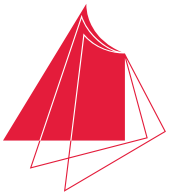


Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Entwicklung eines Frameworks zur Integration von ActivityPub

Bachelor Thesis von

Armin Kunkel

an der Fakultät für Informatik und Wirtschaftsinformatik
Fachrichtung Verteilte Systeme (VSYS)

Erstgutachter:	Prof. Dr. rer. nat. Christian Zirpins
Betreuer:	M. Sc. Robert Schäfer

01. Dezember 2018 – 31. März 2019

Hochschule Karlsruhe Technik und Wirtschaft
Fakultät für Informatik und Wirtschaftsinformatik
Moltkestr. 30
76133 Karlsruhe

Ich versichere wahrheitsgemäß, die Arbeit selbstständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Änderungen entnommen wurde.

Karlsruhe, 19. April 2019

. . . . . .

(Armin Kunkel)

Zusammenfassung

In dieser Arbeit wurde der ActivityPub Standard und die damit in Verbindung stehenden Standards, Protokolle sowie Netzwerke angesehen. ActivityPub ist generell in zwei Protokoll Schichten aufgeteilt. Die Client-zu-Server Interaktionen sind in der „Social API“ festgehalten und im „federated Server-zu-Server“ Protokoll sind Interaktionsmöglichkeiten für Server untereinander festgehalten. ActivityPub baut auf das Datenformat Activity-Streams 2.0 auf. Dieses besteht, wie auch ActivityPub, aus zwei Teilen. Zum einen dem Kernvokabular, welches Grundlegende Modelle festlegt und zum anderen das eigentliche Vokabular, deren Modelle die des Kernvokabulars erweitern. Jedes Modell in Activity-Streams 2.0 wird wiederum als JSON Linked Data Objekt repräsentiert um eine maschinenlesbare Form des Inhalts zu haben und darüber auch Links zwischen Inhalten auf verschiedenen Webseiten zu erstellen.

Mit dem gesammelten Wissen wurde ein Framework entwickelt um eine möglichst einfache Integration des ActivityPub Protokolls in eine bestehende Applikation zu ermöglichen. Die Entwicklung des Frameworks zur ActivityPub Integration beschränkt sich auf die Bereitstellung einer drei Schichten Architektur mit der Datenschicht als Interface. Schicht 1 (*Controller*) beinhaltet Router, welche Funktionalität bereitstellen um ActivityPub konform Inhalte bereitzustellen und zu verarbeiten. Schicht Nummer 2 (*Service*) nimmt Aktivitäten entgegen und holt sowie speichert Inhalte über das IDataSource Interface (*Datenschicht*). In dieser Schicht wird eine Datenbank oder z. B. eine GraphQL Schnittstelle, wie in dieser Arbeit verwendet, angesprochen um Änderungen an einem Datenbestand durchzuführen oder Inhalte zum Bereitstellen zu erhalten.

Zur Authentifizierung der Server untereinander sowie zum Sicherstellen der Datenintegrität werden HTTP-Signaturen verwendet. Hierbei wird für jeden Benutzer ein Schlüsselpaar eines öffentlichen Schlüssel Verfahrens generiert. Mit diesem können dann die Anfragen signiert sowie verifiziert werden.

Für die Signierung sowie Verifizierung wurde eine Performanz-Messung durchgeführt. Diese wurde erstens auf einem Notebook mit 2-Kernen und zweitens auf einem Desktop-Computer mit 16-Kernen durchgeführt. Die Ergebnisse wurden diskutiert und Empfehlungen zu Hashfunktionen abgegeben die Verwendung finden sollten.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

SWWG Social Web Working Group

W3C World Wide Web Consortium

DSA Digital Signature Algorithm

OWP Open Web Platform

RSA Rivest-Shamir-Adleman

API Application Programming Interface

CLI Command Line Interface

BDD Behaviour Driven Development

TDD Test Driven Development

CPU Central Processing Unit

LRDD Link-based Resource Descriptor Discovery

JSON JavaScript Object Notation

JSON-LD JSON Linked Data

URI Uniform Ressource Identifier

AS2 ActivityStreams 2.0

AS2-C AcitivtyStreams Core

AS2-V ActivityStreams Vocab

CORS Cross-Origin Ressource Sharing

JRD JSON Ressource Descriptor

HTTP Hypertext Transfer Protocol

REST Representational State Transfer

URL Uniform Ressource Locator

1. Einleitung

Zentralisierung von Daten und das Vertrauen auf einzelne Instanzen ist heutzutage allgegenwärtig. Im Internet gibt es eine Fülle an sozialen Netzwerken wie Facebook, Twitter, Google+, Instagram oder Pinterest die zumeist das Recht an den Daten, die Sie von ihren Nutzer bekommen, behalten und nicht zuletzt diese Daten auch verkaufen für z. B. Werbezwecke. Für Kriminelle sowie Geheimdienste ist es außerdem leichter an die gesamte Datenbank zu kommen, da bei zentralisierten Netzwerken meist auch eine zentrale Datenbank, bzw. ein zentraler Zugriffspunkt, vorhanden ist. Wenn der Zugriff auf diesen zentralen Punkt erreicht ist kann die ganze Datenbank ausgelesen werden. Um die Kontrolle über Daten in Richtung Nutzer zu lenken kann dezentralisiert werden. Dadurch werden auch mehrere unabhängig administrierte Datenbanken benötigt, was die Sicherheit des Netzwerks erhöht.

Was bedeutet nun dezentral? In der Politik wird unter Dezentralisierung „die Übertragung zentral staatlicher Aufgaben auf subnationale oder subsidiäre Ebene(n) verstanden“ (Vgl. [1]). In der Energiewirtschaft spricht man von einer „dezentralen Stromerzeugung“, wenn der Strom an den Stellen wo er verbraucht auch erzeugt wird. Ein Beispiel hierfür wäre ein Wasserkraftwerk, dass den Strom für die umgebenen Dörfer oder Städte liefert (Vgl. [25]). In der Informatik versteht man unter Dezentralisierung das verteilen von u. a. Daten über mehrere unabhängige Server hinweg.

Durch die Verteilung der Aufgaben wird die zentral staatliche Ebene entlastet und hat somit mehr Ressourcen für andere Aufgaben. Beim Errichten von Wasserkraftwerken nahe an Dörfern und Städten entfällt der Transport des Stroms über größere Distanzen und somit verringert sich der Verlust beim Transport über das Stromnetz. Dezentralisieren von sozialen Netzwerken bringt verschiedene Vorteile mit sich. Ein Vorteil ist die Skalierbarkeit bei dezentralen sozialen Netzwerken. Durch das hinzufügen weiterer Instanzen können dem Netzwerk neue Ressourcen bereitgestellt werden. Ein weiterer Vorteil liegt darin, dass keine zentrale Kontrollinstanz vorhanden ist welche korrumpiert werden kann, sei es durch Kriminelle, wirtschaftliche Interessenvertreter oder staatliche Autoritäten.

Die Arbeit wird in einer gemeinnützigen und durch Spenden finanzierten GmbH namens Human-Connection angefertigt. Hauptsächlicher Arbeitsschwerpunkt der gGmbH[1] liegt in der Umsetzung und Verbreitung eines neuartigen sozialen Wissens- und Aktionsnetzwerkes welches seinen Nutzern die Möglichkeit bieten soll von der Information hin zur Aktion zu kommen. Dabei liegt das Augenmerk auf der Gemeinnützigkeit, Transparenz durch Open-Source, deutschen Serverstandorten, Kommerz-, Werbe- und Zensur-

[1] Gemeinnützige Gesellschaft mit beschränkter Haftung

freiheit, lösungsorientiertem Handeln sowie verzicht auf Datenverkauf. Ziel der Arbeit ist die Erstellung eines Frameworks zur Integration des förderierten Server-zu-Server Protokolls in eine bestehende Applikation mit möglichst wenig Programmieraufwand. Darüber hinaus soll die Arbeit einen Überblick über die Grundlagen des Protokolls, sowie zugehörigen Standards, geben und wie die relevanten Bestandteile zu verstehen sind.

Um sicherzustellen das gesendete Inhalte vom einem zum Anderen Server manipulationsfrei übertragen wurden, wird ein Verfahren benötigt um dies zu gewährleisten. Nicht nur für die manipulationsfreie Übertragung sondern auch das Authentisieren der Nutzer am Empfänger-Server, also das Sicherstellen ob die Inhalte wirklich von einem Nutzer auf einem Server stammen, welcher über den zum öffentlichen Schlüssel zugehörigen privaten Schlüssel zum Verifizieren verfügt, wird ein Verfahren benötigt. Das für die beiden genannten Punkte bei der Implementierung verwendete Verfahren ist das HTTP-Signaturverfahren. Eine Signatur kann mit einem privaten Schlüssel in Kombination mit einer Hashfunktion erstellt werden. Dabei können verschiedene Funktionen zum Signieren sowie Verifizierne verwendet werden.

1.1. Motivation

Bei zentralen sozialen Netzwerken besitzt die Kontrolle über Daten das Unternehmen. Durch die Dezentralisierung bleiben Daten bei den einzelnen Instanzen des sozialen Netzwerkes. Die Datenbanken bei zentralen Netzwerken können zwar verteilt sein und auch das Netzwerk könnte dezentral angelegt sein, doch die Hoheit über Daten liegt bei dem Unternehmen. Bei zentralen sozialen Netzwerken ist es außerdem nicht ohne weiteres möglich Inhalte mit anderen Netzwerken auszutauschen. Mit einem Protokoll wie ActivityPub wird das Verteilen sowie der Zugriff auf Inhalte und der Austausch von Aktivitäten über mehrere soziale Netzwerke hinweg ermöglicht.

Der im März 2018 vom World Wide Web Consortium (W3C) empfohlene Standard namens „ActivityPub“ wurde auf Basis des Wissens im Umgang mit dem OStatus und Pump.io Protokoll entwickelt und besteht aus zwei Teilen. Dem Client-zu-Server Protokoll, auch „Social API“ genannt, und dem förderierten Server-zu-Server Protokoll. Durch die ActivityPub W3C Empfehlung Anfang 2018 und die Social Web Working Group (SWWG) Arbeitsgruppe des W3C sowie weiteren, wird die Verbreitung des Protokolls vorangebracht. Dabei kann der Netzwerkeffekt genutzt werden.

Der Netzwerkeffekt ist ein aus der Volkswirtschaftslehre stammender Begriff. Für die Verständlichkeit sei als Beispiel das Telefonnetz genannt:
Umso mehr Leute ein Telefon besitzen, umso höher ist der Nutzen für jeden einzelnen Telefon Besitzer (Vgl. [20]).
Übertragen auf ActivityPub bedeutet das soviel wie:
„Je mehr Netzwerke den Standard implementieren, desto höher ist der Nutzen (die Reichweite) für ein einzelnes Netzwerk und im Endeffekt für die einzelnen Nutzer“.

Durch ein Framework zur Integration des ActivityPub Standard kann dieser Effekt verstärkt werden. In dieser Abschlussarbeit wird ein solches Framework erstellt um es Entwicklern leichter zu ermöglichen ihre Applikation ActivityPub konform zu machen.

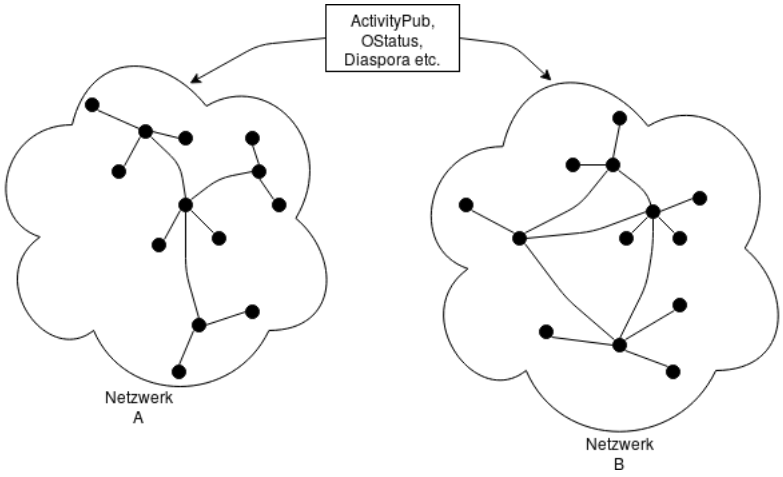


Abbildung 1.1.: Förderieren von Netzwerken über verschiedene Protokolle

Um die Förderierung zweier dezentrale sozialer Netzwerke zu veranschaulichen, wird Abb. 1.1 auf der nächsten Seite betrachtet. In dieser sind zwei Netzwerke zu sehen (Netzwerk A und B). Die Superpeers (Instanzen) eines Netzwerks können, auch wenn dieses noch nicht förderiert wurde, untereinander kommunizieren. Die Regulären Peers stellen hier die einzelnen Clients dar, welche mit dem zugehörigen Superpeer kommunizieren um Inhalte anzuzeigen sowie zu erstellen. Das förderieren der beiden Netzwerke kann über Protokolle wie OStatus (s. 2.1.1.1) oder ActivityPub erreicht werden. Dabei erhalten Superpeers die Fähigkeit auch mit Superpeers anderer Netzwerke zu interagieren anstatt nur mit denjenigen innerhalb des eigenen Netzwerks.

Es wird ein Problemansatz und eine Problemlösung gewählt, welche Entwicklern die Integration des ActivityPub Protokolls so einfach als möglich machen soll. Weder eine extra Datenbank für das Framework, noch viel Vorkenntnis des Protokolls an sich sind notwendig um die Implementierung umzusetzen. Lediglich das Interface muss implementiert werden um einen lauffähigen ActivityPub Service zu erhalten. Dabei werden Hilfsfunktionen bereitgestellt um die Umsetzung möglichst einfach zu gestalten. Zudem wurde auf ein gutes Codeverständnis Wert gelegt.

ActivityPub ist wie „OStatus“ ein Protokoll für dezentrale soziale Netzwerke und wird in dieser Abschlussarbeit implementiert und die wichtigsten Komponenten beschrieben.

1.2. Problemansatz

Folgende Fragestellung wird in dieser Abschlussarbeit behandelt:

Wie wird der Standard möglichst Entwicklerfreundlich implementiert?

Ein wichtiger Aspekt dabei ist die spontane Generierung aller ActivityPub spezifischen Komponenten, wie dem Aktorenobjekt, Sammlungen sowie Objekten und Aktivitäten. Dadurch wird keine zusätzliche Datenbank ausschließlich für das Framework benötigt. Es reicht somit die Nutzung einer bestehenden Datenquelle der Applikation.

Ein weiterer wichtiger Aspekt liegt in der Schaffung eines Interfaces. Über die Implementierung dieses Interfaces sollen Entwickler in der Lage sein ActivityPub in ihre Applikation zu integrieren. Zur weiteren Vereinfachung wird außerdem eine Sammlung von Hilfsfunktionen erstellt.

Außerdem wird auf das Codeverständnis Wert gelegt, was heißt dass die Benennung von Funktionen und Variablen auf eine Weise geschieht die für andere Entwickler auch ohne Dokumentation möglichst leicht verständlich ist.

Als letzter Aspekt wird darauf geachtet, dass die Applikation in einen bestehenden Server integriert sowie als eigenständiger Server gestartet werden kann.

1.3. Problemlösung

Um die spontane Generierung zu erreichen, werden die einzelnen Komponenten als Generator Funktionen implementiert welche die Parameter der Funktion in ein Objekt Literal, mit unter anderem statischen Werten, eintragen.

Aufgeteilt wird das Framework in drei Schichten, der Controller-, Service- sowie Datenschicht. Die zu implementierende Schicht ist die Datenschicht. Da in JavaScript Interfaces nativ nicht existieren, wird stattdessen ein Objekt mit leeren Funktionsrümpfen dafür genutzt. Dadurch wird der Fokus des Entwicklers auf die Anbindung der Datenquelle gelenkt. Im besten Falle muss somit nichts in den anderen Schichten, oder auch Hilfsfunktionen, angepasst werden. Allerdings sind Änderungen nötig wenn weitere Aktivitäten und Objekte verarbeitet werden sollen.

