HOCHSCHULE HEILBRONN

Bachelorthesis
Angewandte Informatik (SPO1a)

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

Robin Hefner*

29. November 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

SQL: Structured Query Language

URL: Uniform Resource Locator

Abbildungsverzeichnis

Abstract

Zusammenfassung

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. Zusätzlich gewinnt die Wahl der zugrunde liegenden Datenbanktechnologie an Bedeutung, da sie maßgeblich beeinflusst, wie effektiv REST und GraphQL implementiert werden können. Relationale Datenbanken, die auf strukturierten Tabellen und SQL basieren, bieten bewährte Mechanismen für komplexe Abfragen und garantieren hohe Datenintegrität. In Kombination mit REST ermöglichen sie eine klare Strukturierung von Endpunkten und eine stabile, vorhersehbare Datenabfrage. In GraphQL hingegen können relationale Datenbanken durch Resolver genutzt werden, um gezielt nur die angeforderten Daten bereitzustellen, was die Abfrageleistung bei komplexen Datenmodellen verbessern kann. Graphdatenbanken, wie Neo4j oder ArangoDB, bieten hingegen eine natürliche Integration für stark vernetzte Daten. Sie zeigen ihre Stärken besonders in Kombination mit GraphQL, da die flexible Abfragesprache direkt auf die Eigenschaften von Graphdatenbanken abgestimmt ist und tiefe, verknüpfte Abfragen effizient ermöglicht. Im Kontext von REST hingegen können Graphdatenbanken ebenfalls verwendet werden, erfordern jedoch oft eine zusätzliche Ebene der Verarbeitung, um die Netzwerkstruktur in flache, hierarchische API-Endpunkte zu überführen. Die Wahl zwischen REST und GraphQL hat signifikante Auswirkungen auf die Entwicklung und den Betrieb der Anwendung, insbesondere im Zusammenspiel mit der zugrunde liegenden Datenbanktechnologie. Unternehmen müssen eine fundierte Entscheidung treffen, welche Kombination aus API-Architektur und Datenbank besser zu ihren Anforderungen im Hinblick auf Leistungsfähigkeit, Skalierbarkeit 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 relationale Algebra, Graphentheorie, APIs, als auch relationale und Graph Datenbanken.

2.1 Relationale Algebra

Die relationale Algebra ist ein mathematisches System, welches 1970 von E.F. Codd entwickelt wurde. Sie wird unter anderem zu Abfrage und Mutation von Daten in relationalen Datenbanken verwendet wird. Durch sie wird eine Menge an Operationen beschrieben, die auf die Relationen angewendet werden können, um neue Relationen zu bilden. (Studer; 2019)

2.1.1 Basisrelation

Um relationale Algebra anzuwenden werden Basisrelationen benötigt. Diese bilden den Grundbaustein, um mit Grundoperationen komplexe Ausdrücke aufzubauen, die neue Relationen definieren. Basisrelationen bestehen aus drei Bausteinen, nämlich Tupel, Attributen und Domänen. Tupel spiegeln die Zeilen der Tabellen woeder, die einzelne Datensätze repräsentieren. Diese werden durch Attribute in einzelne Spalten eingeteilt, welche die Eingenschaften der Tupel beschreiben. Die Wertebereiche, die für die einzelnen Attribute zulässig sind nennt man Domänen. Somit ist jede Relation eine Menge von Tupeln mit spezifischen Attributen und deren Domänen. (Studer; 2019)

2.1.2 Grundoperationen

Grundoperationen in der relationalen Algebra sind einfache mengentheoretische Operationen, die auf die Basisrelationen angewandt werden. Insgesamt gibt es sechs Grundoperationen, die nachfolgend erläutert werden.

- Bei der **Selektion** σ werden die einzelnen Tupel basieren auf einer Bedingung gefiltert. Ein Beispiel hierfür wäre σ Alter > 30 (Person). Hierdurch werden nur Personen mit einem Alter von mehr als 30 Jahren zurückgeliefert.
- Die **Projektion** π ermöglicht es bestimmte Attribute einer Relation auszuwählen oder zu entfernen. Beispielsweise kann durch π Name, Alter (Person), nur der Name und das Alter einer Person zurückgegeben werden.

