listing

Hochschule Bremerhaven

Fachbereich II
Vernetzte Systeme
Informatik B.Sc.

Modul
Vernetzte Systeme

Linkshortner

Entwicklung und Implementierung eines URL-Kürzungsdienstes

Vorgelegt von:	Max Schneider	MatNr. 40605
	Fabian Tober	MatNr. 40607
	Ben Schlickowey	MatNr. 40869
	Evin Gürses	MatNr. 33276
Vorgelegt am:	23. März 2025	
Dozent:in:	Prof. Dr. Karin Vosseberg	

Inhaltsverzeichnis

1 Einleitung

In der heutigen Zeit muss es schnell gehen. Die digitale Welt bereichert sich vermehrt an der schnellen und effizienten Verbreitung von Informationen und Lösungen. Geprägt wird diese Dynamik als zentrales Element durch die Verkürzung von URLs, welche die langen und übersichtlichen Webadressen auf ein handliches aber auch nutzerfreundliches Format zu reduzieren, ermöglichen. Dies hat zur Folge, dass unübersichtliche Webadressen auf ein nutzerfreundliches und handliches Format reduziert werden. Während bekannte URL-Shortening-Dienste wie TinyURL oder Bitly globale Lösungen sich für derartige Probleme aneignen, wächst nichtsdestotrotz in vielen Fällen der Bedarf nach einer lokal gehosteten Alternative. Ein selbst gehosteter URL-Shortener bringt viele Vorteile mit sich. Er gestattet die vollständige Kontrolle über die erzeugten Links, bietet erweiterte Anpassungsmöglichkeiten und erhöht die Datensicherheit. Eine lokale Lösung ist in sensiblen Bereichen, in denen Datenschutz und Zugriffskontrolle oberste Priorität haben, eine verlässliche und anpassungsfähige Alternative. Es kann gewährleistet werden, dass sensible Daten nicht an externe Server weitergeleitet werden, indem auf Drittanbieter-Dienste verzichtet wird. Dies ist besonders relevant für Unternehmen, die hohe Sicherheitsanforderungen erfüllen oder gesetzliche Compliance sicherstellen müssen. Dieses Projekt zielt darauf ab, einen lokalen URL-Shortener zu entwickeln, der eine effiziente Weiterleitung und Skalierbarkeit bietet. Bei der Umsetzung kommen moderne Technologien zum Einsatz, die eine robuste Architektur und hohe Leistungsfähigkeit garantieren. Als Hauptkomponente des Systems fungiert der Webserver Apache, eine Open-Source-Umsetzung der Java-Servlet-Technologie. Apache hat die Aufgabe, eingehende Webanfragen zu empfangen und sie an die entsprechenden Servlets weiterzuleiten, die für die URL-Verarbeitung zuständig sind und geeignete Antworten erstellen. MariaDB, eine relationale Datenbank, die eine zuverlässige und skalierbare Speicherung der Daten bietet, wird für die Speicherung der URL-Zuordnungen verwendet. Um die Performance zu optimieren, wird Redis gleichzeitig als Cache verwendet, mit dem Kurzlinks, die häufig angefordert werden, direkt aus dem Arbeitsspeicher bereitgestellt werden. Redis, eine effiziente In-Memory-Datenbank, beschleunigt den Datenabruf erheblich und verringert so die Serverlast. Dadurch werden die Systemressourcen effizienter genutzt und es kann eine höhere Verarbeitungsleistung bei zahlreichen zeitgleichen Anfragen erzielt werden. Darüber hinaus wäre es möglich, HAProxy als Load Balancer zu verwenden, um eine effiziente Lastverteilung sicherzustellen. HAProxy gewährleistet eine hohe Verfügbarkeit und Ausfallsicherheit des Systems, indem es die Anfragen effizient auf mehrere Server verteilt. Insbesondere für Anwendungen mit hohem Verkehrsaufkommen ist diese Maßnahme von Bedeutung, da sie verhindert, dass einzelne Server, welche überlastet werden und somit die Reaktionszeiten des Systems beeinträchtigt werden. Ein weiterer wesentlicher Punkt dieses Vorhabens ist die Sicherheit des URL-Shorteners. Es sind Mechanismen nötig, die Missbrauch vorbeugen, zusätzlich zur Kontrolle der generierten Links. Hierzu zählen unter anderem Ratenbegrenzungen, um eine exzessive Nutzung durch automatisierte Systeme zu unterbinden, sowie Konzepte zur Authentifizierung und Autorisierung, die den Zugang zu sensiblen Kurzlinks sichern. Außerdem können Blacklist-Mechanismen verwendet werden, um das Verkürzen potenziell schädlicher

oder betrügerischer URLs zu verhindern. Auch eine benutzerfreundliche Oberfläche wird neben der technischen Umsetzung wichtig sein. Eine benutzerfreundliche Webanwendung soll es den Nutzern ermöglichen, Links unkompliziert zu verkürzen, zu verwalten und deren Nutzung nachzuvollziehen. Es ist möglich, weitere Funktionen wie die Auswertung der Link-Nutzung, Verfallsdaten für Kurzlinks oder die Erstellung personalisierter Kurz-URLs zu integrieren.

2 Zielsetzung

Das Projekt zielt darauf ab, einen lokalen URL-Shortener zu entwickeln, der sowohl leistungsfähig als auch sicher ist. Es soll eine effiziente Alternative zu bestehenden Services wie Bitly oder TinyURL angeboten werden, die es Unternehmen, Institutionen und Privatpersonen ermöglicht, Kurzlinks unabhängig und mit vollständiger Kontrolle zu verwalten. Dies ist besonders wichtig in Bereichen, die hohe Sicherheits- und Datenschutzanforderungen aufweisen. Dort sind externe Lösungen aufgrund von Compliance-Vorgaben oder dem Bedarf an individuellen Anpassungen nicht möglich. Um eine robuste, skalierbare und leistungsstarke Architektur zu gewährleisten, beruht die technische Umsetzung auf einer Kombination aus Apache, MariaDB, Redis und HAProxy. Die Bearbeitung von Anfragen, die an das System herangetragen werden, und die Kommunikation mit den unterschiedlichen Systemelementen werden durch Apache gesteuert. MariaDB fungiert als zentrale Datenbank zur Ablage der URL-Zuordnungen und gewährleistet eine verlässliche Verwaltung der Kurzlinks. Zur weiteren Steigerung der Performance wird Redis als Caching-Lösung verwendet, da es häufig genutzte URLs direkt aus dem Arbeitsspeicher liefert und so die Antwortzeiten erheblich verringert. HAProxy kann auch als Load Balancer eingesetzt werden, um die Lastverteilung zu optimieren und die Systemverfügbarkeit bei intensiver Nutzung zu gewährleisten. Auch die Sicherheit ist neben der technischen Infrastruktur entscheidend. Mit einem lokalen URL-Shortener ist es möglich, die erzeugten Links zu kontrollieren und individuelle Sicherheitsvorkehrungen umzusetzen. Hierzu zählen Authentifizierungs- und Autorisierungsmechanismen zur Kontrolle des Zugriffs, Ratenbegrenzungen zum Schutz vor Missbrauch sowie Blacklist-Funktionen, um schädliche oder betrügerische Links zu vermeiden. Es soll neben der Backend-Architektur auch eine Webanwendung entwickelt werden, die sich durch Nutzerfreundlichkeit auszeichnet und eine einfache Verwaltung der Kurzlinks ermöglicht. Zusätzlich zur bloßen Funktion des URL-Kürzens könnten weiterführende Features wie Analysewerkzeuge zur Auswertung von Zugriffen, anpassbare Link-Ablaufzeiten oder die Erstellung personalisierter Kurz-URLs integriert werden. Zusammenfassend zielt dieses Projekt darauf ab, eine leistungsfähige und sichere Alternative zu den bestehenden URL-Shortening-Diensten zu entwickeln. Die Kombination von Apache, MariaDB, Redis und HAProxy ermöglicht die Entwicklung eines Systems mit hoher Performance, Skalierbarkeit und Sicherheit. Mit einer lokalen Lösung erhält man nicht nur mehr Kontrolle über die generierten Links, sondern kann auch sicherstellen, dass Datenschutz- und Sicherheitsanforderungen eingehalten werden. Diese Architektur schafft die Basis für einen flexiblen und anpassungsfähigen URL-Shortener, der vor

allem für Projekte mit besonderen Anforderungen eine wertvolle Alternative zu bestehenden Lösungen bietet.

Eine leistungsstarke Plattform, die den Anforderungen an Skalierbarkeit, Sicherheit und Benutzerfreundlichkeit gerecht wird, entsteht durch die Kombination bewährter Technologien.

3 LinkShortener

Das Internet weist URLs auf, die lang und oft schwer zu merken sind, insbesondere wenn sie viele Parameter enthalten. Hier wird der LinkShortener verwendet – eine Anwendung, die lange URLs in eine kurze und benutzerfreundliche Form umwandelt. Dies erleichtert das Teilen von Links und verbessert die Lesbarkeit, besonders in sozialen Medien, E-Mails oder auf mobilen Geräten.

3.1 Funktionsweise des LinkShorteners

Um eine lange URL in eine kompakte Version zu transformieren, arbeitet der LinkShortener in mehreren Schritten: Eingabe der Original-URL: Der Nutzer gibt eine URL mit langer Zeichenfolge ins System ein, die gekürzt werden soll.

- 1. Generierung eines ShortCodes: Ein eindeutiger kurzer Code (shortCode) wird vom System generiert, um die originale URL zu kennzeichnen. Um Kollisionen zu vermeiden, kann dieser Code entweder zufällig generiert oder anhand eines spezifischen Algorithmus erstellt werden.
- 2. Speicherung in der Datenbank: Der erstellte shortCode und die entsprechende Original-URL werden in einer MariaDB-Datenbank abgelegt. Damit wird eine dauerhafte Zuordnung sichergestellt, die es ermöglicht, dass die Kurz-URL jederzeit auf die Originaladresse verweist.
- 3. Rückgabe der verkürzten URL: Das System gibt nach erfolgreicher Speicherung eine verkürzte URL zurück, die den shortCode enthält, wie z. B.: https://information.hs-bremerhaven.de/kurzcode Sobald ein Nutzer diese Kurz-URL in den Browser eingibt, sucht das System in der Datenbank nach dem zugehörigen Eintrag und leitet somit den Nutzer automatisch zur Original-URL weiter.

Vorteile der URL-Verkürzung Die Verwendung eines LinkShorteners bietet mehrere Vorteile:

- Benutzerfreundlichkeit: Verkürzte URLs haben den Vorteil, dass sie einfach zu merken und zu tippen sind.

- Optimierung für soziale Medien: Kürzere Links sind für Plattformen wie Twitter oder LinkedIn vorteilhaft, da sie weniger Zeichen benötigen.
- Bessere Lesbarkeit: Kurzlinks sind besonders in gedruckten Medien oder QR-Codes vorteilhaft.
- Tracking und Analyse: Viele URL-Verkürzungsdienste bieten die Möglichkeit, Nutzungsstatistiken der Links zu erstellen, wie zum Beispiel die Anzahl der Klicks oder die geografische Herkunft der Nutzer.

3.2 Technische Umsetzung

Ein LinkShortener kann in unterschiedlichen Programmiersprachen implementiert werden. In der Regel werden PHP, Java oder Python zusammen mit einer relationalen Datenbank wie MariaDB verwendet. Um eindeutige und kurze Codes zu generieren, kann Hashing oder Base62-Encoding verwendet werden. Zusammenfassend lässt sich sagen, dass der LinkShortener eine effiziente Lösung für die Verwaltung langer URLs darstellt, und somit eine einfache sowie verlässliche Möglichkeit, Nutzer zu ihren Zielseiten weiterzuleiten, bietet. Er ist eine wertvolle und vor allem interessante Ergänzung für viele Webanwendungen, da er Datenbankspeicherung, Code-Generierung und Nutzerfreundlichkeit kombiniert. [1]

3.3 LinkDAO

Die LinkDAO-Klasse bildet eine grundlegende Komponente eines URL-Verkürzungsdienstes und ist für die Interaktion mit der MariaDB-Datenbank verantwortlich. Sie ist dafür zuständig, URL-Daten zu speichern und abzurufen. Zudem sorgt sie dafür, dass die Kurz-URLs effizient und zuverlässig verwaltet werden. Datenbankoperationen mit LinkDAO Die LinkDAO (Data Access Object)-Klasse hat die Hauptaufgabe, Datenbankoperationen zu kapseln und die direkte Kommunikation zwischen Anwendung und Datenbank zu erleichtern. Sie erfüllt dabei drei zentrale Funktionen:

- 1. Hinzufügen neuer URLs und ihrer Shortcodes: Wenn ein Benutzer eine lange URL zur Verkürzung eingibt, erstellt das System einen Shortcode. Diese Kombination aus Original-URL und ShortCode wird anschließend in die MariaDB-Datenbank eingefügt, um später für Weiterleitungen verwendet zu werden.
- 2. 2. Abruf der ursprünglichen URL unter Verwendung des ShortCodes: Ruft ein Nutzer eine verkürzte URL auf, so wird der ShortCode von dem System aus der Anfrage extrahiert. Die LinkDAO-Klasse sucht in der MariaDB-Datenbank nach der passenden Original-URL, um eine Weiterleitung vorzunehmen.

