HOCHSCHULE HEILBRONN

Bachelorthesis
Angewandte Informatik (SPO1a)

Vergleichende Untersuchung von GraphQL und REST im Kontext relationaler und graphbasierter Datenbanken, hinsichtlich Anpassungs- und Leistungsfähigkeit

Robin Hefner*

9. Oktober 2024

Eingereicht bei Prof. Dr. Fankhauser

*206488, rohefner@stud.hs-heilbronn.de

Inhaltsverzeichnis

Abkürzungsverzeichnis

API: Application Programming Interface

DBMS: Database Management System

HTTP: Hypertext Transfer Protocol

REST: Representational State Transfer

RPC: Remote Prodecdure Call

SOAP: Simple Object Access Protocol

SQL: Structured Query Language

URL: Uniform Resource Locator

Abbildungsverzeichnis

Abstract

1 Einleitung

1.1 Motivation

In der modernen Softwareentwicklung spielen APIs (Application Programming Interfaces) eine entscheidende Rolle bei der Integration und Kommunikation zwischen verschiedenen Diensten und Anwendungen. Traditionell wurde REST (Representational State Transfer) als Standard für die Erstellung und Nutzung von APIs verwendet. Mit der Einführung und zunehmenden Verbreitung von GraphQL, einer Abfragesprache für APIs, die von Facebook entwickelt wurde, stehen Entwickler nun vor der Wahl zwischen diesen beiden Ansätzen. Die Wahl zwischen REST und GraphQL hat signifikante Auswirkungen auf die Entwicklung und den Betrieb der Anwendung. Unternehmen müssen eine fundierte Entscheidung treffen, welche Technologie besser zu ihren Anforderungen, im Hinblick auf Leistungsfähigkeit und Anpassungsfähigkeit, passt.

1.2 Forschungsfragen

Nachfolgend sollen die Forschungsfragen vorgestellt werden, die aus der Motivation abgeleitet wurden. Diese dienen als Grundlage der Forschung für diese Thesis.

- **FF-1: Wie unterscheiden sich GraphQL und REST hinsichtlich der Anfrage- und Antwortzeiten unter verschiedenen Lastbedingungen und Anfragenkomplexitäten ?** Diese Frage zielt darauf ab, die Performance beider Systeme unter variablen Bedingungen zu vergleichen. Beispielsweise könnte untersucht werden, wie schnell eine API auf eine einfache Datenabfrage reagiert, im Vergleich zu einer komplexeren, die mehrere Abhängigkeiten involviert. Diese Untersuchung könnte Einblicke in die Effizienz der beiden Technologien bieten und somit als Entscheidungshilfe für Entwickler dienen, die die beste Lösung für ihre spezifischen Bedürfnisse auswählen möchten.

- **FF-2: Inwiefern bieten GraphQL und REST unterschiedliche Möglichkeiten zur Abfrageanpassung?** Diese Frage beleuchtet die Flexibilität beider Systeme in Bezug auf die Individualisierung von Datenabfragen. Während REST traditionell durch feste Endpunkte gekennzeichnet ist, die jeweils eine bestimmte Datenstruktur zurückgeben, bietet GraphQL eine dynamischere Herangehensweise. Mit GraphQL können Clients genau die Daten anfordern, die sie benötigen, und keine zusätzlichen Informationen, was zu effizienteren Datenübertragungen führen kann.

Diese Fähigkeit, Anfragen präzise anzupassen, könnte die Effizienz und Benutzerfreundlichkeit von Webanwendungen erheblich beeinflussen.

1.3 Vorgehensweise

Für die Untersuchung werden sowohl theoretische Analysen als auch empirische Experimente durchgeführt. Im Rahmen der theoretischen Analyse erfolgt eine umfassende Literaturrecherche und die Analyse bestehender Studien zur Leistungsfähigkeit und Anpassungsfähigkeit von REST und GraphQL. Die empirischen Experimente umfassen die Implementierung von Beispiel-APIs mit beiden Technologien sowie die Durchführung von Leistungs- und Flexibilitätstests. Die Leistungstests konzentrieren sich auf das Messen der Latenz, des Durchsatzes und der Ressourcenauslastung bei verschiedenen Abfrageszenarien. Die Flexibilitätstests bewerten die Anpassungsfähigkeit der APIs an wechselnde Anforderungen und Schemaänderungen.