1.4. Struktur der Arbeit

Kapitel zwei gibt einen Überblick über allgemeine Grundlagen von sozialen Netzwerken. Es wird grob erläutert für was soziale Netzwerke gedacht sind, der Unterschied zwischen zentralen, verteilten, dezentralen und förderierten sozialen Netzwerken herausgestellt sowie auf Sicherheitsaspekte eingegangen. Es wird das Wort Fediverse etwas näher definiert und erläutert worum es sich dabei handelt und im weiteren werden mehrere Protokolle die, sowie auch ActivityPub, zum Fediverse gehören vorgestellt.

Im Anschluss wird ein Kryptographie Unterkapitel eingeführt in dem die benötigten Verfahren für die sichere Umsetzung erläutert werden. Begonnen wird mit einer kur-

zen Einführung in Kryptographie. Darauf folgend wird kurz das RSA Verfahren sowie eine allgemeine Beschreibung von Signatur Algorithmen vorgenommen. HTTP Signaturen werden ausführlicher beschrieben, da diese bei der Implementierung des Prototypen verwendet wurden.

Darauf folgend wird der ActivityPub Standard eingeführt sowie eingeordnet, Bestandteile des Protokolls beschrieben, die Funktionsweise der Client-zu-Server sowie Server-zu-Server Kommunikation und zugehörige Standards kurz erläutert. Des weiteren wird auf die Authentifizierung und Datenintegrität bei ActivityPub eingegangen um damit folgende Fragen zu klären:

- Wie authentifiziert sich ein Benutzer gegenüber dem Server?
- Wie stellt man sicher, dass die übertragenen Daten unverändert angekommen sind?

Zu Beginn von Kapitel drei wird auf die Anforderungen, welche an das Framework gestellt werden eingegangen. Im zweiten Unterkapitel wird dann die Entwurfsentscheidung besprochen und dabei folgende Frage beantwortet:

- Warum wurde die Architektur gewählt?

Das dritte Unterkapitel gibt Aufschluss über die konkrete technische Architektur. Diese wird anschließend anhand von Diagrammen veranschaulicht und erläutert.

In Kapitel vier wird die konkrete Implementierung des Prototypen, welcher für das Unternehmen angefertigt wurde in der die Abschlussarbeit bearbeitet wird, erläutert. Anfangs wird die Abfragesprache „GraphQL“ sowie die genutzte Graph-Datenbank des Projekts vorgestellt, für welches die Implementierung angefertigt wird. Danach gibt ein Klassendiagramm der Service- und Datenschicht Aufschluss über die in diesen Objekten enthaltene Funktionalität welche darauf folgend genauer beschrieben wird. Anschließend werden anhand der Verzeichnisstruktur die Controller-Schicht, sowie die Werkzeug und Sicherheitsfunktionalitäten beschrieben. Es werden zusätzlich auch die genutzten Testwerkzeuge und zwei Vorgehensweisen der Softwareentwicklung kurz erläutert. Unterkapitel 4.6 gibt Aufschluss über die implementierte Funktionalität des ActivityPub Standards. Zuletzt wird ein genauerer Blick auf die Signierung sowie Verifikation geworfen.

Zu Beginn des fünften Kapitels werden verschiedenste Anwendungsbeispiele anhand von Diagrammen veranschaulicht und erklärt. Insgesamt werden die folgenden fünf Anwendungsbeispiele beschrieben:

- Folgen eines Nutzers aus einem anderen Netzwerk
- „Liken“ eines Objekts in einem anderen Netzwerk
- Rückgängig machen einer zuvor gesendeten „Follow“ Aktivität
- Die Umkehrung eines „Likes“ (Dislike)

- Artikel erstellen in einem anderen Netzwerk

Des weiteren wird eine Performanz-Messung für die Signatur Erstellung, sowie Verifikation durchgeführt und die Ergebnisse interpretiert. Bei der Messung werden 4 verschiedene Ergebnistabellen erstellt; Jeweils zwei für eine Hardwarevariation. Zum Einen wird ein älteres Notebook Modell zum testen verwenden, zum Anderen ein Spiele-Computer. Außerdem wird aufgrund der interpretierten Ergebnisse und anderen Aspekten Empfehlungen gegeben, welche Hashfunktion für welchen Zweck am Besten passt.

2. Grundlagen zur Umsetzung dezentraler sozialer Netzwerke

Dieses Kapitel gibt einen Überblick über allgemeine Grundlagen von sozialen Netzwerken. Es wird grob erläutert für was soziale Netzwerke gedacht sind; Der Unterschied zwischen zentralen, verteilten, dezentralen und förderierten sozialen Netzwerken herausgestellt sowie auf Sicherheitsaspekte eingegangen.

Im Anschluss wird ein Kryptographie Unterkapitel eingeführt in dem die benötigten Verfahren für die sichere Umsetzung erläutert werden. Begonnen wird mit einer kurzen Einführung in Kryptographie. Darauf folgend wird kurz das RSA Verfahren sowie eine allgemeine Beschreibung von Signatur Algorithmen vorgenommen. HTTP Signaturen werden ausführlicher beschrieben, da diese bei der Implementierung des Prototypen verwendet wurden.

Darauf folgend wird der ActivityPub Standard eingeführt sowie eingeordnet, Bestandteile des Protokolls beschrieben, die Funktionsweise der Client-zu-Server sowie Server-zu-Server Kommunikation und zugehörige Standards kurz erläutert. Des weiteren wird auf die Authentifizierung und Datenintegrität bei ActivityPub eingegangen um damit folgende Fragen zu klären:

- Wie authentifiziert sich ein Benutzer gegenüber dem Server?
- Wie stellt man sicher, dass die übertragenen Daten unverändert angekommen sind?

2.1. Allgemeine Grundlagen sozialer Netzwerke

In den Sozialwissenschaften versteht man unter einem sozialen Netzwerk mehrere Personen die miteinander wechselwirkend auf soziale weise interagieren (Vgl. [26]). Die Informatik bildet soziale Netzwerke als Plattformen zum Aufbau und der Pflege von Beziehungen ab. Weiter gefasst können auch Mikroblogging-Dienste, Chat Software, Voice Chat-Programme u. s. w. zu sozialen Netzwerken gezählt werden, da auch die Möglichkeit geboten wird auf soziale Weise miteinander zu interagieren. Im Allgemeinen besteht ein soziales Netzwerk aus Softwaretechnischer Sicht aus 3 Komponenten, dies ist aber keine Prämisse. Der Nutzer benutzt das Netzwerk durch eine meist grafische Schnittstelle. Um Zugriff auf dieses zu erlangen benötigt der Nutzer ein ***Frontend***. Dies kann in grafischer Form oder als Command Line Interface (CLI) zur Verfügung stehen. Über das Frontend kann der Nutzer sich nun mit seinen Anmeldeinformationen am ***Backend*** anmelden und

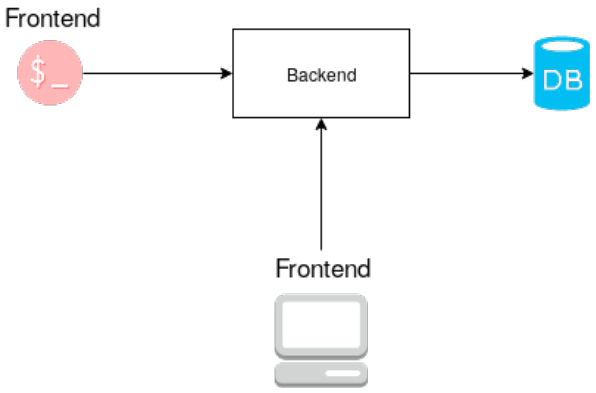


Abbildung 2.1.: 3 Tier Architektur

somit meist eine Sitzung eröffnen. Steht ein Web Interface zur Verfügung, werden Sitzungen oft über Clientseitige Cookie Speicherung aufrecht erhalten. Das Backend hat die Aufgabe eingehende Nutzeranfragen entsprechend zu beantworten und, falls autorisiert, die gewünschten Aufgaben auszuführen. Bei größeren sozialen Netzwerken fällt eine große Menge generierter Daten an die verwaltet werden müssen. Dafür wird meistens eine ***Datenbank*** herangezogen die zumeist auch repliziert betrieben wird. An sich kann ein zentrales Netzwerk auch über den Einsatz von virtuellen Maschinen und Lastenverteilung verteilt oder dezentral sein. Beispielsweise könnte ein zentrales soziales Netzwerk Anfragen länderspezifisch an zuständige Instanzen des Netzwerks weiterleiten. Auch ist es möglich die Daten länderspezifisch zu speichern und somit den Nutzern aus Deutschland andere Inhalte anzubieten als denjenigen aus der Schweiz.

2.1.1. Verwandte Protokolle

Zu Beginn dieses Kapitels wird das Wort Fediverse etwas näher definiert und erläutert worum es sich dabei handelt. Im Anschluss werden mehrere Protokolle die, sowie auch ActivityPub, zum Fediverse gehören vorgestellt.

Als erstes soll der Begriff „förderiertes Netzwerk“ von Förderation hergeleitet werden. Unter einer Förderation versteht man den Verbund von etwas. Deutschland ist z. B. ein förderierter Bundesstaat, welcher 16 Bundesländern verbindet. Ein Verbund von Netzwerken wird demnach als Netzwerkverbund, oder auch „förderiertes Netzwerk“, bezeichnet.

„Fediverse“ ist ein Kofferwort aus „Federated“ und „Universe“. Historisch gesehen beinhaltete der Begriff nur Microblogging Plattformen die das OStatus Protokoll unterstützen. Mehr zu OStatus **s. 2.1.1.1**. Mittlerweile beinhaltet das Fediverse mehr als nur Microblogging Plattformen wie Mastodon[1], sondern auch soziale Netzwerke wie GNU Social[2],

[1] `https://mastodon.social/about`
[2] `https://gnu.io/social/`

sowie „Video hosting“ (PeerTube[3]), Plattformen zum veröffentlichen von Inhalten, wie Wordpress, und viele weitere (Vgl. [8]).

Im „Fediverse“ dreht sich alles um freie (Open Source) Software anstelle von kommerziellen Produkten. Zudem kann ausgesucht werden welchem Administrator man die Kontrolle über seine Daten geben möchte, anstatt auf eine einzelne Instanz vertrauen zu müssen [8]. Ein weiterer wichtiger Aspekt ist das förderieren von Netzwerken über Protokolle wie OStatus oder Pump.io. Somit kann ein dezentrales soziales Netzwerk durch die Implementierung eines dieser Protokolle zum förderierten sozialen Netzwerk werden.

Laut dem Fediverse Netzwerkreport 2018, welcher online zur Verfügung steht, hat sich die Gesamtanzahl der erreichbaren Instanzen von 2.756 auf 4.340 erhöht. Dies entspricht einem Wachstum von 58%. Die Nutzeranzahl ist von 1.786.036 auf 2.474.601 gestiegen (39%). Weitere Details sind im Netzwerk Report nachzulesen. Es ist hinzuzufügen das der Netzwerkreport unvollständig ist und Fehler beinhalten kann [9].

2.1.1.1. OStatus

Der OStatus Standard ist eine Sammlung von verschiedenen nachfolgend gelisteten Protokollen: *Atom, ActivityStreams, WebFinger, WebSub, Salmon, Portable Contacts*

Ausgelegt ist der Standard auf das empfangen und versenden von Status Aktualisierungen für förderierte Microblogging Dienste wie Mastodon. Das Zusammenspiel obiger Protokolle ermöglicht den Austausch von Status Aktualisierungen in fast Echtzeit. Profile werden über den WebFinger Standard entdeckt und zusätzlich ist es möglich über das Link-based Resource Descriptor Discovery (LRDD) Protokoll die Profile zu entdecken. Um Aktualisierungen zu versenden wird WebSub, unter Zuhilfenahme des ActivityStream Datenformats und Atom, verwendet. Mit Salmon werden Benachrichtigungen versendet[4].

2.1.1.2. Diaspora

Diaspora ist ein freier Web Server mit einer integrierten Implementierung eines verteilten sozialen Netzwerkes. In diesem Netzwerk wird ein Knoten als Pod bezeichnet, wobei die Gesamtzahl aller Pod’s das Diaspora Netzwerk ausmachen. Durch die integrierte Funktionalität zum sozialen Netzwerken kann auf dem Web Server aufsetzend eine eigene Applikation entwickelt werden.

Diaspora entdeckt die Profile der Nutzer, wie auch OStatus, über den WebFinger Standard. Zudem wird WebSub verwendet um Echtzeitaktualisierungen zu verteilen. Dies verringert den Verbrauch von Ressourcen auf der Client Seite ausgelöst durch Polling (Vgl. [7]). Die Software ist in der Ruby Programmiersprache unter zu Hilfenahme des „Ruby

[3] https://joinpeertube.org/en/
[4] https://ostatus.github.io/spec/OStatus%201.0%20Draft%202.html

on Rails“ Web Frameworks entwickelt worden. Folgende Modelle sind in dieser Software enthalten (Vgl. [6]): *Benutzer, Personen, Profile, Kontakte, Anfragen, Aspekte, Inhalte, Kommentar, Widerruf*

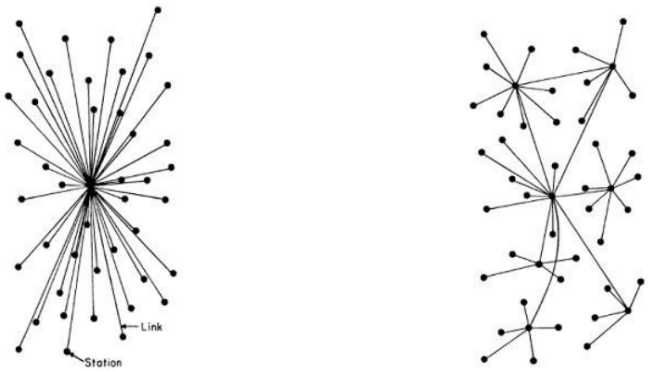
2.1.1.3. Pump.io

Ähnlich dem in dieser Arbeit behandelten Standard werden beim Pump.io Protokoll auch Nutzernachrichteneingänge und Nutzernachrichtenausgänge bereitgestellt. Der Nachrichtenausgang endet hier anstatt auf das Postfix „outbox“, wie bei ActivityPub, auf „feed“. Außerdem nutzen beide das Activity Streams 2.0 Datenformat. Zur Authentisierungen des Clients gegenüber dem Server wird hier das OAuth 1.0 Protokoll verwendet (Vgl. [11]). Die Nutzerprofile sowie Informationen über den Host werden über die „Web Host Meta“ Spezifikation entdeckt (Vgl. [12]). Web Host Meta weist Ähnlichkeiten zu dem Webfinger Protokoll auf[5].

2.1.2. Unterschiedliche Klassen sozialer Netzwerke

Zentrale soziale Netzwerke haben den Nachteil eines „Single point of Failures“. Fällt dieser Knoten aus, bricht das ganze Netzwerk zusammen. Zudem sind sie kaum skalierbar. Ein Vorteil eines zentralen Netzwerks ist die schnelle Bereitstellbarkeit.

Obwohl aus Sicht der Daten die meisten sozialen Netzwerke zentral sind, kann ihre Architektur intern sowohl verteilt als auch dezentral sein.



(a) Zentrales soziales Netzwerk (b) Dezentrales soziales Netzwerk

Quelle: (Zentrale, dezentrale, verteilte Systeme [10])

Abbildung 2.2.: Klassen sozialer Netzwerke

In Abbildung 2.2 ist ein zentrales Netzwerk gezeigt welches aus einem zentralen Knoten (Peer) und verschiedenen Blättern (Benutzern) besteht. Fällt der zentrale Knoten aus, können die Nutzer nicht mehr auf das Netzwerk zugreifen was einen großen Flaschenhals

[5] `https://github.com/pump-io/pump.io/blob/master/API.md`

aufzeigt.

Die Praxis zeigt allerdings, dass ein ausschließlich zentrales System eher selten existiert. Meist sind es Mischformen der verschiedenen Architekturen. Auch große zentrale soziale Netzwerke müssen um Verfügbarkeit u. w. zu gewährleisten verteilte Aspekte haben.

