Ich liebe Java und ich liebe **GraphQL**

Rolf Borst

GraphQL

Abfragesprache, mit der
Clients Anfragen an den Server
formulieren können

Agenda

Server

GraphQL Schema Definition Language (SDL)

```
type Query {
  buecher(suchbegriff: String!): [Buch]
  buch(id: Int!): Buch
}

type Mutation {
  loeschenBuch(id: Int!): Buch
}

type Buch {
  buchId: Int
  isbn: String
  buchtitel: String
  preis: Float
  autoren: [Autor]
}
```

```
type Autor {
  nachname: String
  vorname: String
}
```

GraphiQL ▶ Prettify Merge History Docs

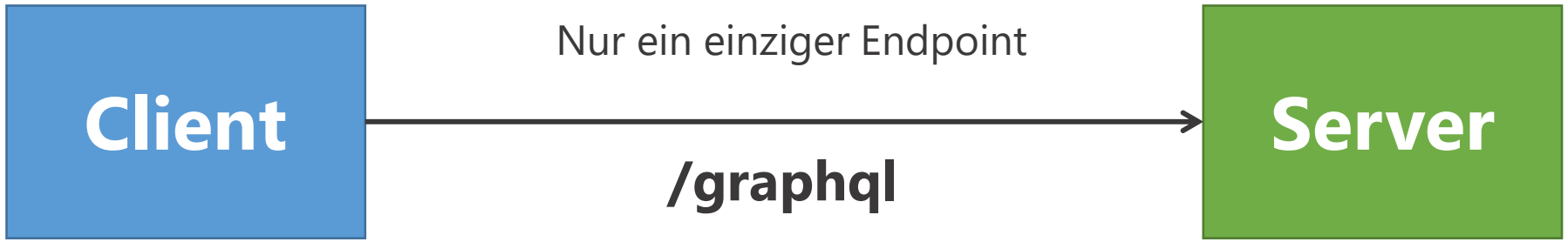
```
query {
  buecher(suchbegriff: "projekt") {
    buchtitel
    preis
  }
}
```

```
{
  "data": {
    "buecher": [
      {
        "buchtitel": "Bärentango",
        "preis": 19.9
      },
      {
        "buchtitel": "Der Termin",
        "preis": 19.9
      }
    ]
  }
}
```

QUERY VARIABLES

Ask for what you want

HTTP-POST




```
POST /graphql HTTP/1.1
Host: localhost:8080
Content-Type: application/json
Content-Length: 115

{
  "query":"query { buecher(
    suchbegriff: \"projekt\") {
    buchtitel preis } }",
  "variables": null
}
```

