[Dokumenttitel]

[Firmenname] | [Firmenadresse]

[Untertitel des Dokuments]

Gianfranco Kessler

[Jahr]

Inhalt

[Einleitung 2](#_Toc458600265)

[Hauptteil 3](#_Toc458600266)

[Webserver anhand des OSI-Modell 3](#_Toc458600267)

[Anwendungsschicht 3](#_Toc458600268)

[Schlussteil 4](#_Toc458600269)

[Literaturverzeichnis 5](#_Toc458600270)

[Anhang 6](#_Toc458600271)

[Schlusserklärung 7](#_Toc458600272)

# Vorwort

Knapp vier Jahre ist es her, seit ich das erste mal mit den Befehlen des cmd-Terminals in Windows herumgespielt habe. Dies war mein erster richtiger Kontakt mit dem «Programmieren». Obwohl ich damit erst meinen PC mit einem Klick herunterfahren oder simple Matrizen erstellen konnte, war dies eine interessante Erfahrung für mich. 2014 dann, in meinem 4. Jahr an der Kantonsschule, hatte ich dann das erste mal richtig Informatik-Unterricht, welcher sich mit dem Programmieren befasste. Wir haben die Programmiersprache Python mithilfe des Python IDLE kennengelernt. Sogleich war ich gepackt vom Programmieren. Leider kamen wir nicht wirklich weit voran und ich hatte einen eher kleinen Einblick. Trotzdem war mir schon dann relativ klar, dass ich Informatik studieren möchte und als Ergänzungsfach für die 6. Klasse Informatik wählen werde, da ich davon einfach fasziniert war. Auch für meine Maturaarbeit wusste ich schnell, dass ich etwas programmieren möchte.

# Einleitung

Ursprünglich wollte ich eine App, präziser ein Spiel programmieren, jedoch wäre dafür notwendig gewesen, eine ganze neue Programmiersprache zu erlernen und sowohl die App- als auch die Netzwerkseite zu programmieren, was ein unrealistischer Aufwand gewesen wäre. Auch eine Arbeit zum Thema «Künstliche Intelligenz» wäre zu aufwendig gewesen und somit entschloss ich mich zum Thema http-Webserver. Um dieses Thema ausführlich zu erklären befasse ich mich auch mit dem OSI-Modell um dem Leser die verschiedenen Schichten und Abläufe während der Benutzung des Internets näher zu bringen, sowie dem http-Protokoll mit dessen Entstehung, den Möglichkeiten und dem http-Protokoll in Python im spezifischen. In meinem Kapitel zum OSI-Modell werde ich zuerst erklären, was Server sind und was so ungefähr passiert, wenn man eine Webseite aufruft. Dann werde ich die einzelnen Schichten des OSI-Modells näher erklären

## Material

Für die Recherche benutze ich das Internet. Obwohl meine Quellen hauptsächlich verschiedene Links sind, habe ich um selbst in das Thema einzusteigen verschiedene Videos auf Youtube geschaut, was ich nur weiterempfehlen kann.

Mein Code ist in Python 3 geschrieben, dafür habe ich Python IDLE Version 3.5 benutzt, sowie PyCharm Community Edition in Version 2016.2.1, da dieses viele Vorteile bietet. Meine verschiedenen Dateien habe ich auf GitHub verwaltet und dort auch mein Protokoll bei jedem Upload mitverfasst.

# Hauptteil

## Webserver anhand des OSI-Modell

Server sind Computer, welche ihre Daten anderen Computern innerhalb eines Netzwerks zur Verfügung stellen. So ähnlich funktioniert dies auch mit Webservern, nur dass die Daten öffentlich zugänglich sind, jeder mit Internetanschluss hat Zugang dazu. Diese Daten werden als Webseite abgerufen. Jede Webseite hat eine URL(Siehe Seite 5), diese funktioniert ähnlich wie eine Hausadresse. Sie besteht meist aus dem zu verwendenden Protokoll, einer Domain, einem Webserver und teilweise aus dem Verzeichnis, in welchem sich die aufzurufenden Dateien befinden. (Aidex)