Bei einem *dezentralen sozialen Netzwerk* verhält sich das anders. Fällt ein Peer aus, können die Nutzer über andere Instanzen trotzdem weiterhin auf aktive Peers des gesamten Netzwerks zugreifen. Dabei ist allerdings eine erneute Registrierung auf einer weiteren Instanz notwendig. Zudem ist die Kontrolle über Daten auf die einzelnen Peers verteilt anstatt das die gesamten Daten des Netzwerks zentral unter Kontrolle einer Entität stehen. Jedes dezentrale, ist auch ein verteiltes Netzwerk und verfügt somit über eine Middleware Schicht zur Kommunikation der einzelnen Instanzen untereinander.

Indirektes Professoren Zitat, wie setzt man das um?

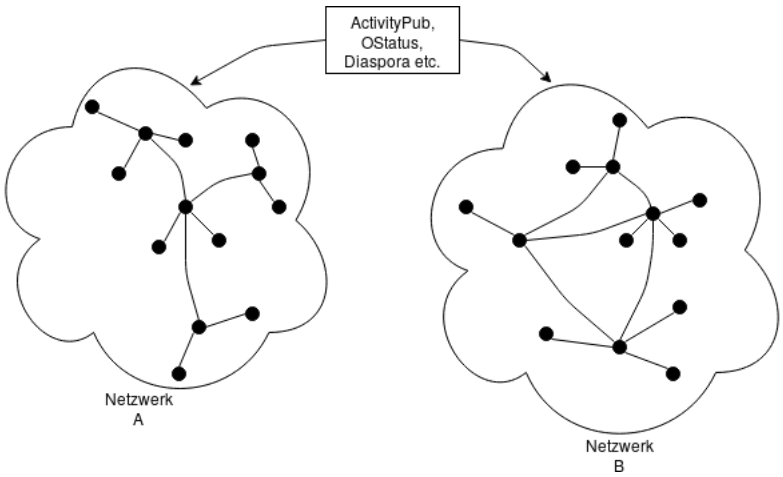


Abbildung 2.3.: Förderieren von Netzwerken über verschiedene Protokolle

Um die Inhalte verschiedener Netzwerke zu verbinden können mehrere soziale Netzwerke zu einem großen verbunden oder „förderiert“ werden. Dies kann über Protokolle und Standards wie OStatus und ActivityPub geschehen oder durch Netzwerkbrücken, im Sinne von Transformatoren, realisiert werden.

Da ein dezentrales Netzwerk zugleich auch ein verteiltes ist, betrachten wir folgende allgemeine Definition verteilter Systeme von Andres S. Tanenbaum und Maarten van Steen:

> *„Ein verteiltes System ist eine Ansammlung unabhängiger Computer, die den Benutzern wie ein einzelnes kohärentes System erscheinen.“*[21]

Ein verteiltes Netzwerk ist demnach eine Ansammlung unabhängiger Computer, in diesem Fall die Instanzen eines sozialen Netzwerks, welche den Benutzern wie ein einzelnes kohärentes Netzwerk erscheinen. Die Instanzen können über das Hypertext Transfer Protocol (HTTP) Protokoll kommunizieren, was somit die Middleware-Schicht darstellt. Es können Inhalte aus mehreren Instanzen zum Darstellen zusammengetragen werden, was der kohärente Aspekt dabei ist.

Das soziale Netzwerk der Firma, für die diese Abschlussarbeit angefertigt wird, hat eine zentrale System Architektur. Die einzelnen Instanzen können also nicht untereinander kommunizieren. Allerdings ist das Netzwerk aus der Sicht her dezentral, als dass jede Person eine eigene Instanz des Netzwerks aufsetzten kann. Zudem ist das Netzwerk bezogen auf die Daten dezentral, da jeder der eine eigene Instanz aufsetzt die Kontrolle über auf dieser generierte Daten hat. Durch die Implementierung des ActivityPub Protokolls wird das System zugleich verteilt und föderiert. Verteilt, weil mit dem Protokoll auch eine Schnittstelle für die Kommunikation der Instanzen des Netzwerks bereitgestellt wird. Förderiert, da diese Schnittstelle auch von anderen Netzwerken implementiert wird und somit die Kommunikation zu Instanzen fremder Netzwerke möglich ist.

2.1.3. Sicherheitsaspekte sozialer Netzwerke

Es gibt einige Sicherheitsaspekte auf die bei der Entwicklung eines sozialen Netzwerks Wert gelegt werden sollte. Dabei sind einige auch optional, da es denkbar ist, dass ein Netzwerk erstellt wird bei dem z. B. keine Authentisierung notwendig ist. Folgend sind einige Aspekte aufgelistet, auf die geachtet werden sollte:

- Die Authentisierung der Nutzer gegenüber dem Server
- Das Sicherstellen der Datenintegrität
- Schutz der Datenbank vor unbefugtem Zugriff
- Filterung (Nutzer bekommt nur das zu sehen, was er sehen darf)

Um die einzelnen Nutzer gegenüber dem Server zu authentisieren, können Technologien wie OAuth[6] oder JSON Web Token[7] verwendet werden. Dadurch ist sichergestellt, das nur derjenige mit korrekten Anmeldedaten das Profil benutzen kann. Außerdem weiß der Server dadurch wie er die Inhalte Filtern muss um dem Nutzer das für ihn erlaubte anzeigen zu können.

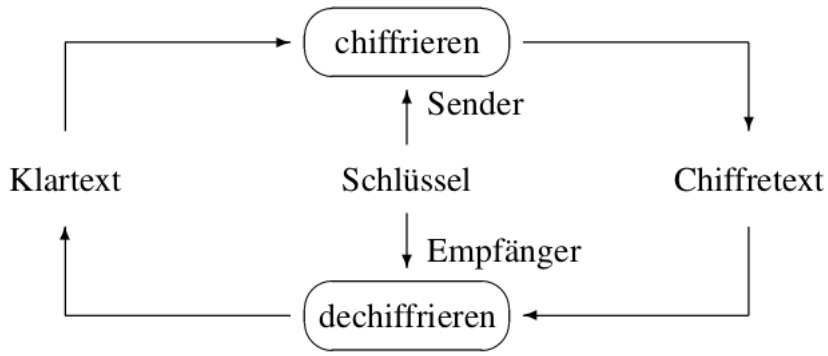
Das Sicherstellen der manipulationsfreien Übertragung von Inhalten ist sowohl bei der Client zu Server, als auch bei der Server-zu-Server Kommunikation zu Empfehlen. Bei beiden Kommunikationsszenarien werden die Daten über das Internet übertragen und sind somit potentiell manipulierbar. Sichergestellt werden kann dies z. B. über HTTP-Signaturen s. 2.2.3.

[6] siehe `https://tools.ietf.org/html/rfc6749`
[7] siehe `https://tools.ietf.org/html/rfc7519`

Ein weiterer wichtiger Sicherheitsaspekt ist das Schützen der Datenbank vor unbefugtem Zugriff. Bei einem sozialen Netzwerk kann man die Datenbank durchaus als das Herzstück des Netzwerkes bezeichnen, da in dieser, gerade bei „sozialen Netzwerken“ eine riesige Menge an Daten liegen. Schaffen es Angreifer auf diese Zugriff zu bekommen und unbemerkt zu bleiben, kann die Datenbank in Ruhe analysiert oder womöglich die Inhalte auf einen eigenen Server transferiert werden. Durch das isolieren der Datenbank, restriktiven oder keinen direkten Zugriff durch Nutzer oder eine Mehrfaktor Authentisierung kann die Sicherheit der Datenbank erhöht werden.

2.2. Kryptographie

„Unter dem Begriff Kryptographie ist die Wissenschaft vom geheimen Schreiben zu verstehen“ (Vgl. [24, S. 1]). Man spricht von symmetrischer Verschlüsselung wenn eine Nachricht im Klartext vom Sender mit einem geheimen Schlüssel, welcher beiden Parteien bekannt ist, verschlüsselt und vom Empfänger, mit demselben Schlüssel, entschlüsselt wird (Vgl. [24, S. 1]). Von asymmetrischer Verschlüsselung ist die Rede, wenn die Teilnehmer



Quelle: [24, S. 1]

Abbildung 2.4.: Symmetrische Ver- und Entschlüsselung

anstatt einen gemeinsamen Schlüssel zu haben, der im vor hinein ausgetauscht werden muss, jeder ein Schlüsselpaar, bestehend aus öffentlichem und privatem Schlüssel, besitzt.

2.2.1. RSA

Das Rivest-Shamir-Adleman (RSA) Verfahren ist ein solches asymmetrisches Verschlüsselungsverfahren welches von den drei namens gebenden Mathematikern 1977 entwickelt wurde (Vgl. [24, S. 77]). Das RSA Verfahren ist wohl das am häufigsten verwendete Public-Key-Kryptosystem.

Beispiel 1. Die Gesprächspartner Alice und Bob, welche beide ein Schlüsselpaar besitzen, wollen miteinander kommunizieren. Alice verschlüsselt einen Nachrichtentext mit

dem öffentlichen Schlüssel von Bob und sendet die verschlüsselte Nachricht an Bob. Dieser kann seinerseits mit seinem privaten Schlüssel die Nachricht entschlüsseln (Vgl. [24, S. 73 f.]).

2.2.2. Signaturen

Für die Sicherstellung der Authentizität können sogenannte Signaturen verwendet werden. Das oben kurz erläuterte RSA Verfahren kann nicht nur zum Ver- und Entschlüsseln von Nachrichten benutzt werden, sondern auch zum signieren. Dabei wird statt des öffentlichen Schlüssels, der private Schlüssel, zusammen mit einer Hashfunktion, benutzt um eine Signatur zu erzeugen. Diese kann dann mit dem öffentlichen Schlüssel des zugehörigen privaten Schlüssels verifiziert werden.

Beispiel 2. Bob möchte eine Nachricht an Alice schicken und sichergehen, dass diese auf dem Weg nicht verändert wurde. Er verwendet seinen privaten Schlüssel und wendet diesen auf eine Nachricht an um eine Signatur zu erzeugen. Beides übermittelt er an Alice. Mit dem öffentlichen Schlüssel von Bob kann die fehlerfreie Übertragung der Nachricht verifiziert werden.

Unter einer Hashfunktion versteht man eine Einwegfunktion welche auch zur Signierung verwendet werden kann. Bei solch einer Funktion wird ein Eingangswert auf eine kryptische Zeichenfolge abgebildet. Dies wird sehr oft bei Passwörtern verwendet um diese nicht Umkehrbar aufzubewahren.

2.2.3. HTTP Signaturen

Eine *HTTP Signatur* wird verwendet um die Authentizität sicherzustellen. Diese wird als Wert einer „Signature“ Kopfzeile eingetragen und besteht aus mehreren Teilen:

- ***keyId***=”https://example.org/activitypub/users/lea#main-key”
- ***algorithm***=”rsa-md4”
- ***headers***=”(request-target) date host content-type”
- ***signature***=”DHeEH0Okmtf1ec/lbM1/F5FiLVfQfbWuoFf9t/TzNZiZ7ak”

Über die *keyId*, was eine Referenz auf einen Schlüssel darstellt, kann der öffentliche Schlüssel angefragt werden. Dies wird beim verifizieren einer Signatur benötigt um die übertragenen Daten auf ihre Authentizität hin zu prüfen.

Welcher Hashing-Algorithmus bei der Erstellung verwendet wurde, kann über das *algorithm* Feld der Signatur nachgeschlagen werden. Zudem muss der Algorithmus bei Erstellung in dieses Feld zum Nachschlagen eingetragen werden.

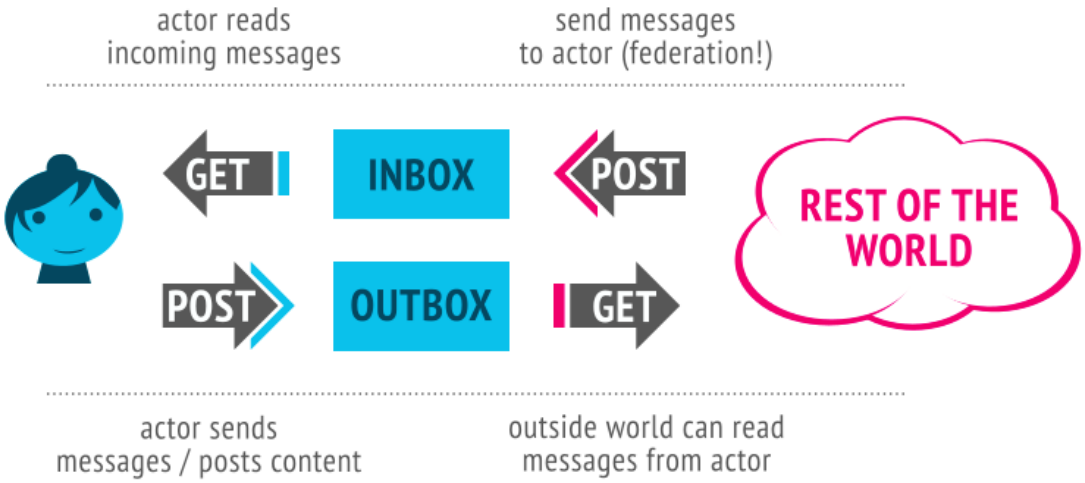
Um die eigentliche Signatur zu erzeugen werden die in *headers* angegebenen Kopfzeilen der HTTP Anfrage verwendet. Somit kann eingeschränkt werden welche Metadaten in die Signierung einfließen.

Die bei der Signierung mit gegebenem Hashing-Algorithmus und Kopfzeilen erzeugte Signatur wird in das *signature* Feld der HTTP Signatur eingetragen sowie die HTTP Signatur an sich als „Signature“ Kopfzeile der HTTP Anfrage gesetzt (Vgl. [4]).

2.3. ActivityPub Standard

Der ActivityPub Standard wurde am 23. Januar 2018 von der W3C empfohlen (Vgl. [14]) und von einer Arbeitsgruppe des W3C, der SWWG (Vgl. [2, 3]), entwickelt. Diese Gruppe war vom 21. Juli 2014 bis zum 13 Februar 2018 aktiv[2] und entwickelte unter anderem ActivityPub, ActivityStreams Core (AS2-C) (Vgl. [18]) und ActivityStreams Vocab (AS2-V)(Vgl. [17]). Die SWWG war eine Arbeitsgruppe des W3C mit dem Ziel neue Protokolle, Vokabulare und Application Programming Interface (API)’s zu definieren für den Zugriff auf soziale Inhalte der sogenannten Open Web Platform (OWP) (Vgl. [23]).

ActivityPub definiert zwei Protokollschichten, sowie Konzepte, Sammlungen und Interaktionen für dezentrale soziale Netzwerke. Eine Protokollschicht ist das Client-zu-Server Protokoll (Social API), um Clients den Zugriff auf die neusten an sie gesendeten Inhalte zu ermöglichen sowie zum entgegennehmen von Anfragen die vom Client abgesetzt wurden (Vgl. [14]). Die zweite Protokollschicht besteht aus dem förderierten Server-zu-



Quelle: ActivityPub 2018 - Overview

Abbildung 2.5.: Schnittstellen des ActivityPub Protokolls

Server Protokoll (Federation Protocol), welches den einzelnen Instanzen von dezentralen sozialen Netzwerken den Austausch von Inhalten untereinander gestattet. ActivityPub setzt auf bereits bestehende Empfehlungen des W3C auf, welche teilweise auch von der

SWWG entwickelt wurden wie z. B. AS2-C und AS2-V (Vgl. [14]). Die zwei Protokollschichten können unabhängig voneinander implementiert werden.

Auch andere Technologien wie JSON Linked Data (JSON-LD) werden verwendet um die Erweiterbarkeit zu gewährleisten. Über neue Ontologien und Vokabulare können weitere syntaktische Definitionen und semantische Beschreibungen zu den bestehenden hinzugefügt werden (Vgl. [14]). Diese Vokabulare können im Kontext des JSON-LD Objektes, angegeben werden. Bei ActivityPub wird das ActivityStreams 2.0 (AS2) Vokabular verwendet welches durch AS2-V erweitert wird.