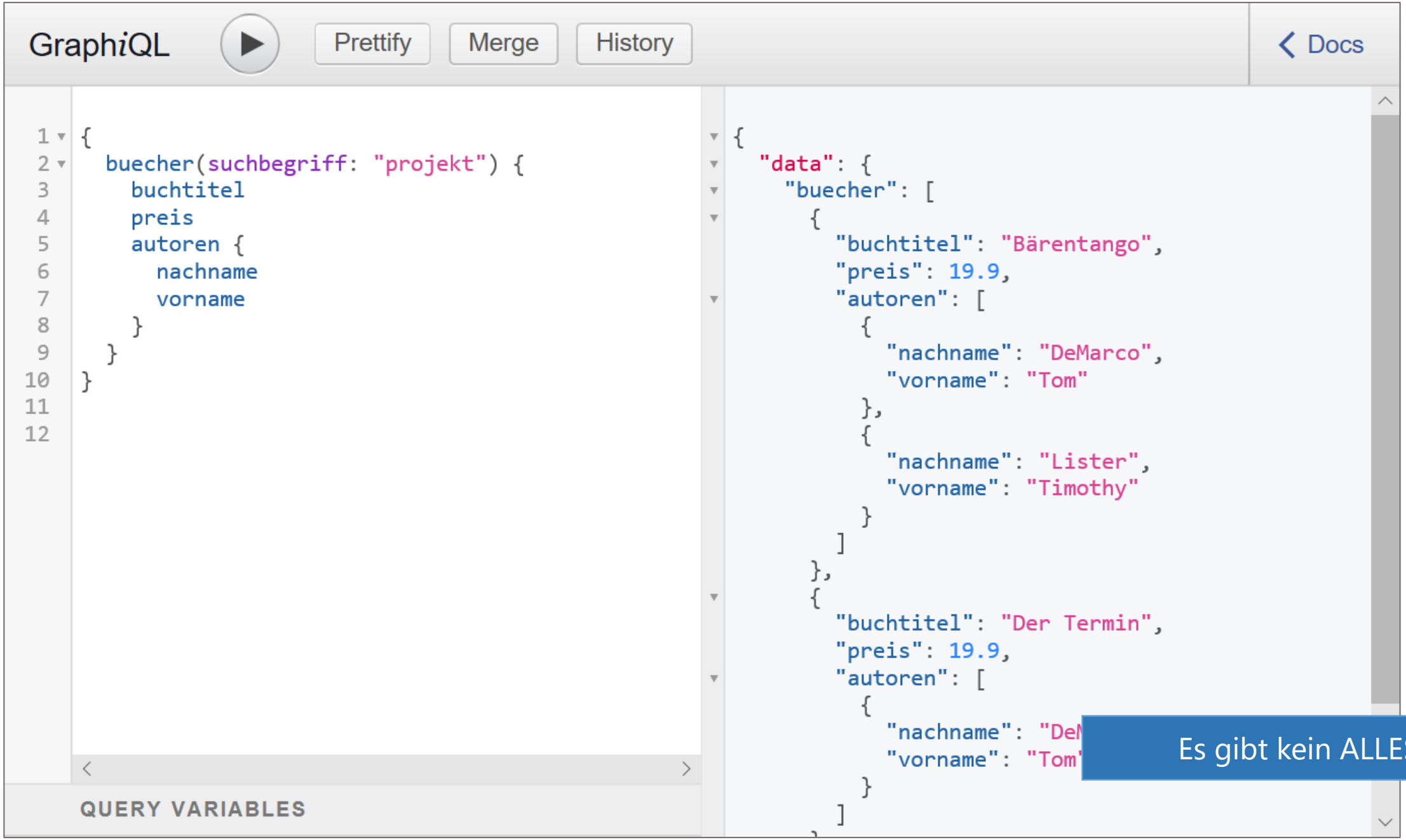
```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 103
Date: Wed, 22 May 2019 16:07:13 GMT

{
  "data": {
    "buecher":[
      {"buchtitel":"Bärentango", "preis":19.9},
      {"buchtitel":"Der Termin", "preis":19.9}]
  }
}
```

Server

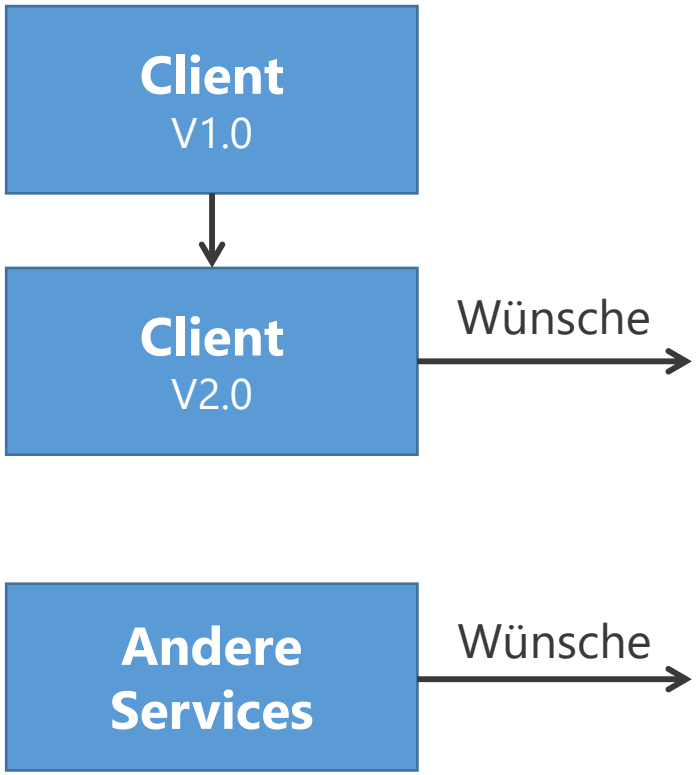
GraphQL Schema Definition Language (SDL)

```
type Query {
  buecher(suchbegriff: String!): [Buch]
  buch(id: Int!): Buch
}

type Mutation {
  loeschenBuch(id: Int!): Buch
}

type Buch {
  buchId: Int
  isbn: String
  buchtitel: String
  preis: Float
  autoren: [Autor]
}
```

```
type Autor {
  nachname: String
  vorname: String
}
```



