Basler
Versicherungen

bluesky
CONSIDER IT DONE

KARAKUN

CSS
Versicherung

magnolia®

JAVA
USER
GROUP
CH

adesso
business.
people.
technology.

KS: kallysoft

sympany
versicherungen

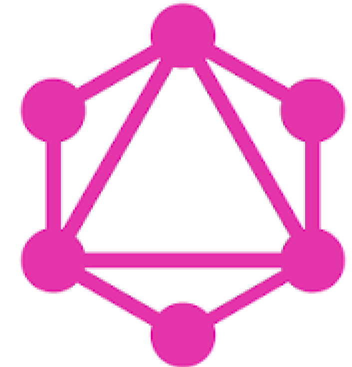
50 ELCA

trivadis

Basel
basel.ch

Beyond REST: GraphQL with Java

- GraphQL Introduction
- GraphQL Demo
- GraphQL Schema Definition Language
- GraphQL Query Language
- GraphQL Java Implementation



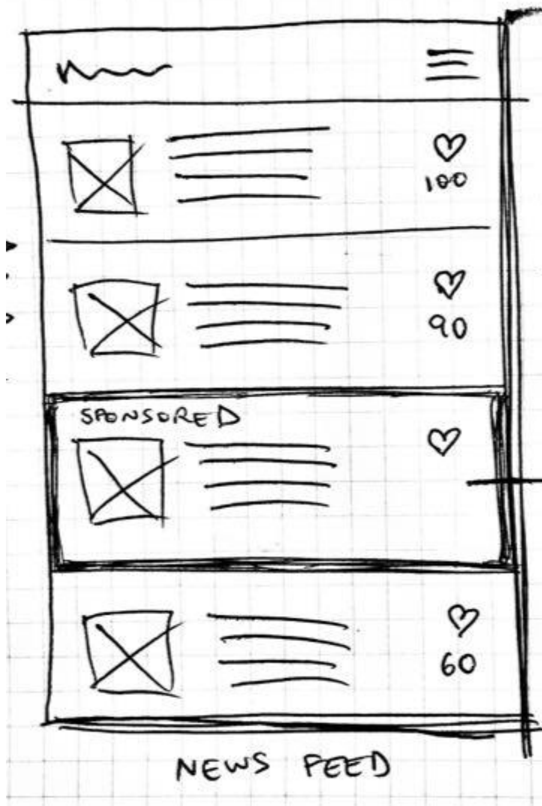
Prof. Dr. Dominik Gruntz
Institute for Mobile and Distributed Systems
University of Applied Sciences Northwestern Switzerland

Fachhochschule Nordwestschweiz

REST

- **Problems**
  - No Standard
    - Fielding's PhD describes the concept
  - Design is difficult
    - PUT vs PATCH
  - Fits well for CRUD applications, but operations do not map well
    - Usually POST is used to model operations
  - HATEOAS is rarely used (typically Maturity Level 2)
  - Documentation has to be added
    - Swagger
  - Over-Fetching
    - Too many data is returned, data which is not needed by the client
  - Under-Fetching
    - Too few data is returned, leading to additional requests (=> n+1 problem)

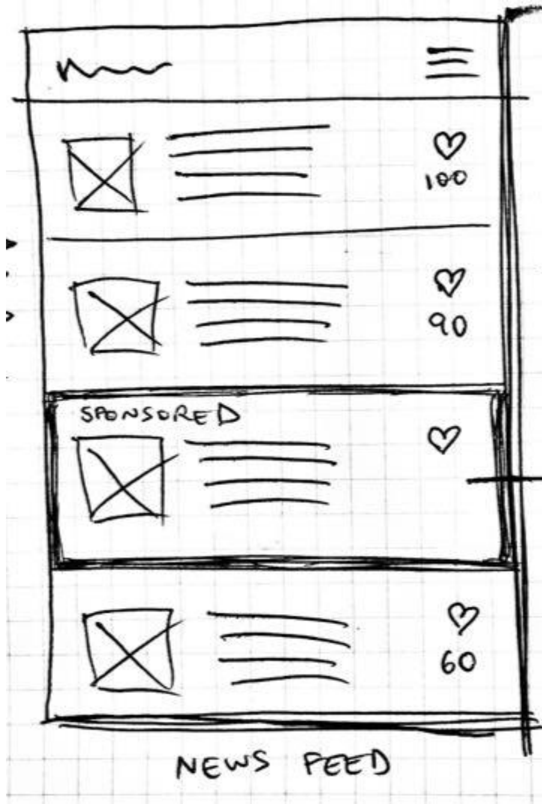
Example Newsfeed App



- **REST endpoints**

```
GET    /posts?limit=10   Query 10 latests posts
POST   /posts            Create a new post
GET    /posts/{id}       Query a particular post
DELETE /posts/{id}       Delete a post
PUT    /posts/{id}       Update a post

GET    /users?q=…        Search for users
POST   /users            Create a new user
GET    /users/{id}       Get a particular user
DELETE /users/{id}       Delete a user
PUT    /users/{id}       Update a user
```

