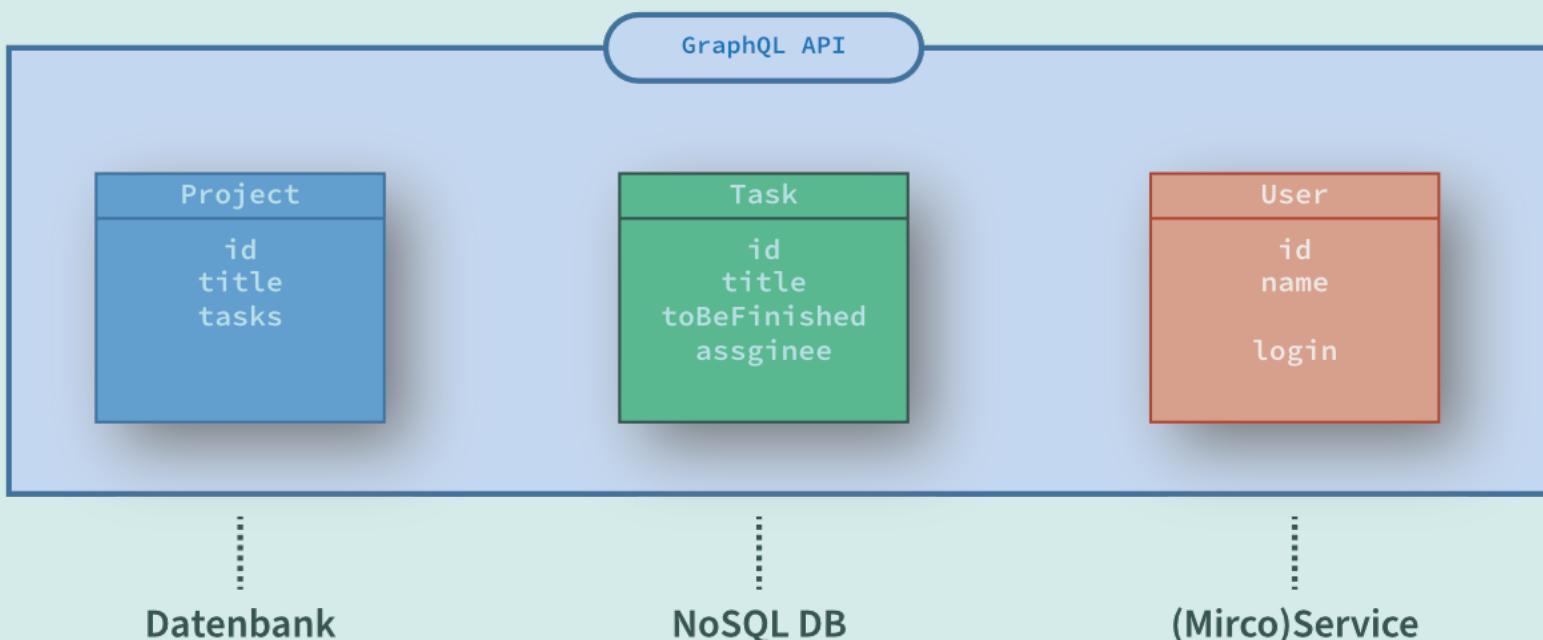
GraphQL (Beispiel: Java)

TEIL 2: RUNTIME-UMGEBUNG (AKA: EURE ANWENDUNG)

GRAPHQL RUNTIME

GraphQL macht keine Aussage, wo die Daten herkommen

- 👉 Ermittlung der Daten ist unsere Aufgabe
- 👉 Müssen nicht aus einer Datenbank kommen



GRAPHQL FÜR JAVA-ANWENDUNGEN

Beispiel: graphql-java

- *Die gezeigten Konzepte sind in GraphQL-Frameworks für andere Programmiersprachen ähnlich!*

DataFetcher

- Ein **DataFetcher** liefert ein Wert für ein angefragtes Feld
- (In anderen Implementierungen auch **Resolver** genannt)

DataFetcher

- Ein **DataFetcher** liefert ein Wert für ein angefragtes Feld
- (In anderen Implementierungen auch **Resolver** genannt)
- DataFetcher ist funktionales Interface (kann als Lambda implementiert werden):