Wie funktioniert nun das Aufrufen einer Webseite genau? Das Ganze funktioniert mit dem sogenannten OSI(Open Systems Interconnection)-Modell. Das Problem bei der Kommunikation von verschiedenen Programmen ist, dass sie verschiedene Sprachen(Protokolle) verwenden. Nun müssen diese verschiedenen Protokolle irgendwie übersetzt werden. Der ganze Vorgang vom Abrufen einer Webseite wird beim OSI-Modell in sieben Schichten unterteilt: Anwendung, Darstellung, Sitzung, Transport, Vermittlung, Sicherung und Bitübertragung. (Wikibooks)

### Anwendungsschicht

Wie der Name schon sagt, besteht diese Schicht aus der Anwendung, welche der Benutzer gerade benutzt. Sie stellt Funktionen zur Verfügung wie Internetkommunikation über http(Siehe Seite 4 ff.) oder SSH[[1]](#endnote-1) und regelt die Dateneingabe und -ausgabe. (Wikibooks)

### Darstellungsschicht

Die Darstellungsschicht übersetzt systemabhängige Zeichen in eine unabhängige Form um, sodass sie von verschiedenen Systemen verstanden werden können. Es ermöglicht durch Übersetzen die Kommunikation mit den unteren Schichten. (Wikibooks)

### Sitzungsschicht

Damit Systeme miteinander kommunizieren können, muss dazu eine Sitzung aufgebaut und instand gehalten werden, dies regelt die Sitzungsschicht. Wenn die Kommunikation vorbei ist, wird die Sitzung wieder beendet. Dies ist notwendig, damit wir Anfragen an den Zielserver senden können und dieser antworten kann. Hier bietet HTTP(Mehr dazu ab Seite 4) den Dienst an, dass wenn man eine URL(Siehe Seite 5) eingibt, es die Daten vom Server holt und einem schickt. (Wikibooks)

### Transportschicht

Die Transportschicht baut eine logische Ende-zu-Ende-Verbindung auf. Das TCP(Transfer-Control-Protocol) nutzt hier die Eigenschaft des Internetprotokolls, einzelne Datenpakete zwischen Systemen mithilfe der IP-Adresse zu versenden. Die Transportschicht wandelt die Datenpakete in Teilpakete um und sorgt für die richtige Zusammensetzung beim Zielsystem. (Wikibooks)

### Vermittlungsschicht

Die Datenpakete gelangen nicht direkt zum Ziel, sie haben verschiedene «Zwischenstationen». Die Vermittlungsschicht sucht also sogenannte «Netzknoten» und baut eine Verbindung auf und danach wieder ab, sie steuert also die Route, welche die Datenpakete nehmen. Diese Schicht läuft meist über einen Router. (Wikibooks)

### Sicherungsschicht

Die Sicherungsschicht besteht aus zwei Unterschichten, nämlich der Medium Access Control-Schicht und der Logical Link Control-Schicht, sie sorgt für den zuverlässigen Austausch von Datenpaketen zwischen den Systemen. (Wikibooks)

### Bitübertragungsschicht

Die Bitübertragungsschicht ist für die Übertragung der Bitströme verantwortlich. Sie definiert die elektrische, mechanische und funktionale Schnittstelle zum Übertragungsmedium. (Wikibooks)

### Beispiel

Was passiert nun also, wenn ich im Webbrowser eine Webseite aufrufen will? Ich suche zum Beispiel mittels Google Chrome nach «youtube.com», das ist die Anwendungsschicht. In der Darstellungsschicht wird die Anfrage nun also umgewandelt. Die Sitzungsschicht baut eine Sitzung auf, die Transportschicht baut eine logische Ende-zu-Ende-Verbindung mithilfe der IP-Adresse, welche jeder Computer besitzt, (in diesem Falle 208.65.153.238) auf, die Anfrage wird an den Server vermittelt, die Vermittlungsschicht routet die Datenpakete zum nächsten Netzknoten. Die Sicherungsschicht fügt die Sender- und Empfängeradresse hinzu und teilt die Pakete in mehrere Frames, sogenannte Teilpakete, ein. Diese werden dann in der Bitübertragungsschicht physisch an den Youtube-Server gesendet. Dort werden die Datenpakete geholt und dann passieren dieselben Vorgänge in umgekehrter Reihenfolge nochmals, bis die Webseite in Google Chrome erscheint.