Example Newsfeed App

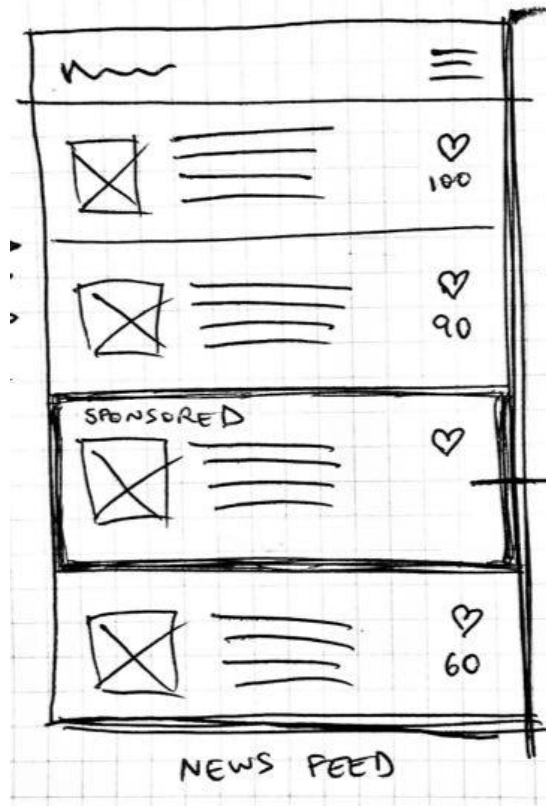


- **Query for the newest posts**

```
GET /posts?limit=10 HTTP/1.1
Accept: application/json

{
   "posts" : [
      {
         "id" : 1,
         "title" : "BaselOne",
         "author" : 10,
         "viewCount" : 100,
         "content" : "Great talks!",
         "reaction" : ["Heart"]
      },
      … 9 additional posts
   ]
}
```


(C) Hochschule für Technik
Fachhochschule Nordwestschweiz

Example Newsfeed App



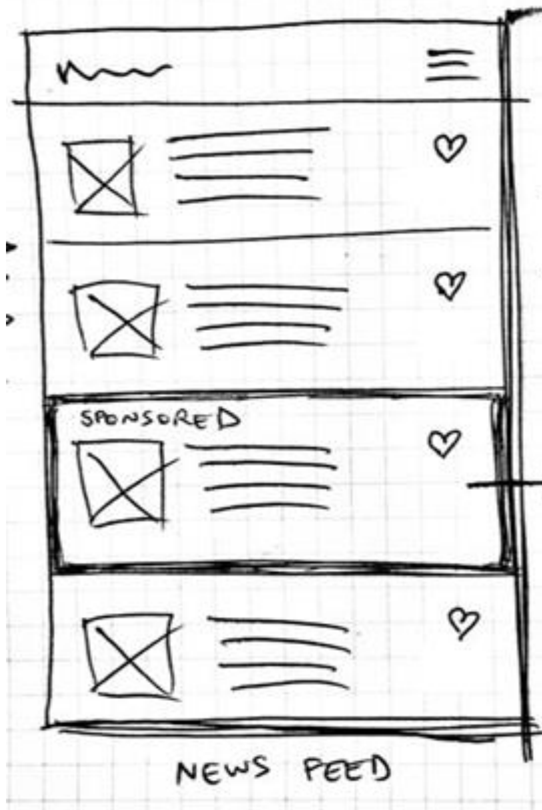
- **Query for the author**

```
GET /users/10 HTTP/1.1
Accept: application/json

{
   "user" : {
      "id" : 10,
      "name" : "Dominik Berger",
      "company" : "bluesky",
      "avatar" : "/avatars/10.png"
   }
}
```

  – 1 request for the latest posts
  – 10 requests for the user's names and avatars