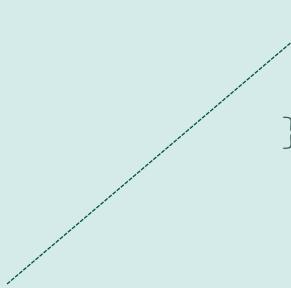
```
interface DataFetcher<T> {  
    T get(DataFetchingEnvironment environment);  
}
```

DATAFETCHER

DataFetcher implementieren

- Beispiel: users-Feld

```
public class UserDataFetchers {  
  
    public DataFetcher<List<User>> usersFetcher() {  
        return environment -> userRepository.findAll();  
    }  
  
}  
  
type Query {  
    users: [User!]!  
}  
}
```



DATAFETCHER

DataFetcher implementieren: Argumente

- environment gibt Informationen über den Query (z.B. Argumente)

```
public class UserDataFetchers {  
  
    type Query {  
        users: [User!]!  
        user(userId: ID!): User  
    }  
  
    public DataFetcher<User> userFetcher() {  
        return environment -> {  
            String userId = environment.getArgument("userId");  
            return userRepository.getUser(userId);  
        };  
    }  
}
```

DATAFETCHER

DataFetcher implementieren: Mutations

- technisch analog zu Query
- dürfen Daten verändern

```
public DataFetcher<Task> addTaskMutationFetcher() {  
    return environment -> {  
        final Map<String, Object> ri =  
            environment.getArgument("taskInput");  
  
        Task t = new Task();  
        t.setTitle((String)taskInput.get("title"));  
        r.setDescription((String)taskInput.get("description"));  
  
        return taskService.addTask(t);  
    };  
}
```

```
type Mutation {  
  addTask  
  (taskInput: AddTaskInput): Task!  
}
```

DATAFETCHER

DataFetcher implementieren: für eigene Typen

- Felder benötigten eigenen DataFetcher, wenn sie nicht am zuvor zurückgegebenen Objekt enthalten sind
- Beispiel: owner-Feld ist nicht am Project POJO definiert, nur dessen Id (User selbst kommt aus "Microservice")

```
query {  
  project(id: 1) {  
    id  
    owner {  
      name  
    }  
  }  
}
```

GraphQL API: Owner Object

Java Klasse: "nur" String

```
public class Project {  
  long id;  
  String title;  
  Category category;  
  String ownerId;  
  ...  
}
```

DATAFETCHER

DataFetcher für eigene Typen

- Für eigene Typen bzw. deren Felder können ebenfalls DataFetcher definiert werden
- Funktionieren wie gesehen, nur dass Parent-Objekt ("Source") übergeben wird

```
query {  
  project(id: 1) {  
    id  
    owner {  
      name  
    }  
  }  
}
```

```
public class ProjectDataFetchers {  
  DataFetcher<User> ownerFetcher = environment -> {  
    Project parent = env.getSource();  
    String ownerId = parent.getOwnerId();  
  
    return userService.getUser(ownerId);  
  };  
}
```

Implementierungen können "herausfordernd" sein

- 1+n-Problem
- Caching
- Performance, ...

Implementierungen können "herausfordernd" sein

- 1+n-Problem
- Caching
- Performance, ...
- **Mögliche Lösungen:**
DataLoader (Aufrufe zusammenfassen und Cachen)

Implementierungen können "herausfordernd" sein

- 1+n-Problem
- Caching
- Performance, ...
- **Mögliche Lösungen:**
 - DataLoader (Aufrufe zusammenfassen und Cachen)
 - Asynchrone DataFetcher (Daten parallel ermitteln)

Implementierungen können "herausfordernd" sein

- 1+n-Problem
- Caching
- Performance, ...
- **Mögliche Lösungen:**
 - DataLoader (Aufrufe zusammenfassen und Cachen)
 - Asynchrone DataFetcher (Daten parallel ermitteln)
 - Persistent Queries (Server kann Queries an Hand von Ids erkennen, kein Standard!)

OUT-OF-SCOPE

GraphQL macht keine Aussage über...

- Security, Paginierung, Sortierung, Filterung
- Muss für die eigene API jeweils konzeptioniert und entwickelt werden

OUT-OF-SCOPE

Ausblick? 😱

 **Nick Schrock**
@schrockn

Folgen ▾

From the talk about the rewrite of fb using Relay and GraphQL. This feature is so amazing and intuitive. Deliver js only if the graphql query returns data that requires that js.