## http-Protokoll

### Was sind Protokolle?

Ein Protokoll ist einfach gesagt, wie eine Sprache. So hat jedes seine eigenen Regeln wie die Grammatik, ohne welche verschiedene Systeme nicht miteinander kommunizieren können. Benutzen beide Systeme die gleiche Sprache und die gleichen Regeln, so können sie miteinander kommunizieren. (Menzerath)

### Das http-Protokoll

1989, kurz nachdem das Internet erfunden wurde, hatte der Brite Tim Berners-Lee die Idee eines weltweiten Netzwerkes dem CERN[[2]](#endnote-2) vorgeschlagen. Innerhalb dieses sollte jeder Computer Daten anbieten und anfragen können, damit Wissenschaftler von überall auf der Welt miteinander kommunizieren können. Dafür mussten die verschiedenen Computer jedoch eine gemeinsame Sprache sprechen. Dazu entwickelte Berners-Lee bis Ende 1990 das http-Protokoll. Nebenbei hat er noch HTML[[3]](#endnote-3) und die URL entwickelt und so die Basis für seinen Webserver und -browser geschaffen. (Webfoundation; Jaschkowski)

Im Laufe der Jahre wurden neue Versionen herausgebracht. Angefangen hat das Ganze 1991 mit HTTP/0.9. Dieses ermöglichte das senden einer Anfrage für eine Datei, also einer GET-Request. Da diese Anfrage nur aus einer Zeile bestand wurde dies oft auch das One-Line Protocol genannt. Die Response bestand nur aus einem Body mit dem Inhalt der Datei. 1996 wurde dann das HTTP/1.0 als RFC[[4]](#endnote-4) veröffentlicht. Die grösste Veränderung war, dass hier viel mehr Daten über Client und Server übertragen werden. Eine Request besteht seither aus der Methode, dem Header und einem Body. Neu gab es nun neben GET auch die Methoden HEAD und POST. Zusätzlich war es jetzt auch möglich eine Authentifizierung zu verlangen, damit nicht jeder auf alles zugreifen konnte. 1999 wurde dann HTTP/1.1 publiziert. Diese neue Version brachte neue Methoden wie DELETE, OPTIONS oder PUT, eine verbesserte Authentifizierung, das ermöglichen von persistenten, daher offenbleibenden Verbindungen und vieles mehr. (O’Reilly; Jaschkowski)

Es ist klar, dass das Internet niemandem gehören kann, doch wer kontrolliert das http-Protokoll? Das HTTP/1.0 wurde ja als RFC 2616 veröffentlicht. Diese RFC’s werden von der sogenannten Internet Society [[5]](#endnote-5)kontrolliert. Zusätzlich sind aber auch noch die Internet Engineering Task Force[[6]](#endnote-6) (IETF) und das Internet Architecture Board[[7]](#endnote-7) (IAB) als Unterorganisationen der Internet Society beteiligt, welche das Ganze auch steuern. (Strickland)

Das http-Protokoll ist ein sogenanntes Übertragungsprotokoll. Im Normalfall sendet ein Browser eine Anfrage an den http-Server und bekommt eine Antwort, danach wird die Verbindung beendet.

Bevor eine solche Anfrage aber überhaupt möglich ist, muss der Client zuerst den Server finden und die zu schickenden Daten angeben. Dies übernimmt die URL, welche gleich auch noch das Protokoll, welches verwendet werden soll, angibt. Eine URL ist meist wie folgt aufgebaut: <http://www.domain.topleveldomain:Port/pfad/Dateiname> wobei der Port optional ist. Das http ist das zu verwendende Protokoll, welches der Browser jedoch meist selbst einfügt. Das [www.domain.topleveldomain](http://www.domain.topleveldomain) ist der Servername, welchen man auch durch die IP ersetzen könnte, da domain.de nur ein im Domain Name System(DNS) eingetragener Name für die betreffende IP ist und welcher von Nameservern übersetzt werden kann. Der Port gehört zum verwendeten Protokoll, bei http zB. Ist es standardmässig der Port 80. Dieser kann jedoch individuell verändert werden. Standardmässig sind auf einem Computer die ersten 1024 Ports festgelegt. Der Pfad gibt an, in welchem Verzeichnis die abzurufende Datei ist, sofern nicht das Stammverzeichnis abgerufen werden soll. (Menzerath; Cyon)