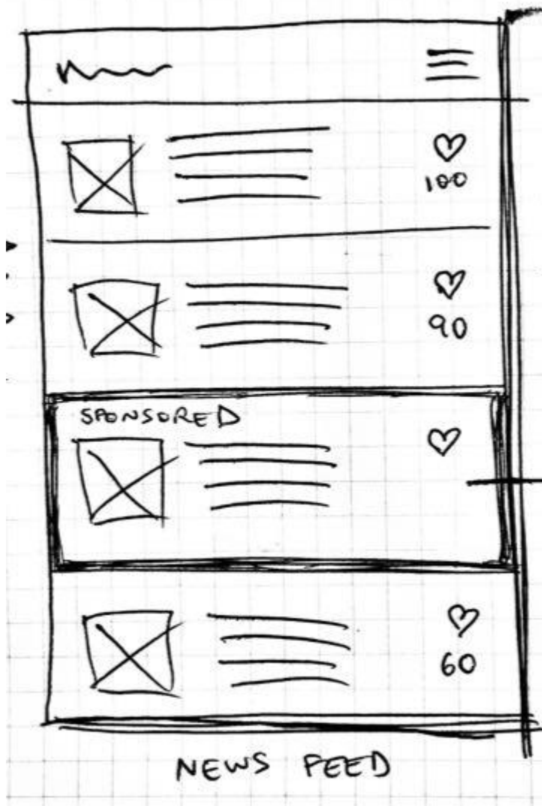
(C) Hochschule für Technik
Fachhochschule Nordwestschweiz

Example Newsfeed App



- **Under-Fetching => multiple round trips**
  - Solution: special endpoint `/newsfeed`
    - Backend and Frontend become tightly coupled
- **Over-Fetching => too many data**
  - Let us assume that the PO decided to remove the view count
    - New version of the API (v2)
    - Simply continue to deliver all date (for old clients)
    - Specify with a query string the data to be delivered

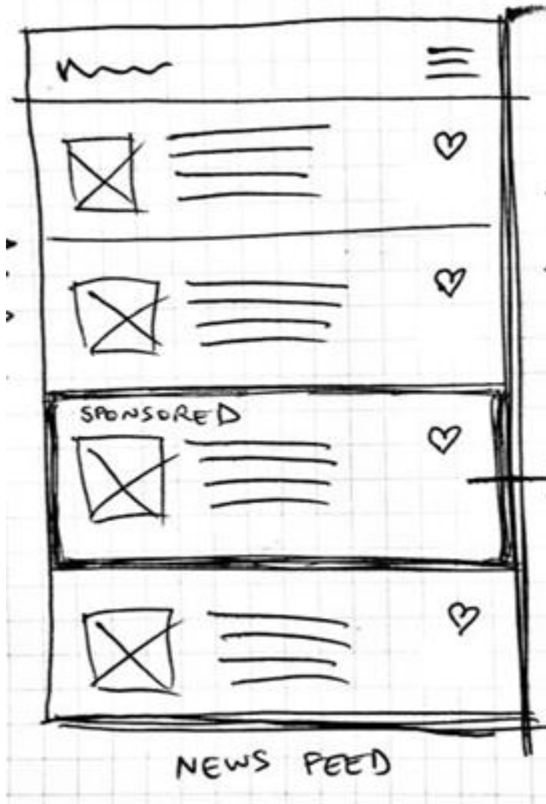
Example Newsfeed App



- **GraphQL query**

```
POST /graphql HTTP/1.1

query {
  posts(limit: 10, offset: 0) {
    id
    title
    content
    author {
        name,
        avatar
    }
    viewCount
    reactions(limit: 10) { icon }
  }
}
```

Example Newsfeed App



- **GraphQL query**

```
POST /graphql HTTP/1.1

query {
  posts(limit: 10, offset: 0) {
    id
    title
    content
    author {
        name,
        avatar
    }

    reactions(limit: 10) { icon }
  }
}
```

GraphQL

- **GraphQL Query**

Operation

Operation name (with argu

Shape of the result

```
POS
query {
  posts(limit: 10, offset:
    id
    title
    content
    author {
        name,
        avatar
    }

    reactions(limit: 10) { icon }
  }
}
```

- **GraphQL Response**

```
{ "data" : {
  "posts" :  [
   {
     "id" : 1,
     "title": "BaselOne",
     "content": "Great Talks!",
     "author" : {
       "name": "Dominik Berger",
       "avatar": "/avatars/10.png"
     },
     reactions: null
   },
   …  9 other posts
   ]
}}
```

GraphQL

- **History**
  - GraphQL is a data query language created by Facebook
    - 2012 internal product
    - 2015 released as open-source product
    - 2018 project moved to the GraphQL foundation
- **Core Ideas**
  - A query is shaped just like the data it returns
  - Data are graphs of objects which are navigated using queries
- **Specification**
  - Schema Definition Language (SDL)
  - Query Language
  - http://facebook.github.io/graphql/draft/ [Working Draft, Oct 2019]
  - https://github.com/graphql/graphql-spec [Spec Repo]