2 Grundlagen

Die folgenden Abschnitte sollen die theoretischen Grundlagen vermitteln, die notwendig sind, um das Thema dieser Thesis zu betrachten. Die Konzepte, die hier beschrieben werden sind APIs, wie REST und GraphQL, als auch relationale und Graph Datenbanken.

2.1 API

Nachfolgend werden die Grundlagen von APIs thematisiert. Hierbei werden die grundlegenden Definitionen im Zusammenhang mit APIs und die verschiedenen Typen vorgestellt.

2.1.1 Definition API

Der Begriff „API“ steht für „Application Programming Interface“. Eine API bezeichnet eine Schnittstelle, welche Entwicklern den Zugriff auf Daten und Informationen ermöglicht. Bekannte Beispiele für häufig genutzte APIs sind die Twitter- und Facebook-APIs. Diese sind für Entwickler zugänglich und ermöglichen die Interaktion mit der Software von Twitter und Facebook. Zudem ermöglichen APIs die Kommunikation zwischen Anwendungen. Sie bieten den Anwendungen einen Weg, miteinander über das Netzwerk, überwiegend das Internet, in einer gemeinsamen Sprache zu kommunizieren. (Jacobson et al.; 2012)

2.1.2 API Typen

APIs können anhand von Verfügbarkeit, Anwendungszweck oder der Spezifikation in verschiedene Typen eingeteilt werden.

API Typen nach Anwendungszweck

- **Datenbanken APIs** ermöglichen die Kommunikation zwischen einer Datenbank und einer Anwendung die deren Daten benötigt. (Gözneli; 2020)
- **Betriebssystem APIs** definieren wie Ressourcen und Services von Betriebssystemen von einer Anwendung benutzt werden, die auf deren Daten zugreift. (Gözneli; 2020)
- **Remote APIs** definieren die Regeln, wie Anwendungen auf verschiedenen Host-Maschinen miteinander interagieren. (Gözneli; 2020)
- **Web APIs** sind die verbreitetsten APIs. Sie stellen Daten bereit und übermitteln diese zwischen Web-basierenden Systemen über eine Client-Server Verbindung. (Gözneli; 2020)

API Typen nach Verfügbarkeit

Im Bezug auf Verfügbarkeit können APIs public (öffentlich), privat oder für Partner bereitgestellt werden.

Abbildung 1: API Typen nach Verfügbarkeit

- **Public APIs:** Öffentliche APIs sind für jeden Entwickler bzw. jegliche Dritte verfügbar. Eine öffentliche API kann zu einer Steigerung der Markenbekanntheit beitragen und eröffnet bei einer Monetarisierung zusätzliche Einnahmequellen.(Jacobson et al.; 2012)

- **Partner APIs:** Partner-APIs stellen eine Art Schnittstelle zwischen privaten und öffentlichen APIs dar. Ihr Zweck besteht darin, die Entwicklung von Unternehmensanwendungen zu fördern. Die Unternehmen haben dabei eine hohe Nutzerkontrolle.(Jacobson et al.; 2012)

- **Private APIs:** Dieser Typ erlaubt es Entwicklern, die innerhalb einer Firma tätig sind, die API zu nutzen, um interne Produkte, Systeme oder Apps zu integrieren. Des Weiteren können bei der Entwicklung neuer Systeme bereits vorhandene Ressourcen verwendet werden. Dadurch lassen sich die Kosten erheblich reduzieren und es entsteht eine größere Flexibilität. (Jacobson et al.; 2012)

API Typen nach Spezifikation/Protokoll

Das Ziel der Spezifikation von APIs ist die Kommunikation zwischen verschiedenen Services zu standardisieren