Damit Daten gesendet werden können muss zuerst aber eine Anfrage geschickt werden, eine sogenannte http-Request. Sie besteht aus der Methode(meist GET oder POST) und dem Request-Header. Ein Beispiel für einen http-Request-Header wäre folgendes:

«

GET /hello.txt HTTP/1.1

User-Agent: curl/7.16.3 libcurl/7.16.3 OpenSSL/0.9.7l zlib/1.2.3

Host: www.example.com

Accept-Language: en, mi

» (Reschke, 2016)

Die erste Zeile benennt die Methode[[8]](#endnote-8), welche verwendet wird, die angeforderte Datei und die verwendete Version des Protokolls. Die zweite Zeile ist da, damit die Antworten auf den Browser zugeschnitten sind. Die Dritte wählt den Host aus, bei welchem die Datei geholt werden soll, falls der Server mehrere Domains verwaltet. Die letzte Zeile gibt an, welche Sprachen (und evtl. Formate) akzeptiert werden. (Menzerath)

Auf die Anfrage sendet der http-Server dann eine Antwort zurück, welche aus einem dreistelligen Status-Code über die Verfügbarkeit und einer Klartextmeldung besteht.

Ein Beispiel für einen Response-Header wäre:

«

HTTP/1.1 200 OK

Date: Mon, 27 Jul 2009 12:28:53 GMT

Server: Apache

Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT

ETag: "34aa387-d-1568eb00"

Accept-Ranges: bytes

Content-Length: 51

Vary: Accept-Encoding

Content-Type: text/plain

Hello World! My payload includes a trailing CRLF.

» (Reschke, 2016)

Bei der Response beschreibt die erste Zeile die Protokollversion sowie einen Statuscode (Mehr zum Statuscode im nächsten Abschnitt). In der zweiten Zeile findet man das Datum, in der dritten den verwendeten Server. Darunter findet man noch Angaben zum Inhalt, zB. die Länge in Bytes, der Typ des Inhalts, etc. All diese Angaben haben weder in der Request noch in der Response eine bestimmte Reihenfolge ausser der ersten Zeile.

Wie bereits erwähnt findet man im Response-Header verschiedene Statuscodes. Ein Statuscode gibt an, ob die Anfrage erfolgreich war, oder ob ein Fehler vorliegt und wenn, welcher. Verschiedene wichtige Statuscodes sind folgende:

2XX Erfolgreiche Operation

200 OK: Die Anfrage wurde erfolgreich bearbeitet.

3XX Umleitung

301 Moved Permanently: Die benutze Adresse ist nicht mehr gültig. Die neue wird unten im Location-Feld angegeben.

307 Temporary Redirect: Die benutzte Adresse ist vorübergehend nicht verfügbar. Die vorübergehende wird im Location-Feld angegeben.

4XX Client-Fehler

400 Bad Request: Die Anfrage war fehlerhaft.

403 Forbidden: Kein berechtigter Zugriff

404 Not Found: Die angeforderte Ressource konnte nicht gefunden werden.

5XX Server-Fehler

500 Internal Server Error: Unerwarteter Server-Fehler.

504 Gateway Time-out: Keine Antwort innerhalb der festgelegten Zeitspanne.

(iana.org; W3.org)

Was ist aber, wenn bestimmte Daten nicht für alle bestimmt sind? http hat dafür das Basic-http-Authentication-Schema, welches einen Benutzernamen und ein Passwort verwendet. Wenn dies der Fall ist, dann kommt bei der http-Response der Status-Code 401. Der Browser öffnet dann ein Fenster, wo man aufgefordert wird, den Benutzernamen und das Passwort einzugeben, diese werden dann an den Server gesendet. Falls die Eingaben stimmen kommen dann die Daten zurück.

Das aktuelle HTTP verfügt jedoch noch über viele andere Funktionen wie Message Syntax, Routing, Caching, etc.

### Das http-Protokoll im OSI-Modell

Wo kommt das http-Protokoll eigentlich vor in den Schichten des OSI-Modells? Die Bitübertragungsschicht läuft über Kabel, Lan oder Licht, die Sicherungsschicht über das Ethernet, die Vermittlungsschicht über das Internet Protocol, die Transportschicht über TCP. Die Kommunikation- und Darstellungsschichten laufen nun über http und die Anwendungsschicht über das Internet. Nun wird klar, dass http dem TCP und IP übergeordnet ist. Es verarbeitet die Daten der beiden Systeme und benutzt TCP für die Verbindung und IP für die Adressierung.