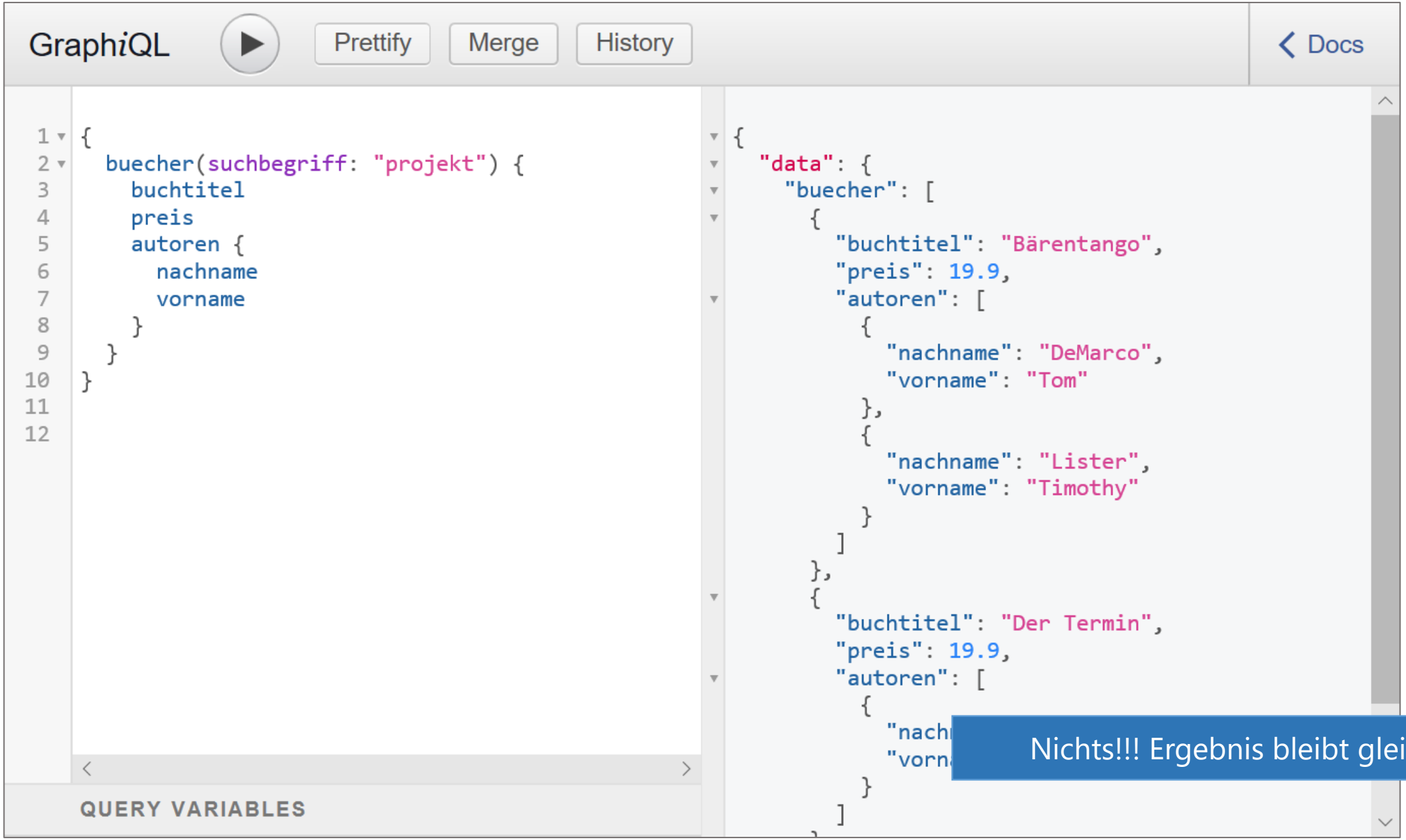
Es gibt kein ALLES




```
type Query {
  buecher(suchbegriff:
     String!): [Buch]
  buch(id: Int!): Buch
}

type Mutation {
  loeschenBuch(id: Int!): Buch
}

type Buch {
  buchId: Int
  isbn: String
  buchtitel: String
  preis: Float
  autoren: [Autor]
  cover: Base64Image
  bewertungen: [Bewertung]
}
```

```
type Autor {
  nachname: String
  vorname: String
  biographie: String
}

type Bewertung {
  benutzer: String
  kommentar: String
  sterne: Int
}

scalar Base64Image
```