- 3. Verwaltung und Optimierung der Datenbankabfragen: LinkDAO kommt optimierte SQL-Abfragen zum Einsatz, um eine effiziente Performance zu gewährleisten. Durch Indizes auf den Tabellen und Caching-Techniken wird es möglich, Datenbankzugriffe schnell und ressourcenschonend auszuführen.

3.4 Vorteile der DAO-Architektur

Das Anwenden des DAO-Patterns hat verschiedene Vorzüge:

- Klare Trennung zwischen Geschäftsanwendung und Datenbankzugriff: Die Applikation bleibt modular, indem Datenbankoperationen von der Hauptlogik getrennt werden.
- Wiederverwendbarkeit: Die LinkDAO-Klasse ermöglicht es verschiedenen Teilen der Anwendung, sie zu verwenden, ohne dass eine Duplizierung des Datenbankcodes erforderlich ist.
- Erweiterbarkeit: Sollte eine Migration zu einer anderen Datenbank notwendig sein (z. B. von MariaDB zu PostgreSQL), sind lediglich Anpassungen in der DAO-Klasse erforderlich.

Somit kann geschlussfolgert werden, dass die LinkDAO-Klasse das Bindeglied zwischen der Anwendung und der Datenbank darstellt, und gewährleistet eine sichere sowie leistungsstarke Verwaltung von kurzen und langen URLs. Sie kümmert sich effizient um das Speichern und Abrufen von URLs, wodurch sie die Basis für einen funktionierenden URL-Verkürzungsdienst bildet. [1]

3.5 DBUtil

Die DBUtil-Klasse ist ein grundlegender Bestandteil von datenbankgestützten Anwendungen, der eine zuverlässige und effiziente Verbindung zur MariaDB-Datenbank ermöglicht. Sie fungiert als Verbindung zwischen der Anwendung und der Datenbank und sorgt dafür, dass Daten sicher gespeichert und auch abgerufen werden können. Herstellung der Datenbankverbindung DBUtil hat unter anderem die Aufgabe, eine Verbindung zur MariaDB-Datenbank aufzubauen. Hierzu wird eine Verbindung über den JDBC-Treiber (Java Database Connectivity) aufgebaut, der Java-Anwendungen die Kommunikation mit relationalen Datenbanken wie MariaDB oder MySQL ermöglicht. Eine derartige Verbindung wird benötigt, um Datenbankoperationen wie bspw. das Einfügen, Aktualisieren oder Abrufen von Informationen aus Tabellen durchzuführen. Nutzung von Zugangsdaten Um eine erfolgreiche Verbindung herzustellen, benötigt DBUtil bestimmte Zugangsdaten. Diese umfassen:

- Die Datenbank-URL, die angibt, wo sich die MariaDB-Instanz befindet. Diese URL beinhaltet Angaben wie den Hostnamen, den Port und den Namen der Datenbank.

- Der Benutzername, den man für die Authentifizierung bei der Datenbank braucht.
- Das Passwort, das als Sicherheitsvorkehrung dient, um den Zugang nur für autorisierte Nutzer zu ermöglichen. Um Sicherheitsrisiken zu minimieren und die Wartung zu erleichtern, werden diese Zugangsdaten aus einer Konfigurationsdatei oder einer Umgebungsvariable geladen.

3.6 Effizienz und Stabilität der Verbindung

Die DBUtil-Klasse gewährleistet nicht nur die Verbindung zur Datenbank, sondern auch deren Effizienz und Stabilität. Hierfür können verschiedene Optimierungsmethoden verwendet werden, darunter:

- Verbindungs-Pooling, um es zuzulassen, dass Verbindungen erneut verwendet werden, und um die Leistung zu verbessern.
- Fehlermanagement, um Verbindungsprobleme rechtzeitig zu identifizieren und passende Schritte einzuleiten.
- Sichere Speicherung von Zugangsdaten, um zu vermeiden, dass Unbefugte Zugriff auf die Datenbank erhalten.

Unser Fazit ist, dass in vielen Anwendungen die DBUtil-Klasse das technische Rückgrat für die Kommunikation mit der Datenbank darstellt. Sie sorgt für eine sichere und effiziente Verbindung zur MariaDB-Datenbank, indem sie Zugangsdaten verwaltet, Verbindungen optimiert und Stabilität garantiert. Eine funktionierende Datenverwaltung in modernen Anwendungen wäre ohne eine zuverlässige Verbindung zur Datenbank kaum realisierbar. Diese Mechanismen garantieren die Stabilität, Sicherheit und Leistungsfähigkeit des Systems. [2]

4 ShortenServlet

Innerhalb eines URL-Verkürzungsdienstes stellt das ShortenServlet eine grundlegende Komponente dar. Es hat die Funktion, Anfragen zur URL-Verkürzung zu empfangen, die eingegebenen Daten auf ihre Validität zu prüfen, den Verkürzungsprozess auszuführen und die erzeugte Kurz-URL als Antwort zurückzusenden.

4.1 Empfangen und Prüfen von Anfragen

Wenn eine Anfrage beim ShortenServlet eintrifft, kontrolliert es zuerst, ob eine gültige URL übermittelt wurde. Um zu garantieren, dass nur funktionierende Webadressen verarbeitet werden und Fehler vermieden werden, ist diese Validierung entscheidend. Das System erkennt ungültige oder fehlerhaft eingegebene URLs und zeigt dem Nutzer eine entsprechende Fehlermeldung, damit dieser eine korrekte URL eingeben kann.

4.2 Verarbeitung durch die LinkShortener-Klasse

Das ShortenServlet übergibt die eingegebene URL nach erfolgreicher Überprüfung an die LinkShortener-Klasse, welche für die Generierung eines kurzen Codes (shortCode) verantwortlich ist. Dieser Code fungiert als eindeutiger Identifikator für die originale URL. Danach werden die Original-URL sowie der generierte shortCode in einer MariaDB-Datenbank abgelegt. Dank der Datenbank kann eine dauerhafte Verbindung zwischen Kurz- und Original-URL hergestellt werden, was eine zuverlässige Bearbeitung späterer Anfragen ermöglicht. Rückgabe der verkürzten URL Nach Vollzug des Verkürzungsprozesses gibt das ShortenServlet die verkürzte URL als Antwort an den Nutzer aus. Diese Kurz-URL kann in E-Mails, Nachrichten oder sozialen Medien geteilt werden, um den Zugriff auf lange und unübersichtliche Webadressen zu erleichtern. Der shortCode, der in der Datenbank gespeichert ist, ermöglicht es dem System jederzeit, die ursprüngliche URL abzurufen und den Nutzer entsprechend weiterzuleiten. Letzten Endes ist das ShortenServlet für den gesamten Prozess der URL-Verkürzung von zentraler Bedeutung, da es die Eingaben validiert und mit der Datenbank kommuniziert. Es gewährleistet außerdem, dass jede übermittelte URL schnell verkürzt und sicher gespeichert wird, indem es eng mit der LinkShortener-Klasse zusammenarbeitet. Das ShortenServlet leistet somit einen erheblichen Beitrag zur Usability und Funktionsfähigkeit des Gesamtsystems. [3]

4.3 RedirectServlet

Das RedirectServlet stellt eine essentielle Komponente in einem URL-Verkürzungsdienst dar. Es garantiert die zuverlässige Weiterleitung von Nutzern, die eine verkürzte URL aufrufen, zur ursprünglichen, vollständigen URL. Es extrahiert dazu den ShortCode aus der Anfrage, sucht in der MariaDB-Datenbank die entsprechende Original-URL und führt dann die Weiterleitung durch. Das RedirectServlet übernimmt drei wichtige Aspekte:

- Extrahierung des ShortCodes aus der Anfrage: Ruft ein Nutzer eine verkürzte URL auf (z. B. https://short.ly/abc123), so wird der ShortCode (abc123) aus dieser URL extrahiert. Dieser Code fungiert als eindeutiger Identifikator für die Original-URL, die in der Datenbank gespeichert ist.

- Suche der Original-URL in der MariaDB-Datenbank: Nach der Ermittlung des Short-Codes aus der Anfrage schickt das RedirectServlet eine SQL-Abfrage an die Datenbank, um die dazugehörige Original-URL abzurufen.
- Benutzerweiterleitung zur Original-URL: Bei Auffinden einer gültigen Original-URL verwendet das Servlet einen HTTP-Redirect (Statuscode 301 oder 302), um den Benutzer auf die ursprüngliche Webadresse weiterzuleiten.

Wird die URL nicht gefunden oder tritt ein Fehler auf, so gibt das System eine Fehlermeldung (HTTP 404 – Not Found) zurück. Vorteile und Optimierungsmöglichkeiten Die Nutzung eines RedirectServlets bietet zahlreiche Vorteile:

- Effizienz und Geschwindigkeit: Die Weiterleitung erfolgt in Millisekunden, da pro Anforderung lediglich eine Datenbankabfrage durchgeführt wird.
- SEO-Vorteile durch Statuscode 301: Ein permanenter Redirect (301) sorgt dafür, dass Suchmaschinen die Kurz-URL als Alias der Original-URL ansehen und den PageRank weiterleiten.
- Fehlertoleranz: Sollte ein ShortCode nicht vorhanden sein oder die Datenbank nicht erreichbar, bekommt der Nutzer eine aufklärende Fehlermeldung.
- Erweiterbarkeit: Das System lässt sich um Analysefunktionen ergänzen, die es ermöglichen, Klicks auf Kurz-URLs zu zählen und nachzuvollziehen.

Zusammenfassend ist zu erläutern, dass RedirectServlet die Verbindung zwischen einer verkürzten URL und der eigentlich ursprünglichen Zieladresse herstellt. Es stellt sicher, dass Nutzer durch Extraktion des ShortCodes aus der Anfrage, Durchführung einer effizienten Datenbankabfrage und zuverlässige HTTP-Weiterleitung immer die korrekte Webseite erreichen. Es ist ein essentielles Element aller URL-Shortener und garantiert eine reibungslose Verwendung. [1]

4.4 HTML-Seite für einen URL-Verkürzungsdienst

Eine einfache und benutzerfreundliche Oberfläche ist für einen URL-Verkürzungsdienst erforderlich, damit Nutzer lange URLs eingeben und in eine kürzere Form umwandeln können. Diese HTML-Seite fungiert als zentrale Benutzeroberfläche für den Dienst und ermöglicht auf einfache Weise das Verkürzen von URLs sowie das Abrufen der erzeugten Kurz-URL.

4.5 Aufbau und Funktionalität der HTML-Seite

Die HTML-Seite umfasst mehrere wesentliche Komponenten, die eine einwandfreie Verwendung des URL-Shorteners gewährleisten:

- Feld für die lange URL: Hier können Nutzer ihre URL im Long-Format angeben, die sie kürzen wollen. Das Feld für die Eingabe
- Sorgt dafür, dass ausschließlich gültige URLs bearbeitet werden, indem es beispielsweise ein Webadressmuster vorschreibt.
- Button zur URL-Verkürzung: Nachdem die URL eingegeben wurde, kann der Nutzer auf den „Verkürzen“-Button klicken. Dies sendet die URL an den Server, wo sie bearbeitet wird und ein ShortCode erstellt wird.
- Präsentation der verkürzten URL: Sobald die URL erfolgreich verkürzt wurde, wird sie dem Nutzer präsentiert. Die URL mit verkürzter Form kann entweder angeklickt oder direkt kopiert werden.
- Zusammenfassung: Die HTML-Seite des URL-Verkürzungsdienstes stellt eine unkomplizierte und selbsterklärende Option zur Verfügung, um lange URLs zu kompakten Versionen zu konvertieren. Mit HTML, CSS und JavaScript entsteht eine interaktive Webanwendung, die einfach zu erweitern und anzupassen ist. Sie ist die Schnittstelle zwischen Server und Nutzer und sorgt dafür, dass die URL-Verkürzung effizient und schnell erfolgt.