- Das **Kartesisches Produkt** $\times$ kombiniert jede Zeile der ersten Relation mit jeder Zeile der zweiten Relation. Somit erzeugt $R \times S$ alle möglichen Kombinationen aus R und S.
- Eine **Vereinigung** $\cup$ verknüpft die Tupel zweier Relationen, die eine gleiche Struktur aufweisen. $R \cup S$ kombiniert somit alle Tupel aus beiden Relationen mit gleicher Struktur, ohne Duplikate zu erzeugen.
- Die **Differenz** $\setminus$ zweier Relationen liefert alle Tupel, die in der ersten Relation vorkommen, aber nicht in der Zweiten. Sinngemäß gibt $R \setminus S$ alle Tupel R aus die nicht in S enthalten sind.
- Der **Schnitt** $\cap$ findet die Tupel, die in beiden Relationen vorhanden sind. Somit gibt $R \cap S$ die Tupel, die sowohl in R als auch in S entahlten sind zurück.

Durch die Verbindung dieser Grundoperationen können andere Operationen, wie beispielhaft ein Theat-Join $\bowtie$, welcher durch eine Kombination aus kartesischem Produkt und Selektion alle Tupel zweier Relationenen aufgrund einer Bedingung miteinander verbindet, erstellt werden. (Studer; 2019)

2.2 Graphentheorie

Ein Graph besteht im allgemeinen aus Knoten und verbindenden Kanten (vgl. Abb 1). In der Informatik bietet diese Datenstruktur einen großen Vorteil gegenüber der relationalen Algebra, wenn es sich um stark verzweigte Daten handelt. (Saidur; 2017)

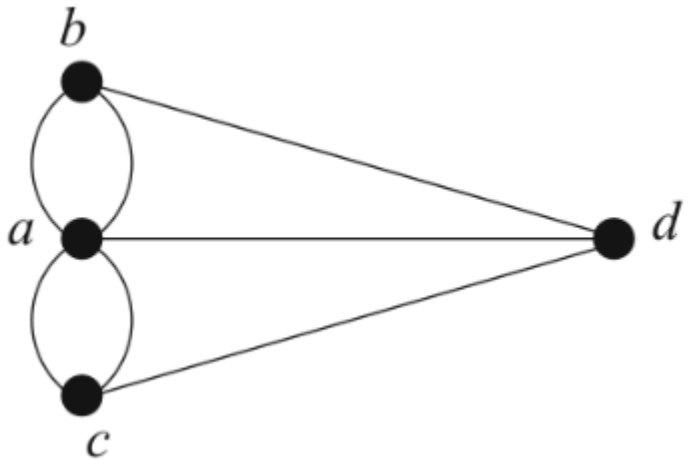

Abbildung 1: Modell eines ungerichteten Graphen. (Saidur; 2017)

2.2.1 Knoten

Knoten sind Punkte innerhalb eines Graphen, die beispielsweise als Objekte der realen Welt verstanden werden können. Diese Objekte können zum Beispiel geographische Koordinaten sein, um einen Distanz-Graphen zu erstellen oder auch Webseiten, um einen Web-Graphen zu erhalten, der die Verbindung verschiedener Webseiten aufzeigt. Innerhalb der Knoten unterscheidet man verschiedene Typen. Zum einen gibt es Isolierte Knoten. Diese besitzen keine Verbindung zu anderen Knoten des Graphens und können zur Identifizierung nicht-verbundener Teile eines Netzwerks verwendet werden. Außerdem gibt es verbundene Knoten, welche mindestens eine Verbindung zu einem andern Knoten besitzen. Dadurch ergibt sich eine dirkete Verbindung zu einem benachbarten Knoten. Außerdem kann eine Indirekte Verbindung enstehen, indem ein Pfad zwischen Ausgangs und Endknoten existiert, der über mehrere Verbindungen führt. (Saidur; 2017)