- **Remote Procedure Call (RPC)** stellt eine einfache und zugleich die älteste Form von Application Programming Interfaces dar. Ihr Zweck besteht in der Initiierung von Prozeduren auf unterschiedlichen Computern über das Netzwerk hinweg. Zu diesem Zweck übermittelt eine Anwendung eine oder mehrere Nachrichten an eine andere Anwendung, um eine Prozedur zu starten. Im Anschluss daran sendet die empfangende Anwendung dem Sender eine oder mehrere Nachrichten zurück, sobald die Prozedur abgeschlossen ist. Obwohl es konzeptionell einfach und leicht zu implementieren ist, gibt es eine Menge verschiedener und subtile Probleme, die zu unterschiedlichen RPC-Implementierungen führen. (Tay and Ananda; 1990)

- **Simple Object Access Protocol (SOAP)** stellt ein leichtgewichtiges Protokoll für den Austausch von Informationen in einer dezentralen, verteilten Umgebung dar. Es handelt sich um ein XML-basiertes Protokoll, welches aus drei Teilen besteht: einem Umschlag, welcher einen Rahmen für die Beschreibung des Inhalts einer Nachricht sowie ihrer Verarbeitung definiert, einer Reihe von Kodierungsregeln für die Darstellung von Instanzen anwendungsdefinierter Datentypen sowie einer Konvention für die Darstellung von Remote-Prozeduraufrufen und Antworten. SOAP kann potenziell in Kombination mit einer Vielzahl von anderen Protokollen verwendet werden. (Box et al.; 2000)

- **Representational State Transfer (REST)** wurde erstmals im Jahr 2000 in einer Dissertation von Roy Fielding beschrieben. Hierbei handelt es sich um einen Software-Architekturstil für APIs. REST basiert auf einer Ressourcenorientierung, bei der jede Entität als Ressource betrachtet und durch eine eindeutige Uniform Resource Locator (URL) identifiziert wird. Die Architektur basiert auf sechs grundlegenden Beschränkungen, darunter die Client-Server-Architektur, bei der Client und Server unabhängig voneinander agieren. Ein wesentliches Charakteristikum von REST ist die Zustandslosigkeit, d. h. jede Anfrage beinhaltet sämtliche für die Verarbeitung erforderlichen Informationen, wodurch die Interaktion zwischen Client und Server vereinfacht wird. Die Umsetzung der CRUD-Operationen (Create, Read, Update, Delete) erfolgt durch die HTTP-Methoden (POST, GET, PUT, DELETE). REST nutzt das in HTTP integrierte Caching, um die Antwortzeiten und die Leistung zu optimieren. Dabei besteht die Möglichkeit, Serverantworten als cachefähig oder nicht cachefähig zu kennzeichnen. Des Weiteren ist eine einheitliche Schnittstelle zu nennen, welche die Interaktionen zwischen unterschiedlichen Geräten und Anwendungen erleichtert und sichtbar macht. Darüber hinaus erfordert REST ein mehrschichtiges System, bei dem jede Komponente

lediglich mit der unmittelbar vorgelagerten Schicht interagiert. Die Bereitstellung von ausführbarem Code durch den Server ist optional. RESTful APIs, die diesen Prinzipien folgen, nutzen HTTP-Anfragen, um Ressourcen effizient zu bearbeiten. (Fielding; 2000) (Vadlamani et al.; 2021)

- **GraphQL** wurde 2012 von Facebook für den internen Gebrauch entwickelt. Im Jahr 2015 erfolgte die Veröffentlichung als Open-Source-Projekt für die Allgemeinheit. Das Kernkonzept von GraphQL basiert auf client-getriebenen Abfragen, bei denen der Client die Struktur der Daten präzise definiert und nur die erforderlichen Daten erhält. Dies resultiert in einer Reduktion von Datenübertragungen und ermöglicht effizientere Netzwerkaufrufe. Die hierarchische Struktur der Abfragen, welche die Graph-Struktur widerspiegelt, erlaubt eine intuitive Datenmodellierung. Die starke Typisierung in GraphQL wird durch ein Schema definiert, welches die Typen der Daten spezifiziert. Dadurch wird eine bessere Validierung und Dokumentation ermöglicht. Im Gegensatz zu REST, bei dem für verschiedene Operationen mehrere Endpunkte erforderlich sind, verwendet GraphQL lediglich einen einzigen Endpunkt für alle API-Abfragen. (Vadlamani et al.; 2021)

2.2 Datenbank

Im Folgenden werden die Grundlagen von Datenbanken behandelt. Es werden grundlegende Definitionen im Zusammenhang mit Datenbanken und die verschiedenen Arten von Datenbanken vorgestellt.