[4] [1]

4.6 PHP und Apache

Mit einem URL-Verkürzungsdienst, der auf PHP und Apache basiert, können lange URLs einfach und effizient in kurze und vor allem einprägsame Links umgewandelt werden. Während PHP die Backend-Logik zur Erstellung und Verwaltung der Kurz-URLs übernimmt, fungiert Apache als Webserver, der die Anfragen bearbeitet und weiterleitet.

4.7 PHP und Apache in einem URL-Verkürzungsdienst

PHP und Apache als Ganzes, was als URL-Verkürzungsdienst fungiert, spielt sowohl im Back- als auch im Frontend eine wichtige Rolle.

Backend:
Apache bearbeitet eingehende Anfragen, und leitet diese anschließend an das PHP-Backend weiter. Kurz-URLs werden mit `mod_rewrite` in der .htaccess-Datei auf eine zentrale index.php umgeleitet, die die entsprechende Lang-URL aus der Datenbank abruft und eine HTTP-302-Weiterleitung vornimmt. Auch kann die Performance von Apache durch `mod_cache` oder einen Redis-Cache verbessert werden.

Frontend:
Apache fungiert im Frontend als Webserver und liefert statische Inhalte wie HTML, CSS, JavaScript und Bilder aus. Sollte der URL-Verkürzungsdienst eine Verwaltungsoberfläche für Kurzlinks anbieten, kann Apache diese ebenfalls bereitstellen. Darüber hinaus gewährleistet die Aktivierung von HTTPS (SSL/TLS) mittels `mod_ssl` eine sichere Kommunikation. PHP und Apache arbeiten zusammen, um dynamische Webanwendungen wie einen URL-Verkürzungsdienst bereitzustellen.

4.8 Funktionen von PHP und Apache

Apache als Webserver:

- Nimmt HTTP-Anfragen von Nutzern entgegen und übermittelt sie weiter.
- Verwendet `mod_rewrite`, um Kurz-URLs auf eine zentrale PHP-Datei umzuleiten.
- Bietet statische Inhalte wie HTML, CSS und JavaScript an.
- Ermöglicht eine sichere HTTPS-Verbindung (SSL/TLS) mithilfe von `mod_ssl`.

PHP als Backend-Skriptsprache:

- Bearbeitet die Anfragen, wie das Erstellen und Abrufen von Kurz-URLs.
- Interagiert mit der Datenbank (wie z. B. MariaDB), um URLs zu speichern und abzurufen.
- Setzt HTTP-Weiterleitungen um (z. B. 302-Redirects für Kurzlinks).
- Kann Caching-Methoden wie Redis verwenden, um die Performance zu optimieren.
- Apache und PHP bieten zusammen eine effiziente und skalierbare Verarbeitung von Webanfragen – vom URL-Handling bis zur Bereitstellung von Webseiten.

4.9 Vor- und Nachteile von PHP und Apache

4.9.1 Vorteile von PHP:

- Leichte Implementierung: Verständlich und häufig genutzt in der Webentwicklung.
- Dynamische Inhalte: Bietet die Möglichkeit zur serverseitigen Verarbeitung und zur Interaktion mit Datenbanken.
- Umfangreiche Community und diverse Frameworks: Hilfe von vielen Entwicklern sowie Frameworks wie Laravel oder CodeIgniter.

- Unabhängig von der Plattform: Funktioniert auf unterschiedlichen Betriebssystemen (Linux, Windows, macOS).

4.9.2 Vorteile von Apache:

- Robuste und erprobte Technologie: Seit vielen Jahren als verlässlicher Webserver in Gebrauch.
- Flexibel erweiterbar: Bietet Unterstützung für Module wie `mod_rewrite` zur URL-Umschreibung oder `mod_ssl` für HTTPS.
- Einfache Konfiguration: Anpassbar über .htaccess-Dateien.
- Open Source kostenlos: breite Unterstützung, sowie keine Lizenzkosten.

4.9.3 Nachteile von PHP:

- Performance-Limitierungen: PHP kann im Vergleich zu modernen Sprachen wie Node.js oder Go bei hoher Last langsamer sein.
- Sicherheitsrisiken: Code-Implementierungen, die nicht sicher sind, können Schwachstellen wie SQL-Injection oder XSS verursachen.
- Kein echtes Multithreading: Die Verarbeitung geschieht synchron, was bei zahlreichen gleichzeitigen Anfragen zu Engpässen führen kann.

4.9.4 Nachteile von Apache:

- Speicherverbrauch: Bei hoher Last kann es mehr RAM benötigen als schlankere Optionen wie Nginx.
- Geringere Effizienz bei zahlreichen gleichzeitigen Verbindungen: Greift auf einen prozessbasierten Ansatz zurück, der nicht die Leistungsfähigkeit von eventbasierten Webservern besitzt.
- Komplexere Skalierung: Notwendig ist ein zusätzlicher Aufwand für die Lastverteilung (z. B. mit HAProxy oder Load Balancing).

So schlussfolgern wir, dass PHP und Apache eine perfekte Kombination für Webanwendungen darstellen, da die Kombination aus beiden leicht zu konfigurieren ist und dem Nutzer umfassende Unterstützung bietet. MariaDB: MariaDB ist eine effektive relationale Datenbank, die sich perfekt für die Speicherung strukturierter Daten eignet. MariaDB erfüllt in einem URL-Verkürzungsdienst eine zentrale Funktion: Sie sorgt dafür, dass die Verbindung zwischen kurzen und langen URLs gespeichert wird und Benutzer verlässlich auf ihre ursprünglichen Links weitergeleitet werden können. Die Rolle von MariaDB in einem URL-Verkürzungsdienst zeigt, dass diese aus mehreren Komponenten besteht, darunter Rolle von MariaDB in einem URL-Verkürzungsdienst Ein URL-Verkürzungsdienst besteht aus mehreren Komponenten, darunter das Frontend (die Nutzeroberfläche) sowie das Backend, welches die Anfragen bearbeitet. MariaDB ist in diesem System zentral, da es die Datenbank für die Speicherung und Verwaltung der URL-Zuordnungen darstellt.

4.10 Speicherung der URLs:

- Bei jeder Eingabe einer langen URL durch einen Nutzer erstellt das System einen Short-Code.
- In einer Tabelle der MariaDB-Datenbank werden diese Daten abgelegt.
- Abruf der ursprünglichen URL:
  - Bei einem Aufruf einer verkürzten URL durch einen Nutzer sucht das System den ShortCode in der Datenbank.
  - Bei einer gefundenen Übereinstimmung gibt MariaDB die lange URL zurück, um den Benutzer dorthin weiterzuleiten.
- Zusätzliche Metadaten-Speicherung:
  - Neben der URL selbst speichert MariaDB weitere Daten ab, wie:
  - Das Erstellungsdatum der URL
  - Das Ablaufdatum im Falle einer gewünschten Löschung
  - Anzahl der Aufrufe für eine gewünschte statischen Analyse

Die Vorteile von MariaDB sind für strukturierte Daten gut geeignet, da diese als relationale Datenbanken optimal für effiziente Abfragen mit SQL, sind. Hinzu kommt die zuverlässige Speicherung, da MariaDB die Transaktionssicherheit sicher darstellt und somit gewährleistet, dass die Daten dauerhaft gespeichert bleiben. Die Nachteile von Maria DB sind, dass bei hohem Traffic Leistungseinbußen auftreten können, besonders dann, wenn vermehrte Abfragen die Datenbank belasten. Fehlen Optimierungsmaßnahmen wie Indexierung oder Caching, kann dies zu einer merklichen Verlangsamung führen. Damit die Performance konstant hoch bleibt,

sind manuelle Optimierungen nötig. Hierzu zählen das präzise Tuning der Datenbank sowie das Nutzen von Load-Balancing-Methoden, um die Belastung effizient zu verteilen und Engpässe zu verhindern. Welches uns zum Fazit bringt, dass MariaDB eine effiziente und verlässliche Methode zur Ablage von Metadaten verkürzter URLs sowie diesen selbst zur Verfügung darstellt. Sie ermöglicht somit eine rasche Datenablage und -wiederherstellung dank ihrer leistungsstarken Abfragefunktionen. Ohne die nötigen Optimierungen kann es jedoch bei hoher Last zu Leistungseinbußen kommen. [5]

5 Redis

Redis (Remote Dictionary Server) ist ein leistungsfähiger In-Memory-Datenspeicher, der als Cache zur Verbesserung der Performance in einem URL-Verkürzungsdienst eingesetzt werden kann. Da Redis die Daten direkt im Arbeitsspeicher speichert, ermöglicht es extrem schnelle Lese- und Schreibvorgänge. Dies macht es besonders nützlich für häufig wiederholte Datenbankabfragen, wie sie bei einem URL-Shortener vorkommen. Rolle von Redis in einem URL-Verkürzungsdienst Redis erfüllt in einem URL-Verkürzungssystem verschiedene Funktionen, darunter die eines Caches, während andere Komponenten eine Datenbank (wie MariaDB) und ein Backend (wie Java-Servlets) sind. Redis dient hier dazu, Daten, die häufig abgerufen werden, temporär zu speichern. Dadurch wird die Geschwindigkeit der Abfrage gesteigert und die Datenbank entlastet.

Durch Caching in Redis verbessert das System den Zugriff auf verkürzte URLs. Zunächst wird bei einer Anfrage überprüft, ob die Original-URL bereits im Cache vorhanden ist. Tritt dieser Fall ein, so wird die Rückgabe direkt aus Redis vorgenommen, ohne dass eine Datenbankabfrage erfolgt. Ist die URL nicht im Cache vorhanden, wird sie aus der MariaDB-Datenbank abgerufen und in Redis gespeichert, um weitere zukünftige Anfragen zu beschleunigen. Die Verwendung von Redis verringert die Last auf der Datenbank erheblich, da oft angeforderte Links direkt aus dem Cache bereitgestellt werden. Dadurch wird die Zahl der Datenbankabfragen verringert, was vor allem bei hohem Traffic und in Lastspitzen zu einer besseren Systemperformance führt. Dank Redis als In-Memory-Cache erfolgen Datenabrufe direkt aus dem RAM, was dem Benutzer deutlich schnellere Antwortzeiten ermöglicht. Dies führt im Vergleich zu herkömmlichen Datenbankabfragen zu einer deutlich besseren Performance.

Kurz erklärt, bedeutet es, dass Redis rasche Antworten liefert und auch die Belastung der Datenbank verringert, bringt jedoch zusätzlichen Aufwand für die Verwaltung mit sich und speichert Daten lediglich vorübergehend im RAM.

Durch den gezielten Einsatz von TTL, LRU-Cache-Verwaltung und Replikation kann Redis jedoch optimal für hohe Skalierbarkeit und schnelle Antwortzeiten in einem URL-Verkürzungssystem genutzt werden. [6]

6 HAProxy

HAProxy (High Availability Proxy) ist eine Open-Source-Software, die als Load Balancer und Reverse Proxy den Datenverkehr effizient verteilt. In einer modernen Webarchitektur verbessert HAProxy die Skalierbarkeit und Verfügbarkeit, indem es zwischen Clients und mehreren Server-Instanzen vermittelt. In einem URL-Shortener-System mit mehreren Apache-Instanzen übernimmt HAProxy die Lastverteilung der HTTP-Anfragen. Dadurch wird sichergestellt, dass die Server gleichmäßig ausgelastet sind und kein einzelner Server überlastet wird, insbesondere bei hohem Traffic. Im weiteren wird erläutert, welche Funktionen HAProxy mit sich bringt, wie u.a. das Load Balancing zwischen den vielen Apache-Instanzen, wie:

- HAProxy leitet eingehende Anfragen auf mehrere Apache-Server weiter, die das Backend für die URL-Verkürzung bereitstellen. Dadurch wird die Ausfallsicherheit und Performance verbessert, da keine einzelne Instanz überlastet wird.
- HAProxy dient als Reverse Proxy und nimmt alle Nutzeranfragen entgegen, um sie an die Backend-Server weiterzuleiten. Dadurch bleibt die interne Infrastruktur geschützt, und die Server-Kommunikation wird effizienter.
- HAProxy gewährleistet Hochverfügbarkeit und Ausfallsicherheit, indem es bei einem Ausfall einer Apache-Instanz den Datenverkehr automatisch auf die verbleibenden Server umleitet und somit Betriebsausfälle minimiert.