2.3.1. Bestandteile des Protokolls

Die Hauptbestandteile des ActivityPub Standards sind die folgenden:

- Aktoren
- Objekte
- Sammlungen
- Aktivitäten

In ActivityPub werden Benutzer als „Aktoren“(actors) dargestellt (siehe Abb. **??**). Diese können nicht nur Personen, sondern auch Applikationen, Organisationen, Gruppen und Services sein (Vgl. [18]). Jedes Aktoren Objekt muss eine „Inbox“ und „Outbox“, welche geordnete Sammlungen sein müssen, sowie einen Identifikator und ein Typ besitzen (Vgl. [14]). Der Identifikator muss global einzigartig sein. Dies kann garantiert werden durch eine Domänen und Protokoll bezogene URI oder IRI wie z. B. „https://example.org/users/alice“ oder „https://example.org/users/álìcê“. Der Typ eines Aktor (z. B. ”type”: ”Person”) kann variieren zwischen den fünf oben genannten. Ein Beispiel Aktoren Objekt kann auf der nächsten Seite (Abb. **??**) begutachtet werden. Dabei fällt auf das im Kontext des JSON Linked Data (JSON-LD) Objektes zwei Einträge zu finden sind. Der erste erweitert das Objekt um die Funktionalität des AS 2.0 Vokabulars und der zweite fügt dem Objekt sicherheitsrelevante Syntax und Semantik hinzu. Der benötigte Identifikator ist hier ein Uniform Ressource Identifier (URI). Das Aktoren Objekt bildet eine Person mit Namen macu und verschiedenen benötigten sowie optionalen Sammlungen ab. Desweiteren enthält das Objekt eine Beschreibung des Profils (summary) und ein Link zu diesem (url). Aus dem Sicherheitsvokabular wird das „publicKey“ Attribut genutzt. Der Wert ist ein Objekt eines öffentlichen Schlüssels mit einem Identifikator, dem Besitzer und dem eigentlichen Schlüssel im ASCII Format.

2.3.2. Zugehörige Standards und Komponenten

Am 22. April 2016 hat die „W3C Community Group“ einen Entwurfsbericht herausgebracht. Durch diesen wird neue Syntax und Semantik definiert um Internet basierten Applikationen das Verschlüsseln, Entschlüsseln sowie digitale Signieren und Verifizieren von

```
{
  "@context": [
    "https://www.w3.org/ns/activitystreams",
    "https://w3id.org/security/v1"
  ],
  "id":"https://human-connection.org/activitypub/users/macu",
  "type":"Person",
  "following":"https://human-connection.org/activitypub/users/macu/following",
  "followers":"https://human-connection.org/activitypub/users/macu/followers",
  "inbox":"https://human-connection.org/activitypub/users/macu/inbox",
  "outbox":"https://human-connection.org/activitypub/users/macu/outbox",
  "preferredUsername":"macu",
  "name":"Marco Curosa",
  "summary":"This is my Profile!",
  "url":"https://human-connection.org/activitypub/@macu",
  "publicKey":{
    "id":"https://human-connection.org/activitypub/users/macu#main-key",
    "owner":"https://human-connection.org/activitypub/users/macu",
    "publicKeyPem":"-----BEGIN PUBLIC KEY-----\
        nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAvGOWsb/tTRCd5BcZOZYa\
        nqQ6os0vMJmOeg9kNvB5KXZhZwiwjTbXf7VEshUoDnoYUAF1jFnhfP5ChODVq9r9X\nS+
        Wx3o65vjdhjXTrZrVHicTFoB4RaC9sCEHfTinR8ywewzo7VrEVQ2KnbeXzzf+v\na+
        Dr2h1qkwNv9E9QJ5Mq6RfCKdoFzOFhPqdFOKmgCPuaUOnzgirKoLkwJz0Qe8ky\
        neFDNT4uQKoFPmjXt3tl5ZMlR/0FEuk9Vn0l+rrLXqNUrsq7AINoY3rZf97d/CMDy\
        nChNbN/6N9B1xFN+AvQSAoLWjbD0Zrkm3vy6/dnyDcgXQmrpR0selWQvV/PgbeSgP\
        nhQIDAQAB\n-----END PUBLIC KEY-----\n"
  },
  "tag":[],
  "attachment":[],
  "endpoints":{
   "sharedInbox":"https://human-connection.org/activitypub/inbox"
  }
}
```

Abbildung 2.6.: Beispiel Aktoren Objekt

verlinkten Daten (Linked Data) zu ermöglichen. Es enthält auch Vokabeln für die Erstellung und Verwaltung einer dezentralen Public-Key-Infrastruktur über das Internet (Vgl. [16]). Ein Anwendungsfall ist das holen des öffentlichen Schlüssels eines Nutzers, über dessen Aktoren Objekt, um eine vom Nutzer gesendete Nachricht zu verifizieren. ActivityPub benutzt die AS2 Daten Syntax und das Vokabular. Zusätzlich kann ein weiteres, z. B. das im vorherigen Absatz erwähnte, Sicherheitsvokabular[8] benutzt werden. Auch in Abbildung 3.6 wurde obig beschriebenes Vokabular genutzt.

2.3.2.1. ActivityStreams 2.0

„AS2“ beinhaltet Modelle für Aktoren, Aktivitäten, Intransitiven Aktivitäten, Objekte, Links, Sammlungen, Natürliche Sprachwerte (Strings) und für Internationalisierung. Das Kernvokabular von AS2 wird durch AS2-V erweitert. Dazu gehören verschiedene Aktivitätstypen wie z.B. „Accept“, „Add“, „Remove“, „Delete“ und „Create“, um Aktorentypen

[8]Eine Ontologie die Sicherheitsaspekte definiert wie öffentliche Schlüssel, Signaturen u.v.m.

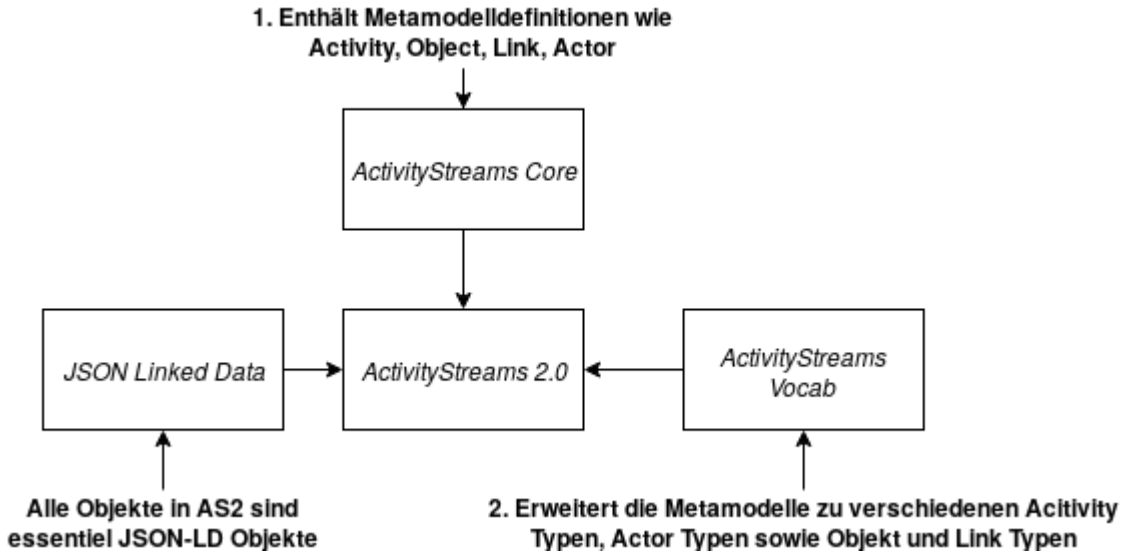


Abbildung 2.7.: AS2 Komponenten

wie „Person“, „Application“ und „Group“ sowie um verschiedenste Objekttypen wie „Article“, „Event“, „Note“ und „Relationship“[9].

Objekte werden innerhalb von AS2 in Aktivitäten eingehüllt um mitteilen zu können das ein Objekt, wie z. B. „Article“, erstellt wurde. Dies geschieht mit der „Create“ Aktivität. Der Mittelpunkt bilden somit Objekte, welche die eigentlichen Inhalte repräsentieren wie Audio oder Video Dateien, Bilder, Artikel und mehr. Die umhüllenden Aktivitäten sind Metadaten um den Objekten erweiterte semantische Informationen zu geben.

In den Abbildungen **??** und **??** sind zwei Beispiel AS2 Objekte abgebildet um die Struktur solcher zu veranschaulichen.

```
1 {
2   "@context": "https://www.w3.org/ns/activitystreams",
3   "type": "Note",
4   "id": "http://example.org/note/124",
5   "summary": "A note by Sally",
6   "content": "Everything is OK here."
7 }
```

Abbildung 2.8.: Beispiel Notiz Objekt

Notizobjekte werden eher für kürzere Texte verwendet. Das prinzipiell wichtigste bei einem Notizobjekt ist der Identifikator sowie der Inhalt. Außerdem kann wie im nachfol-

[9] siehe `https://www.w3.org/TR/activitystreams-vocabulary/`, 3

genden Beispiel noch ein veröffentlichungs Attribut, sowie weitere, angegeben werden[10].

```
{
  "@context": "https://www.w3.org/ns/activitystreams",
  "type": "Article",
  "id": "http://example.org/note/124",
  "summary": "A article by Sally",
  "content": "Lorem ipsum dolor sit amet. Lorem ipsum dolor ist amet.",
  "published": "2019-03-14T21:25:40.081Z"
}
```

Abbildung 2.9.: Beispiel Artikel Objekt

Artikel bilden, im Gegensatz zu Notizen, längere Texte ab. Hier ist, wie oben angedeutet, das Veröffentlichungsdatum mitangegeben.

2.3.2.2. JSON Linked Data

Etwas genauer beschreiben

JSON-LD ist eine Erweiterung des JSON Formates um verlinkte Daten zu Repräsentieren. JSON an sich, ist ein Format welches im Web häufig Anwendung findet um Daten auszutauschen. Im Kern sind AS2 auch JSON-LD Objekte. Der AS2 Kontext definiert verschiedene Klassen und Eigenschaften, von denen nicht alle benutzt werden. Typische Klassen sind „Activity“, „Link“ und „OrderedCollection“.

2.3.2.3. WebFinger

Beim OStatus Protokoll wird zum Entdecken von Profilen für z. B. eine Suchfunktion das WebFinger Protokoll benutzt. Auch dieses Framework verwendet, obwohl der Standard das nicht festlegt, das WebFinger Protokoll. Dabei wird ein JSON Ressource Descriptor (JRD) bereitgestellt, welcher einen Link auf das Aktoren-Objekt enthält. Über diesen kann eine AcitivtyPub konforme Repräsentation eines Nutzern (Aktoren-Objekt) erhalten werden. Beispielsweise wird WebFinger bei Mastodon benutzt um Nutzer zu suchen.

2.3.3. Authentisierung und Datenintegrität

Für die Authentisierung und zum sichern der Datenintegrität definiert der Standard keine Mechanismen. Es gibt allerdings „Best Practices“ für die Umsetzung dieser Anforderungen.

Zum einen werden bei der Client-zu-Server Authentisierung „OAuth 2.0 Bearer Tokens“ benutzt, zum anderen, auf der Server Seite, „HTTP“ oder „Linked Data Signatures“ zur

[10] siehe `https://www.w3.org/TR/activitystreams-core/`, 4.1

Sicherstellung der Datenintegrität.
Bei„OAuth 2.0 Bearer Tokens“ handelt es sich um eine Methode um auf geschützte Ressourcen zugreifen zu können (Vgl. [15]). ActivityPub nutzt diese für jegliche Interaktionen mit dem Server.

„Die Datenintegrität umfasst Maßnahmen damit geschützte Daten während der Verarbeitung oder Übertragung nicht durch unautorisierte Personen entfernt oder verändert werden können. Sie stellt die Konsistenz, die Richtigkeit und Vertrauenswürdigkeit der Daten während deren gesamten Lebensdauer sicher und sorgt dafür, dass die relevanten Daten eines Datenstroms rekonstruierbar sind“ (Vgl. [**data-integrity**]).

Um sicherzustellen das HTTP Anfragen beim Transport nicht verändert wurden, können HTTP Signaturen verwendet werden. Diesen verwenden einen kryptografischen Algorithmus um aus ausgewählten Kopfzeilen einer HTTP Anfrage eine kryptischen Zeichenfolge zu generieren. Auf der Empfängerseite kann die Zeichenfolge mit mitgelieferten und auch nachschlagbaren Information verifiziert werden.

Wenn ein Objekt nicht nur vom Client zum Server gesendet, sondern auch zwischen Servern untereinander weitergeleitet werden soll wird zum Sicherstellen der Datenintegrität ein anderes Verfahren benötigt als HTTP Signaturen. Die „Best Practices“ empfehlen für solche Fälle „Linked Data Signatures“. Der größte Unterschied zwischen HTTP Signaturen und „Linked Data Signatures“ besteht darin, welche Daten zum Erstellen der Signatur verwendet werden. Bei HTTP Signaturen sind es die Kopfzeilen. Mit „Linked Data Signatures“ kann auch das Objekt selbst, also der Payload einer HTTP Anfrage, anstatt nur die Kopfzeilen, zum signieren verwendet werden.

3. Entwurf einer Lösung für Server-zu-Server Interaktion mit ActivityPub

Zu Beginn dieses Kapitels wird auf die Anforderungen, welche an das Framework gestellt werden eingegangen. Im zweiten Unterkapitel wird dann die Entwurfsentscheidung besprochen und dabei folgende Frage beantwortet:

- Warum wurde die Architektur gewählt?

Das dritte Unterkapitel gibt Aufschluss über die konkrete technische Architektur. Diese wird anschließend anhand von Diagrammen veranschaulicht und erläutert.

3.1. Anforderungen an das Framework

An das Framework werden verschiedene Anforderungen gestellt. Zum einen soll es möglichst Entwicklerfreundlich erstellt werden, sodass Entwickler, welche das Framework benutzen möchten um Ihre Applikation ActivityPub konform zu machen, es möglichst einfach haben dieses zu Integrieren oder darauf aufzubauen. Dabei wird auf das Codeverständnis Wert gelegt sowie auf die Schaffung eines einzelnen Interfaces.

Durch das Erstellen von Nutzern passend zum verwendeten Datenschema soll zudem verhindert werden, dass weitere Mutationen, Anfragen sowie Entitäten für Relationen zwischen Schema-Entitäten angelegt werden müssen. Die Änderungen am vorhandenen Datenschema sollen somit so gering wie möglich gehalten werden.

Eine weitere Anforderung ist die Integrationsfähigkeit in bestehende Applikationen sowie die Möglichkeit das Framework als Grundlage einer neuen Applikation zu nutzen.

Auch die Nutzung verschiedener Datenquellen soll ermöglicht werden durch die Erstellung des Interfaces auf Ebene der Datenschicht. Die Controller- und Serviceschicht sollen im besten Falle nicht geändert werden müssen; Lediglich beim hinzufügen neuer Funktionalität sollen diese beiden Schichten angepasst werden müssen.

3.2. Entwurfsentscheidung

Aus Gründen der Einfachheit wurde sich in dieser Arbeit für eine drei Schichten Architektur, bestehend aus Controller-, Service- und Datenschicht, entschieden. Hierbei wird die

letzte Schicht implementiert und ist somit die Schnittstelle für Entwickler zum integrieren des Frameworks in eine bestehende Applikation. Es wurde sich für die Datenschicht (IDataSource) entschieden um eine breite Fläche an Datenbanken und API's ansprechen zu können; Abhängig davon was bereits besteht.

Um außerdem Entwicklern zu ermöglichen auch ohne bestehenden Express Web Server, in den das Framework integriert wird, mit dem implementieren zu beginnen kann der Service ohne Parameter initialisiert werden und startet damit einen separaten Web Server.

Ein besseres Codeverständnis wird erlangt durch die Erstellung eines Fassaden Objekts, welches Weiterleitungen zum Interface enthält. Dabei handelt es sich um Funktionalität welche mit Sammlungen zu tun hat. Das Objekt heißt dementsprechend „Collections".

Außerdem wurde sich für das Erstellen von Nutzern für jeden Aktor des Fediverse, welche mit dieser Instanz kommuniziert, entschieden, da dadurch die automatische Erstellung von Relationen erhalten bleibt. Somit entfallen größere Änderungen am verwendeten Datenschema.

3.3. Technische Architektur

Die technische Architektur ist in drei Schichten aufgeteilt. Der *Controller* hat die Aufgabe HTTP Anfragen entgegen zu nehmen. In der *Service* Schicht bekommen alle Methoden eine Aktivität als Parameter übergeben. Zudem nutzen alle Service Methoden die IDataSource Implementierung, welche die *Datenschicht* darstellt.