Fachhochschule Nordwestschweiz

GraphQL Example: GitHub

GraphQL API Explorer | GitHub D

developer.github.com/v4/explorer/

GitHub GraphQL API

Signed in as **dgruntz**. You're ready to explore! Sign out

Heads up! GitHub's GraphQL Explorer makes use of your **real, live, production data.**

GraphiQL Prettify History

```
query {
  viewer {
    login
    avatarUrl
    repositories(first:10) {
      nodes {
        name
      }
    }
  }
}
```

QUERY VARIABLES

```
{}
```

```
{
  "data": {
    "viewer": {
      "login": "dgruntz",
      "avatarUrl": "https://avatars0.githubusercontent.com/u/1516800?v=4",
      "repositories": {
        "nodes": [
          {
            "name": "scala"
          },
          {↔},
          {
            "name": "scala.github.com"
          },
          {
            "name": "snakey"
          },
          {
            "name": "docs.scala.ide"
          },
          {
            "name": "scala-dist"
          },
          {
            "name": "scalapuzzlers.github.com"
          },
          {
            "name": "macrocosm"
```

Documentation Explorer

Search Schema...

A GraphQL schema provides a root type for each kind of operation.

ROOT TYPES

query: Query

mutation: Mutation

GraphQL Example: GitHub

- **GraphQL Query**

```
query {
  viewer {
    login
  }
}
```

- **GraphQL Response**

```
{
  "data": {
    "viewer": {
      "login": "dgruntz"
    }
  }
}
```


(C) Hochschule für Technik
Fachhochschule Nordwestschweiz

GraphQL Example: GitHub

- **HTTP Request**

```
POST /graphql/ HTTP/1.1
Host: graphql-explorer.githubapp.com
Content-Length: 73
Content-Type: application/json

{"query":"query {viewer {\n login\n }\n}\n ","variables":{}}
```

- **HTTP Response**

```
HTTP/1.1 200 OK
Server: Cowboy
Date: Thu, 17 Oct 2019 07:30:43 GMT
Content-Type: application/json; charset=utf-8
Cache-Control: max-age=0, private, must-revalidate
Transfer-Encoding: chunked

{"data":{"viewer":{"login":"dgruntz"}}}
```


(C) Hochschule für Technik
Fachhochschule Nordwestschweiz

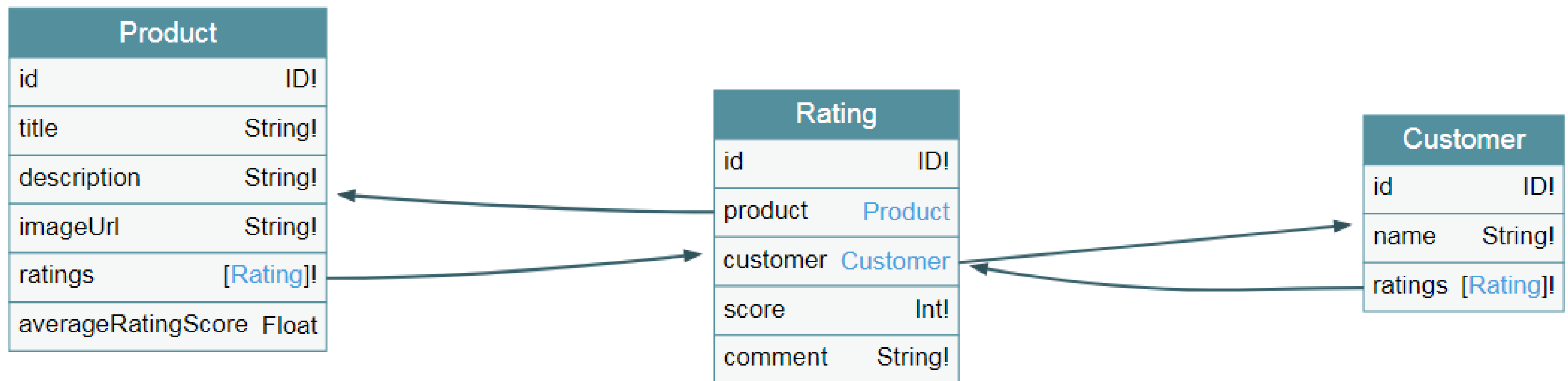
Outline

- **GraphQL Introduction**
- **GraphQL Schema Language**
- **GraphQL Query Language**
- **GraphQL Java Implementation**