HAProxy bietet eine effiziente Lastverteilung, indem es Anfragen auf mehrere Apache-Instanzen verteilt und so eine Überlastung einzelner Server verhindert. Zudem sorgt es für hohe Verfügbarkeit, da ausgefallene Server automatisch erkannt und der Datenverkehr auf funktionierende Instanzen umgeleitet wird. Dank der einfachen horizontalen Skalierung können bei steigenden Anforderungen problemlos weitere Apache-Instanzen hinzugefügt werden. Die Nachteile hingegen sind, das mit dem Einsatz von HAProxy ein zusätzlicher Konfigurationsaufwand verbunden ist, da es individuell eingerichtet und gewartet werden muss. Bei kleineren Systemen, die nur eine Apache-Instanz nutzen, ist es oft nicht notwendig und bringt lediglich einen begrenzten Nutzen. Außerdem braucht HAProxy Server-Ressourcen für sich selbst, wodurch die Komplexität des Systems zunimmt. Summiert man die Vorteile und auch die Nachteile, so ergibt es, dass HAProxy ein leistungsstarker Load Balancer und Reverse Proxy ist, es ermöglicht eine effiziente Lastverteilung, verbesserte Performance und hohe Verfügbarkeit in einer skalierbaren URL-Shortener-Architektur. Bei geringem Traffic und nur einem Apache-Server ist es oft nicht erforderlich, aber bei höherer Last und mehreren Servern wird es zu einer wichtigen Komponente. HAProxy benötigt jedoch eine zusätzliche Konfiguration und Wartung, weshalb seine Verwendung je nach Systemanforderungen sorgfältig abgewogen werden sollte. [7]

Hauptablauf der URL-Verkürzung Ein URL-Shortener-Dienst hat die URL-Verkürzung als entscheidenden Prozess. Dies beinhaltet die Umwandlung einer langen URL in eine kurze, leicht zu teilende Adresse. Dadurch wird das Linkteilen einfacher und die Usability verbessert.

7 Ablauf der URL-Verkürzung

- Eingabe der URL durch den Nutzer:
  - Der Nutzer gibt eine lange URL in das Web-Interface des Dienstes ein.
- Generierung eines einzigartigen Shortcodes:
  - Das Backend erstellt einen eindeutigen Shortcode für die eingegebene URL.
  - Dies kann durch Algorithmen wie UUID oder eine hashbasierte Methode erfolgen und dient als Identifier
- Speicherung in der MariaDB-Datenbank:
  - Die generierte Kurz-URL und die zugehörige Original-URL werden in einer relationalen MariaDB-Datenbank gespeichert. Zusätzlich können sowohl Erstellungsdatum oder gar ein Ablaufdatum hinzugefügt und gespeichert werden.
- 4. Bereitstellung der Kurz-URL für den Nutzer:
  - Diese URL kann von nun an überall geteilt werden, z. B. in Socialmedia, E-Mails, oder auch Nachrichten.

7.1 Vor- und Nachteile der URL-Verkürzung

Mit Java Servlets kann URL-Shortening einfach umgesetzt werden und die Shortcodes sind durch UUIDs eindeutig. Auch die Speicherung in einer relationalen Datenbank gewährleistet eine sichere Datenverwaltung. Jede Anfrage benötigt jedoch eine Verbindung zur Datenbank. Bei hohem Traffic kann dies zu Skalierungsproblemen führen. Fehlt das Caching, könnte sich die Performance bei zahlreichen Anfragen noch weiter verschlechtern. Sowohl die Vor- als auch die Nachteile zeigen auf, dass mit Java Servlets und MariaDB die URL-Verkürzung stabil und sicher realisiert werden kann. Dabei erfolgt die Datenspeicherung strukturiert und die Verarbeitung der Anfragen gestaltet sich effizient. Eine skalierbare Architektur entsteht durch klare Shortcodes und eine API. Es sollten jedoch Optimierungen wie Caching mit Redis oder Load Balancing durch HAProxy in Betracht gezogen werden, um Performance-Probleme bei hoher Last zu vermeiden.

7.2 URL-Auflösung Weiterleitung

Ein URL-Shortener-Service umfasst unbedingt die Weiterleitung und Auflösung von URLs. Durch diesen Prozess wird gewährleistet, dass Nutzer beim Aufruf einer Kurz-URL zur ursprünglichen Lang-URL weitergeleitet werden. Hierbei kommen Caching-Techniken zum Einsatz, um die Leistung zu verbessern und Anfragen effizient zu bearbeiten.

Ablauf der URL-Auflösung Weiterleitung

- Aufruf der Kurz-URL durch den Nutzer:
  - Beim Aufruf einer kurzen URL sucht das Backend die zugehörige lange URL zuerst im Redis-Cache, alternativ in MariaDB, leitet bei Erfolg per HTTP 302 weiter oder gibt bei fehlender Zuordnung eine 404-Fehlermeldung aus.
- Prüfung im Redis-Cache:
  - Das System prüft zunächst, ob die lange URL im Redis-Cache vorhanden ist, und leitet sie bei einem Treffer direkt weiter, ohne die Datenbank zu belasten.
- Suche in der MariaDB-Datenbank (falls nicht im Cache):
  - Wird die URL nicht im Redis-Cache gefunden, erfolgt eine Suche in der MariaDB-Datenbank. Die gefundene lange URL wird zurückgegeben und gleichzeitig im Redis-Cache gespeichert, um zukünftige Anfragen schneller zu verarbeiten.
- Automatische Weiterleitung (HTTP 302):
  - Bei einer Übereinstimmung erfolgt eine HTTP-302-Weiterleitung, sodass der Nutzer direkt zur ursprünglichen URL gelangt.
- Fehlermeldung bei nicht vorhandener URL:
  - Existiert keine passende Zuordnung, gibt das System eine HTTP-404-Fehlermeldung aus, die darauf hinweist, dass die Kurz-URL nicht existiert oder abgelaufen ist.

8 Datenbank MariaDB

MariaDB wird als Datenbank genutzt, um sowohl die Zuordnung zwischen Kurz-URLs als auch Original-URLs zu gewährleisten und zu speichern. Folglich werden URLs aufgelistet mit denen wir die meiste Zeit gearbeitet haben. Des Weiteren werden sowohl die Vor- als auch die Nachteile hervorgehoben.

- Tabelle urls:

* Felder:
* id: Primärschlüssel, automatisch inkrementiert.
* shortcode: Eindeutiger Kurzcode (VARCHAR).
* `long_url: Die ursprüngliche lange URL (TEXT).`
* `created_at: Zeitpunkt der Erstellung (TIMESTAMP).`
* `expires_at: Ablaufdatum (TIMESTAMP, standardmäßig 24 Stunden` nach Erstellung. Veraltete Links werden automatisch entfernt).

Das System ist durch eine einfache und klare Struktur gekennzeichnet und bietet eine automatische Ablauffunktion, die nach 24 Stunden veraltete URLs entfernt. Es kann zudem flexibel erweitert werden, zum Beispiel durch die Einbindung eines Zählers zur Erfassung der Nutzungshäufigkeit. Bei hohem Traffic können jedoch eine Zunahme von Datenbankabfragen und damit einhergehend Leistungseinbußen auftreten. Fehlt eine optimierte Indexierung der shortcode-Spalte, kann die Performance bei großen Datenmengen weiter beeinträchtigt werden.

8.1 Erweiterungen und Optimierungen

Ablauf nach 24 Stunden:

* URLs werden nach 24 Stunden autmoatischgelöscht, um Speicherplatz zu sparen.
* Das Backend prüft bei jeder Anfrage, ob die URL noch gültig ist.
* Falls die URL abgelaufen ist, wird eine Fehlermeldung (HTTP 404) zurückgegeben.

8.2 Counter für Nutzungshäufigkeit (Optional):

* `Eine zusätzliche Spalte usage_count könnte hinzugefügt werden,` um zu erfassen, wie oft eine Kurz-URL aufgerufen wurde.
* Dies wäre nützlich für Statistiken, ist aber für die grundlegende Funktionalität nicht zwingend erforderlich.

Als Fazit wissen wir, dass die Architektur eine stabile und skalierbare Basis für einen effizienten URL-Shortener darstellt. MariaDB sorgt hier für eine verlässliche Speicherung, während Redis durch Caching eine Verbesserung der Performance gewährt. Obwohl HAProxy für die zukünftige Skalierbarkeit sorgen kann, ist diese nicht zwingend notwendig. Die automatische Ablauffunktion, die nach 24 Stunden aktiviert wird, ermöglicht eine einfache Datenbereinigung und sorgt dafür, dass das System somit schnell und flexibel erweiterbar bleibt.

9 Lasttests und Skalierbarkeit

Im Rahmen der Leistungs- und Stabilitätsprüfung der URL-Shortener-Anwendung wurden zwei verschiedene Lasttest-Ansätze gewählt: k6 und Artillery. Beide Werkzeuge ermöglichen die Simulation von realistischen Lastszenarien und helfen dabei, die Skalierbarkeit und Belastbarkeit des Systems zu analysieren. Die Wahl dieser Tools wurde basierend auf den spezifischen Anforderungen der Anwendung getroffen, insbesondere hinsichtlich der Art der erwarteten Lastspitzen und des allgemeinen Benutzerverhaltens.

9.1 Einführung in Lasttests und deren Bedeutung

Lasttests sind eine essentielle Methode der Performance-Analyse in der Softwareentwicklung und dienen dazu, ein System unter verschiedenen Lastbedingungen zu testen. Ziel ist es, Engpässe zu identifizieren, Antwortzeiten, Fehlerraten und Stabilität zu messen sowie sicherzustellen, dass die Anwendung auch unter hoher Last korrekt funktioniert.

Zu den wichtigsten Fragestellungen, die durch Lasttests beantwortet werden sollen, gehören:

* Wie verhält sich das System bei einer hohen Anzahl von gleichzeitigen Nutzern?
* Gibt es signifikante Verzögerungen in der Antwortzeit?
* Gibt es Speicher- oder CPU-Engpässe?
* Wie viele Anfragen pro Sekunde kann der Server verarbeiten, bevor er überlastet ist?

Da die URL-Shortener-Anwendung mit einer Vielzahl von kurzen API-Anfragen arbeitet (eine URL wird eingegeben und ein Shortlink generiert oder aufgelöst), ist ein Test der Reaktionsfähigkeit des Systems unter realistischen Bedingungen essenziell.

9.2 Warum wurden k6 und Artillery gewählt?

Es gibt viele verschiedene Lasttest-Tools, darunter Apache JMeter, Gatling, Locust, Siege und weitere. k6 und Artillery wurden jedoch aus folgenden Gründen bevorzugt:

* k6: Ein leistungsstarkes Lasttest-Tool, ideal für skalierbare Systeme.
* Artillery: Flexibel einsetzbar, besonders für Spike-Tests geeignet und optimiert für moderne Web-APIs.

Beide Tools bieten leichtgewichtige, aber leistungsfähige Skripting-Möglichkeiten und sind für die heutige Webentwicklung ausgelegt. Zudem ermöglichen sie eine detaillierte Auswertung der Testergebnisse, was für eine fundierte Analyse entscheidend ist.

9.3 k6 – Hochleistungsfähiges Lasttest-Tool für Stresstests

k6 ist ein modernes, entwicklerfreundliches Lasttest-Tool, das für die Analyse von Webanwendungen, APIs und Microservices konzipiert wurde. Es zeichnet sich durch eine hohe Performance, einfache Skripterstellung mit JavaScript und eine ressourcenschonende Architektur aus.

Hauptmerkmale von k6:

* Moderne Skriptsprache (JavaScript) → Keine komplizierte Konfiguration erforderlich
* Geringer Ressourcenverbrauch → Hohe Skalierbarkeit für große Tests
* CLI-Support → Tests lokal ausführbar
* Detaillierte Metriken und Statistiken → Gute Analyse- und Auswertungsmöglichkeiten

9.3.1 Warum ist k6 für den Stresstest geeignet?

Ein Stresstest hat das Ziel, die maximale Kapazität des Systems zu bestimmen. Dabei wird eine kontinuierlich steigende Last erzeugt, bis das System an seine Grenzen stößt. Dies hilft dabei, herauszufinden, wann:

* die Antwortzeiten kritisch ansteigen,
* erste Fehler auftreten,

* der Server nicht mehr reagiert.