2.2.2 Kanten

Kanten dienen in einem Graphen dazu, die Knoten miteinander zu Verbinden, um eine Relation zwischen ihnen zu visualisieren. Jede Kante besitzt einen Start- und Endknoten, der auch derselbe Knoten sein kann. Somit würde man in diesem Fall von einer Schleife sprechen. So wie es verschiedene Arten von Knoten gibt, exestieren auch verschiedene Kanten. In Abb. 1 sind *ungerichtete Kanten* im Graphen zu sehen. Sie verbinden die Knoten auf die trivialste Art, indem sie ohne Richtung, Gewicht oder andere Beschränkung eine Verbindung herstellen. Durch ungerichtete Kanten ensteht ein ungerichteter Graph. *Gerichtete Kanten* werden in Abb. 2 (a) genutzt. Diese werden durch einen Pfeil visualisiert. Dieser gibt an, in welcher Richtung der Graph eine Beziehung zwischen den Knoten herstellt. Es ist ebenfalls möglich, dass zwei Knoten eine beidseitige Beziehung durch zwei gerichtete Kanten, also eine Kante pro Richtung, eingehen. Ein Graph, der durch gerichtete Kanten verbunden ist wird gerichteter Graph genannt. Werden Zahlenwerte zu einer Kante hinzugefügt (vgl. Abb 2 (b)), so spricht man von *gewichteten Kanten*. Diese können verwendet werden, um die Strecke zwischen zwei Kanten darzustellen oder die Kosten für die Nutzung der Kante anzugeben. Wird ein Graph mit gewichteten Kanten verbunden, spricht man von einem gewichteten Graphen.(Saidur; 2017)

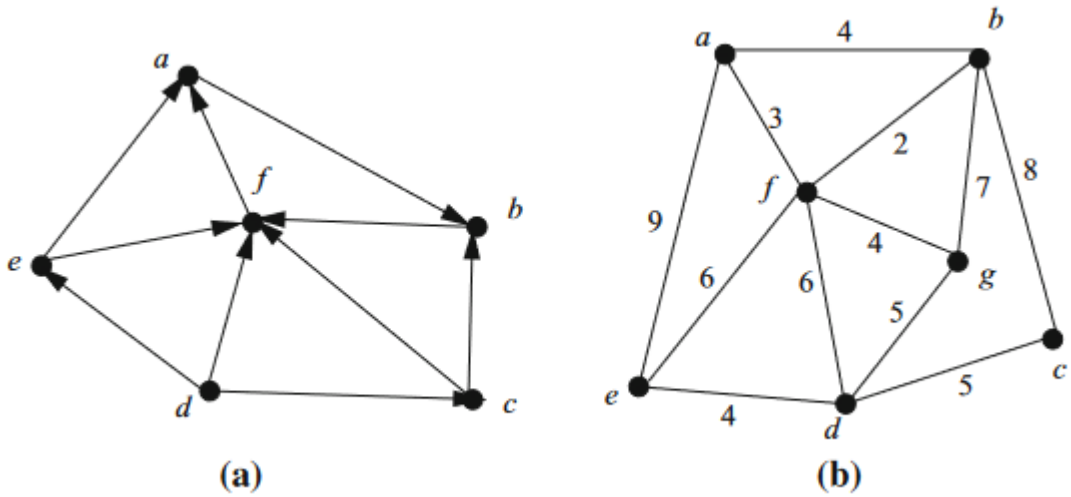

Abbildung 2: Modell eines gerichteten (a) und eines gewichtet Graphen (b). (Saidur; 2017)

2.3 API

Nachfolgend werden die Grundlagen von APIs thematisiert. Hierbei werden die grundlegenden Definitionen sowie die Typen vorgestellt, die für diese Arbeit von Relevanz sind.

2.3.1 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)

2.3.2 REST API

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)

2.3.3 GraphQL

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)

2.4 Datenbank

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

2.4.1 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)

2.4.2 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)