Fachhochschule Nordwestschweiz

GraphQL Schema Language

- **Sample Model**



  – Generated from the GraphQL schema using Voyager

GraphQL Schema Language

- **Sample Schema**

```
type Product {
   id: ID!
   title: String!
   description: String!
   imageUrl: String
   ratings: [Rating]!
   averageRatingScore: Float
}

type Rating {
   id: ID!
   product: Product
   customer: Customer
   score: Int!
   comment: String!
}
```

```
type Customer {
   id: ID!
   name: String!
   ratings: [Rating]!
}

type Query {
   products: [Product!]!
   product(id: ID!): Product
}

schema {
   query: Query
}
```


(C) Hochschule für Technik
Fachhochschule Nordwestschweiz

GraphQL Schema Language

- **Scalar-Types**
  - `Int` (32 bit signed)
  - `Float` (double IEEE754)
  - `String` (UTF-8 character sequence)
  - `Boolean` (false/true)
  - `ID` (unique identifier, serialized as String)
- **List-Type**
  - Arrays can be applied to scalars or to object types
- **Non-Null-Type**
  - ! – marked fields are mandatory (non-null), fields are by default nullable
- **Enum-Type**
- **Interface-Type**
- **Union-Type**

GraphQL Schema Language

- **Object-Types**

```
ObjectTypeDefinition ::=
   type Name [implements NamedType { & NamedType }]
        { Name [ ( Name : Type [ = Value ] ) ]: Type }
```

  - Object fields are conceptually functions which yield values
  - Object fields can accept arguments to further specify the return value; the arguments are defined as a list of name-type pairs
  - Arguments can have default values
  - Example

```
type Query {
   products(limit: Int = 10): [Product]!
   product(id: ID!): Product
}
```


(C) Hochschule für Technik
Fachhochschule Nordwestschweiz

GraphQL Schema Language

- **Schema**

```
SchemaDefinition ::=
    schema { OperationType : NamedType }
```

  – Defines the starting points for navigating through the data
  – Three types of operations (=> Root types)
    - Query
    - Mutation
    - Subscription
  – Root types are syntactically just regular types with fields
  – Each GraphQL implementation *must* at least define a Query type

```
schema {
   query: Query
   mutation: Mutation
}
```

Outline

- GraphQL Introduction
- GraphQL Schema Language
- GraphQL Query Language
- GraphQL Java Implementation

GraphQL Query Language

- **Query Language**
  - Clients use the query language to interact with the API
    - => Query which describes the structure of the result
    - <= JSON Document
  - Query consists of an operation and one or more patterns which are matched against the big graph containing all the data
  - Operations
    - **`query`** a read-only fetch of data
    - **`mutation`** a write followed by a fetch
    - **`subscription`** a long-lived request used for notifications
  - Pattern
    - Expressed in terms of the relationships between objects and the fields the objects contain

(C) Hochschule für Technik
Fachhochschule Nordwestschweiz

GraphQL Query Language

- **Query**

```
query {
  products {
    title
  }
}
```

```
{
  "data": {
    "products": [
      {
        "title": "PHILIPPI Paco"
      },
      {
        "title": "Driade Nemo"
      },
      {
        "title": "MonkeyUmbrella"
      }
    ]
  }
}
```

  - Query describes the structure of the response, comparable to a template in a template language
  - Listing the fields in a query corresponds to a SELECT in SQL
  - "query" key-word could be omitted, this is the default
  - When objects are referenced, at least one field must be specified

GraphQL Query Language

- **Field References**

```
query {
  products {
    title
    ratings {
      score
    }
  }
}
```

```
{
  "data": {
    "products": [
      {
        "title": "PHILIPPI Paco",
        "ratings": [
          {
            "score": 4
          },
          {
            "score": 3
          },
        ]
      }, …
    ]
} }
```

  – It is also possible to retrieve multiple fields and to follow references
  – The resulting tree gets deeper by nested field access

GraphQL Query Language

- **Arguments**

```
query {
  product(id: "3") {
    title
    ratings {
      comment
    }
  }
}
```

```
{
  "data": {
    "product": {
      "title": "Monkey Umbrella",
      "ratings": [
        { "comment":
          "I like this product"
        },
        { "comment":
          "what the hack is this"
        },
        { "comment":
          "awsome!!!"
        }
      ]
} } }
```

  - Some fields my accept arguments which restrict the set of returned objects
  - Arguments are named
  - An argument in a GraphQL is comparable to a WHERE clause in SQL

GraphQL Query Language

- **Mutations**

```
mutation {
  createProduct(
    title: "Tea Egg",
    description:
        "Samba rattle for tea",
    imageUrl: "a052191.jpg") {
      id
  }
}
```

```
{
  "data": {
    "createProduct": {
      "id": "6"
    }
  }
}
```