9.3.2 Unser K6 Skript:

```
import http from 'k6/http';
import { check, sleep } from 'k6';

export let options = {
    stages: [
        { duration: '30s', target: 50 },
        { duration: '2m', target: 100 },
        { duration: '30s', target: 0 },
    ],
    thresholds: {
        http_req_failed: ['rate<0.01'],
        http_req_duration: ['p(95)<500'],
    },
};

export default function () {
    // 1. Kurzlink erstellen (POST)
    const longUrl = `https://example.com/${__VU}-${Date.now()}`;
    const postRes = http.post(
        'http://localhost:8080/shorten.php',
        `url=${encodeURIComponent(longUrl)}`,
        { headers: { 'Content-Type': 'application/x-www-form-urlencoded'
        ↪  } }
    );

    check(postRes, {
        'POST /shorten.php → 200 OK': (r) => r.status === 200,
        'Kurzlink generiert': (r) => r.body.includes('http://'),
    });

    // 2. Kurzlink aufrufen (GET)
    const shortCode = postRes.body.split('/').pop().trim();
    const getRes =
    ↪  http.get(`http://localhost:8080/redirect.php?code=${shortCode}`,
    {
        redirects: 0
```

```
    });

    check(getRes, {
        'GET /redirect.php → 302 Redirect': (r) => r.status === 302,
        'Redirect zur Original-URL': (r) => r.headers.Location ===
        ↪  longUrl,
    });

    sleep(1);
}
```

Das folgende K6-Skript führt eine Lasttestsimulation für einen URL-Kürzungsdienst durch. Es simuliert Nutzer, die Kurzlinks erstellen und anschließend aufrufen, um die Weiterleitung zu testen.

9.4 Testkonfiguration

- **Laststufen:**
  - **Stufe 1:** 30 Sekunden lang werden bis zu 50 virtuelle Nutzer hochgefahren.
  - **Stufe 2:** 2 Minuten lang steigt die Nutzeranzahl auf 100.
  - **Stufe 3:** 30 Sekunden lang wird die Last wieder auf 0 reduziert.
- **Schwellenwerte:**
  - `Die Fehlerrate von HTTP-Anfragen muss unter 1\% liegen: http_req_failed < 0.01.`
  - `Die 95. Perzentil-Latenz der Anfragen muss unter 500 ms bleiben: http_req_duration < 500 ms.`

9.5 Testablauf

Jede virtuelle Nutzerinstanz führt die folgenden Schritte aus:

1. **Erstellung eines Kurzlinks (POST-Anfrage)**
   - Eine zufällige lange URL wird in der Form `https://example.com/{{__VU}}-{Timestamp}` generiert.

   * Diese URL wird als `POST`-Anfrage an `http://localhost:8080/shorten.php` gesendet.
   * Die Anfrage enthält den Parameter `url` mit der langen URL, kodiert im `application/x-www-form-urlencoded`-Format.
   * Es werden folgende Prüfungen durchgeführt:
     · Statuscode der Antwort ist 200 (`OK`).
     · Die Antwort enthält eine generierte Kurz-URL.

2. **Abruf des Kurzlinks (GET-Anfrage)**
   * Der aus der Antwort extrahierte Kurzlink-Code wird für den Abruf verwendet.
   * Eine `GET`-Anfrage wird an `http://localhost:8080/redirect.php?code={shortCode}` gesendet.
   * Die Anfrage erlaubt keine Weiterleitungen (`redirects: 0`).
   * Es werden folgende Prüfungen durchgeführt:
     · Der Statuscode der Antwort ist 302 (`Found` – Weiterleitung).
     · Der `Location`-Header der Antwort verweist auf die ursprüngliche lange URL.

3. **Pause zwischen den Anfragen**
   * Nach jeder Iteration schläft der Nutzer für 1 Sekunde, bevor der nächste Testdurchlauf beginnt.

9.6 Artillery – Hochgradig konfigurierbares Tool für Spike-Tests

Während k6 für Stresstests optimal ist, bietet Artillery eine ausgezeichnete Möglichkeit, Spike-Tests durchzuführen.

9.6.1 Was ist ein Spike-Test?

Ein Spike-Test simuliert plötzliche, kurzfristige Lastspitzen – beispielsweise wenn eine URL plötzlich in sozialen Medien viral geht. Das System muss solche unerwarteten Lastschwankungen ohne Abstürze bewältigen.

9.6.2 Warum ist Artillery hier besonders gut?

* Event-Driven Architektur → Simuliert realistische Lastpeaks
* Einfache YAML-Skripte → Leichte Konfiguration
* Realistische Benutzerflüsse → Kombination aus GET- und POST-Anfragen

Hier ist unser Artillery Skript:

```yaml
config:
  target: "http://localhost:8080"
  phases:
    - duration: 10
      arrivalRate: 10
    - duration: 5
      arrivalRate: 500
scenarios:
  - flow:
      - post:
          url: "/index.php"
          form:
            longUrl: "https://example.com/{{ uuid() }}"
```

Das folgende Artillery-Skript beschreibt eine Lasttestsimulation für einen lokalen Server unter `http://localhost:8080`. Es besteht aus zwei Testphasen mit unterschiedlichen Lastanforderungen und sendet HTTP-POST-Anfragen an die URL `/index.php` mit einer zufällig generierten URL als Parameter.

9.7 Konfiguration

* **Ziel-Server:** `http://localhost:8080`
* **Phasen:**
  - **Phase 1:** Dauer von 10 Sekunden mit einer Ankunftsrate von 10 Anfragen pro Sekunde.
  - **Phase 2:** Dauer von 5 Sekunden mit einer stark erhöhten Ankunftsrate von 500 Anfragen pro Sekunde.

9.8 Szenario

Während des Tests wird ein HTTP-POST-Request an den Endpunkt `/index.php` gesendet. Die Anfrage enthält ein Formular mit dem folgenden Parameter:

* `longUrl`: Eine zufällige URL in der Form `https://example.com/{{ uuid() }}`, wobei `uuid()` eine zufällig generierte eindeutige Identifikationsnummer repräsentiert.

9.9 Fazit – Warum sind k6 und Artillery die beste Wahl?

Tool	Test-Typ	Warum es passt?
K6	Stresstest	Skalierbarkeit und Performance gut messbar
Artillery	Spike-Test	Simuliert plötzliche Lastspitzen realistisch

Die Kombination aus k6 und Artillery ermöglicht eine gute Bewertung der Systemleistung.

[8], [9], [10], [11]

9.10 Lasttest-Ergebnisse

Im nächsten Schritt präsentieren und erläutern wir die Ergebnisse des Lasttests.

9.10.1 Testergebnis: Artillery

Das erste Lasttestergebnis stammt aus einem Testlauf mit Artillery. Der Fokus lag darauf, das System plötzlichen Lastspitzen auszusetzen, um zu beobachten, wie es darauf reagiert. Dabei ging es insbesondere darum, mögliche Engpässe wie erhöhte Antwortzeiten oder Fehlerquellen zu erkennen.

Die untenstehende Abbildung 9.1 zeigt die Ergebnisse des Tests.

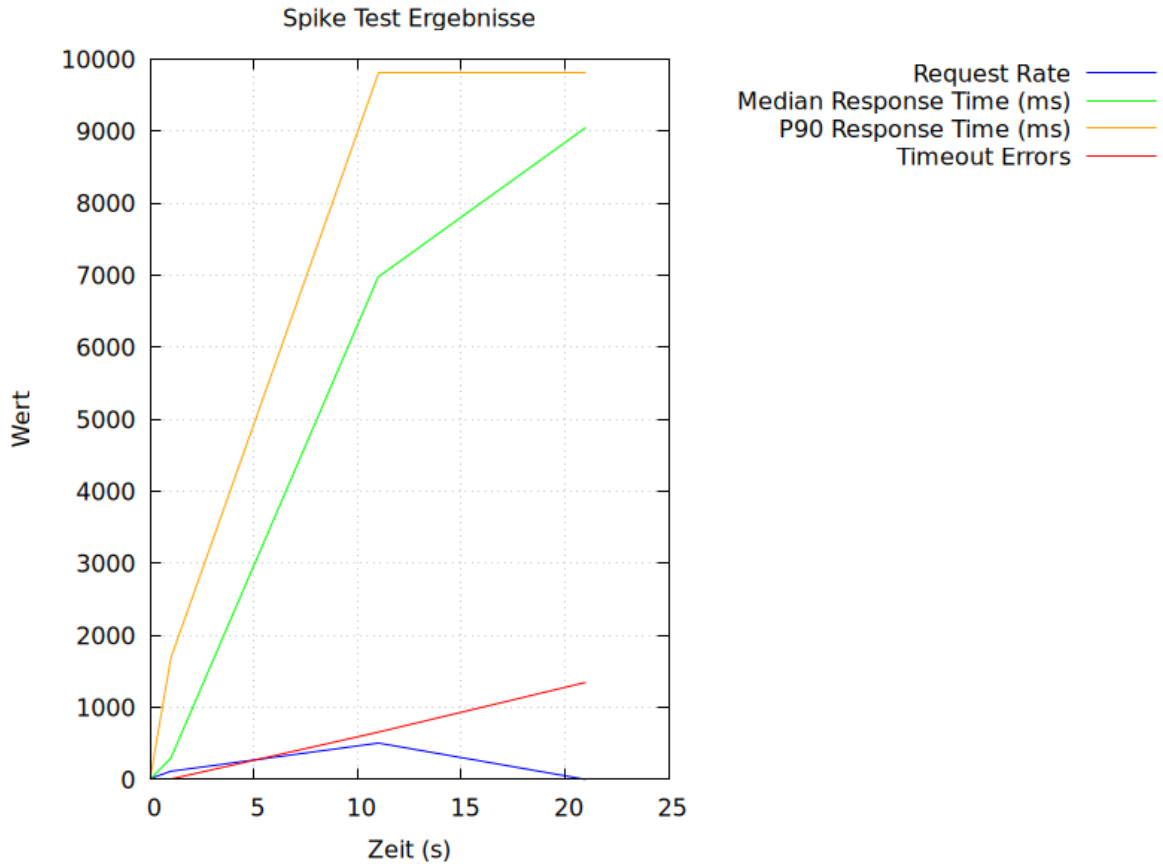

Abbildung 9.1: Ergebnisse des Artillery-Lasttests

Wie in Abbildung 9.1 zu sehen ist, wurden folgende Leistungskennzahlen gemessen:

* **Anforderungsrate (blau):** Die Anzahl der Anfragen pro Sekunde. Diese beginnt niedrig, erreicht einen Höhepunkt nach 5 Sekunden und fällt anschließend ab.
* **Median-Antwortzeit (grün):** Die mittlere Antwortzeit in Millisekunden. Diese steigt während des Tests kontinuierlich an.
* **90. Perzentil der Antwortzeiten (orange):** Diese Kennzahl gibt an, dass 90% der Anfragen in circa 10.000 ms beantwortet wurden. Die Antwortzeiten stabilisieren sich schnell auf diesem Wert.
* **Timeout-Fehler (rot):** Die Anzahl der Zeitüberschreitungsfehler steigt gleichmäßig über die gesamte Testdauer.

Der Artillery-Test zeigt deutlich, wie die Systemleistung unter extremer Last beeinträchtigt wird. Insbesondere die stark ansteigende Median- und P90-Antwortzeit sowie die Zunahme der Timeout-Fehler weisen auf mögliche Engpässe hin.

9.10.2 Analyse des K6-Testergebnis

Die durchgeführten Lasttests mit K6 haben umfangreiche Daten zur Leistung der getesteten Webanwendung geliefert. Die generierte Grafik stellt die Antwortzeiten der Anfragen über die Zeit hinweg dar. In diesem Abschnitt wird die Grafik analysiert und ihre Bedeutung für die Bewertung der Systemperformance herausgearbeitet.