(Elektronik Kompendium; Menzerath)

|  |  |
| --- | --- |
| OSI | TCP/IP |
| 7 | Anwendungsschicht | Anwendungen (FTP, SMTP, http, etc) |
| 6 | Darstellungsschicht |
| 5 | Sitzungsschicht |
| 4 | Transportschicht | TCP (Host to Host) |
| 3 | Vermittlungsschicht | IP |
| 2 | Sicherungsschicht | Netzwerkzugriff (Normalerweise Ethernet) |
| 1 | Bitübertragungsschicht |

### Das http-server-Modul in Python

In Python kann man mithilfe des Socketserver- und des http-Moduls relativ einfach einen simplen Webserver programmieren, welcher GET- und POST-Requests erhalten und bearbeiten kann. Welche Rolle spielt das http-Modul dabei?

Das http-Modul übernimmt die Rolle des Handlers(Bearbeiter), welcher eingehende Requests analysiert und dann beantwortet. Dieser prüft, welche Methode er verwenden muss, er schaut, welche Protokollversion verwendet wird, was angefordert wird und ob es verfügbar ist. Er liest die Host-Adresse, prüft, für welchen User-Agent er die Antwort optimieren muss und welche Sprachen akzeptiert werden. Dann schreibt er eine Request, welche sagt, ob die Anfrage erfolgreich war oder nicht und sie je nachdem dann beantwortet mit Infos zur angeforderten Datei.

### Mein Webserver in Python

Ich habe als Ausgangslage einen Standard-http-Webserver-Skript verwendet.

«

**import** **http.server**

**import** **socketserver**

PORT = 8000

Handler = http.server.SimpleHTTPRequestHandler

httpd = socketserver.TCPServer(("", PORT), Handler)

print("serving at port", PORT)

httpd.serve\_forever()

» (Python Software Foundation, 2017)

Dieses Programm hostet einen Server unter localhost:8000. Alle Requests welche eingehen werden hier automatisch vom SimpleHTTPRequestHandler bearbeitet. Meine Arbeit bestand nun darin, diesen zu ersetzen. Dazu muss ich wissen was ein Handler alles beinhalten muss und welche Prozesse wie ablaufen. Zuerst kommt die Request als Bytestring im Webserver an. Diese muss mit utf-8, der am weitesten verbreiteten Unicode-Kodierung decodiert werden. Dann wird sie geparsed, also in Teile zerlegt, damit der Handler die einzelnen Elemente der Request verarbeiten kann. Dann wird die Anfrage bearbeitet und eine Response zusammengestellt. Danach muss diese jedoch auch wieder verschlüsselt werden. Der Responsebody wird jeweils wieder mit utf-8 codiert, der Body hängt von der Dateiart ab.

Nun musste ich mir Gedanken machen, was mein Webserver braucht. Ich habe die Maindatei wie oben behalten. Dann brauche ich einen Requestparser, eine Klasse welche testet welche Methode verwendet wird und eine Responsebauklasse. Ich habe nun also einen Requestparser erstellt, welcher die Zeilen der Request auseinandernimmt und dann ein Dictionary mit Headerfeld(z.B. Host) und Inhalt(z.B. www.google.com) für den Inhalt des Headers erstellt und sofern vorhanden den Body speichert. Danach wird die Methode getestet. Je nach Methode wird dann ein entsprechender Responseheader mithilfe des Dictionarys erstellt und das File bearbeitet oder gesendet. Mein Handler kann nur PUT und GET bearbeiten. Falls nun also eine andere Methode eingeht oder das File bei GET nicht vorhanden ist, wird eine Errorresponse mit entsprechendem Statuscode zurück gesendet. Mein Programm ist so aufgebaut, das man die Maindatei startet. Diese hostet dann einen Webserver und sobald eine Request eingeht wird diese erfasst. Dann startet die Maindatei den Requestparser, welcher die Request bearbeitet. Dann wird die Testmethode aufgerufen. Die Methode aus der Request wird getestet und die Testmethode ruft den Responsebau auf, welcher die jeweilige Response zusammenstellt. Dann wird die Response zurück an die Maindatei gegeben, welche diese dann an den Empfänger zurückschickt. Wie bei Webservern üblich wird, falls bei der GET-Methode kein Pfad angegeben ist die index.html-Datei aufgerufen. In dieser kann man dann auf die anderen Dateien im Verzeichnis mit einem Klick zugreifen.