Was ändert sich für den ursprünglichen Client?


GraphiQL
Prettify
Merge
History
Docs
{
buecher(suchbegriff: "projekt") {
buchtitel
preis
autoren {
nachname
vorname
}
}
}
QUERY VARIABLES
{
"data": {
"buecher": [
{
"buchtitel": "Bärentango",
"preis": 19.9,
"autoren": [
{
"nachname": "DeMarco",
"vorname": "Tom"
},
{
"nachname": "Lister",
"vorname": "Timothy"
}
]
},
{
"buchtitel": "Der Termin",
"preis": 19.9,
"autoren": [
{
}
]
Nichts!!! Ergebnis bleibt gleich

```
type Query {
  buecher(suchbegriff:
     String!): [Buch]
  buch(id: Int!): Buch
}

type Mutation {
  loeschenBuch(id: Int!): Buch
}

type Buch {
  buchId: Int
  isbn: String
  buchtitel: String
  preis: Float
  autoren: [Autor]
  cover: Base64Image
  bewertungen: [Bewertung]
}
```

```
type Autor {
  nachname: String
  vorname: String
  biographie: String
}

type Bewertung {
  benutzer: String
  kommentar: String
  sterne: Int
}

scalar Base64Image
```

Welche Entscheidungsmöglichkeiten hat hier der Client?

```
type Query {
  buecher(suchbegriff:
     String!): [Buch]
  buch(id: Int!): Buch
}

...

type Buch {
  buchId: Int
  isbn: String
  buchtitel: String
  preis: Float
  autoren: [Autor]
  cover: Seite
  bewertungen: [Bewertung]
}

...
```

```
type Seite {
  max: Base64Image
  skaliert(breite: Int, hoehe: Int): Base64Image
  url: String
  urlSkaliert (breite: Int, hoehe: Int): String
  beschreibung: String
}

...
```

Neue Typen schaffen neue Möglichkeiten

```
type Query {
  buecher(suchbegriff:
     String!): [Buch]
  buch(id: Int!): Buch
}

...

type Buch {
  buchId: Int
  isbn: String
  buchtitel: String
  preis: Float
  autoren: [Autor]
  cover: Seite
  rueckseite: Seite
  seite(nummer: Int): Seite
  seiten(von: Int, bis: Int): [Seite]
  bewertungen: [Bewertung]
}
```

```
type Seite {
  max: Base64Image
  skaliert(breite: Int, hoehe: Int): Base64Image
  url: String
  urlSkaliert (breite: Int, hoehe: Int): String
  beschreibung: String
}

...
```