2.2.1 Definition Datenbank und Datenbank Management System

Eine Datenbank stellt eine Sammlung von Daten und Informationen dar, welche für einen einfachen Zugriff gespeichert und organisiert werden. Dies umfasst sowohl die Verwaltung als auch die Aktualisierung der Daten. Die in der Datenbank gespeicherten Daten können nach Bedarf hinzugefügt, gelöscht oder geändert werden. Die Funktionsweise von Datenbanksystemen basiert auf der Abfrage von Informationen oder Daten, woraufhin entsprechende Anwendungen ausgeführt werden. DBMS bezeichnet eine Systemsoftware, die für die Erstellung und Verwaltung von Datenbanken eingesetzt wird. Zu den Funktionalitäten zählen die Erstellung von Berichten, die Kontrolle von Lese- und Schreibvorgängen sowie die Durchführung einer Nutzungsanalyse. Das DBMS fungiert als Schnittstelle zwischen den Endnutzern und der Datenbank, um die Organisation und Manipulation von Daten zu erleichtern. Die Kernfunktionen des DBMS umfassen die Verwaltung von Daten, des Datenbankschemas, welches die logische Struktur der Datenbank definiert, sowie der Datenbank-Engine, welche das Abrufen, Aktualisieren und Sperren von Daten ermöglicht. Diese drei wesentlichen Elemente dienen der Bereitstellung standardisierter Verwaltungsverfahren, der Gleichzeitigkeit,

der Wiederherstellung, der Sicherheit und der Datenintegrität. (Hassan; 2021)

2.2.2 Datenbank Modelle

Datenbanken können in SQL und NoSQL (Not Only SQL) Datenbanken unterteilt werden. Eine relationale Datenbank stellt eine SQL Datenbank dar, eine Graph Datenbank eine NoSQL Datenbank. Im Folgenden werden beide Vertreter der Technologien vorgestellt.

Relationale Datenbank

Relationale Datenbanken basieren auf dem von E. F. Codd eingeführten relationalen Modell. Es verwendet relationale Algebra und Tupel-Relationen und speichert Daten in tabellarischer Form, wobei Zeilen als Tupel und Spalten als Attribute bezeichent werden. Dies hat zur Folge, dass die Struktur, in der die Daten gespeichert werden sollen vor der Speicherung in der Datenbank bekannt sein müssen. Falls Werte nicht vorkammen werden diese auf null gesetzt. Die Tabellen sind durch Primär- und Fremdschlüssel miteinander verknüpft. Diese Art von Datenbanken sind einfach zu entwerfen und umzusetzen und zeichnen sich durch Benutzerfreundlichkeit, Konsistenz und Flexibilität aus. Diese Datenbanken eignen sich besonders für normalisierte Daten und solche, die Transaktionsintegrität erfordern. (Győrödi et al.; 2016) (Hassan; 2021)

Graph Datenbanken

Graphdatenbanken basieren auf dem Konzept der Graphentheorie, die Daten in Form von Graphen schemafreien Weise speichern. Eine Graphdatenbank ist eine Sammlung von Knoten und Kanten, wobei die Knoten Entitäten oder Objekte darstellen und die Kanten die Beziehungen zwischen den Knoten darstellen. Der Graph enthält auch Informationen über die Eigenschaften der mit den Knoten verbundenen Objekte. Graphdatenbanken bieten eine effiziente Datenspeicherung speziell für semistrukturierte Daten. Die Formulierung von Abfragen als Traversale in Graphen-Datenbanken sind sie schneller als relationale Datenbanken. Graphdatenbanken befolgen ACID-Bedingungen und bieten Rollback-Unterstützung, die die Konsistenz der Informationen Informationen garantiert. (Hassan; 2021)

