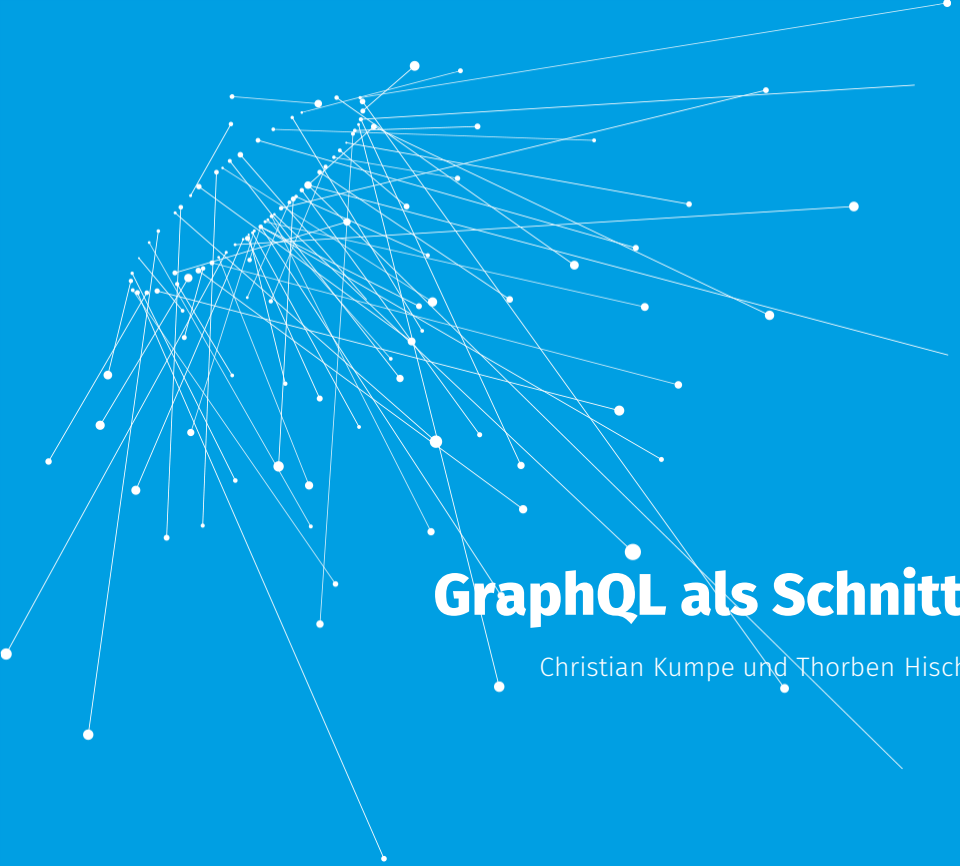


diva e
Digital Value Excellence

GraphQL als Schnittstelle zum Backend

Christian Kumpe und Thorben Hischke, Java Forum Suttgart, 4. Juli 2019

diva-e
Digital Value Excellence

Agenda

1

Vorstellung
- **diva-e**
- **Referenten**

2

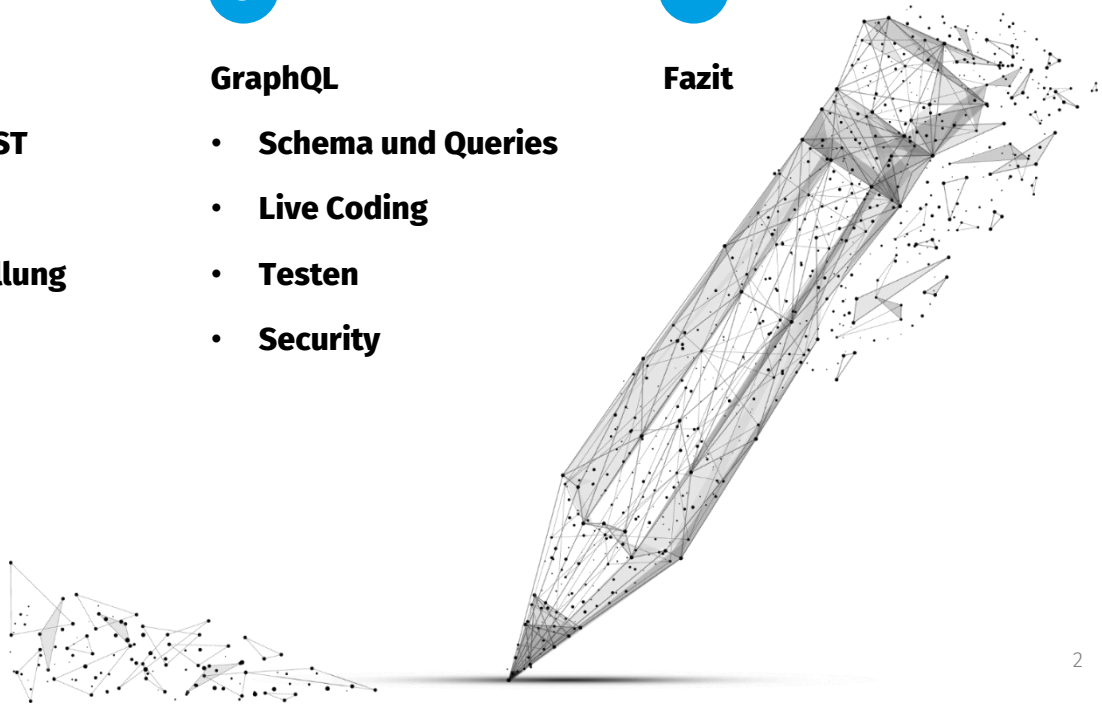
Motivation
- **Überblick REST**
- **GraphQL**
- **Aufgabenstellung**

3

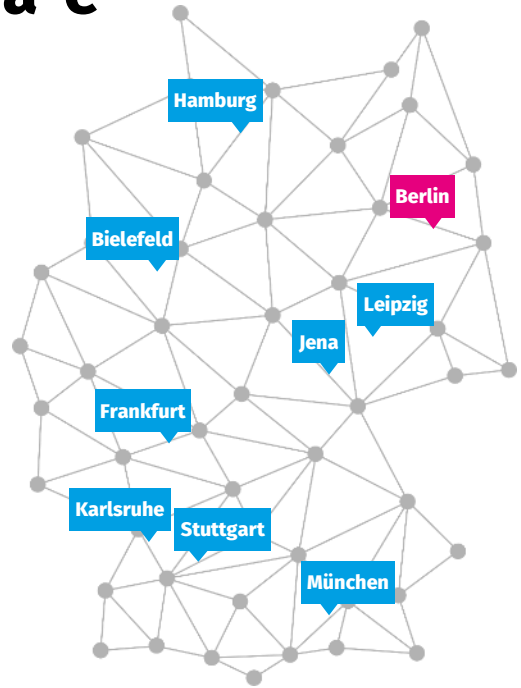
GraphQL
- **Schema und Queries**
- **Live Coding**
- **Testen**
- **Security**

4

Fazit



diva-e



9 Standorte

Wir halten eine umfangreiche Vor-Ort-Präsenz bei unseren Kunden als einen wichtigen Erfolgsfaktor. Deshalb sind unsere Standorte über die gesamte Bundesrepublik verteilt.

500+ Mitarbeiter

Bei uns arbeiten derzeit über 500 Mitarbeiter.

100% Know-How

Von E-Commerce über Content- und Digital-Marketing-Services bis hin zur Retail-Kompetenz: Bei uns erhalten Sie Leistungen aus allen wichtigen E-Business-Disziplinen, vernetzt unter einem Dach.