Der ActivityPub Service wird als Express Middleware, sowie als allein stehender Web Server bereitgestellt. Bei der letzteren Variante wird anstatt die Middleware in ein bestehenden Express Server zu integrieren, ein eigener Server gestartet.

In der oben gezeigten Abbildung wurde der ActivityPub Service in einen bestehenden Express Server einer GraphQL API integriert.

Benutzer der API kommunizieren mit dem Web Server über das HTTP Protokoll. Der Web Server leitet die Anfragen dann an den entsprechenden Router weiter, welcher wiederum den Anfrage Inhalt transformiert und „Handler" Funktionen der Service Schicht, mit entsprechenden Parametern, aufruft.
Die Abbildung 3.2 zeigt eine detailliertere Variante des ActivityPub Service Diagramms aus Abb. 3.1.

Der Router stellt hier die Controller Schicht dar bei der die HTTP Anfragen der Nutzer eingehen. Die ActivityPub Komponente ist der eigentliche Service und stellt zudem die gleich benannte Schicht dar (Serviceschicht).

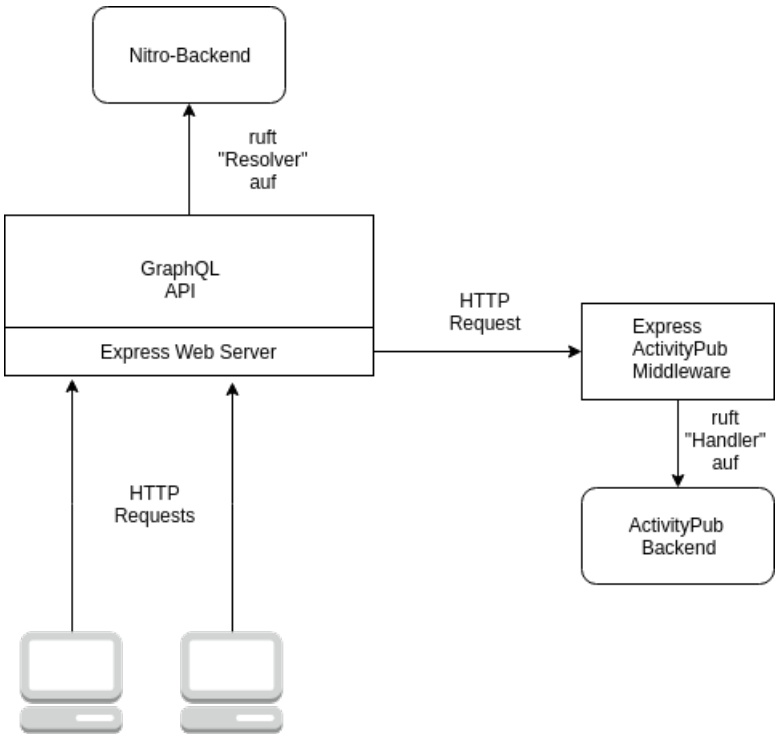


Abbildung 3.1.: Technische Architektur ActivityPub

Bei der Architektur wird weitestgehend darauf Wert gelegt, dass alle ActivityPub konformen Inhalte dynamisch generiert werden können. Damit wird vermieden eine weitere Datenbank extra für den Service einrichten zu müssen.

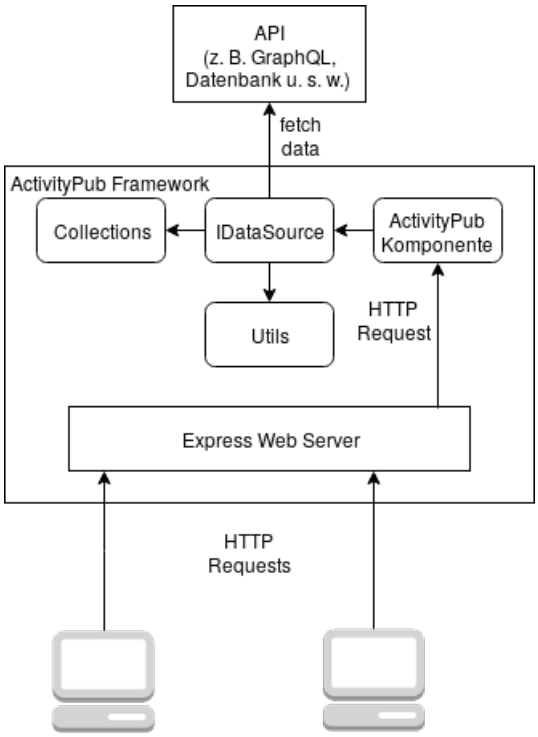


Abbildung 3.2.: Technische Architektur als allein stehender Server

4. Implementierung eines ActivityPub Prototyps

In diesem Kapitel wird die konkrete Implementierung des Prototypen, welcher für das Unternehmen angefertigt wurde in der die Abschlussarbeit bearbeitet wird, erläutert. Zu Beginn wird die Abfragesprache „GraphQL“ sowie die genutzte Graph-Datenbank des Projekts vorgestellt, für welches die Implementierung angefertigt wird. Danach gibt ein Klassendiagramm der Service- und Datenschicht Aufschluss über die in diesen Objekten enthaltene Funktionalität welche darauf folgend genauer beschrieben wird. Anschließend werden anhand der Verzeichnisstruktur die Controller-Schicht, sowie die Werkzeug und Sicherheitsfunktionalitäten beschrieben. Es werden zusätzlich auch die genutzten Testwerkzeuge und zwei Vorgehensweisen der Softwareentwicklung kurz erläutert. Unterkapitel 4.6 gibt Aufschluss über die implementierte Funktionalität des ActivityPub Standards. Zuletzt wird ein genauerer Blick auf die Signierung sowie Verifikation geworfen.

Im Jahre 2015 wurde eine Abfragesprache und Laufzeitumgebung namens GraphQL veröffentlicht, welche schon seit 2012 von mobilen Facebook Applikationen verwendet wird und somit auch von diesem Unternehmen entwickelt wurde. GraphQL nutzt zum Beschreiben der Daten ein Typsystem. Das Datenschema wird über eine Schema-Datei angegeben (Vgl. [19]). Der Vorteil gegenüber einer HTTP- oder REST-API liegt in der Nutzung eines einzelnen Endpunktes und einem Typsystem. Damit entfallen viele HTTP Anfragen, da sich mehrere Datensätze verschiedenen Typs bequem über eine Anfrage anfragen lassen. Möchte man beispielsweise für eine Statistik alle Nutzer sowie Artikel erhalten, kann dies in einer einzigen HTTP Anfrage an die GraphQL API erledigt werden. Bei einer REST-API müssen im schlimmsten Falle zuerst alle Identifikatoren abgerufen werden um mit diesen die eigentlichen Nutzer und Artikel abzufragen.

Die GraphQL-Schnittstelle kann verschiedene relationale-, nicht relationale sowie Graph-Datenbanken nutzen. Das Projekt verwendet eine Graph-Datenbank mit dem Namen „Neo4j“. Im Allgemeinen besteht ein Graph aus Knoten und Kanten wobei ersteres z. B. einen Nutzer mit verschieden Eigenschaften wie Passwort und Email darstellt sowie eine Kante Relationen zwischen Knoten repräsentiert wie z. B. Nutzer A ist befreundet mit Nutzer B. Speziell für das Repräsentieren und Abfragen von Relationen ist eine Graph-Datenbank bestens geeignet. Außerdem bleibt die Performanz der Abfragen konstant auch wenn die in der Datenbank enthaltene Menge an Daten steigt (Vgl. [13]).
Abbildung **??** zeigt ein Klassendiagramm mit zwei Hauptkomponenten des Frameworks; Dies sind die ActivityPub und IDataSource Objekte. Dabei enthält das Collections Objekt das Interface, sowie dieses das ActivityPub Objekt enthält.

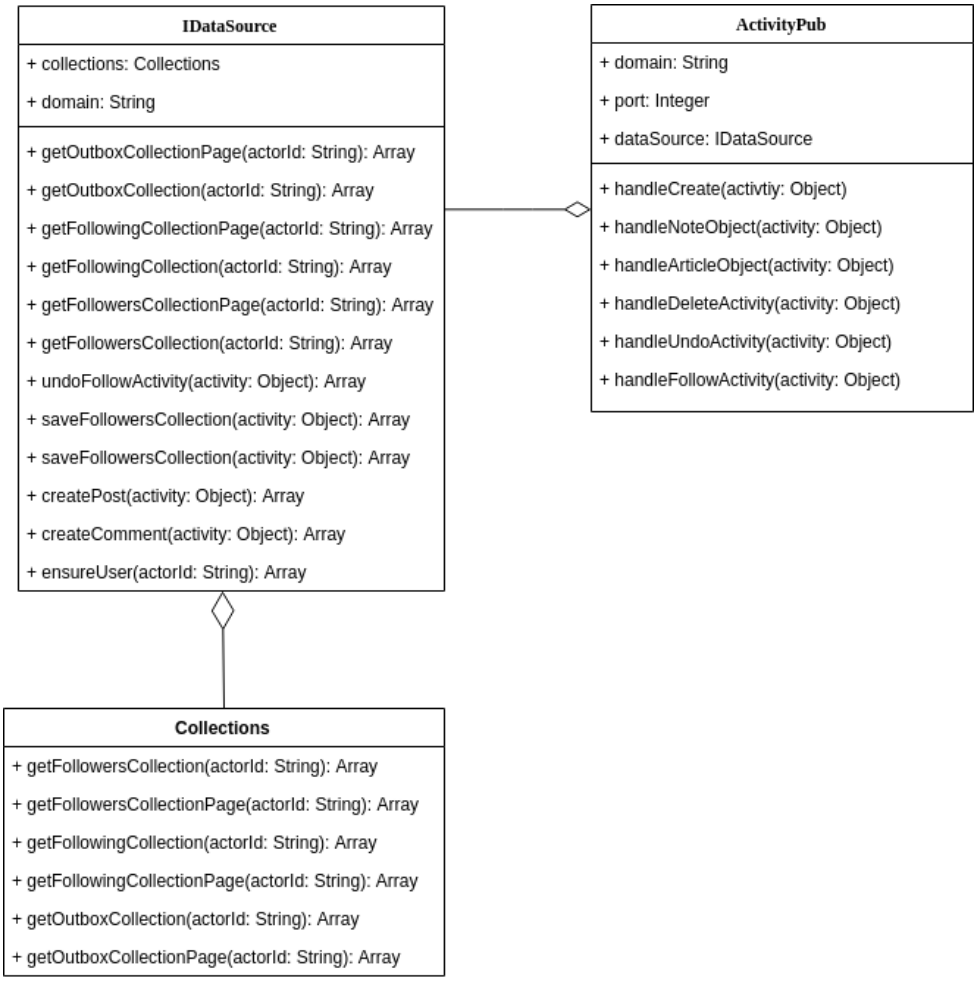


Abbildung 4.1.: Hauptkomponenten des förderierten Servers

Das „Collections“ Objekt ist ein Fassaden Objekt[1] und enthält Funktionen die auf das IDataSource Interface zugreifen um folgende Sammlungen zu erhalten:

- Followers Collection
- Following Collection
- Outbox Collection

Das Fassaden Objekt dient ausschließlich dem Codeverständnis für Entwickler.

Beim ActivityPub Objekt handelt es sich um das Hauptobjekt der Implementierung. Die entsprechenden Anfrage-Handler der Controller-Schicht wandeln die Anfrage in einen Service Aufruf um. Dieses Objekt enthält nicht nur Funktionen zum Verarbeiten eingehender Anfragen, sonder auch Funktionalität zum Senden von Aktivitäten.

[1] Abgeleitet von Fassaden Klasse

Das Hauptobjekt aus Sicht der Entwickler ist allerdings das IDataSource Interface. In diesem stehen zu implementierende Funktionsrümpfe bereit. Es enthält Rümpfe zum Beziehen und Speichern von Sammlungen, Erstellen und Speichern von Objekten sowie geteilten Nachrichteneingängen. Weiter sind Rümpfe zum Beziehen der beiden Schlüsselpaar-Komponenten und dem Identifikator eines Aktors vorhanden. Um prüfen zu können ob ein Nutzer bereits existiert und um einen solchen anzulegen stehen außerdem Rümpfe bereit.

4.1. Verzeichnisstruktur

Die nachfolgende Abbildung zeigt die Verzeichnisstruktur des Frameworks welche in den folgenden Unterkapiteln beschrieben wird und die enthaltene Funktionalität erläutert wird.

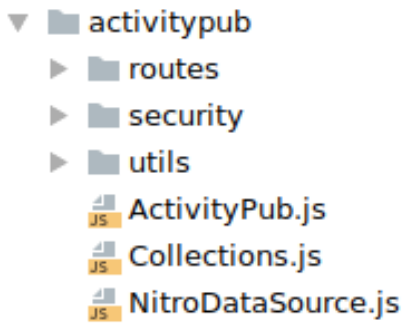


Abbildung 4.2.: Ordnerstruktur des Prototypen

4.1.1. Inhalt routes Verzeichnis

Um zu veranschaulichen wie die einzelnen Teile der Controller-Schicht zusammengesetzt werden und welche zusätzliche Funktionalität jedem Controller anzufügen ist, wird die Abbildung am Anfang der nächsten Seite betrachtet. Doch zuvor wird anhand der folgenden Abbildung gezeigt welche Dateien sich innerhalb des „routes“ Verzeichnisses befinden:

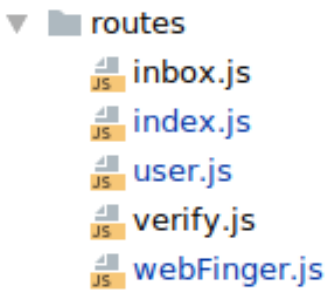


Abbildung 4.3.: Dateien im „routes“ Verzeichnis

Alle Controller befinden sich im *router* Ordner. Jede Datei beinhaltet einen Express Router, welche alle in der *index.js* Datei zu einem einzigen zusammengefasst werden (siehe Abb. 4.4). Dabei nutzt jeder Router zusätzlich die Express Cross-Origin Ressource Sharing (CORS) Middleware um entsprechend die HTTP Verben *Post*, *GET* oder beide für den Zugriff von anderen Domains freizugeben. Außerdem wird bei allen Nutzer betreffenden Routen, sowie beim geteilten Nachrichteneingang ein Parser genutzt um Anfrage Inhalte mit den drei gegebenen Inhaltstypen (*application/json, application/ld+json, application/activity+json*) in Objekte umzuwandeln und eine weitere Middleware für das Parsen der Uniform Ressource Locator (URL) Parameter. Bei eingehenden Nachrichten, welche Daten verändern wird des weiteren ein Router benutzt um die HTTP Signaturen zu verifizieren (*verify.js*).

```
1  const router = express.Router()
2
3  router.use('/.well-known/webfinger',
4    cors(),
5    express.urlencoded({ extended: true }),
6    webFinger
7  )
8  router.use('/users',
9    cors(),
10   express.json({ type: ['application/activity+json', '
       application/ld+json', 'application/json'] }),
11   express.urlencoded({ extended: true }),
12   user
13 )
14 router.use('/inbox',
15   cors(),
16   express.json({ type: ['application/activity+json', '
       application/ld+json', 'application/json'] }),
17   express.urlencoded({ extended: true }),
18   verify,
19   inbox
20 )
21
22 export default router
```

Abbildung 4.4.: Wie die einzelnen Teile zusammengefügt werden

4.1.2. Inhalt utils Verzeichnis

Zu Beginn dieses Unterkapitels wird anhand folgender Abbildung der Inhalt des „utils“ Verzeichnisses dargestellt:
Der „utils“ Ordner beinhaltet verschiedene Dateien welche Funktionen mit dem Namen entsprechender Funktionalität exportieren um dies zu bewerkstelligen. In der Datei *collection.js* sind Funktionen zum Erstellen und Senden von Sammlungen zu finden. Die Datei *actor.js* beherbergt Funktionen zum Erstellen von Aktoren Objekt und WebFinger Datensatz. Funktionalität zum Erstellen, Beantworten und Validieren von Aktivitäten befindet

Abbildung 4.5.: Dateien im „utils“ Verzeichnis

sich in der *activity.js* Datei. Jegliche weiteren Hilfsfunktionen sind in der *index.js* Datei. Die zuletzt genannte Datei exportiert insgesamt 6 Funktionen. Zwei davon Erstellen Objekte (Note und Article) durch das einfügen von Werten in ein Objekt Literal. Als Parameter wird den Funktionen jeweils eine activity- sowie objectId, der Inhalt, Name des Absenders sowie das Veröffentlichungsdatum übergeben. Eine weitere ist mit dem Namen des gewünschten Aktors parametrisiert für welchen der Identifikator erhalten werden soll. Des weiteren sind zwei Funktionen zum Senden von *Accept* und *Reject* Aktivitäten enthalten welche beide jeweils das akzeptierte oder abgelehnte Objekt, den Namen des Empfängers, sowie die Domain und URL an welche die Anfrage gesendet wird, als Parameter übergeben bekommen. Die letzte exportierte Funktion gibt einen Wahrheitswert zurück, welcher angibt ob das Notiz oder Artikel Objekt an die Öffentlichkeit gesendet wird.