Testaufbau und Testziel Der K6-Lasttest simulierte eine steigende Anzahl gleichzeitiger Benutzer, die eine Kurz-URL generieren und aufrufen. Ziel des Tests war es, die Skalierbarkeit und Stabilität der Anwendung unter Last zu untersuchen. Die wichtigsten Kennzahlen waren:

* Antwortzeiten (ms)
* Anzahl der Anfragen pro Sekunde
* Systemverhalten bei steigender Nutzerzahl

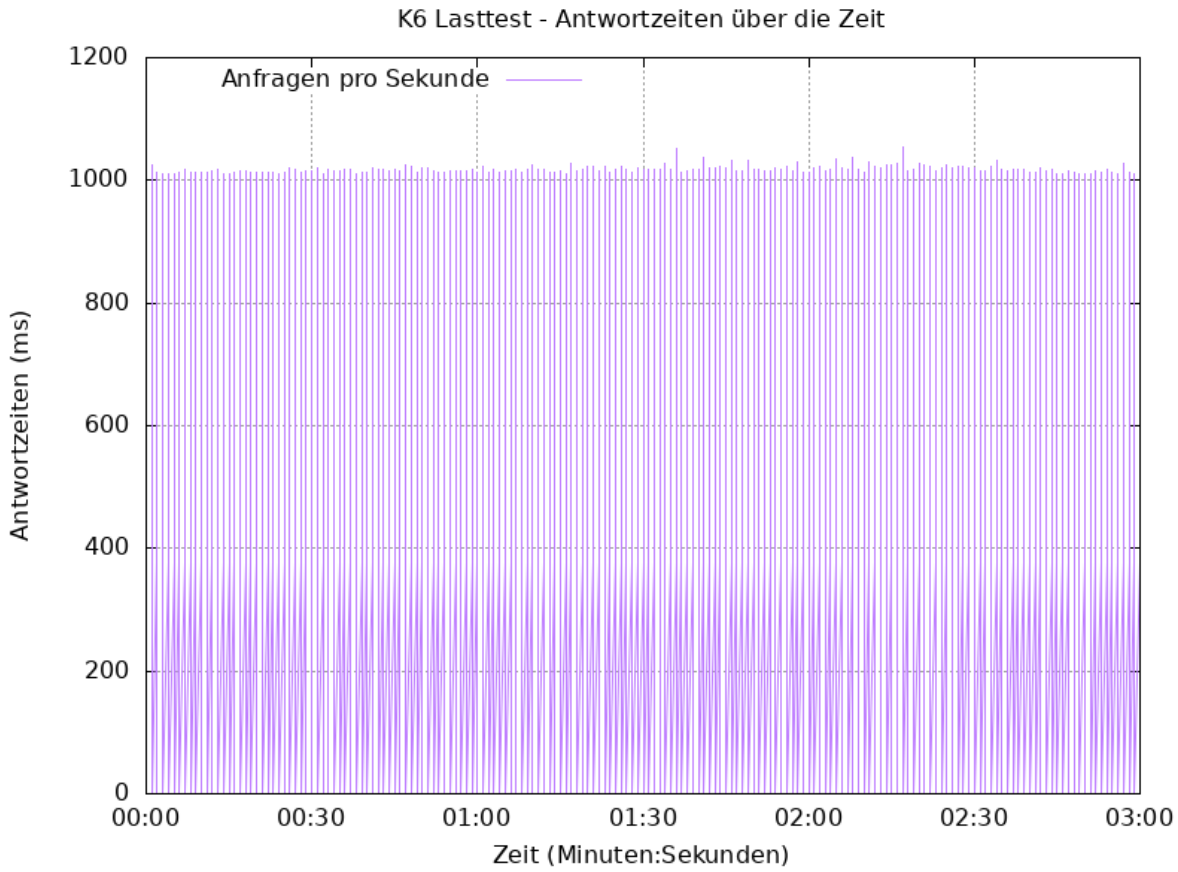

Abbildung 9.2: Ergebnisse des K6-Lasttests

Ergebnisse und Interpretation Die generierte Grafik zeigt die Antwortzeiten in Millisekunden über die Testdauer hinweg. Auffällig ist, dass die Antwortzeiten relativ stabil bleiben und keine signifikanten Ausreißer oder Latenzspitzen zu erkennen sind. Dies deutet darauf hin, dass die Anwendung eine gleichbleibend hohe Performance liefern kann, selbst unter Last.

* **Stabile Antwortzeiten:** Die Antwortzeiten bleiben konstant bei etwa 1000 ms im Durchschnitt, was eine zuverlässige Verarbeitung der Anfragen bedeutet.
* **Konstante Lastverteilung:** Die Anzahl der Anfragen pro Sekunde bleibt stabil, was zeigt, dass der Server die eingehenden Anfragen effizient verwaltet.
* **Keine Performance-Einbrüche:** Es treten keine spürbaren Verzögerungen oder Ausfälle auf, was bedeutet, dass die Webanwendung auch unter hoher Last stabil läuft.

9.10.3 Fazit - Lasttests

Der K6-Lasttest zeigt, dass die getestete Anwendung auch unter Last eine stabile und gleichbleibende Performance liefert. Es gab weder kritische Verzögerungen noch Systeminstabilitäten. Insgesamt zeigt sich das System als zuverlässig und gut skalierbar. Die aktuellen Serverantwortzeiten sind ausreichend, und keine weiteren Maßnahmen zur Optimierung sind erforderlich, solange keine Werte oberhalb der getesteten Schwellen erreicht werden.

9.11 Skalierbarkeit

Die durchgeführten Lasttests mit K6 und Artillery liefern wertvolle Erkenntnisse über die Skalierbarkeit des Systems unter verschiedenen Belastungsszenarien.

Das K6-Testergebnis zeigt eine konstante Anzahl von Anfragen über einen längeren Zeitraum, wobei die Antwortzeiten stabil bleiben. Dies deutet darauf hin, dass das System eine gleichmäßige Last effizient verarbeiten kann, ohne signifikante Leistungseinbrüche zu erleiden. Die geringe Streuung der Antwortzeiten spricht für eine stabile Skalierung innerhalb der getesteten Lastgrenzen. Sollte die Nutzerzahl jedoch weiter ansteigen, müssten zusätzliche Skalierungsmaßnahmen in Betracht gezogen werden.

Das Artillery-Testergebnis verdeutlicht die Reaktion des Systems auf plötzliche Lastspitzen. Es zeigt sich, dass die Antwortzeiten mit steigender Last signifikant zunehmen. Insbesondere die P90-Response-Zeit und die Timeout-Fehlerrate deuten darauf hin, dass das System ab einem bestimmten Punkt an seine Kapazitätsgrenzen stößt. Dies legt nahe, dass in Szenarien mit plötzlich ansteigendem Traffic Engpässe auftreten können, die zu einer Verschlechterung der Nutzererfahrung führen können.

Zusammenfassend lässt sich feststellen, dass das System unter konstanter Last stabil läuft. Bei stark schwankenden oder extrem hohen Lastspitzen können jedoch Herausforderungen auftreten. Solange jedoch die getesteten Werte nicht überschritten werden, funktioniert das System einwandfrei und zeigt keine signifikanten Probleme.

10 Kommunikation im Netzwerk

10.1 Netzwerk

Das Netzwerk besteht in unserem Projekt aus verschiedenen Docker-Containern. Sie haben alle die gleiche Grundstruktur, wie bereits beschrieben. Diese Container sind über ein gemeinsames virtuelles Netzwerk miteinander verbunden und können über eine bidirektionale TCP-Verbindung Daten austauschen. Dadurch ist eine direkte und effiziente Kommunikation zwischen den einzelnen Containern gewährleistet.

Eine der großen Stärken dieses Ansatzes ist die Skalierbarkeit. Theoretisch könnte das System so erweitert werden, dass nicht mehr einzelne Container die Verarbeitung übernehmen, sondern jeder Container als eigenständiger Server agiert. Diese Server müssten sich nicht zwangsläufig an einem einzigen Standort befinden, sondern könnten über verschiedene Rechenzentren oder sogar weltweit verteilt sein. Dies würde nicht nur die Ausfallsicherheit erhöhen, sondern auch die Latenzzeiten für Benutzer in verschiedenen Regionen reduzieren.

Wenn das Netzwerk über mehrere Standorte verteilt wird, müssen jedoch zusätzliche Maßnahmen getroffen werden. Eine der wichtigsten Herausforderungen ist die Sicherheit der Verbindungen. Während eine lokale Verbindung innerhalb eines Docker-Netzwerks relativ sicher ist, müssen Verbindungen zwischen Servern an verschiedenen Standorten besonders geschützt werden. Hier kommen Technologien wie VPNs (Virtual Private Networks), TLS-Verschlüsselung oder SSH-Tunnel zum Einsatz, um sicherzustellen, dass Daten nicht von Dritten abgefangen oder manipuliert werden können.

Ein weiterer wichtiger Aspekt ist die Netzwerklatenz. Damit der Datenaustausch zwischen den Servern möglichst schnell und effizient bleibt, müssen geeignete Protokolle und Mechanismen zur Optimierung der Kommunikation implementiert werden. Caching-Strategien, Load-Balancing-Mechanismen und intelligente Routing-Algorithmen können dabei helfen, die Performance des Systems zu verbessern.

Zudem wäre es sinnvoll, ein Monitoring-System zu integrieren, das die Netzwerkverbindungen kontinuierlich überwacht. Falls Engpässe oder Verbindungsprobleme auftreten, könnte ein solches System automatisch Gegenmaßnahmen einleiten, beispielsweise durch das Umleiten des Datenverkehrs über alternative Routen oder das automatische Neustarten fehlerhafter Dienste.

Schließlich spielt auch die Skalierung der Infrastruktur eine entscheidende Rolle. Während ein kleineres System mit wenigen Containern noch manuell verwaltet werden kann, ist dies bei einer größeren verteilten Architektur kaum noch möglich.

Hier können Container-Orchestrierungswerkzeuge wie Kubernetes oder Docker Swarm helfen, indem sie automatisch neue Instanzen starten, Lasten verteilen und Fehlerfälle selbstständig beheben.

Zusammenfassend lässt sich sagen, dass unser aktuelles Container-Netzwerk bereits eine solide Grundlage für eine skalierbare Architektur bietet. Durch gezielte Erweiterungen in den Bereichen Sicherheit, Performance und Verwaltung könnte es jedoch noch weiter optimiert werden, um auch in groß angelegten, verteilten Umgebungen zuverlässig und effizient zu funktionieren.

10.2 Kommunikation im Projekt

In unserem Projekt haben wir mehrere Container in unserer Netzwerkstruktur integriert, um eine effiziente und skalierbare Umgebung für die Verarbeitung der Anfragen zu gewährleisten. Ein zentraler Bestandteil dieser Architektur ist HAProxy, der als Load Balancer fungiert. Er verteilt die ankommenden Client-Verbindungen gleichmäßig auf mehrere Apache-Webserver, sodass die vorhandenen Ressourcen optimal genutzt werden und kein einzelner Server überlastet wird.

Die Apache-Server übernehmen die Verarbeitung der HTTP-Anfragen und greifen dabei auf den MariaDB-Server zu, der als zentrale Datenbank dient. Hier werden unter anderem die Informationen zu den verkürzten Links gespeichert und abgerufen. Wenn ein Benutzer einen verkürzten Link aufruft, durchläuft die Anfrage denselben Weg:

Der Client stellt eine Anfrage an HAProxy, der entscheidet, welcher Apache-Server die Anfrage bearbeiten soll. Der ausgewählte Apache-Server verarbeitet die Anfrage und holt die benötigten Daten aus der MariaDB-Datenbank. Falls der Kurzlink in der Datenbank vorhanden ist, erfolgt eine Weiterleitung zur Original-URL. Falls die Anfrage nicht gefunden wird, gibt der Server eine entsprechende Fehlermeldung aus. Durch den Einsatz von tcpdump kann die gesamte Kommunikation innerhalb des Netzwerks detailliert analysiert werden. So lassen sich die einzelnen Pakete und ihre Routen zwischen den Containern nachverfolgen. Besonders interessant ist dabei, wie HAProxy die Anfragen verteilt und wie die Antwortzeiten der Datenbank im Lastbetrieb aussehen.

Ein weiteres wichtiges Element in unserer Infrastruktur ist die Skalierbarkeit. Falls der Traffic zunimmt, können wir problemlos weitere Apache-Server in das System einbinden, die dann automatisch von HAProxy berücksichtigt werden. Gleichzeitig ist es möglich, eine Caching- wie Redis hinzuzufügen, um häufig genutzte Daten schneller bereitzustellen und die Datenbank zu entlasten.