Referenten

Christian Kumpe

Expert Developer

- → Informatikstudium am KIT (Universität Karlsruhe)
- → Freelancer im Bereich Web und Java
- → Seit Mai 2011 bei diva-e in Karlsruhe
- → Seit 2002 in der Java-Welt unterwegs

Referenten

Thorben Hischke

Senior Architekt

- → Dipl.-Inf. (FH), seit 2007 bei Netpioneer GmbH / diva-e
- → Schwerpunkt CMS (Imperia, FirstSpirit)
- → Integration und Kundenzufriedenheit

Überblick REST

Vorteile

- Dissertation in 2000
- Architekturkonzept
- HTTP-Direktive
  - Verbs
  - Status-Code
  - Content-Type
  - Caching
- 4 Maturity-Levels
- Swagger

Nachteile

- In der realen Welt wird meist Maturity Level 2 und kein *HATEOS* eingesetzt.
- *HATEOS* erzeugt viele Requests
- Applikation braucht meist mehrere Zugriffe, um alle Daten zu erfragen
  - Kein Subset
  - Netzwerk-Overhead
  - Längere Laufzeit

Überblick GraphQL

Vorteile

- Facebook veröffentlicht 2016
- Abfragesprache (GraphQL Schema)
- Spezifikation
- Einen Endpunkt via HTTP
- Applikation kann Subsets erfragen
- Applikation braucht nur einen Zugriff

Nachteile

- „Umdenken“ in der Modellierung
- Query of death

Operationen in GraphQL

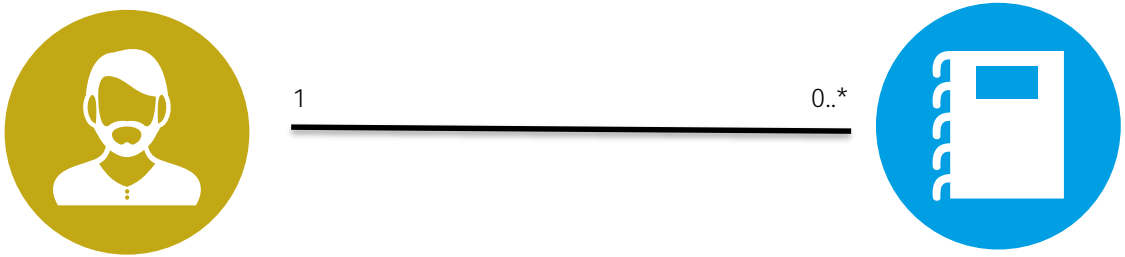
- **Queries**
  Mit Queries können Daten abgefragt werden.

- **Mutations**
  Mit Mutations können Daten verändert werden.

- **Subscribtions**
  Mit Subscription kann der Client auf Events warten.

Unser Beispiel

Bei unserer Versicherung kann der Kunde (Person) Versicherungen (Verträge) abschließen.



Person

- Vor- und Nachname
- Email
- Verträge

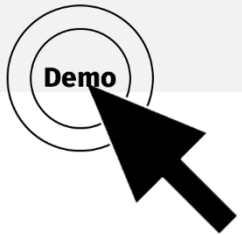
Vertrag

- Name

GraphQL Schema

```
type Person {
    id: ID!
    firstName: String
    lastName: String
}
```

```
type Query {
    person( id: ID!): Person
}
```



Live Coding

Beispiel Code in GitHub verfügbar

https://github.com/diva-e/graphql-spring-boot

GraphQL Backend mit Spring Boot

1. **GraphQL Kickstarter als Abhängigkeit hinzufügen**
   https://www.graphql-java-kickstart.com/spring-boot/
   `com.graphql-java-kickstart:graphql-spring-boot-starter`

2. **Schema erstellen**

3. **Resolver implementieren**

4. **Debugging über GraphiQL**
   http://localhost:8080/graphiql
   https://github.com/graphql/graphiql

Testen

GraphiQL

- Als Abhängigkeit im eigenen Projekt
  `com.graphql-java-kickstart:graphiql-spring-boot-starter`
- Electron based App (https://github.com/skevy/graphiql-app)

Unit Tests

- Entsprechende Abhängigkeit ins Projekt einfügen:
  `com.graphql-java-kickstart:graphql-spring-boot-starter-test`
- Testklasse mit `@GraphQLTest` annotieren
- Schnittstelle über `GraphQLTestTEmplate` ansteuern