2.4.3 Graphdatenbanken

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 Datenmodellierung

3.1 Datenmodell

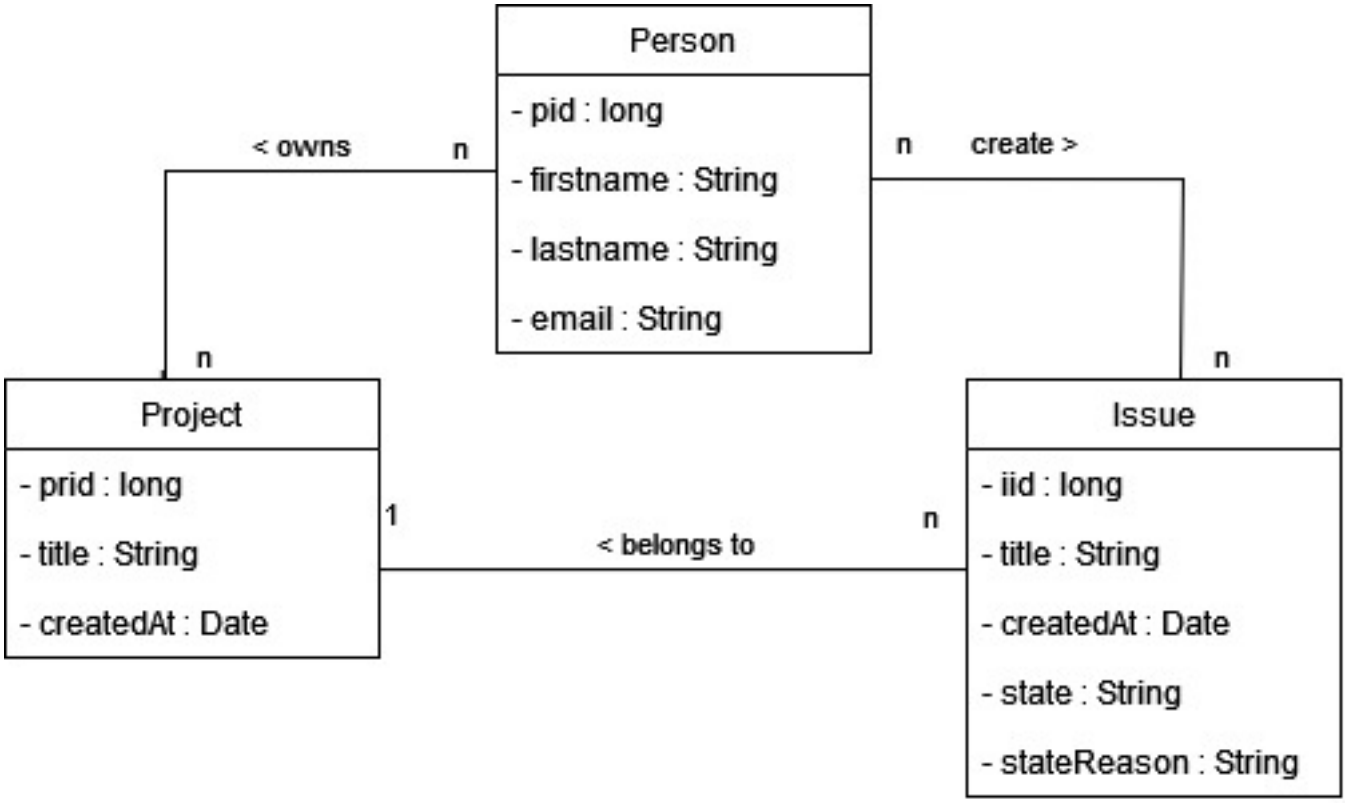

Abbildung 3: Klassenstruktur

4 Systemdesign

4.1 Schnittstellendesign

5 Implementierung

5.1 Grundprinzipien während der Implementierung

6 Ergebnisse

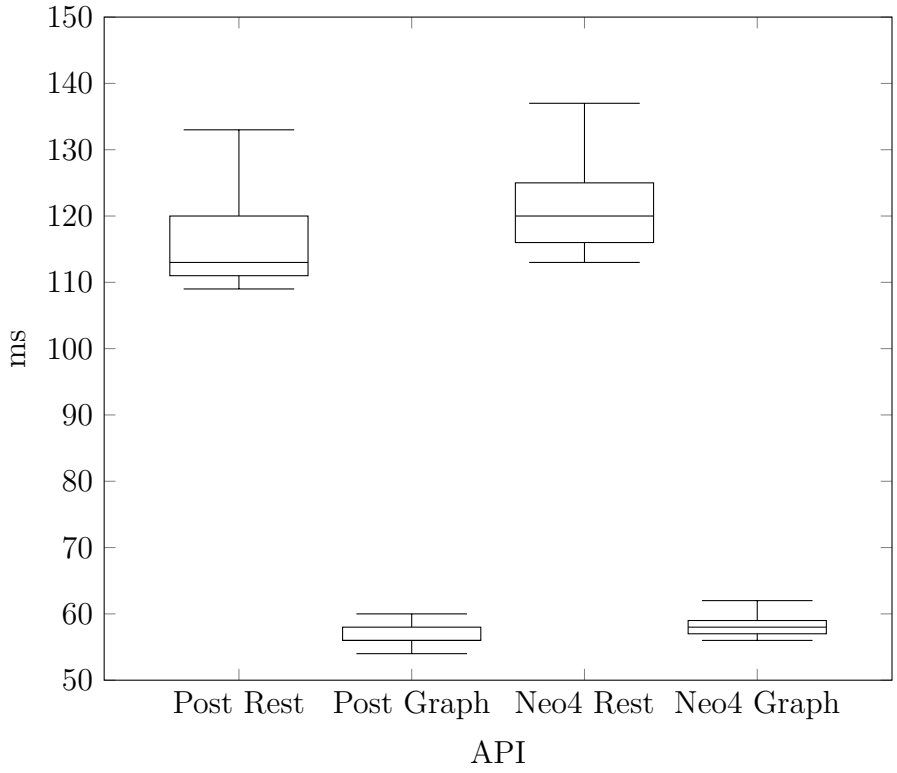