Noch mehr Felder

GraphiQL ▶ Prettify Merge History Docs

```
{
  buecher(suchbegriff: "projekt") {
    buchtitel
    preis
    cover {
      skaliert(breite: 300)
    }
    rueckseite {
      skaliert(breite: 100)
      url
    }
    seiten(von: 1, bis: 5) {
      url
    }
  }
}
```

QUERY VARIABLES


```
{
  "data": {
    "buecher": [
      {
        "buchtitel": "Bärentango",
        "preis": 19.9,
        "cover": {
          "skaliert": "BASE64"
        },
        "rueckseite": {
          "skaliert": "BASE64",
          "url": "/bilder/22/9999"
        },
        "seiten": [
          {
            "url": "/bilder/22/1"
          },
          {
            "url": "/bilder/22/2"
          },
          {
            "url": "/bilder/22/3"
          },
          {
            "url": "/bilder/22/4"
          },
```


Unendliche Möglichkeiten

graphql-java

https://github.com/graphql-java
https://github.com/graphql-java-kickstart

```
<dependency>
  <groupId>com.graphql-java</groupId>
  <artifactId>graphql-java</artifactId>
  <version>${graphql-java.version}</version>
</dependency>
<dependency>
  <groupId>com.graphql-java-kickstart</groupId>
  <artifactId>graphql-java-servlet</artifactId>
  <version>>${graphql-java-servlet.version}</version>
</dependency>
```

```
@WebServlet("/graphql")
public class GraphQLServlet extends GraphQLHttpServlet {

   @Override
   protected GraphQLConfiguration getConfiguration() {

      File schemaFile = new File(getClass()
              .getResource("/graphql/schema.graphqls").getFile());
      TypeDefinitionRegistry typeDefinitionRegistry = new SchemaParser()
              .parse(schemaFile);

      RuntimeWiring runtimeWiring = ...  // nächste PowerPoint-Seite

      GraphQLSchema schema = new SchemaGenerator()
              .makeExecutableSchema(typeDefinitionRegistry, runtimeWiring);

      return GraphQLConfiguration.with(schema).build();

   }

}
```

```
@Inject
GraphQLServices services;   // nächste PowerPoint-Seite
...

RuntimeWiring wiring = RuntimeWiring.newRuntimeWiring()

  .type("Query", typeWiring -> typeWiring
    .dataFetcher("buecher", services::resolveBuecher)
    .dataFetcher("buch", services::resolveBuch))

  .type("Mutation", typeWiring -> typeWiring
    .dataFetcher("einfuegenBuch", services::einfuegenBuch))

  .type("Buch", typeWiring -> typeWiring
    .dataFetcher("cover", services::resolveBuchCover)
    .dataFetcher("rueckseite", services::resolveBuchRueckseite)

  .type("Seite", typeWiring -> typeWiring
    .dataFetcher("max", services::resolveSeiteMax)
    .dataFetcher("skaliert", services::resolveSeiteSkaliert)
    .dataFetcher("url", services::resolveSeiteUrl)

  .build();
```

```
public class GraphQLServices {

  @Inject
  private BuchService buchService;

  public List<Buch> resolveBuecher(DataFetchingEnvironment env) throws Exception {
    String suchbegriff = env.getArgument("suchbegriff");
    return buchService.suchenBuecher(suchbegriff);
  }
  ...

  public Seite resolveBuchCover(DataFetchingEnvironment env) throws Exception {
    Buch buch = env.getSource();
    return new Seite(buch.getBuchId(), COVER_SEITE);
  }
  ...

  public byte[] resolveSeiteSkaliert(DataFetchingEnvironment env) throws Exception {
    Seite seite = env.getSource();
    Integer breite = env.getArgument("breite");
    Integer hoehe = env.getArgument("hoehe");
    ...
  }
```

Zusammenfassung

Der Server definiert ein Angebot

Der Client fragt explizit Daten ab

Daraus ergeben sich
fantastische Möglichkeiten und
es gibt viel zu entdecken

GraphQL is super!

Schaut es Euch unbedingt an.