Security

1. **Spring Security einbinden**
   Einfügen von `org.springframework.boot:spring-boot-starter-security` in die Projektabhängigkeiten

2. **Method Based Security aktivieren**
   Über die entsprechende Annotation: `@EnableGlobalMethodSecurity(prePostEnabled = true)`

3. **Absichern der GraphQLResolver-Methoden**
   Bspw. mit `@PreAuthorize("hasRole('USER')")`

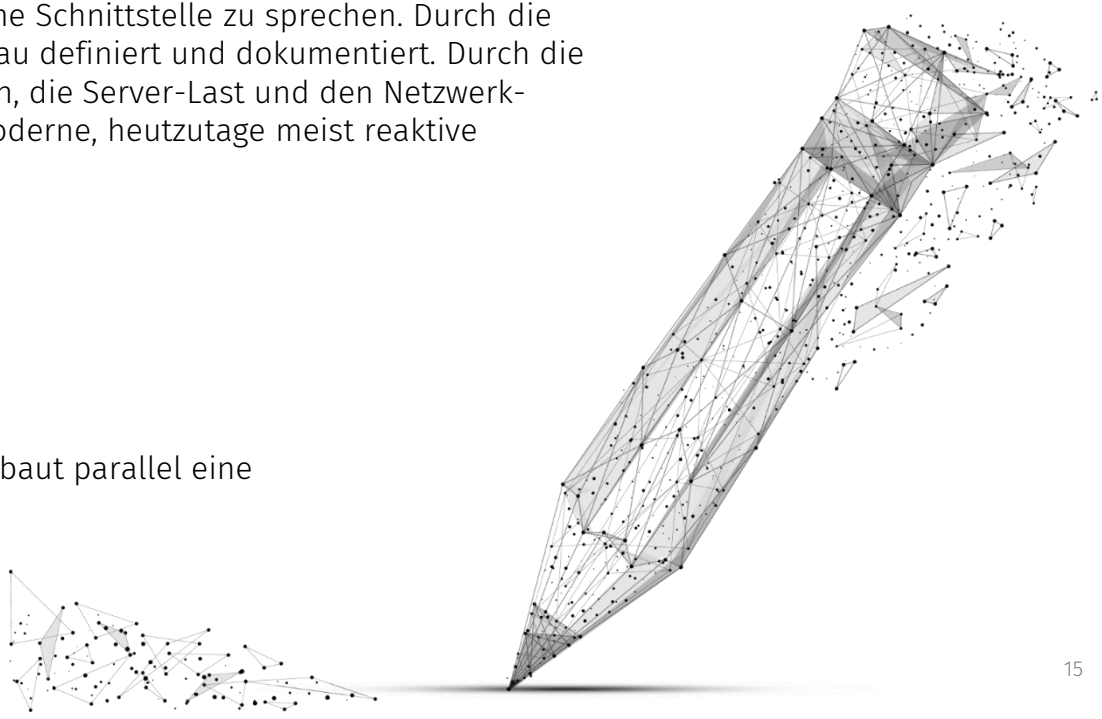
Fazit

GraphQL ist interessante Spezifikation, die es gerade in agilen und unabhängig voneinander agierenden Teams erlaubt, über eine gemeinsame Schnittstelle zu sprechen. Durch die Verwendung eines Schema sind die Zugriffe genau definiert und dokumentiert. Durch die flexible Abfragesprache reduziert es die Anfragen, die Server-Last und den Netzwerk-Traffic. Damit eignet es sich hervorragend für moderne, heutzutage meist reaktive Webanwendungen.

- **Schema**
- **Flexible Abfragesprache**

Ist GraphQL das „neue, bessere“ REST?

Nein. Probiert es aus, experimentiert damit und baut parallel eine GraphQL-Schnittstelle auf.



diva-e. You can’t buy it. You can’t make it. And you sure can’t fake it.

diva-e
Digital Value Excellence

Danke

Bitte geben sie uns Feedback!