4.1.3. Inhalt security Verzeichnis

Wie in der folgenden Abbildung gezeigt, befindet sich im „security“ Verzeichnis sowohl eine Index Datei als auch eine Testdatei um die Funktionalität, welche die Indexdatei beinhaltet, auf korrekte Funktionsweise zu prüfen. Anders als bei den Akzeptanztests wird hier Jest[2] als Test Framework verwendet.

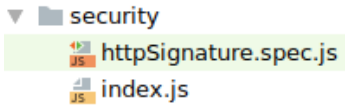


Abbildung 4.6.: Dateien im „security“ Verzeichnis

Bei ausgehenden Anfragen wir Funktionalität benötigt um Signaturen zu erstellen welche im *security* Ordner bereitgestellt wird. Nicht nur zur Signierung ausgehender, sonder auch zum Verifizieren eingehender Anfragen ist eine Funktion vorhanden. Signiert sowie Verifiziert werden können die Anfragen mit verschiedenen bekannten Hashfunktionen, wie z.B. *SHA-256*, *MD5* oder *SHA-512*, in Kombination mit einem Schlüsselpaar eines öffentlichen Schlüssel Verfahrens. Genutzt wird dazu die in Node.js enthaltene *crypto* Bibliothek.

Die *index* Datei im *security* Ordner exportiert drei Funktionen und ein Array mit unterstützten Hashfunktionen. Wird ein RSA Schlüsselpaar benötigt kann dieses mit der

[2] s. `https://jestjs.io/`

generateRsaKeyPair Funktion generiert werden. Diese bekommt als Parameter ein Options Objekt mit *passphrase* Eigenschaft übergeben; Es genügt auch ein parameterloser Aufruf wobei die Umgebungsvariable „PRIVATE_KEY_PASSPHRASE“ ausgelesen wird.

Zur Erstellung einer HTTP Signatur wird die *createSignature* Funktion verwendet. Wie die vorherige bekommt auch diese als Parameter ein *options* Objekt übergeben. Dieses benötigt einen privaten Schlüssel welcher zur Signierung verwendet wird, sowie eine *keyId* (Referenz auf den öffentlichen Schlüssel) und *url* als Eigenschaften. Mit der *headers* Eigenschaft kann optional angegeben werden, welche Kopfzeilen bei der Signierung Verwendung finden. Die Datums Kopfzeile wird standardmäßig genutzt und muss somit nicht angegeben werden. Weitere optionale Eigenschaften sind *algorithm* und *passphrase*, wobei letzteres wie zuvor aus der Umgebung ausgelesen wird. Eine Hashfunktion kann unter den im exportierten Array enthaltenen gewählt werden. Beim Aufruf der *createSignature* Funktion wird zuerst der verwendete Algorithmus validiert und anschließend unter Verwendung des privaten Schlüssels und einer Passphrase, sowie einer zuvor konstruierten Zeichenkette die Signatur in der Base64[3] Kodierung erstellt.

Die Parameter der dritten exportierten Funktion, *verifySignature*, sind einmal der Endpunkt mit, falls vorhanden, Suchanfrage an welchen die HTTP Anfrage gesendet wurde und zweitens die Kopfzeilen. Aus den Kopfzeilen werden beim Aufruf der Funktion die Signatur genommen und aus dieser alle Werte extrahiert. Danach wird die Zeichenkette welche neben privatem Schlüssel und Passphrase zum Signieren verwendet wurde rekonstruiert. Über die *keyId* wird der öffentliche Schlüssel angefragt und mit diesem, der Signatur, der Zeichenfolge und dem verwendeten Algorithmus als Parameter, die Funktion zum Erstellen der Komponente zur Verifikation und dem eigentlichen verifizieren aufgerufen.

4.2. Das IDataSource Interface

Für eine Integration in verschiedenste Applikationen wurde das *IDataSource* Interface, welches die Datenschicht repräsentiert, erstellt. Durch die Implementierung dieses Interfaces können Aktivitäten ActivityPub konform empfangen werden. Das senden wiederum muss jedes Backend selbst implementieren. Dafür stehen im ActivityPub Objekt Funktionen zum Senden von Aktivitäten bereit. Durch importieren des ActivityPub Objekts erhält man eine Instanz, worüber die Funktionen verfügbar sind. Zum Senden werden die Daten in eine Aktivität verpackt, was über die Nutzung einer AS2 Bibliothek geschehen kann oder durch Erstellung mit Objekt Literalen. Bei letzterer Variante muss sich zusätzlich um Kontexte und weiteres gekümmert werden.

Ein Austausch der Datenquelle ist durch die Nutzung von Datenbanken oder ‚api's im Interface möglich. Es kann eine relationale oder auch nicht-relationale Datenbank, wie z. B. MySQL oder MongoDB, genutzt werden. Bei der Prototyp Implementierung in dieser Abschlussarbeit wird eine GraphQL Schnittstelle angesprochen.

[3] Base64 ist ein Verfahren zu Kodierung von 8-Bit Binärdaten in ASCII-Zeichen

4.2.1. Beziehen sowie Speichern von Sammlungen

Das IDataSource Interface enthält Funktionen zum Beziehen sowie Speichern von Sammlungen. Als Parameter bekommen die Funktionen zum Beziehen von Sammlungen die „actorId“ übergeben. Aus dieser kann der Nutzername extrahiert und so die entsprechenden Daten verarbeitet werden, welche unter Zuhilfenahme von Hilfsfunktionen aus dem Werkzeug Ordner zu Sammlungen konvertiert werden. Die Funktionen zum speichern bekommen eine Sammlung übergeben welche zuvor mit den Hilfsfunktionen erstellt wurden. Optional kann mit einer Flagge angeben werden, ob nur das neuste Element gespeichert werden soll.

4.2.2. Erstellen und Löschen von Posts

Weiter enthält das IDataSource Interface Funktionen zum Erstellen und Löschen von „Post's“, Kommentaren sowie „Like's“[4] und zusätzlich zum Updaten von „Post's“. Alle in diesem Absatz genannten Funktionen bekommen eine Aktivität als Parameter übergeben. Diese wird über das Umwandeln in eine GraphQL Anfrage und ausführen dieser in einer dem Hauptsystem entsprechenden Repräsentation gespeichert.

4.2.3. Weitere Funktionalität

Um zu wissen an welche Server eine ausgehende Aktivität gesendet werden soll, ist auch Funktionalität zum beziehen und hinzufügen von geteilten Nachrichteneingängen vorhanden. Bei Eingehende Anfragen, welche bei der Verarbeitung das Aktoren-Objekt eines Nutzers beziehen, wird geprüft ob der geteilte Nachrichteneingang bereits hinzugefügt wurde. Ist das nicht der Fall wird dieser den bekannten hinzugefügt. So entsteht ein „bekanntes Netzwerk“.

Des weiteren wird Funktionalität bereitgestellt um sicherzustellen ob ein Nutzer bereits existiert sowie zum rückgängig machen vorheriger Aktivitäten. Für jeden gesichteten Nutzer wird eine der Implementierung entsprechende Repräsentation im Netzwerk angelegt. Dabei wird der Nutzer ohne Passwort und E-Mail erstellt, sodass unterschieden werden kann ob es sich um einen Nutzer der Instanz handelt, oder um einen über ActivityPub erstellten Nutzer. Bei diesem vorgehen müssen keine weiteren Typen dem GraphQL Schema hinzugefügt und außerdem keine neuen Relationen angelegt werden.

4.3. Nötige Änderungen am Datenschema

Ein paar Änderungen sind dennoch nötig. Der Nutzertyp wird um drei Eigenschaften erweitert: Erstens der „actorId“, zweitens um einen privaten und drittens einen öffentlichen Schlüssel. Außerdem muss für alle Modelle, im GraphQL Schema Typen genannt, eine Aktivitäten- und Objektidentifikator Eigenschaft angelegt werden. Dadurch wird die vollständige Rekonstruktion entsprechender AS2 Objekte und Aktivitäten sichergestellt.

[4] Ein „Like“ wird zu einem „Shout“ umgewandelt

4.4. Testwerkzeuge

Getestet wird die Implementierung mit dem Cucumber Test-Framework, welches ein Werkzeug zum ausüben von Behaviour Driven Development (BDD) ist. Es wurden Akzeptanz Tests erstellt um zu prüfen ob der Service ActivityPub konform ist. Die Tests selbst werden im Rahmen von Cucumber als sogenannte „Feature"'s bezeichnet und haben die Dateiendung „.feature". Geschrieben sind die Tests in der Gherkin Sprache welche eine Sammlung von Syntax ist um diese auf entsprechende Aktionen, sogenannte „Steps" abzubilden. Nicht nur einzelne „Step" Definitionen sonder auch Welt Dateien können genutzt werden um einen Ort für wiederverwendbare Funktionalität zu haben und einen Testzustand einzuführen. Es wurden bei der Gherkin Syntax extra leicht verständliche Schlüsselworte gewählt um die Tests auch für Benutzer, Produktbesitzer und weitere lesbar zu halten. Somit muss nur die Gherkin Sprache bekannt sein um die Features schreiben zu können. Das Verständnis beim lesen ergibt sich aus der Verwendung natürlicher Sprachkonstrukte als Schlüsselwörter. Die „Step" Definitionen zu implementieren setzt allerdings, im Gegensatz zum schreiben von „Features" Dateien, Programmierkenntnisse voraus.

4.5. Vorgehensweisen in der Softwareentwicklung

Unter Test Driven Development (TDD) versteht man eine Vorgehensweise beim Entwickeln von Software. Dabei werden zuerst fehlschlagende Tests geschrieben, um diese im zweiten Schritt durch eine Implementierung der Komponenten zur erfolgreichen Ausführung zu bringen. Im dritten Schritt werden die Tests angepasst und der Zyklus wiederholt sich.

Bei der Softwareentwicklung mit dem BDD Ansatz wird im Grunde genommen Wert auf eine hohe Kommunikation zwischen allen beteiligten im Entwicklungsprozess gelegt. Werkzeuge wie Cucumber fördern dabei die Lesbarkeit von Softwaretests und somit auch die Kommunikation zwischen Programmierern und dem Rest des Teams. Zudem ist es für Neueinsteiger leichter einen Überblick über die Funktionalität des Systems zu bekommen (Vlg. [5]).

4.6. Server-zu-Server Protokoll

Es wird eine Teilmenge der AS2 Objekte und Aktivitäten implementiert, die angepasst ist auf die Anforderungen des Unternehmens in welcher diese Arbeit angefertigt wird. Dabei handelt es sich um folgende Aktivitäten: *Create*, *Update*, *Delete*, *Follow*, *Undo*, *Accept*, *Reject*, *Like*, *Dislike*. Als Objekt Eigenschaft der Aktivitäten werden folgende Typen implementiert: *Note*, *Article*.

Die ersten drei Aktivitäten werden zum Manipulieren von Objekten wie z. B. einem Artikel benötigt. *Follow* und *Undo* sind zum folgen eines Nutzers so wie zum rückgängig machen verschiedener Aktivitäten. Wenn einem Nutzer gefolgt wurde, wird eine *Accept* Aktivität an die *Inbox* des Folgenden gesendet mit einer *Follow* Aktivität als Objekt. Im

Falle eines Fehlers wird die Anfrage mit einer *Reject* Aktivität beantwortet.

Der Funktionsumfang des förderierten Servers beschränkt sich auf den folgenden:

- Empfangen von ***Article*** und ***Note*** Objekten am Geteilten Nachrichteneingang (sharedInbox)
- Empfangen von ***Like*** und ***Follow*** Aktivitäten am Geteilten Nachrichteneingang
- Empfangen von ***Undo*** und ***Delete*** Aktivitäten für ***Articles*** und ***Notes*** am Geteilten Nachrichteneingang
- Bereitstellen eines ***Webfinger*** und ***Aktoren*** Objektes
- Bereitstellen der ***Followers***, ***Following*** und ***Outbox*** Sammlungen

Artikel und Notizen werden bei der in dieser Arbeit angefertigten Implementierung gleich behandelt. Beim empfangen einer Create Aktivität, die eine Notiz oder Artikel als Objekt Attribut hat, wird der Artikel über die IDataSource Implementierung erstellt und entsprechende Metadaten zum rekonstruieren der ID's gespeichert. Somit können empfangene „Posts", welche an die Öffentlichkeit gerichtet sind, im Netzwerk angezeigt werden. Die Logik zum Empfangen von „Posts" die an einzelne Nutzer gerichtet sind ist noch nicht vorhanden.

4.7. Signierung und Verifikation

Eingehende HTTP POST Anfragen werden auf eine vorhandene Signatur Kopfzeile geprüft. Wird diese nicht gefunden, wird die Anfrage verworfen. Ist sie vorhanden, wird die Signatur geprüft indem das Aktoren Objekt über die zugehörige *keyId* angefragt wird und die Signatur gegen den öffentlichen Schlüssel geprüft wird. Dies stellt sicher, das die Inhalte der Nachricht beim Transport nicht verändert wurde und zudem das die Aktivität von dem Nutzer stammt, der den passenden privaten Schlüssel besitzt. Eine Signatur bei der Übertragung von Servern untereinander wird demnach verwendet um die Authentizität der Daten überprüfen zu können.

Für die Nutzung eines Schlüsselpaars pro Aktor spricht, dass es für potentielle Angreifer schwieriger wird die Schlüssel zu erhalten. Bei der Nutzung eines einzigen Server Schlüsselpaars zur Signierung und Verifikation, reicht es dieses zu entwenden. Es sei angemerkt das die privaten Schlüssel mit einer Einweg Hashfunktion, in Verbindung mit einem Passwort, verschlüsselt werden und dieses Passwort somit zusätzlich entwendet werden muss.

Bei dieser Implementierung gibt es keine Server-zu-Server Interaktionen die nicht vom Nutzer initiiert wurden, abgesehen von der Anfrage nach Aktoren Objekten oder WebFinger Deskriptoren welche auch Nutzer initiiert sind, allerdings von Nutzern anderer Instanzen. Somit ist die Generierung eines weiteren Schlüsselpaars für den Server nicht

notwendig da die Inhalte mit den entsprechenden privaten Schlüsseln der Autoren signiert werden. Darüber wird festgestellt ob der Server von dem die Anfrage kam, über den privaten Schlüssel verfügt mit welcher die Signatur erstellt wurde.

5. Evaluation

Zu Beginn dieses Kapitels werden verschiedenste Anwendungsbeispiele anhand von Diagrammen veranschaulicht und erklärt. Insgesamt werden die folgenden fünf Anwendungsbeispiele beschrieben:

- Folgen eines Nutzers aus einem anderen Netzwerk
- „Liken“ eines Objekts in einem anderen Netzwerk
- Rückgängig machen einer zuvor gesendeten „Follow“ Aktivität
- Die Umkehrung eines „Likes“ (Dislike)
- Artikel erstellen in einem anderen Netzwerk

Des weiteren wird eine Performanz-Messung für die Signatur Erstellung, sowie Verifikation durchgeführt und die Ergebnisse interpretiert. Es wird sich für die Messung der Signatur Erstellung sowie Verifikation entschieden, da diese beiden Schritte eine nicht geringe Zeit der gesamten Anfrage in Anspruch nehmen. Damit wird versucht eine Hilfestellung für verschiedene Situationen zu bieten. Wird z. B. auf die Performanz großen Wert gelegt, kann anhand der Ergebnisse die Entscheidung für eine Hashfunktion gefällt werden.

