Hochschule Konstanz
Department of Computer Science

Eingereicht von
David Wolpers
Matrikel-Nummer 295416

david.wolpers@htwg-konstanz.de

Bachelor These

Entwicklung von Entscheidungskriterien für die Verwendung von GraphQL gegenüber REST in modernen Webanwendungen

Konstanz, 30ter März 2020

Bachelor These

Entwicklung von Entscheidungskriterien für die Verwendung von GraphQL gegenüber REST in modernen Webanwendungen

von

David Wolpers

zur Erlangung des Grades

Bachelor of Science
in Angewandte Informatik

an der Hochschule Konstanz Technik, Wissenschaft und Gestaltung,

Matrikel-Nummer:	295416
Abgabedatum:	30ter März 2020
1. Betreuer:	**Prof. Dr. Marko Boger**
2. Betreuer:	**Michael Wagner**

Eine elektronische Version dieser Thesis ist Verfügbar auf https://github.com/Seitenbau/GraphQL-.

Abstract

Abstract. . .

Inhaltsverzeichnis

1

Einleitung

Parallel zur Verbreitung des Internets, ist die versendete Datenmenge stark angestiegen. Immer mehr Nutzer wollen auf immer größere werdende Datenmengen möglichst schnell zugreifen. Gleichzeitig ist der Anteil der Nutzer, welche über mobile Endgeräte auf die Daten zugreifen, immer weiter angestiegen. [**Enge**; **2019**] Diese Entwicklungen resultieren in neuen Anforderungen für die Datenübertragung. Zum Einen brauchen vor allem mobile Endnutzer eine schnelle und datensparende Verbindung. Zum Anderen führt die erhöhte Vielfalt der Geräte auch dazu, dass die benötigten Daten sich von Nutzer zu Nutzer stark unterscheiden können.
Diese Anforderungen müssen von der Schnitstelle zwischen Server und Client erfüllt werden. Mit der aktuell wohl am weitesten verbreiteste Technologie REST wird dies schnell sehr aufwändig. Den hier treten häufig Probleme wie Over- und Underfetching auf. Auch Facebook hatte damit zu kämpfen und entschied sich daraufhin eine neue Technologie zu entwickeln - GraphQL. Diese sollten einen flexibilen, datensparenden Zugriff auf die entsprechenden APIs ermöglichen.

1.1. Ziel der Arbeit

Wann der Einsatz von GraphQL sinnvoll ist lässt sich aber nicht pauschal sagen. Die Sprache ermöglicht zwar neue Herangehensweisen an eine API, doch könnten andere bekannte Funktionalitäten darunter leiden. Ziel der nachfolgenden Arbeit ist es, für Unternehmen und Entwickler eine Entscheidungshilfe zu formulieren, wann sich der Einsatz von GraphQL gegenüber REST lohnen könnte. Konkret sollen dabei folgende

Fragen beantwortet werden:

- Welche funktionellen Aspekte sind für den Aufbau der meisten APIs erforderlich?
- Wie können diese Aspekte in GraphQL und REST umgesetzt werden?
- In welchen Fällen ist der Einsatz welcher Technologie zu empfehlen? (??)

1.2. Vorgehensweise

Um diese Fragen zu beantworten wird erst aus aktueller Literatur entnommen, welche Aspekte entscheidendend für den Aufbau einer guten API sind. Anschließend wird über Codebeispiele, welche sich auf eine bereits existiernde Segel-API beziehen, untersucht(??), ob und wie diese in GraphQL und REST implementiert werden können. Diese Implementierungen werden daraufhin gegenübergestellt und analysiert. Zuletzt werden praxisnahe Use-Cases formuliert und mit den Ergebnissen der vorherigen Analyse gewichtet(??).

2

Technischer Hintergrund

Um die Arbeit verstehen zu können, müssen erst eine wichtige Technologien, welche die Arbeit behandelt, beschrieben werden. Zuerst wird erklärt was eine API eigentlich ist. Anschließend wird mit REST und GraphQL die verschiedenen Herangehensweise an eine API behandelt. Zuletzt wird noch gezeigt inwiefern das Grundprinzip von GraphQL sich von REST unterscheidet, aber auch welche Gemeinsamkeiten sie haben.