  - Mutations are used to modify data, i.e. adding or mutating data on the server
  - Arguments are named and can be specified in any order
  - Similar to a query, but the difference is that several mutations must be executed sequentially in the order given

GraphQL Query Language

- **Subscriptions**

```
subscription {
  productAdded {
    id
    title
  }
}
```

  - Subscriptions are used to get data from the server (typically triggered by modifications on the server)
  - The request specifies the data to be sent from the server to the client when the event is triggered
  - GraphQL does not prescribe any technology binding. Typically subscriptions are implemented over WebSockets using the graphql-ws subprotocol

Data
↑{"type":"connection_init","payload":{}}
↓{"type":"connection_ack"}
↑{"id":"1","type":"start","payload":{"query":"subscription {\n bookAdded {\n id\n title\n }\n}","variables":null}}
↓{"type":"data","id":"1","payload":{"data":{"bookAdded":{"id":"5","title":"The New Book"}}}}

GraphQL Query Language

- **GraphQL Introspection**

```
query {
  __type (name: "Product") {
    name
    fields {
      name
    }
  }
}
```

```
{
  "data": {
    "__type": {
      "name": "Product",
      "fields": [
        { "name": "id" },
        { "name": "title" },
        { "name": "description"},
        { "name": "imageUrl" },
        { "name": "ratings" },
        { "name":
           "averageRatingScore" }
      ]
    }
  }
}
```

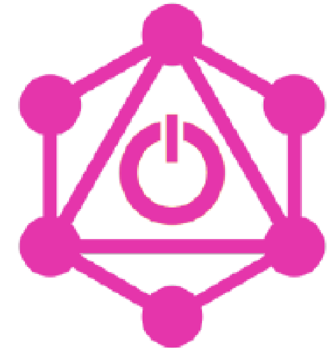
  - Introspection feature is provided by the GraphQL implementation
  - Keywords for introspection start with _ _, i.e. _ _schema or _ _type

Outline

- GraphQL Introduction
- GraphQL Schema Language
- GraphQL Query Language
- GraphQL Java Implementation

GraphQL Java Implementation

- **GraphQL-Java & Kickstart SpringBoot Starters**
  - https://www.graphql-java-kickstart.com/
  - https://github.com/graphql-java-kickstart
  - https://github.com/graphql-java



- **Dependencies**

```
implementation
         'com.graphql-java-kickstart:graphql-spring-boot-starter:5.10.0'
runtime 'com.graphql-java-kickstart:graphiql-spring-boot-starter:5.10.0'
runtime 'com.graphql-java-kickstart:voyager-spring-boot-starter:5.10.0'
runtime 'com.graphql-java-kickstart:altair-spring-boot-starter:5.10.0'

implementation 'com.graphql-java-kickstart:graphql-java-tools:5.6.1'
```


(C) Hochschule für Technik
Fachhochschule Nordwestschweiz

GraphQL Java Implementation

- **graphql-java**
  – Low-level api, wiring has to be done by hand

- **Spring Boot GraphQL Starter & GraphQL Java Tools**
  – Automatic Schema detection (`*.graphqls` on classpath)
  – GraphQL service published on `/graphql` endpoint
  – Object fields are automatically fetched
  – Additional field resolver may be provided

(C) Hochschule für Technik
Fachhochschule Nordwestschweiz

GraphQL Java Implementation

- **Entities**

```
@Data
@AllArgsConstructor
public class Customer {
   private String id;
   private String name;
}

@Data
@AllArgsConstructor
public class Product {
   private String id;
   private String title;
   private String description;
   private String imageUrl;
}
```

```
@Data
@AllArgsConstructor
public class Rating {
  private String id;
  private Product product;
  private Customer customer;
  private int score;
  private String comment;
}
```

GraphQL Java Implementation

- **Repository**

```
@Component
public class ShopRepository {
   public Product createProduct(String title, String description,
                                String url) { … }
   public Collection<Product> getAllProducts() { … }
   public Optional<Product> getProductById(String id) { … }

   public Customer createCustomer(String name) { … }
   public Optional<Customer> getCustomerById(String id) { … }

   public Rating rateProduct(String productId, String customerId,
                             int score, String comment) {… }
   public List<Rating> getRatingsForCustomer(Customer c) { … }
   public List<Rating> getRatingsForProduct(Product p) { … }
}
```

GraphQL Java Implementation

- **Query-Resolver**

```
@Component
public class Query implements GraphQLQueryResolver {

   @Autowired
   private ShopRepository shopRepository;

   public Collection<Product> products() {
      return shopRepository.getAllProducts();
   }

   public Optional<Product> product(String id) {
      return shopRepository.getProductById(id);
   }
}
```