Bei der Messung werden 4 verschiedene Ergebnistabellen erstellt; Jeweils zwei für eine Hardwarevariation. Zum Einen wird ein älteres Notebook Modell zum testen verwenden, zum Anderen ein Spiele-Computer. Außerdem wird aufgrund der interpretierten Ergebnisse und anderen Aspekten Empfehlungen gegeben, welche Hashfunktion für welchen Zweck am Besten passt.

5.1. Anwendungsbeispiel

Bei jedem der folgenden Anwendungsbeispiele muss sich der Server mit einer HTTP Signatur authentisieren. Die Signaturen werden beim ActivityPub Protokoll mit dem privaten Schlüssel des interagierenden Nutzers erstellt und mit dem öffentlichen verifiziert. Es sei außerdem erwähnt, dass die in diesem Abschnitt gezeigten Aktivitäten und Objekte keines Wegs alle zur Verfügung stehenden Attribute nutzen.

Im folgenden Diagramm wird durch den Anwendungsfall „HTTP Signatur verifizieren“ ersichtlich, dass jede eintreffende Aktivität von einem Nutzer eines anderen Server zuerst auf die Integrität geprüft wird:

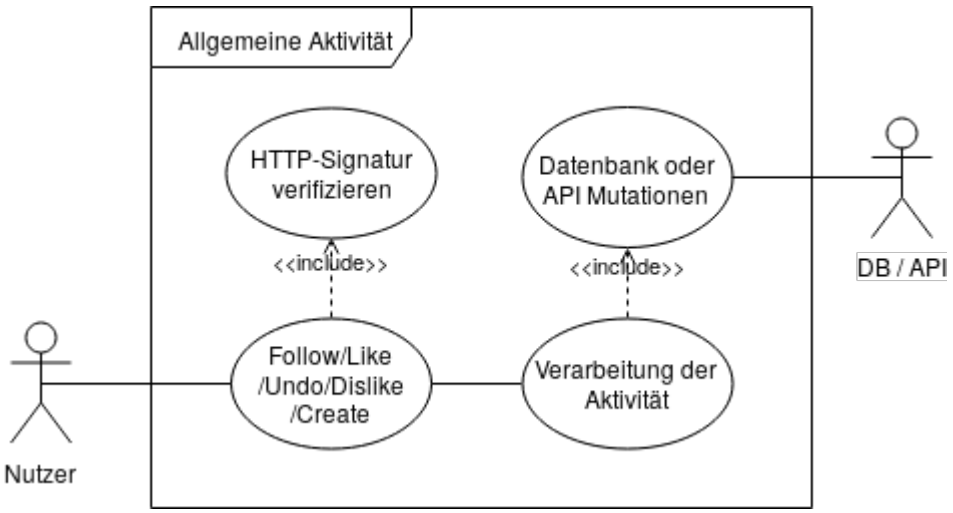


Abbildung 5.1.: Generelles Anwendungsfalldiagramm für das Empfangen von Aktivitäten

5.1.1. **Beispiel 1**: Nutzer wird gefolgt

Eine Nutzerin Alice aus einem anderen Netzwerk oder von einer anderen Instanz des selben Netzwerks, genannt Instanz A, möchte einem Nutzer auf Instanz B folgen. Dafür muss Alice dem zu folgenden Nutzer Bob über die Suchfunktion auf Instanz A finden und anschließend auf den „folgen“ Knopf drücken. Dadurch wird auf der Instanz A eine Anfrage erstellt, welche mit dem privaten Schlüssel von Alice signiert wird. Trifft die signierte Anfrage bei Instanz B ein wird zuerst die Signatur mit dem öffentlichen Schlüssel von Alice geprüft. Bei einer erfolgreichen Verifikation wird die Anfrage weiter verarbeitet und vom Server entsprechend eine „Accept“ oder „Reject“ Aktivität gesendet. Schlägt die Signatur-Prüfung fehl wird ein „Forbidden“ Statuscode zurückgegeben.

In der Testdatei zu diesem Anwendungsbeispiel wird im Körper der HTTP-Anfrage folgend abgebildeter Inhalt mitgesendet:

```
1 {
2   "@context": "https://www.w3.org/ns/activitystreams",
3   "id": "http://localhost:4123/api/users/stuart-little/status
      /83J23549sda1k72fsa4567na42312455kad83",
4   "type": "Follow",
5   "actor": "http://localhost:4123/api/users/stuart-little",
6   "object": "http://localhost:4123/api/users/tero-vota"
7 }
```

Abbildung 5.2.: Eine Follow Aktivität

Der Typ des Inhaltes dieser Anfrage muss auf „application/activity+json“ gesetzt werden. Vor dem Absenden ist die „Follower“ Sammlung leer; Nach dem erfolgreichen Absenden der HTTP Anfrage wurde zur Sammlung der folgende Nutzer hinzugefügt wie in Abbildung ?? zu sehen ist.

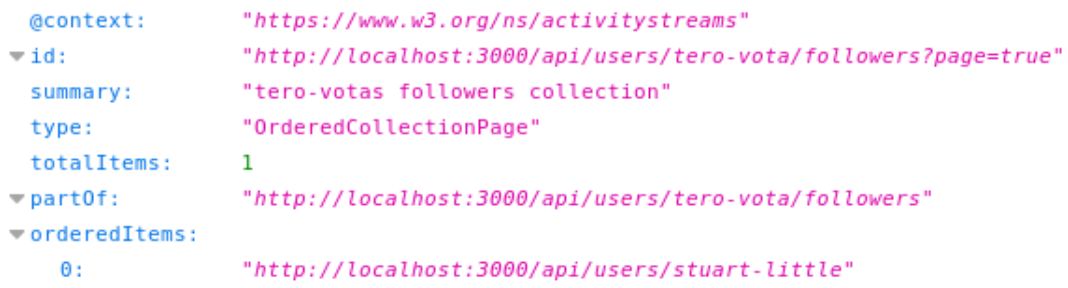


Abbildung 5.3.: Follower Sammlung von tero-vota

Auch in den folgenden Beispielen werden die Benutzer Alice und Bob als Beispielnutzer genannt welche entsprechend auf Instanz A (Alice) und Instanz B (Bob) residieren. Zudem sind die Inhaltstypen aller weiteren Anwendungsbeispiele mit dem dieses Beispiels identisch.

5.1.2. **Beispiel 2**: Rückgängig machen eines vorherigen folgens

Da sich Alice und Bob gestritten haben möchte Alice keine weiteren Benachrichtigungen von Bob erhalten, deshalb folgt Sie Bob nun nicht mehr. Instanz A erstellt, signiert und sendet somit eine Anfrage mit einer „Undo“ Aktivität an Instanz B.

Bei diesem Anwendungsbeispiel sendet die Testdatei eine HTTP-Anfrage mit folgendem Inhalt:

```
{
  "@context": "https://www.w3.org/ns/activitystreams",
  "id": "http://localhost:4123/api/users/stuart-little/status
     /a4DJ2afdg323v32641vna42lkj685kasd2",
  "type": "Undo",
  "actor": "http://localhost:4123/api/users/stuart-little",
  "object": {
    "id": "http://localhost:4123/api/users/stuart-little/
       status/83J23549sda1k72fsa4567na42312455kad83",
    "type": "Follow",
    "actor": "http://localhost:4123/api/users/stuart-little",
    "object": "http://localhost:4123/api/users/tero-vota"
  }
}
```

Abbildung 5.4.: Eine Undo Aktivität eines vorherigen folgens

Anders als z. B. bei 5.1.5 ist das Objekt der Aktivität ebenfalls wieder eine Aktivität.

Der Ausgangspunkt in diesem ist das vorherige Beispiel. Nach dem Rückgängig machen der Aktivität aus 5.1.1 sieht die „Follower“ Sammlung wie folgt aus:

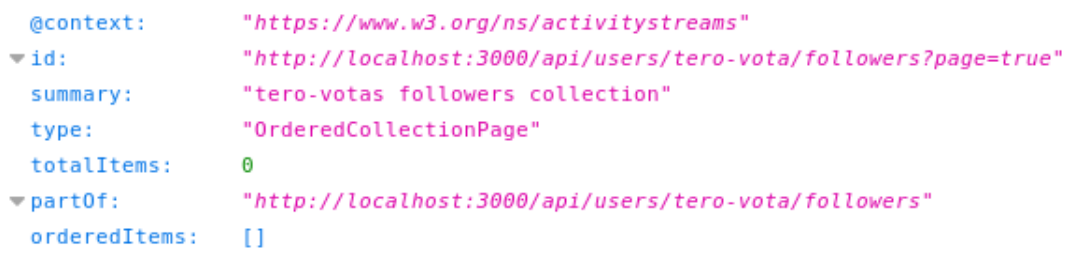


Abbildung 5.5.: Follower Sammlung von tero-vota

Im ersten Beispiel wurde also der Zähler inkrementiert und den Items eines hinzugefügt. In diesem Beispiel wurde der Zähler dekrementiert und das entsprechende Item entfernt.

5.1.3. **Beispiel 3**: Objekt eines Nutzers „liken“

Alice durchsucht ihre Zeitleiste nach neuen Inhalten. Sie stößt auf einen Inhalt von Bob, den Sie sehr mag. Daraufhin „liked“ Sie den Inhalt. Instanz A erstellt eine „Like“ Aktivität und signiert diese mit Alice ihrem privaten Schlüssel. Die Anfrage wird nun auf Empfängerinstanz B verifiziert und, bei Erfolg, verarbeitet.

Beim „liken“ eines Objekts, Aktors etc., wird der nachfolgende Inhalt von der Testdatei versendet:

```
1 {
2   "@context": "https://www.w3.org/ns/activitystreams",
3   "id": "http://localhost:4123/api/users/karl-heinz/status/83
        J23549sda1k72fsa4567na42312455kad83",
4   "type": "Like",
5   "actor": "http://localhost:4123/api/users/karl-heinz",
6   "object": "http://localhost:4123/api/users/theodor/status/
        dkaaadsfljsdfaaaffg9843jknsdf"
7 }
```

Abbildung 5.6.: Eine Like Aktivität

Die Implementierung des IDataSource Interfaces in dieser Abschlussarbeit konvertiert einen „like“ eines Artikels zu einem „shout“. Man kann sagen, dass ein „like“ und ein „shout“ Äquivalent sind. Nach der Verarbeitung des in Abbildung **??** gezeigten Anfrage Inhalts, sieht ein für diese Aktivität erstellter Artikel wie in der Abbildung zu Beginn von Seite 39 aus.

Abbildung 5.7.: Ausgerufener Test Artikel

5.1.4. **Beispiel 4**: Rückgängig machen eines vorherigen „likes“

Der zuvor von Alice gemochte Inhalt wurde in der Zwischenzeit von Bob verändert. Alice sieht dies, liest den Inhalt erneut und wundert sich warum Bob den Inhalt verändert hat. Ihrer Ansicht nach war er vorher viel Aussagekräftiger, daher „Disliked“ Sie den Inhalt. Außerdem tut sie verärgert ihrer Meinung in einem Kommentar kund.

Nachfolgend der Körper einer weiteren „Undo“ Aktivität, welche eine „Like“ Aktivität als Objekt hat:

```
{
  "@context": "https://www.w3.org/ns/activitystreams",
  "id": "http://localhost:4123/api/users/karl-heinz/status/
     faskjh6sda1k72fsa4567nfgdj367s83",
  "type": "Undo",
  "actor": "http://localhost:4123/api/users/karl-heinz",
  "object": {
    "id": "http://localhost:4123/api/users/karl-heinz/status
       /83J23549sda1k72fsa4567na42312455kad83",
    "type": "Like",
    "actor": "http://localhost:4123/api/users/karl-heinz",
    "object": "http://localhost:4123/api/users/theodor/status
       /dkaaadsfljsdfaaaffg9843jknsdf"
  }
}
```

Abbildung 5.8.: Eine Undo Aktivität eines vorherigen likens

Der Ausgangspunkt beim senden der HTTP Anfrage mit obigem Körper ist die Anfrage in Beispiel 3. Nach dem Absenden sieht der aktualisierte Artikel wie in Abbildung **??** aus.

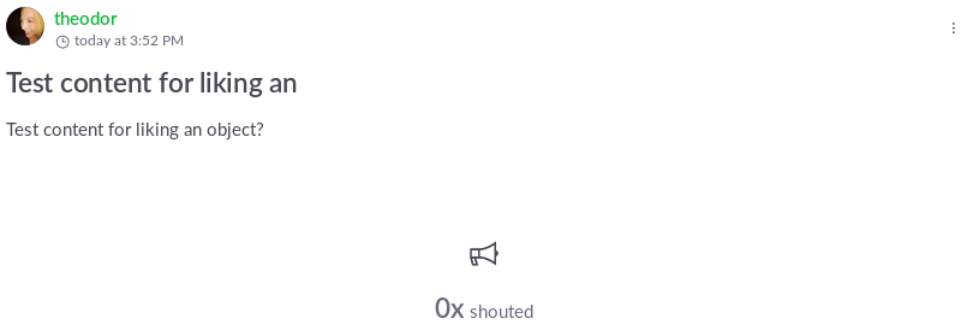


Abbildung 5.9.: Nicht ausgerufener Artikel

5.1.5. **Beispiel 5**: Erstellen eines Inhalts

Nach einem schönen Wochenende möchte Alice ihre neuen Erfahrungen in einem Inhalt zusammenfassen und mit ihren Freunden teilen. Sie meldet sich bei Instanz A an und verfasst ihre Erfahrungen. Beim Absenden erstellt Instanz A eine „Create“ Aktivität mit einem Artikel als Objekt und sendet signierte Anfragen an jeden ihrer Freunde.

Der HTTP Körper beim Erstellen eines Artikel Objekts sieht wie folgt aus:

```
1  {
2    "@context": "https://www.w3.org/ns/activitystreams",
3    "id": "https://localhost:4123/api/users/marvin/status/
       lka7dfzafdgkjnasg2398hsfd",
4    "type": "Create",
5    "actor": "https://localhost:4123/api/users/marvin",
6    "object": {
7      "id": "https://localhost:4123/api/users/marvin/status/
         kljsasddfg9843jknsdfasd",
8      "type": "Article",
9      "published": "2019-02-07T19:37:55.002Z",
10     "attributedTo": "https://localhost:4123/api/users/marvin"
         ,
11     "content": "Content for article object creation!",
12     "to": "as:Public"
13   }
14 }
```

Abbildung 5.10.: Eine Create Aktivität mit einem Artikel als Objekt

Zu sehen ist eine „Create“ Aktivität mit einem Identifikator um diese eindeutig zu identifizieren, sowie einem Artikel als Objekt. Dieses enthält seinerseits einen Identifikator um in diesem Falle das Objekt eindeutig zu identifizieren, das Veröffentlichungsdatum, den Identifikator des Senders und Empfängers, sowie den Inhalt selbst.

Um nach dem obigen Erstellen des Artikel Objekts zu validieren ob dieses erstellt wurde, wird die „Outbox“ Sammlung des Senders der Aktivität betrachten:

```
{
  "@context":"https://www.w3.org/ns/activitystreams",
  "id":"http://localhost:3000/api/users/gerald/outbox?page=
     true",
  "summary":"geralds outbox collection",
  "type":"OrderedCollectionPage",
  "totalItems":1,
  "partOf":"http://localhost:3000/api/users/gerald/outbox",
  "orderedItems":[
    {
      "id":"https://localhost:4123/api/users/gerald/status/
         lka7dfzafdgkjnasg2398hsfd",
      "type":"Create",
      "actor":"http://localhost:3000/api/users/gerald",
      "object":{
        "id":"https://localhost:4123/api/users/gerald/status/
           kljsasddfg9843jknsdfasd",
        "type":"Article",
        "published":"2019-04-19T15:17:52.920Z",
        "attributedTo":"http://localhost:3000/api/users/
           gerald",
        "content":"Content for article object creation!",
        "to":"https://www.w3.org/ns/activitystreams#Public"
      }
    }
  ]
}
```

Abbildung 5.11.: Outbox Sammlung mit einem Artikel-Objekt-Item

Auf der Abbildung ist das Artikel-Objekt aus **??** als Item der Sammlung zu sehen. Durch die „Create“ Aktivität wurde dieses Erstellt.