2.1. Application Programming Interface (API)

Unter einer Anwendungsprogrammierschnittstelle (API) versteht man die Kommunikationsstelle zwischen zwei Softwarekomponenten. Die API wird dabei von einer der Komponenten festgelegt und gibt an wie die andere Komponente mit ihr komunizieren kann. Konkret werden von der API alle Daten und Funktionalitäten zur Verfügung gestellt, welche von außen zugänglich sein sollen. API ist damit aber ein sehr weitgefächerter(??) Begriff. Für diese Arbeit werden APIs behandelt, welche eine Server-Client Architektur ermöglichen. Dabei kommuniziert der Client, häufig eine App oder Webseite, mit der API eines externen Servers. GraphQL und REST sind beides Verfahren, die entwickelt worden, um diese Web-APIs aufzubauen. (?? QUELLE: API Design Buch, sehr gut und auf deutsch, wohl auch weiter unten verwenden)

2.2. Representational State Transfer (REST)

REST ist für sich gesehen eigentlich keine Technologie, sonden nur ein Architekturstil zum Aufbau der API verteilter Systeme. Nach Roy Fielding handelt es sich dabei, um eine Hybride Mischung verschiedener Netzwerkbasierten Architekturstile, welche durch weitere Einschränkungen eine einheitliche Anschlussschnittstelle beschreiben. Damit sich eine API RESTful nennen darf, müssen die folgenden Prinzipien erfüllt werden [**Fielding**; **2000**]:

- **Client-Server:** Trennung der User-Oberfläche von der Datenspeicherung, wordurch die Portabilität und die Skalirbarkeit ansteigt
- **Zustandslos:** Jede Anfrage des Clienten zum Server muss alle benötigten Daten enthalten, um die Anfrage zu verstehen
- **Cacheable:** Die Daten des Servers müssen mit Cachable oder Nicht-Cachable beschriftet werden. Cachable Daten dürfen dann vom Client Cache verwendet werden
- **Schichtenaufbau:** Aufbau in Schichten, so dass das die einzelnen Schichten nur ihre direkten Interaktionspartner sehen
- **Code auf Abfrage (Optional):** Client Funktionalität soll durch downloadbare Skripte und Applets erweitert werden
- **Einheitliche Schnittstelle:** Durch den Einsatz des Allgemeingültigkeitprinzips wird die Architektur vereinfacht und die die Sichtbarkeit der Interaktionen verbessert(??)

Die einheitliche Schnitstelle ist dabei nach Fielding das zentrale Feature, welches REST von anderen Architekturstilentilen unterscheidet. Um dieses Ziel zu erreichen muss die Schnitstelle weitere Einschränkungen erfüllen: Identifikation von Resourcen, Manipulation von Resourcen über Repräsentationen, Selbsterklärende Nachrichten und das Hypermedia as the engine of application state Prinzip (HATEOAS).
(?? Restfulapi.net, quelle?) Hierbei werden die Begriffe Resourcen und Repräsentationen verwendet. Um diese Einschränkungen vollständig zu verstehen müssen diese Begriffe erläutert werden. Eine Resource ist jede Form einer benenbaren Information. Dabei kann es sich sowohl um klassische Dateien handelt, als auch um Services oder reale nicht virtuelle Personen. Jeder dieser Resourcen muss über einer Uniform Resource Identifier(URI) identifizierbar sein. Der Client greift dann über diese URI auf die Resource zu und erhält als Antwort eine Repräsentation der Resource. Diese Repräsentation

kann verschiedene Formate annehmen, wie z.B JSON, HTML oder XML. Jedoch sollten diese Repräsentationen zur Erfüllung des HATEOAS-Prinzip wie Hypertext (??) aussehen. [**Fielding**; **2008**] Unter Hypertext versteht man, dass der Text bzw. die Repräsentationen auf weitere Resourcen verlinken sollen.(??) [**Fielding**; **2008**] Diese Definitionen geben die Einschränkungen eigentlich schon vor [**Fielding**; **2000**]:

- **Identifikation von Resourcen:** Trennung der User-Oberfläche von der Datenspeicherung, wordurch die Portabilität und die Skalirbarkeit ansteigt
- **Manipulation von Resourcen über Repräsentationen:** Jede Anfrage des Clienten zum Server muss alle benötigten Daten enthalten, um die Anfrage zu verstehen
- **Selbsterklärende Nachrichten:** Die Daten des Servers müssen mit Cachable oder Nicht-Cachable beschriftet werden. Cachable Daten dürfen dann vom Client Cache verwendet werden
- **HATEOAS:** Aufbau in Schichten, so dass das die einzelnen Schichten nur ihre direkten Interaktionspartner sehen

3

Aspekte von Web-APIs

Welche Aspekte sind wichtig?

Was hat Auswirkungen darauf ob sie wichtig sind(Daten, Architektur, etc.)

4

Prototypische Analyse

Analyse der Aspekte für die beiden Technoligien

Entweder ein großes oder mehrere kleinere Prototypen

5

Evaluation

Evaluation der Analyse anhand von Use-Cases

6

Fazit

7

Ausblick