GraphQL Java Implementation

- **Mutation-Resolver**

```
@Component
public class Mutation implements GraphQLMutationResolver {

   @Autowired
   private ShopRepository shopRepository;

   public Product createProduct(String t, String d, String u) {
      return shopRepository.createProduct(t, d, u);
   }

   public Rating rateProduct(String pId, String cId,
                                        int score, String comment) {
      return shopRepository.rateProduct(pId, cId, score, comment);
   }
}
```

GraphQL Java Implementation

- **Customer-Resolver**

```
@Component
public class CustomerResolver implements GraphQLResolver<Customer> {

   @Autowired
   private ShopRepository shopRepository;

   public List<Rating> ratings(Customer c) {
      return shopRepository.getRatingsForCustomer(c);
   }
}
```

Fachhochschule Nordwestschweiz

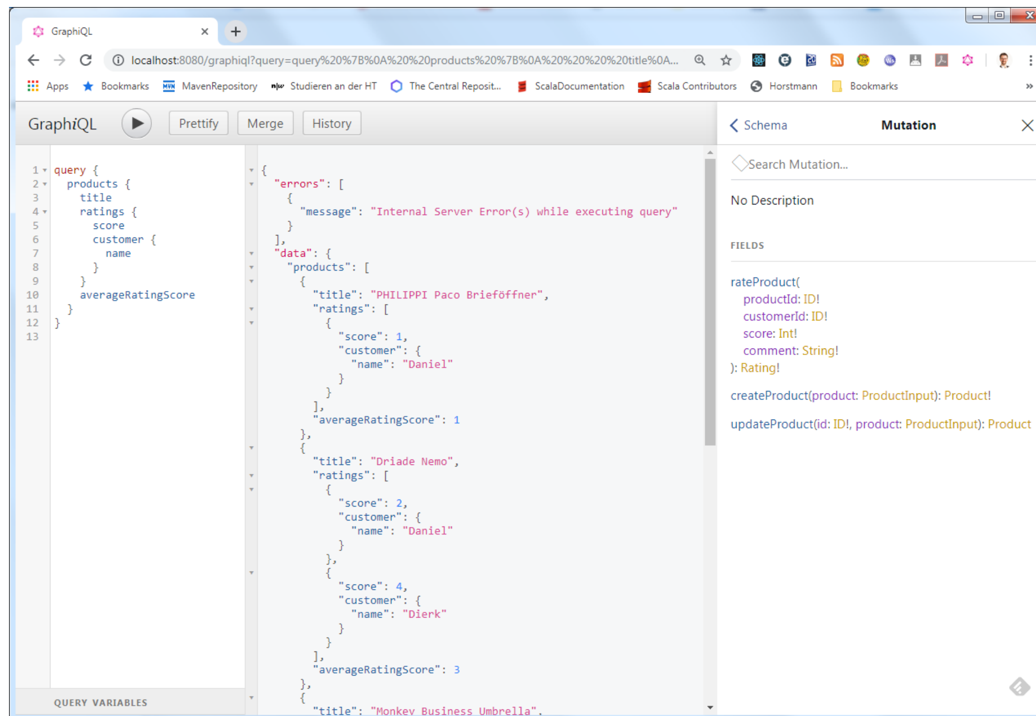
GraphQL Java Implementation

- **Product-Resolver**

```
@Component
public class ProductResolver implements GraphQLResolver<Product> {

   @Autowired
   private ShopRepository shopRepository;

   public List<Rating> ratings(Product p) {
      return shopRepository.getRatingsForProduct(p);
   }

   public double averageRatingScore(Product p) {
      return shopRepository.getRatingsForProduct(p).stream()
         .mapToInt(r -> r.getScore()).average().getAsDouble();
   }
}
```

Tools: GraphiQL

- **GraphiQL: A graphical interactive in-browser GraphQL IDE**
  - Repository: https://github.com/graphql/graphiql
  - Live Demo: http://graphql.org/swapi-graphql/

