Information System
Lab 2

T. Back
s3218147

R. Bwana
s3232352

15th January, 2018

1 Introduction

For this assignment we implemented and compared 2 different API technologies, namely a traditional REST API system and GraphQL as outlined in the assignment specification document.

We modeled our data on a simple article - author relationship.

The authors, named 'dudes' in our schema would have an *id, name* and a link to the articles they have authored.

The article, named 'article' in our schema would have an *id, title, dude_id*[1] and a counter for *votes*.

We also filling in some dummy data at start-up to ease testing.

To focus on the important aspects of this assignment, we implemented the REST and GraphQL Interface with subqueries, which can access and also change data. The data is stored persistent in a sqlite database. The project itself is written in NodeJS using the ExpressJS web framework[2].

[1] *dude_it* is a foreign key to the 'dudes' schema

[2] Please consult the source code and also the included README.md file for more information

2 REST

In order to implement the RESTful API we took advantage of the service that `Swagger.io` provides. This allowed us to quickly get an API definition up an running in a short amount of time.

We then downloaded the file and ran the server locally for ease of testing and use. The used library takes care of the routing within the application, so that only the implementation on of the functionality need to be done in the server.

We hereby show an example of a request and response from the REST API.

The request sent was:

```
curl -X GET --header 'Accept: text/html'
'http://localhost:7700/article/1'
```

Returning a response of:

```
{
  "id": 1,
  "title": "Amageddon",
  "votes": 13,
  "dude_id": 1
}
```

3 GraphQL

For GraphQL, we recreated the schema based on our previous implementation of Swagger. The library again takes care of the routing. We only had to write the so called resolvers that are able to gather the data. This also handles automatically nested queries.

We once again ran the server locally for ease of use and testing.

An example of a request and response from the GraphQL API is as follows.

```
curl -XPOST -H 'Content-Type:application/json' -d
'{"query": "query {dudes {id name articles {id title votes}}}"}'
localhost:7700/graphql
```

returns

```
{
  "data": {
    "dudes": [
      {
        "id": "1",
        "name": "DudeA",
        "articles": [
          {
            "id": "1",
            "title":"Amageddon",
            "votes": 13
          },
          {
            "id": "2",
            "title":"Beta Stuff",
            "votes":0
          },
          {
            "id": "4",
            "title":"Amageddon",
            "votes":0
          },
          {
            "id": "5",
            "title":"Beta Stuff",
            "votes":0
          }
        ]
      },
      {
        "id": "2",
        "name": "DudeB",
        "articles": [
          {
            "id": "3",
            "title":"Casear",
            "votes":0
          },
          {
            "id": "6",
            "title":"Casear",
            "votes":0
          }
        ]
      }
    ]
  }
}
```

Second example using the mutation functionality of GraphQL:

```
curl -XPOST -H 'Content-Type:application/json' -d
'{"query": "mutation { upvoteArticle(articleId: 1) { id title votes } }"}'
localhost:7700/graphql
```

Returning a response of:

```
{
  "data": {
    "upvoteArticle": {
      "id": "1",
      "title": "Amageddon",
      "votes": 14
    }
  }
}
```

4 SWOT analysis

For this section, we compare the advantages and disadvantages of each API. While not an exact 'SWOT' anlysis, we do consider the Opportunity of one to be the weakness of the other and vice versa.

Strength of REST	**Strength of GraphQL**
• Common and established to a greater extent than GraphQL. • No schema required	• Requester specifies (only) the wanted data. • The links between the data are specified and used. • Can handle nested or complex queries much easier and simpler. • Hides the API call details making querying simpler.
Weakness of REST	**Weakness of GraphQL**
• It's simplicity means more queries than necessary. • REST is only specified by a developer writing documentation, no actual schema.	• Somewhat different to REST meaning slow adoption. • More complex to initially set up. • Functionality is hidden behind the GraphQL abstraction, making it less transparent.

5 Extra context feature

After developing and testing both API technologies we also decided to test how easy it would be to establish authentication requirements on both of them.

This was done using a simple header field. The field *Authorization* needs to be set to *secret* in our case for all the REST API call to paths including */dude*. This demonstrates a simple way of implementing a weak security rule. This is not perfect as not all routes are protected and also a standard password is used via an unencrypted connection, but it can further be extended.

6 Learning Experience

What we learned from this experience was that REST API is easy to get up and running compared to GraphQL, but once both services are established, GraphQL is more feature rich and less resource intensive when it comes to calls made over the network.

Also, we can imagine a scenario whereby multiple data sources are integrated within the same service. In such a scenario using an API, or a series of APIs together would allow for a much easier to maintain experience than any other option.

Additionally, we enjoyed being able to specify the data we want in the GraphQL requests and only receive that. The validation of the request against a schema also makes it more fail-safe compared to changing REST APIs.

Interestingly enough, the connection to a persistent data store like our SQLite database did not pose a problem. As calls from the REST API are straight forward to handle, figuring out how to write the correct so-called resolvers for GraphQL pay out. Cross-linking between the data schema is easily done.

7 Which would we choose?

Unfortunately, this depends on the situation. We concluded that if something needs to be implemented quickly then a REST API would be best suited. Should we be developing a system that is meant to handle the complexity of modern information systems however, we would rather use GraphQL.

We are happy to be familiar with GraphQL now and the invested time will pay off at some point in the future inside a professional project.