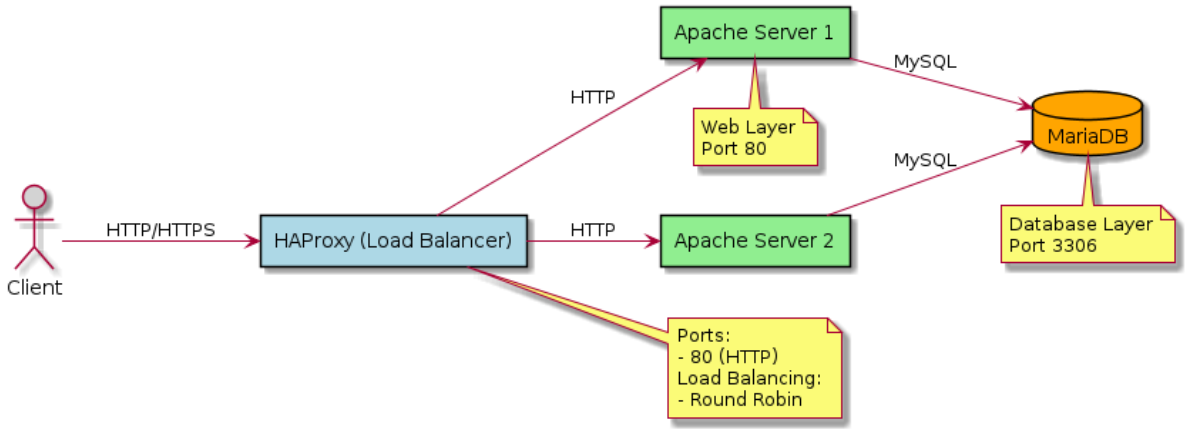

Abbildung 10.1: Veranschaulichung der Stuktur des Netzwerkes im Projekt

10.3 Zentrale Netzwerkprotokolle im System

In modernen Netzwerken ist eine zuverlässige und effiziente Kommunikation essenziell. Dies wird durch verschiedene Netzwerkprotokolle ermöglicht, die je nach Anwendungsfall unterschiedliche Aufgaben übernehmen. In einer typischen IT-Infrastruktur spielen vor allem vier Protokolle eine zentrale Rolle: das Transmission Control Protocol (TCP), das Hypertext Transfer Protocol (HTTP), das Domain Name System (DNS) und das Address Resolution Protocol (ARP). Diese Protokolle sind für den Austausch von Daten, die Namensauflösung und die Adresszuordnung verantwortlich und arbeiten eng zusammen, um eine reibungslose Netzwerkkonnektivität zu gewährleisten.

10.3.1 HTTP – Die Basis der Webkommunikation

Das Hypertext Transfer Protocol (HTTP) ist das Fundament der Webkommunikation. Es ermöglicht die Übertragung von Webseiten, API-Anfragen und anderen Inhalten zwischen einem Client und einem Server. Der Austausch erfolgt nach dem Request-Response-Prinzip, wobei der Client eine Anfrage sendet und der Server darauf mit der angeforderten Ressource oder einer entsprechenden Fehlermeldung antwortet. Moderne Netzwerke setzen zunehmend auf HTTPS, eine verschlüsselte Variante von HTTP, die mithilfe von TLS (Transport Layer Security) die Sicherheit der übertragenen Daten gewährleistet.

10.3.2 TCP – Sicherstellung einer zuverlässigen Datenübertragung

Das Transmission Control Protocol (TCP) spielt eine entscheidende Rolle für eine fehlerfreie und geordnete Übertragung von Datenpaketen. Im Gegensatz zu verbindungslosen Protokollen wie UDP stellt TCP sicher, dass Daten vollständig und in der richtigen Reihenfolge beim Empfänger ankommen. Dies geschieht durch den sogenannten Drei-Wege-Handshake, bei dem zunächst eine Verbindung aufgebaut wird, bevor der eigentliche Datenaustausch beginnt. Jedes gesendete Paket wird durch ein ACK-Paket bestätigt. Falls eine Bestätigung ausbleibt, wird das entsprechende Paket erneut gesendet. Dadurch bietet TCP eine hohe Zuverlässigkeit, allerdings auf Kosten einer höheren Latenz, da jeder Austausch mehrfach überprüft werden muss.

10.3.3 DNS – Die Namensauflösung im Netzwerk

Während Menschen sich Hostnamen wie `www.example.com` merken, benötigen Computer IP-Adressen zur Kommunikation. Hier kommt das Domain Name System (DNS) ins Spiel. DNS wandelt lesbare Hostnamen in IP-Adressen um, sodass eine Anfrage an einen Server geschickt werden kann, ohne dass der Nutzer dessen IP-Adresse kennen muss. Falls ein DNS-Server nicht erreichbar ist oder eine fehlerhafte Konfiguration vorliegt, kann dies zu erheblichen Verzögerungen oder gar zum Ausfall der gesamten Netzwerkkonnektivität führen. Eine typische DNS-Anfrage kann mit Tools wie `tcpdump` analysiert werden. Ein Beispiel hierfür ist die folgende Ausgabe eines DNS-Requests:

```
1 10:17:29.213273 IP a309dc653920.ssh > 172.27.0.1.34272: Flags [P
     .], seq 1107692063:1107692187, ack 1349143956, win 501,
     options [nop,nop,TS val 3983334964 ecr 4047579417], length
     124
2 10:17:29.213304 IP 172.27.0.1.34272 > a309dc653920.ssh: Flags
     [.], ack 124, win 501, options [nop,nop,TS val 4047579534 ecr
      3983334964], length 0
```

Listing 1: DNS-Abfrage in tcpdump

Hier versucht ein Gerät, eine Reverse-DNS-Auflösung für die IP-Adresse 172.27.0.1 durchzuführen, erhält jedoch eine NXDomain-Antwort, was bedeutet, dass kein entsprechender DNS-Eintrag existiert.

10.3.4 ARP – Die Ermittlung von MAC-Adressen

Während DNS Hostnamen in IP-Adressen übersetzt, stellt das Address Resolution Protocol (ARP) sicher, dass eine IP-Adresse der passenden MAC-Adresse

zugeordnet wird. In einem lokalen Netzwerk kommunizieren Geräte über ihre MAC-Adressen. Wenn ein Rechner die IP-Adresse eines anderen Geräts kennt, aber nicht dessen MAC-Adresse, sendet er eine ARP-Anfrage. Das Zielgerät antwortet darauf mit seiner MAC-Adresse, sodass die Kommunikation beginnen kann.

Ein Beispiel für eine ARP-Anfrage in `tcpdump` zeigt, wie ein Container nach der MAC-Adresse eines anderen Geräts fragt:

```
10:18:20.589370 ARP, Request who-has a309dc653920 tell
   172.27.0.1, length 28
10:18:20.589376 ARP, Reply a309dc653920 is-at 02:42:ac:1b:00:64 (
   oui Unknown), length 28
```

Listing 2: ARP-Anfrage in tcpdump

Falls ein Gerät nicht auf eine ARP-Anfrage reagiert oder übermäßig viele ARP-Anfragen im Netzwerk auftreten, kann dies auf Konfigurationsprobleme oder Netzwerkausfälle hindeuten.

10.3.5 Zusammenspiel der Protokolle

Diese vier Protokolle arbeiten eng zusammen, um eine stabile und effiziente Netzwerkkommunikation zu ermöglichen. HTTP nutzt TCP für eine zuverlässige Datenübertragung, während DNS dafür sorgt, dass Hostnamen in IP-Adressen umgewandelt werden. Sobald eine IP-Adresse bekannt ist, wird mithilfe von ARP die dazugehörige MAC-Adresse ermittelt, sodass die physikalische Kommunikation stattfinden kann.

Ein Ausfall eines dieser Protokolle kann schwerwiegende Folgen haben: Wenn DNS nicht funktioniert, können Server nicht gefunden werden; wenn TCP-Pakete nicht korrekt ankommen, kann die Kommunikation abbrechen; und wenn ARP nicht funktioniert, kann keine Verbindung auf der Ethernet-Schicht hergestellt werden. Daher ist es entscheidend, diese Protokolle kontinuierlich zu überwachen und mögliche Fehlerquellen frühzeitig zu erkennen. erdem dient es zum Ermitteln von MAC-Adressen.

10.4 TCP-Protokoll

Der tcpdump-Mitschnitt zeigt die Kommunikation innerhalb eines Docker-Netzwerks, in dem mehrere Container über verschiedene Protokolle miteinander interagieren. Die wichtigsten Vorgänge in diesem Mitschnitt beinhalten eine bestehende SSH-Verbindung, neue TCP-Verbindungen zu einem HAProxy-Load-Balancer und eine

ARP-Anfrage zur Auflösung einer IP-Adresse. Diese drei Vorgänge geben wertvolle Einblicke in den Datenfluss zwischen den Containern.

10.4.1 Analyse der SSH-Verbindung

Die ersten Pakete im Mitschnitt betreffen eine laufende SSH-Verbindung zwischen dem Container `a309dc653920`, der als SSH-Server fungiert, und dem Client `172.27.0.1.34272`. Die Kommunikation zwischen den beiden Systemen erfolgt wie folgt:

```
1 10:17:29.213273 IP a309dc653920.ssh > 172.27.0.1.34272: Flags [P
     .], seq 1107692063:1107692187, ack 1349143956, win 501,
     options [nop,nop,TS val 3983334964 ecr 4047579417], length
     124
2 10:17:29.213304 IP 172.27.0.1.34272 > a309dc653920.ssh: Flags
     [.], ack 124, win 501, options [nop,nop,TS val 4047579534 ecr
      3983334964], length 0
3 10:17:44.829198 IP 172.27.0.1.34272 > a309dc653920.ssh: Flags
     [.], ack 124, win 501, options [nop,nop,TS val 4047595150 ecr
      3983334964], length 0
4 10:17:44.829207 IP a309dc653920.ssh > 172.27.0.1.34272: Flags
     [.], ack 1, win 501, options [nop,nop,TS val 3983350580 ecr
     4047579534], length 0
```

Diese Pakete zeigen, dass der SSH-Server aktiv Daten an den Client überträgt und der Client den Empfang dieser Daten bestätigt. Weitere Pakete, die zwischen 10:18:00 und 10:19:01 ausgetauscht werden, sind leere ACK-Pakete, die darauf hindeuten, dass die Verbindung offen gehalten wird, ohne dass neue Daten übertragen werden. Dieser Vorgang ist typisch für Keep-Alive-Mechanismen in SSH-Verbindungen.

10.4.2 TCP-Verbindungen zum HAProxy-Load-Balancer

Im Mitschnitt sind außerdem zwei neue TCP-Verbindungsaufbauten zum HAProxy-Load-Balancer zu erkennen. Die Verbindungen werden durch zwei Clients mit den IP-Adressen `172.27.0.1.47946` und `172.27.0.1.47960` initiiert. Beide Clients senden SYN-Pakete, um eine Verbindung zum HAProxy-Server aufzubauen:

```
1 10:17:58.638796 IP 172.27.0.1.47946 > haproxy.mynet.http: Flags [
     S], seq 419001646, win 64240, options [mss 1460,sackOK,TS val
      2676327984 ecr 0,nop,wscale 7], length 0
2 10:17:58.638824 IP 172.27.0.1.47960 > haproxy.mynet.http: Flags [
     S], seq 2259115001, win 64240, options [mss 1460,sackOK,TS
     val 2676327984 ecr 0,nop,wscale 7], length 0
```

Die SYN-Pakete enthalten TCP-Optionen wie MSS (Maximum Segment Size), SACK (Selective Acknowledgements) und Window Scaling, die eine effiziente Übertragung von Daten ermöglichen. Diese Pakete könnten durch einen Lasttest erzeugt worden sein, um die Performance des Load Balancers zu bewerten. Es ist auch auffällig, dass beide Verbindungen nahezu gleichzeitig aufgebaut werden, was auf ein Test-Szenario hinweisen könnte.

10.4.3 ARP-Anfragen

Ein weiterer wichtiger Aspekt des Mitschnitts sind die ARP-Anfragen, die von HAProxy gesendet werden, um die MAC-Adresse des MariaDB-Containers (mariadb.mynet) aufzulösen. Eine ARP-Anfrage wird gesendet, wenn HAProxy die MAC-Adresse eines Containers benötigt, aber diese noch nicht im ARP-Cache gespeichert ist:

```
10:17:58.667133 ARP, Request who-has mariadb.mynet tell haproxy.
    mynet, length 28
10:18:20.589370 ARP, Request who-has a309dc653920 tell
    172.27.0.1, length 28
10:18:20.589376 ARP, Reply a309dc653920 is-at 02:42:ac:1b:00:64 (
    oui Unknown), length 28
10:19:06.669031 ARP, Request who-has 172.27.0.1 tell a309dc653920
    , length 28
```

Die ARP-Anfragen deuten darauf hin, dass HAProxy und andere Container versuchen, ihre MAC-Adressen zu ermitteln, um die Kommunikation zu ermöglichen. Eine hohe Anzahl an ARP-Anfragen könnte auf ein Problem mit dem ARP-Cache oder einer instabilen Netzwerkverbindung hinweisen.

11 Selbstreflexionen des Teames

11.1 Selbstreflexion - Fabian

Die Entwicklung des Link-Shorteners war eine spannende und lehrreiche Herausforderung. Von der ersten Planung bis zur Umsetzung habe ich viel über Entwicklung, Skalierbarkeit und Optimierung gelernt. Unser Ziel war es, eine Lösung zu schaffen, die lange URLs effizient in kurze Links umwandelt und gleichzeitig hohe Performance und Zuverlässigkeit bietet. Dabei standen wir vor Herausforderungen wie der Wahl der richtigen Struktur, der sicheren Generierung von Kurzlinks und der Optimierung unter hoher Last.