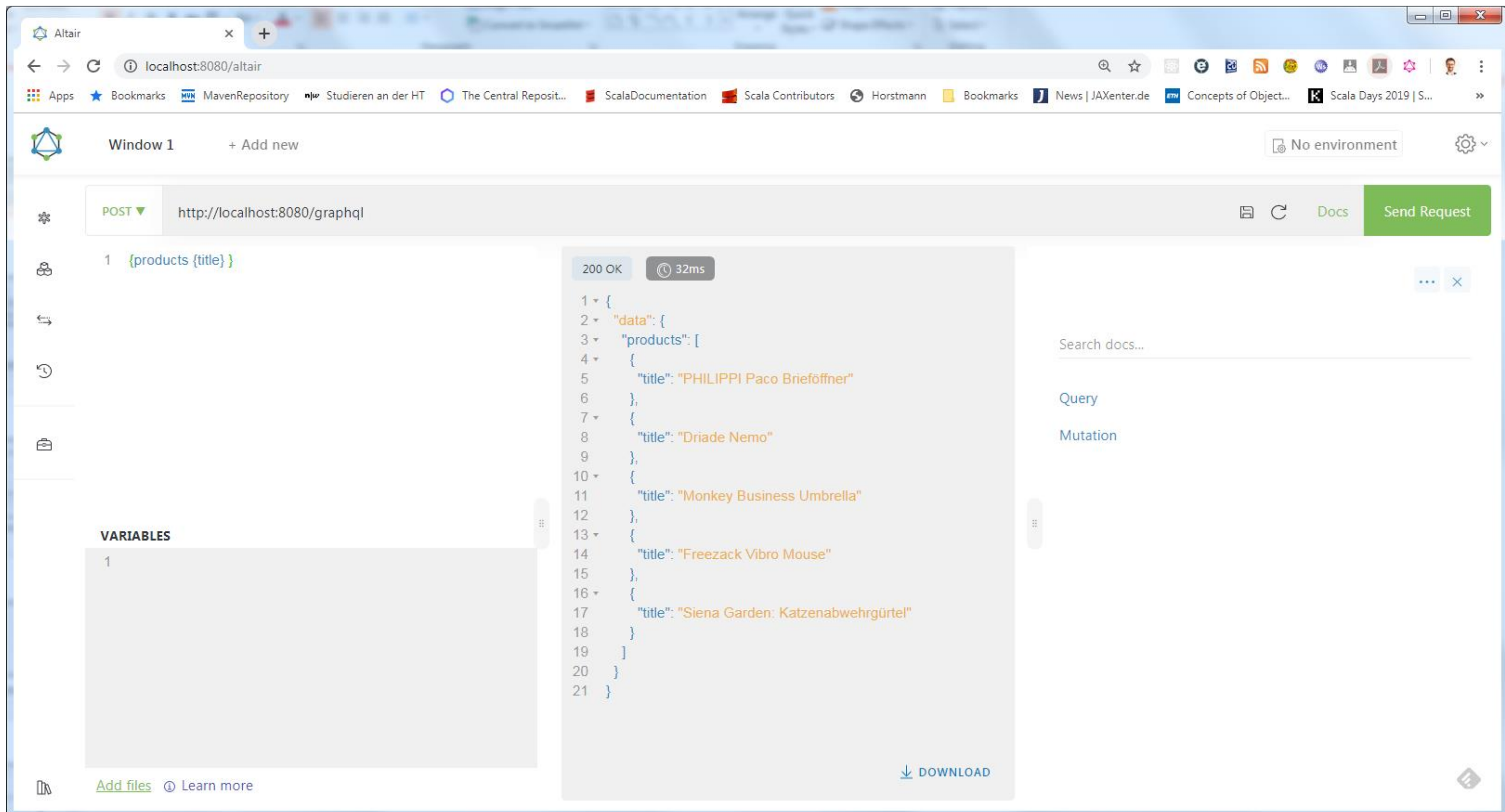


Tools: Voyager

- **Voyager: Represent any GraphQL API as an interactive graph**

Tools: Altair

- **Altair: A beautiful feature-rich GraphQL Client for all platforms**



Outline

- GraphQL Introduction
- GraphQL Schema Language
- GraphQL Query Language
- GraphQL Java Implementation
- GrphqQL Summary

GraphQL vs REST

- **Advantages**
  - Solves over-fetching and under-fetching problem => Performance
    - Query contains shape of the result
    - Queries are use-case specific
  - Evolvability
    - Server can provide additional data, clients do not need to consume them
  - Monitoring
    - Server can monitor which attributes are being used!
    - Can be used to identiffy legacy types/fields
- **Disadvantage**
  - GraphQL has no automatic caching system (POST Requests)
  - Error handling (typically all responses are 200 OK)

GraphQL Resources

- **Specification**
  - https://github.com/graphql/graphql-spec
  - https://graphql.github.io/graphql-spec/
- **Collection of links to resources including tutorials & libraries**
  - https://github.com/chentsulin/awesome-graphql *Resources*
  - https://github.com/APIs-guru/graphql-apis *Public APIs*
  - http://graphql.org/users/ *GraphQL Users*
  - https://graphql.github.io/learn/
- **Graphql-java**
  - https://www.graphql-java.com/documentation/v12/
  - https://www.graphql-java-kickstart.com/
  - https://github.com/dgruntz/baselone-graphql