3 Implementierung

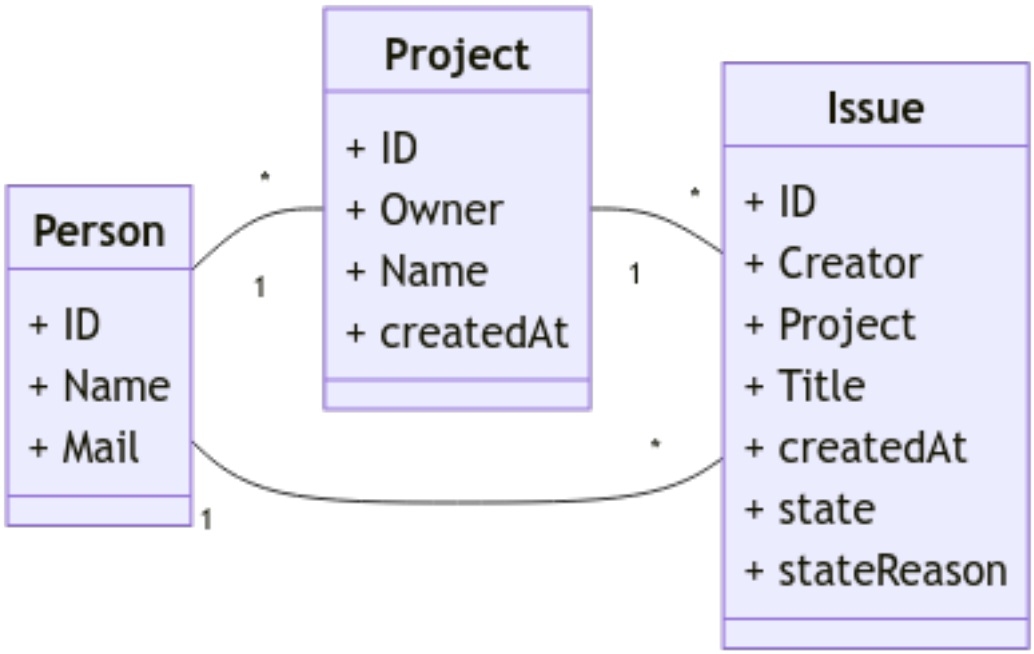

Abbildung 2: Klassenstruktur

4 Methodik

...

5 Ergebnisse

6 Diskussion

...

7 Fazit

...

Literaturverzeichnis

Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen, H. F., Thatte, S. and Wine, D. (2000). Simple object access protocol (soap) 1.1.
URL: *https://www.w3.org/TR/2000/NOTE-SOAP-20000508/*

Fielding, R. T. (2000). Architectural styles and the design of network-based software architectures.
URL: *https://ics.uci.edu/ fielding/pubs/dissertation/top.htm*

Győrödi, C., Ștefan, A., Győrödi, R. and Bandici, L. (2016). A comparative study of databases with different methods of internal data management, *International Journal of Advanced Computer Science and Applications (IJACSA) Vol. 7, No. 4, .*

Gözneli, B. (2020). *Identification and evaluation of a process for transitioning from rest apis to graphql apis in the context of microservices architecture*, Master's thesis, Technische Universität München, München, BY.

Hassan, M. A. (2021). Relational and nosql databases: The appropriate database model choice, *2021 22nd International Arab Conference on Information Technology (ACIT)*, pp. 1–6.

Jacobson, D., Brail, G. and Woods, D. (2012). *APIs: A Strategy Guide*, O'Reilly Medi, Sebastopol, CA. 978-1-449-30892-6.

Tay, B. H. and Ananda, A. L. (1990). A survey of remote procedure calls.
URL: *https://doi.org/10.1145/382244.382832*

Vadlamani, S. L., Emdon, B., Arts, J. and Baysal, O. (2021). Can graphql replace rest? a study of their efficiency and viability.
URL: *https://ieeexplore.ieee.org/abstract/document/9474834*