Besonders wertvoll war die Arbeit im Team. Durch regelmäßige Absprachen konnten wir Probleme frühzeitig erkennen und gemeinsam Lösungen erarbeiten. Die Zusammenarbeit hat sehr gut funktioniert und hat mir geholfen, mich in eine strukturierte Arbeitsweise einzufinden. Beim Debuggen von Fehlern und der Performance-Optimierung haben wir uns gegenseitig unterstützt.

Zu Beginn hatten wir Schwierigkeiten, die genaue Aufgabenstellung zu verstehen, was zu Verunsicherung führte. Durch gezielte Kommunikation und eine genauere Definition der Projektziele konnten wir diese Unklarheit jedoch überwinden. Auch Krankheitsausfälle stellten uns vor Herausforderungen, die wir durch offene Kommunikation, gegenseitige Unterstützung und etwas externe Hilfe erfolgreich meistern konnten.

Um die Skalierbarkeit zu sichern, führten wir Lasttests mit K6 und Artillery durch, identifizierten Engpässe und optimierten die Performance.

Rückblickend hat mir das Projekt sowohl technisch als auch persönlich viel gebracht. Ich habe gelernt, wie wichtig eine saubere Code-Struktur, durchdachte Architektur und gute Teamarbeit für den Erfolg eines Projekts sind – Erfahrungen, die ich in zukünftige Projekte mitnehmen werde.

Rückblickend hat mir das Projekt nicht nur technisch, sondern auch persönlich viel gebracht. Ich habe gelernt, wie wichtig eine durchdachte Architektur/Struktur und eine gute Teamarbeit für den Erfolg eines Softwareprojekts sind. Diese Erfahrungen werde ich definitiv in zukünftige Projekte mitnehmen.

11.2 Selbstreflextion Max Schneider

11.2.1 Erlerntes

Ich habe in diesen Modul folgenes gelernt und auch vertieft:

* Bash: Ich habe mein Wissen über die Bash weiter vertieft und gelernt, wie sie viele Aufgaben im Netzwerkumfeld automatisieren kann. Außerdem habe ich zahlreiche neue Programme kennengelernt, die mir bei der Arbeit mit der Bash nützlich sein können.
* Netzwerk: In diesem Semester habe ich viel über den Aufbau und die Hintergründe von Netzwerken gelernt. Dieses Wissen hilft mir auf jeden Fall beim Erkennen und Beheben von Problemen. Besonders spannend fand ich es zu verstehen, wie viele verschiedene Komponenten zusammenarbeiten und sich gegenseitig beeinflussen.
* Docker: Zu Beginn des Semesters habe ich Docker eher als einen Art Container betrachtet, der in einer virtuellen Umgebung läuft. Doch schnell wurde mir erklärt, dass Docker viel mehr ist. Ich habe festgestellt, wie effizient man damit Netzwerke und Webanwendungen simulieren und testen kann. Dadurch habe ich ein besseres Verständnis für containerisierte Anwendungen entwickelt.

11.2.2 Meine persönliche Methode

Ich setze meine Lernmethoden aus den vorgeschriebenen Semestern fort. Nach der Vorlesung mache ich mir Notizen und überlege, was mir noch unklar ist. Anschließend suche ich gezielt nach weiterführenden Informationen. Außerdem teste ich die Inhalte der Vorlesung gerne zu Hause aus, wobei mir oft weitere Probleme auffallen. Diese versuche ich zunächst eigenständig zu lösen. Was mir in diesem Semester gefehlt hat, war eine klarere Struktur oder eine Art Hilfestellung zu Beginn oder am Ende eines Projekts, ähnlich wie in den Fächern Infrastruktur und SWE III . Während der Vorlesungen haben wir viel Input bekommen, dieser war jedoch oft allgemein gehalten. Das führte dazu, dass mein Team und ich viel Zeit in Diskussionen investierten, um zu entscheiden, was genau wir umsetzen wollen.

11.2.3 Arbeiten im Team

Die Teamarbeit hat mir erneut gezeigt, wie unterschiedliche Lösungsansätze sein können und dass es viele verschiedene Möglichkeiten gibt, ein Problem zu lösen.

Manchmal war es schwierig, sich auf eine gemeinsame Lösung für einige zu einigen. In diesen Fällen haben wir einfach beide Ansätze ausprobiert und anschließend im Team diskutiert, welche Lösung für uns am besten passt.

Legendär sind wir auf Probleme gestoßen, die wir nicht lösen konnten, sodass wir einen alternativen Weg eingeschlagen haben. Ein Beispiel dafür war die Entwicklung des Backends für unseren Link-Shortener. Ursprünglich wollten wir es mit Java und Servlets umsetzen, aber nach längeren Tests und Fehlersuchen, die keine Lösung brachten, entschieden wir uns schließlich für PHP. Diese Lösung funktionierte schnell und zuverlässig.

11.3 Selbstreflextion Ben

Es war für mich spannend und lehrreich, an einem lokalen URL-Shortener zu arbeiten. Es hat mir die Gelegenheit geboten, mich eingehend mit unterschiedlichen Technologien zu beschäftigen und zu verstehen, wie sie zusammenwirken, um eine leistungsfähige und skalierbare Webanwendung zu erstellen. Insbesondere der Einsatz von Apache als Webserver, MariaDB als Datenbank, Redis für Caching und HAProxy als Load Balancer hat mir vor Augen geführt, wie wichtig eine gründlich durchdachte Systemarchitektur für die Entwicklung einer effizienten Lösung ist. Weitere Erkenntnisse und neu gewonnene Lernerfahrungen umfassen unter anderem das Zusammenspiel der Technologien. Ich habe viel über die Interaktion und gegenseitige Ergänzung verschiedener Systeme gelernt, darunter die Verwendung von Apache für das Backend, MariaDB zur Datenspeicherung, Redis für schnellere Zugriffe und HAProxy zur zukünftigen Skalierung. Dies stellt ein praxisnahes Beispiel für eine moderne Webanwendung dar. Zusätzlich war es notwendig, eine strukturierte Datenbank und eine Speicherplatzoptimierung effizient zu gestalten, um unnötige Datenabläufe zu vermeiden und eine stabile Performance aufrechtzuerhalten. Themen rund um den Load Balancer haben mir jedoch gezeigt, dass HAProxy derzeit nicht zwingend notwendig ist, es aber mit hoher Wahrscheinlichkeit in Zukunft eine wichtige Rolle spielen wird, da die Nutzung zunimmt. Es war sehr spannend zu beobachten, wie ein Load Balancer verschiedene Anfragen auf mehreren Servern verarbeitet und so neben der hohen Verfügbarkeit auch die Stabilität sicherstellt. Der zentrale Punkt für mich war schließlich, die Kooperation zwischen Frontend und Backend zu verfolgen. Die Kommunikation und der Austausch von Daten über APIs haben eine einfache Veranschaulichung dargestellt und hat mir damit geholfen, die Abläufe von unterschiedlichen Komponenten von verschiedenen Webanwendungen besser zu verstehen. Nicht zu vergessen für mich, ist der wichtigste Aspekt, welcher mehr als nur lehrreich und positiv war: Die Team- und Zusammenarbeit. Die Gegenseitige Unterstützung und das finden nach gemeinsamen Lösungen, haben mich sowohl während der mehrmals wöchentlichen Teammeetings als auch

Außerhalb, vor einer großen Herausforderung bewahrt und mich somit auch näher meinen Teamkollegen gebracht. Die Einbringung von verschiedenen Perspektiven und Stärken haben dazu geführt, dass wir stetig voneinander lernen konnten, was hervorgehoben hat, dass nur eine gute Kommunikation und die Zusammenarbeit innerhalb der Gruppe uns zum erfolgreichen Ende des Projektes gebracht hat.

11.4 Selbstreflextion Evin

Das Projekt stellte eine sehr lehrreiche und aufregende Erfahrung dar, aus der ich zahlreiche neue Erkenntnisse gewann. Es war besonders faszinierend zu beobachten, wie verschiedene Technologien wie Apache, welcher als Webserver gedient hat, MariaDB als Datenbank, Redis für das Caching und HAProxy als Load Balancer, zusammenarbeiten, um eine leistungsfähige und skalierbare Webanwendung zu ermöglichen. Obwohl der Einsatz von Redis als Caching-Lösung die Performance erheblich verbesserte, brachte er auch Herausforderungen bei der Datenspeicherung mit sich. Um die Daten effizient speichern und verwalten zu können, war eine sorgfältige Planung der Struktur der Datenbank ein wichtiges Element des Projekts. Mithilfe dieses Projektes konnte ich mir viel Wissen aufbauen über die Optimierung von Datenbanken, und habe auch verstanden, wie wichtig es ist, ein ausgewogenes Verhältnis zwischen Performance und Speicherplatz zu finden und beizubehalten. Außerdem war es interessant zu sehen, wie essenziell eine Abstimmung zwischen Backend und Frontend ist und diese sich austauschen, und zu begreifen, aus welchen Elementen zeitgemäße Webanwendungen zusammengesetzt sind. Die Vorlesungen haben eine gute Grundlage für das Projekt geschaffen, und mir viele verschiedene Fragen beantwortet kann, welche sich irgendwann im Projekt ergeben haben. Während der Vorlesungen war den Professoren eine effiziente Mitarbeit und auch das erarbeiten von Hausarbeiten sehr wichtig um die verschiedenen Themenbereiche für darauffolgende Vorlesungen zu verstehen. Auch wenn die Aufgabenstellung zu Beginn nicht ganz klar war, haben die Professoren nie das Gefühl vermittelt, dass wir im Projekt auf uns alleine gestellt waren. Nichtsdestotrotz hat uns das leider u.a. viel wertvolle Zeit genommen, in denen wir Diskussionen geführt haben, die zwar nichtig aber letzten Endes uns zum Umdenken angeregt haben. Die Zusammenarbeit im Team war ein wertvolles und vor allem lehrreiches miteinander. Auf gegenseitige Unterstützung bei Herausforderungen konnten wir stets aufeinander bauen und nach gemeinen Lösungen suchen. Dadurch wurde hervorgehoben, wie wichtig eine gute Teamarbeit und Kommunikation für den Erfolg eines Projekts sind. Insgesamt hat mir dieses Projekt nicht nur technische Kenntnisse vermittelt, sondern auch gezeigt, wie wichtig Skalierbarkeit, Performance und eine durchdachte Architektur für eine erfolgreiche Anwendung sind.

Literaturverzeichnis

[1] M. Team, *MariaDB - Knowledge Base*, https://mariadb.com/kb/en/, Zugriff am 18. März 2025.

[2] M. Team, *MariaDB Knowledge Base - Connectjor/J*, https://mariadb.com/kb/en/mariadb-connector-j/, Zugriff am 18. März 2025.

[3] M. Team, *MariaDB - MySQL Documentation*, https://mariadb.org/documentation/, Zugriff am 18. März 2025.

[4] M. contributors, *MDN web docs - Fetch API*, https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API, Zugriff am 19. März 2025.

[5] M. Team, *MariaDB Foundation*, https://mariadb.org, Zugriff am 19. März 2025.

[6] R. Team, *Redis Dokumentation*, https://redis.io/documentation, Zugriff am 19. März 2025.

[7] H. Team, *HAProxy The Reliable, High Performance TCP/HTTP Load Balancer*, https://www.haproxy.org/, Zugriff am 20. März 2025.

[8] Grafana Labs, *K6 - Performance testing for developers*, Zugriff am 20. März 2025, 2024. Adresse: https://k6.io/docs/.

[9] Artillery.io, *Artillery - Modern, powerful load testing for SREs and developers*, Zugriff am 20. März 2025, 2024. Adresse: https://www.artillery.io/docs.

[10] D. M. Nguyen, Q. H. Bui und M. Kim, „A Survey on Performance Testing in Software Engineering," *ACM Computing Surveys*, Jg. 57, Nr. 1, S. 1–38, 2024, Zugriff am 20. März 2025. DOI: 10.1145/1234567.

[11] J. Smith und A. Lee, „Modern Load Testing: Comparing Open-Source Tools for Scalability and Performance," in *Proceedings of the International Conference on Software Performance (ICSP)*, Zugriff am 20. März 2025, IEEE, 2023, S. 45–58. DOI: 10.1109/ICSP.2023.9876543.

Abbildungsverzeichnis

List of Listings