Abbildung 4: /api/person/?id

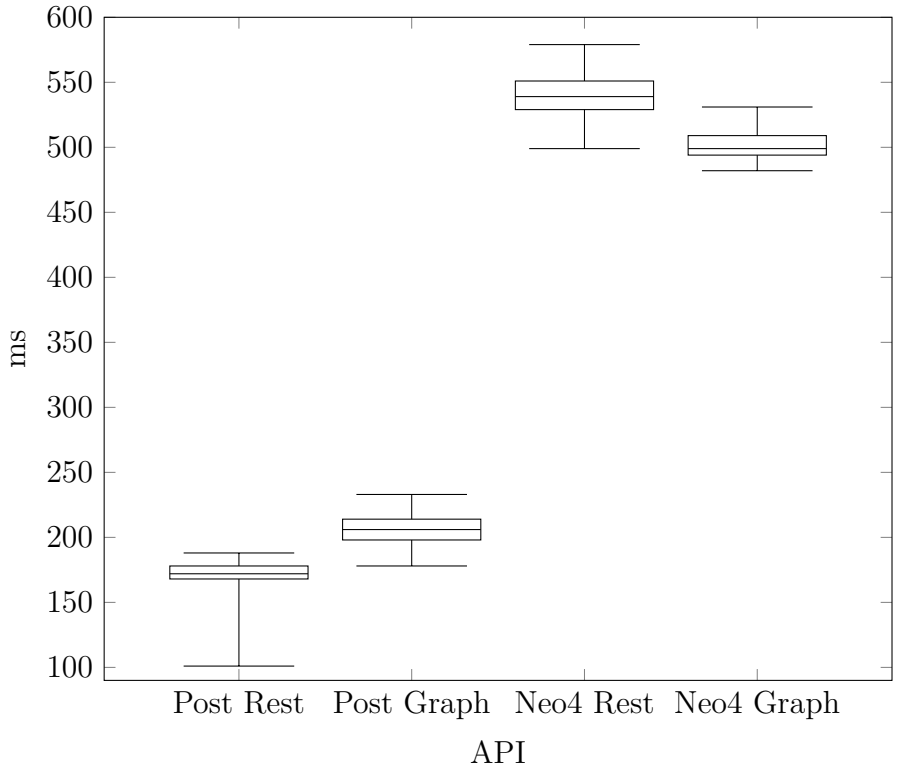

Abbildung 5: GET /api/person

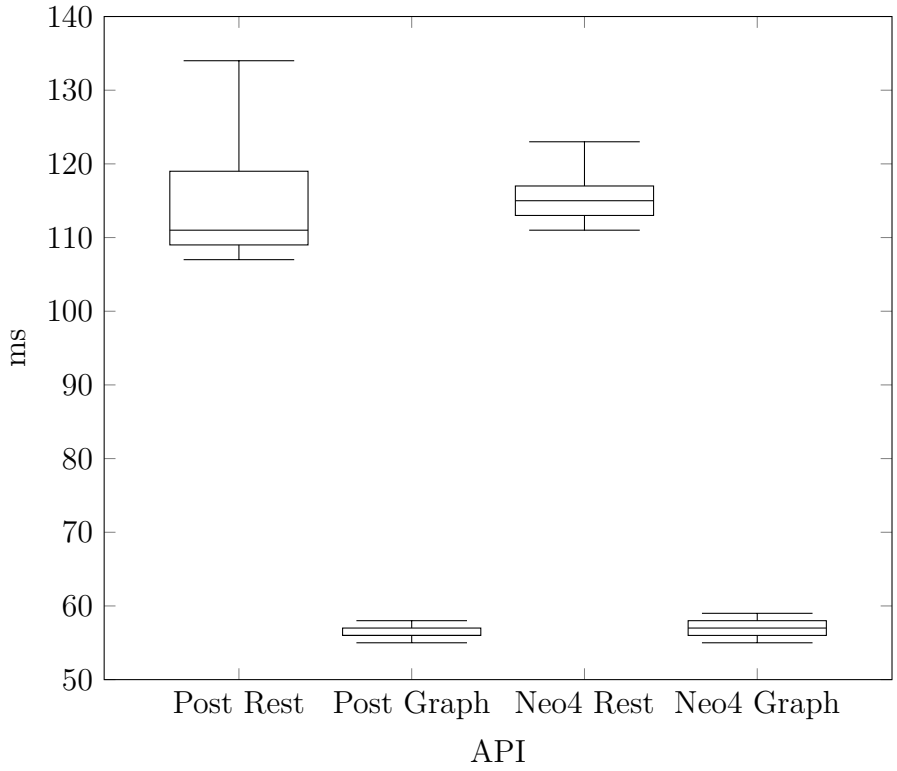

Abbildung 6: GET /api/persons/:pid/projects/issues

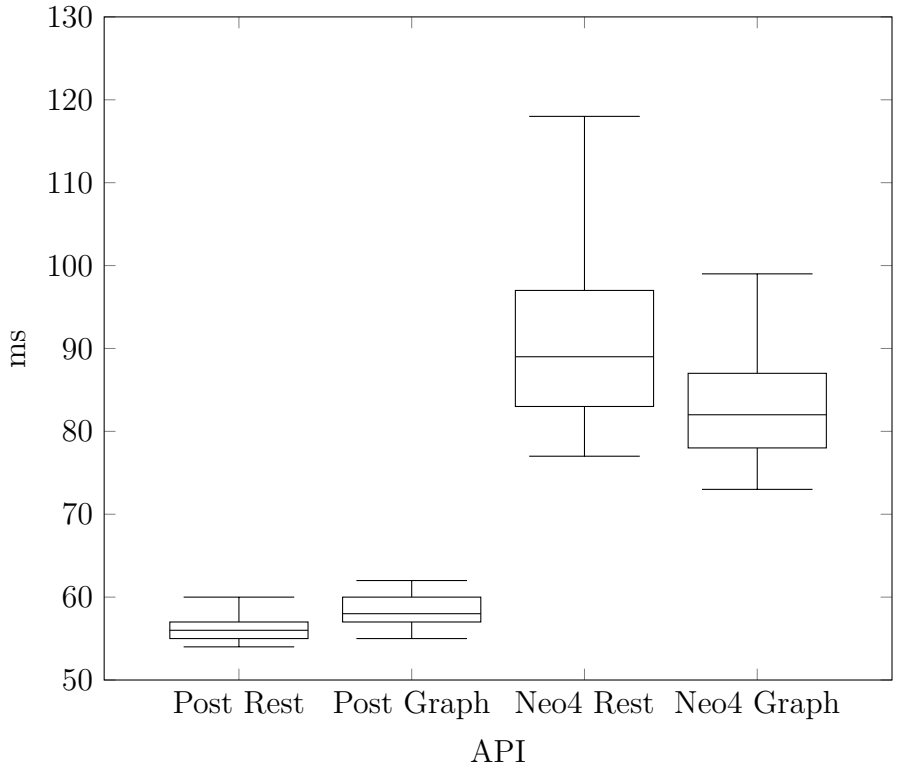

Abbildung 7: POST /api/persons/:pid/projects/:prid/issues

7 Diskussion

...

8 Fazit

...

Literaturverzeichnis

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, .*

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. ISBN: 978-1-449-30892-6.

Saidur, R. (2017). *Basic Graph Theory*, Springer, Cham, Switzerland. ISBN: 978-3-319-49474-6.

Studer, T. (2019). *Relationale Datenbanken*, Springer Vieweg, Berlin, Germany. ISBN: 978-3-662-58976-2.

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*