5.2. Performanz-Messungen

Performanz Messung mit JMeter

Es wurde die Performanz der Erstellung von Signaturen, sowie der Verifikation solcher gemessen. Insgesamt wurden 2 Messungen mit je 10 Testdurchläufen pro Hashfunktion durchgeführt. Die erste Messung wurde auf einem Notebook mit eingebauter Intel 2-Kern Central Processing Unit (CPU) der Serie Core i5 und einer Taktfrequenz von 2,4 GH durchgeführt[1]. Bei der zweiten Messung kam eine 16-Kern AMD CPU aus der „Ryzen“ Serie mit der Bezeichnung „Threadripper 2950X“ und einer Taktfrequenz von 3,5 GH zum Einsatz[2].

Performanz-Messung in die Implementierung verschieben

[1] https://ark.intel.com/content/www/de/de/ark/products/47341/intel-core-i5-520m-processor-3m-cache-2-40-ghz.html
[2] https://www.amd.com/de/products/cpu/amd-ryzen-threadripper-2950x

Die Performanzmessungen wurden in der Node.js Laufzeitumgebung mit der Version 10.15.3 durchgeführt unter Zuhilfenahme der internen process.hrtime() Funktion. Diese wird vor und nach dem Erstellen sowie Verifizieren aufgerufen. Die Rückgabe des zweiten Aufrufs nach der jeweiligen Funktion enthält die Differenz zum ersten Aufruf in Nanosekunden. Beim betrachten der arithmetischen Mittel fällt auf, dass die Messwerte sehr

	RSA-MD4	RSA-MD5	RSA-SHA256	RSA-SHA512
Testlauf 1	24.254 ms	24.123 ms	24.171 ms	24.173 ms
Testlauf 2	22.833 ms	23.006 ms	23.626 ms	22.948 ms
Testlauf 3	27.186 ms	23.113 ms	22.873 ms	22.781 ms
Testlauf 4	23.630 ms	24.256 ms	22.597 ms	22.667 ms
Testlauf 5	25.012 ms	24.027 ms	24.343 ms	30.278 ms
Testlauf 6	30.221 ms	23.032 ms	22.827 ms	22.768 ms
Testlauf 7	27.050 ms	29.950 ms	22.629 ms	22.774 ms
Testlauf 8	23.630 ms	24.282 ms	24.625 ms	25.495 ms
Testlauf 9	23.151 ms	22.958 ms	22.854 ms	26.816 ms
Testlauf 10	22.654 ms	24.067 ms	23.637 ms	23.579 ms
Arithmetisches Mittel	25,122 ms	24,281 ms	23,418 ms	24,427 ms

Tabelle 5.1.: Testergebnisse der ersten Messung (Signatur Erstellung)

nahe beieinander liegen. Die SHA256 Hashfunktion schneidet dabei mit einer Funktionslaufzeit von 23,418 ms am Besten ab. Auf Platz zwei liegt die MD5 Funktion mit 24,281 ms gefolgt von SHA512 mit 24,427 ms. Das Schlusslicht bildet MD4 mit einer Laufzeit von 25,122 ms.
Aus Sicht der Performanz wird bei der ersten Messung somit die SHA256 Hashfunktion zur Signierung empfohlen. Werden die Arithmetischen Mittel der Verifikation betrachtet

	RSA-MD4	RSA-MD5	RSA-SHA256	RSA-SHA512
Testlauf 1	107,177 ms	128.993 ms	121.406 ms	114.669 ms
Testlauf 2	43,660 ms	51.178 ms	35.524 ms	39.633 ms
Testlauf 3	32.101 ms	32.691 ms	30.461 ms	32.293 ms
Testlauf 4	36.096 ms	30.279 ms	32.018 ms	43.842 ms
Testlauf 5	31.869 ms	34.882 ms	30.877 ms	26.555 ms
Testlauf 6	39.218 ms	67.507 ms	38.734 ms	31.330 ms
Testlauf 7	40.996 ms	39.646 ms	45.433 ms	27.290 ms
Testlauf 8	25.977 ms	39.302 ms	26.778 ms	53.478 ms
Testlauf 9	23.641 ms	37.833 ms	30.485 ms	36.284 ms
Testlauf 10	31.097 ms	52.725 ms	28.943 ms	28.634 ms
Arithmetisches Mittel	41,183 ms	51,503 ms	42,065 ms	43,400 ms

Tabelle 5.2.: Testergebnisse der ersten Messung (Signatur Verifikation)

ist eine Zeitspanne von 10 ms Sekunden zu Beobachten in welcher sich alle Messwerte

befinden. Dabei ist der MD4 Algorithmus mit 41,183 ms auf Position eins, während sich MD5 mit 51,503 ms auf der letzten Position befindet. Die beiden Funktionen SHA256 und SHA512 liegen mit Laufzeiten von 42,065 und 43,400 ms näher an der MD4 Funktion. Zudem fällt auf, dass bei der Verifikation die ersten Messwerte stark abweichen von allen weiteren. Das ist dadurch zu erklären, dass zuerst eine HTTP Anfrage gesendet werden muss um den benötigten öffentlichen Schlüssel zu erhalten. Jegliche weitere Anfragen werden aus dem Cache bedient und sind somit um einen groben Faktor von 2 schneller als die erste Anfrage.

Anders als bei der ersten Messung liegt bei Betrachtung der arithmetischen Mittel kei-

	RSA-MD4	RSA-MD5	RSA-SHA256	RSA-SHA512
Testlauf 1	7.152 ms	7.137 ms	8.178 ms	7.002 ms
Testlauf 2	6.955 ms	7.134 ms	6.714 ms	7.048 ms
Testlauf 3	6.910 ms	6.762 ms	6.703 ms	6.860 ms
Testlauf 4	7.044 ms	6.800 ms	6.827 ms	6.982 ms
Testlauf 5	6.935 ms	7.831 ms	6.854 ms	6.958 ms
Testlauf 6	7.191 ms	7.697 ms	6.873 ms	7.050 ms
Testlauf 7	7.195 ms	7.108 ms	6.728 ms	6.840 ms
Testlauf 8	7.237 ms	6.864 ms	6.743 ms	6.722 ms
Testlauf 9	7.136 ms	7.016 ms	6.740 ms	6.834 ms
Testlauf 10	7.128 ms	6.939 ms	6.811 ms	6.819 ms
Arithmetisches Mittel	7,088 ms	7,128 ms	6,917 ms	6,911 ms

Tabelle 5.3.: Testergebnisse der zweiten Messung (Signatur Erstellung)

ne Zeitspanne vor, sondern die Messwerte liegen sehr nahe beieinander. Dabei ist die SHA512 Hashfunktion am schnellsten mit 6,911 ms, dicht gefolgt von der SHA256 Funktion mit einer Laufzeit von 6,917 ms. Auf Platz drei ist die MD4 Funktion mit 7,088 ms und auf Platz vier MD5 mit 7,128 ms.

Wie auch in der ersten Messung sehen wir bei der Verifikation, dass die beginnenden Messwerte einen deutlichen Unterschied zu weiteren aufweisen. Der Grund ist derselbe wie bei der ersten Messung. Betrachten wir die arithmetischen Mittel, sehen wir Messwerte innerhalb einer Zeitspanne von ungefähr 3 ms in der sich die Messwerte bewegen. Die MD5 Hashfunktion schneidet hierbei am Besten ab mit einer Zeit von 17,900 ms, gefolgt von der MD4 Funktion mit 18,144 ms. Die Schlusslichter bilden hier SHA256 mit 18,515 ms und SHA512 mit 18,656 ms.

Wird bei der Erstellung und Verifikation auf die Laufzeit Wert gelegt, gelten folgende Empfehlungen:

- Bei der Betrachtung von Messung eins der Signatur Erstellung kann die SHA256 Funktion und von Messung zwei SHA512 empfohlen werden.
- Wird die Signatur Verifikation betrachtet, kann die MD4 Funktion bei der ersten Messung und MD5 bei der zweiten empfohlen werden.

	RSA-MD4	RSA-MD5	RSA-SHA256	RSA-SHA512
Testlauf 1	45.639 ms	50.422 ms	53.230 ms	50.732 ms
Testlauf 2	16.919 ms	18.139 ms	15.841 ms	16.793 ms
Testlauf 3	19.744 ms	12.443 ms	15.318 ms	16.947 ms
Testlauf 4	18.103 ms	14.628 ms	15.803 ms	13.946 ms
Testlauf 5	16.982 ms	13.784 ms	13.126 ms	12.239 ms
Testlauf 6	18.338 ms	14.137 ms	24.549 ms	19.943 ms
Testlauf 7	14.566 ms	11.247 ms	11.772 ms	12.191 ms
Testlauf 8	17.689 ms	13.369 ms	12.277 ms	16.769 ms
Testlauf 9	13.462 ms	14.808 ms	11.173 ms	14.314 ms
Testlauf 10	15.422 ms	16.029 ms	12.069 ms	12.693 ms
Arithmetisches Mittel	18,144 ms	17,900 ms	18,515 ms	18,656 ms

Tabelle 5.4.: Testergebnisse der zweiten Messung (Signatur Verifikation)

Aus Sicht der Sicherheit sei allerdings gesagt, dass die Funktionen MD4 und MD5 als unsicher gelten. Da SHA-1 auch die MD4 Funktion verwendet und diese bereits als unsicher gilt, ist somit auch SHA-1 nicht sicher (Vgl. [24, S. 103]). Wird also Wert auf kollisionsfrei erzeugte Hashes gelegt, sollte zulasten der Laufzeit vorzugsweise die SHA256 oder SHA512 Funktion Verwendung finden (Vgl. [22, S. 40, Tabelle 4.1]).

5.3. Diskussion von Vor- und Nachteilen der Lösung

6. Fazit und Ausblick

Im Fazit die Arbeit nochmal Zussammenfassen, blos kennt der Leser die Arbeit bereits

6.1. Fazit

Ergebnisse, wie sind sie einzuordnen,

6.2. Ausblick

Da das Interface sehr groß werden kann beim hinzufügen weiterer Funktionalität könnte dieses neu Strukturiert oder aufgeteilt werden. Beispielsweise kann eine weitere Arbeit innerhalb des Interfaces Fassaden-Klassen anlegen. Es könnte sich auch ein neues Interface Design überlegt werden.

Die Implementierung beinhaltet zur Authentifizierung der Server untereinander sowie zur Sicherstellung der Datenintegrität Funktionalität zum Erstellen und Verifizieren von HTTP-Signaturen. Eine weitere Arbeit kann mit einer Implementierung der Funktionalität zum Signieren und Verifizieren von unterschiedlichen *Linked Data* Signatur Typen und einer Performanz-Messung dieser Implementierung auf diese Arbeit aufbauen.

Zur Erweiterung der Funktionalität des Frameworks können weitere Werkzeugklassen sowie Logik zum Verarbeiten zusätzlicher Aktivitäten und Objekte hinzugefügt werden. Das Framework enthält bis dato keine seitens des Standards definierten Aktivitäten zum hinzufügen zu und entfernen von Aktivitäten aus Sammlungen (Add, Remove). Auch weitere Objekttypen wie *Audio* oder *Video* können hinzugefügt werden[1].

Auch eine Komponente zum Schutz vor förderierten *denial-of-service* Attacken des förderierten Servers könnte implementiert werden. Der Nutzerinhalt von Objekten kann außerdem so bereinigt werden, dass kein *cross-site-scripting* stattfinden kann.

[1] siehe `https://www.w3.org/TR/activitystreams-vocabulary/`, 3.3

Literatur

[1] Bundeszentrale für politische Bildung. *Dezentralisierung | bpb*. Date accessed: 29.03.2019. 2019. URL: https://www.bpb.de/nachschlagen/lexika/das-junge-politik-lexikon/204292/dezentralisierung.

[2] W3 Consortium. *Social Web Working Group*. Date accessed: 12.12.2018. 2018. URL: https://www.w3.org/Social/WG.

[3] W3 Consortium. *W3C launches push for social web application interoperability*. Date accessed: 21.07.2014. 2014. URL: https://www.w3.org/blog/news/archives/3958.

[4] W3C Consortium. *HTTP Signatures*. Date accesed: 14.03.2019. 2014. URL: https://tools.ietf.org/id/draft-cavage-http-signatures-01.html.

[5] Cucumber Gemeinschaft. *BDD Overview*. Date accessed: 19.03.2019. 2019. URL: https://docs.cucumber.io/bdd/overview/.

[6] Diaspora Gemeinschaft. *An introduction to the Diaspora source - diaspora* project wik*. Date accessed: 30.03.2019. 2019. URL: https://wiki.diasporafoundation.org/An_introduction_to_the_Diaspora_source.

[7] Diaspora Gemeinschaft. *Über - Das Projekt Diaspora**. Date accessed: 30.03.2019. 2019. URL: https://diasporafoundation.org/about.

[8] Fediverse Gemeinschaft. *Fediverse*. Date accessed: 17.01.2019. 2019. URL: https://fediverse.party/en/fediverse/.

[9] Fediverse Gemeinschaft. *Fediverse Network Report 2018*. Date accessed: 17.01.2019. 2019. URL: https://fediverse.network/reports/2018.

[10] hagengraf. *Zentrale, dezentrale, verteilte Systeme*. Date accessed: 14.03.2019. 2017. URL: https://blog.novatrend.ch/2017/12/25/zentrale-dezentrale-und-verteilte-systeme/.

[11] E. Hammer-Lahav. *RFC 5849 - The OAuth 1.0 Protocol*. Date accessed: 30.03.2019. 2019. URL: https://tools.ietf.org/html/rfc5849.

[12] E. Hammer-Lahav. *RFC 6415 - Web Host Metadata*. Date accessed: 30.03.2019. 2019. URL: https://tools.ietf.org/html/rfc6415.

[13] Neo4j Inc. *Why Graph Databases? - Neo4j Graph Database Platform*. Date accessed: 01.04.2019. 2019. URL: https://neo4j.com/why-graph-databases/.

[14] Christopher Lemmer Webber u. a. *Der ActivityPub Standard*. Date accessed: 12.12.2018. 2018. URL: https://www.w3.org/TR/activitypub/.

[15] D. Hardt M. Jones. *RFC 6750 - OAuth 2.0 Authorization Framework: Bearer Tokens Usage*. Date accessed: 24.01.2019. 2019. URL: https://tools.ietf.org/html/rfc6750.

[16] Inc. Manu Sporny Digital Bazaar. *The Security Vocabulary*. Date accessed: 22.04.2016. 2016. URL: https://web-payments.org/vocabs/security.

[17] James M. Snell und Evan Prodromou. *Activity Vocabulary*. Date accessed: 23.05.2017. 2017. URL: https://www.w3.org/TR/activitystreams-vocabulary/.

[18] James M. Snell und Evan Prodromou. *ActivityStreams 2.0*. Date accessed: 23.05.2017. 2017. URL: https://www.w3.org/TR/activitystreams-core/.

[19] Facebook Open Source. *GraphQL | A query language for your API*. Date accessed: 01.04.2019. 2019. URL: https://graphql.org/.

[20] Christian Stobitzer. *Netzwerkeffekt*. Date accessed: 29.12.2018. 1986. URL: http://www.wirtschafts-lehre.de/netzwerkeffekte.html.

[21] Maarten van Steen Tanenbaum Andrew S. *Verteilte Systeme*. Pearson Studium, 2008.

[22] Bundesamt f ur Sicherheit in der Informationstechnik. *Kryptographische Verfahren: Empfehlungen und Schlüssellängen*. Date accessed: 18.03.2019. 2019. URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102.pdf?__blob=publicationFile&v=10#table.4.1.

[23] W3C. *Social Web Working Group Charter*. Date accessed: 17.12.2018. URL: https://www.w3.org/2013/socialweb/social-wg-charter.

[24] Dietmar Wätjen. *Kryptographie*. Springer Vieweg, 2018.

[25] Wikipedia. *Dezentrale Stromerzeugung*. Date accessed: 29.03.2019. 2019. URL: https://de.wikipedia.org/wiki/Dezentrale_Stromerzeugung.

[26] Wikipedia. *Soziales Netzwerk (Soziologie)*. Date accessed: 14.03.2019. 2018. URL: https://de.wikipedia.org/wiki/Soziales_Netzwerk_(Soziologie).

A. Anhang

A.1. First Appendix Section