 Tweet übersetzen



AT&T LTE 8:52 AM 84%
Data-Driven Code-Splitting
Relay
... on Post {
 ... on PhotoPost {
 @module('PhotoComponent.js')
 photo_data
 }
 ... on VideoPost {
 @module('VideoComponent.js')
 video_data
 }
 ... on SongPost {
 @module('SongComponent.js')
 song_data
 }
18:06 - 1. Mai 2019

<https://twitter.com/schrockn/status/1123619660732047360>

Zusammenfassung

GRAPHQL - HEILSBRINGER ODER TEUFELSZEUG?

GraphQL != Mainstream

- Implementierungen und Einsatz noch "bleeding edge" (?)
- Wenig erprobte Best-Practices (?)

GraphQL != Mainstream

- Implementierungen und Einsatz noch "bleeding edge" (?)
- Wenig erprobte Best-Practices (?)

[« Alle Veranstaltungen](#)

betterCode () – Workshop: GraphQL für Java-Anwendungen

—
24. März

Fällt mangels Nachfrage aus!

GraphQL - Zusammenfassung

- **GraphQL != SQL**

- kein SQL, keine "vollständige" Query-Sprache
- wir definieren explizit eine API
- keine Datenbank!
- kein Framework!

GraphQL - Zusammenfassung

- **Schema**

- Erzwingt API Design (ob gut oder schlecht, aber man muss es tun)
- Ermöglicht sehr gutes Tooling und Typsicherheit in der Entwicklung
- Exploration von unbekannten APIs möglich

GraphQL - Zusammenfassung

- **Interessante, aber noch relativ junge Technologie**
Bricht mit einigen Gewohnheiten aus REST
Erfordert umdenken
- Für APIs, die von Externen verwendet werden sollen, vielleicht noch nicht richtig

GraphQL - Zusammenfassung

- **Bibliotheken und Frameworks für viele Sprachen**
Prototyp zum Ausprobieren in der Regel schnell gebaut
- **Spannende Experimente und Ideen**
Schema-Merging (Federation), Code-Generierungen, ...
-

GraphQL - Zusammenfassung

- **(noch) kein Mainstream**
- **Meine Empfehlung: ausprobieren, beobachten**

Vielen Dank!

Beispiel-Code: <https://github.com/nilshartmann/graphql-java-talk>

Slides: <https://react.schule/bettercode2021-graphql>

Kontakt & Fragen: nils@nilshartmann.net

Weitere GraphQL Projekte im Java-Umfeld

- **HTTP Endpunkt:** graphql-java-servlet (<https://github.com/graphql-java-kickstart/graphql-java-servlet>)
- **Spring Boot Starter:** <https://github.com/graphql-java-kickstart/graphql-spring-boot>
- **GraphQL Schema mit Java Annotations beschreiben:** <https://github.com/Enigmatis/graphql-java-annotations>

GraphQL MicroProfile

- **Spezifikation:** <https://github.com/eclipse/microprofile-graphql>
- **Quarkus:** <https://quarkus.io/guides/microprofile-graphql>
- **Open Liberty:** <https://openliberty.io/blog/2020/06/05/graphql-open-liberty-20006.html#GQL>

GraphQL Code Generator

- **Generator für zahlreiche Sprachen und Bibliotheken:**
<https://graphql-code-generator.com/>
- **Generator für Queries und Antworten (Java):**
<https://github.com/adobe/graphql-java-generator>
- **Spring Boot Starter:** <https://github.com/graphql-java-kickstart/graphql-spring-boot>

GraphQL APIs für bestehende Datenbanken

- **GraphQL als ORM Ersatz (JavaScript, Go):**

<https://prisma.io/>

- **Instant GraphQL Schema für PostgresDB (Node.JS):**

<https://www.graphile.org/postgraphile/>

- **Instant GraphQL Schema für PostgresDB:**

<https://hasura.io/>

Meine GraphQL-Artikel bei heise Developer

(Bisschen älter schon, aber Grundlagen noch aktuell)

- **Teil 1, Java Backend:**

<https://www.heise.de/developer/artikel/Java-Anwendungen-mit-GraphQL-Teil-1-4205852.html>

- **Teil 2, Frontend mit Apollo, React und TypeScript:**

<https://www.heise.de/developer/artikel/GraphQL-Clients-mit-React-und-Apollo-Teil-2-4273017.html>