### Probleme bei der Programmierung

Da es beim Programmieren so ist, dass nur schon ein kleiner Fehler, wie fehlende Klammern, dafür sorgen kann, dass das Ganze Programm nicht funktioniert, kam es mir gelegen, dass ich PyCharm benutzte, da dieses viele solcher Fehler erkennen kann.

Trotzdem kam es bei verschiedenen Teilen meines Programms zu Problemen. Der vermutlich schwierigste Teil war der Requestparser. Zuerst habe ich ihn so umgesetzt, dass er Elemente aus einzelnen bestimmten Zeilen als Variablen speicherte. Jedoch war mir bis dahin noch nicht klar, dass die Zeilen ausser der ersten keine bestimmte Reihenfolge haben müssen. Auch habe ich nicht beachtet, dass die Anzahl Zeilen nicht definiert war, ein möglicher Body nicht berücksichtigt wurde und die Anzahl Zeilen nicht vordefiniert war. Dann habe ich zuerst versucht die Zeilen von 2 bis zum Ende mit einem Loop in einem Dictionary zu speichern. Doch auch hier war der Body und allgemein Zeilen ohne einem Doppelpunkt nicht berücksichtigt. Schlussendlich habe ich alles in eine Schleife eingebaut, welche jede Zeile einzeln auf Doppelpunkt oder http im Inhalt testet und diese dann je nachdem unterschiedlich parsed. Falls ein http vorkommt, ist es die erste Zeile und muss in Methode, Version und Pfad zerlegt werden, welche dann als Variablen abgespeichert wurden. Fall ein Doppelpunkt vorkommt war es eine Zeile aus dem Header und konnte ins Dictionary abgespeichert werden. Falls die Zeile weder Doppelpunkt noch http enthält, ist es eine Zeile des Body und wird deshalb dem String «Body» hinzugefügt.

Ein anderes Problem tauchte beim Responsebau auf. Ich habe zuerst jede Response einzeln zusammengefügt. Da dies sehr unübersichtlich ist, habe ich eine Vorlage mit mehreren Lücken gemacht, welche dann an die jeweilige Methode angepasst formatiert werden kann.

Nachdem der Webserver eigentlich fertig sein sollte, habe ich bei einem Test festgestellt, dass die Request verschlüsselt als Bytestring ankommt. Daher musste ich noch Code einbauen, damit die Request mit UTF-8 decodiert wird und am Schluss die Response wieder encodiert wird. Dabei war zu berücksichtigen, dass der Body bei der Response je nach Typ der Datei nicht immer mit UTF-8 encodiert werden muss.

# Schlussteil

Im Verlaufe meiner Maturaarbeit konnte ich mir viel neues Wissen aneignen. Sowohl Wissen über die Funktionsweise des Internets und das http-Protokoll, als auch Erfahrung im fortgeschrittenen Umgang mit Python, Objektorientierte Programmierung. Da die meisten Programmiersprachen ähnlich aufgebaut sind, wird mir dies beim Erlernen dieser viel helfen. Auch hat mich diese Arbeit im Vorhaben Informatik zu studieren gestärkt und auch für dieses viel Erfahrung gebracht.

# Literaturverzeichnis

# Glossar

# Anhang

# Glossar

1. SSH oder Secure Shell ist ein Netzwerkprotokoll für verschlüsselte Netzwerkkommunikation. [↑](#endnote-ref-1)
2. Das CERN ist die Europäische Organisation für Kernforschung mit Sitz in der Schweiz. [↑](#endnote-ref-2)
3. HTML ist die sogenannte Hypertext Markup Language. Mit HTML kann man Dokumente unter anderem im WWW strukturiert darstellen. [↑](#endnote-ref-3)
4. Ein RFC ist ein sogenannter Request for Comment ist eine Art der Publikation für das IETF. [↑](#endnote-ref-4)
5. Die Internet Scoiety ist Veranwortlich für Pflege und Weiterentwicklung des Internets zuständig. Sie besteht aus Einzelpersonen sowie mehreren anderen Unterorganisationen wie z.B. IETF. [↑](#endnote-ref-5)
6. Die Internet Engineering Task Force arbeitet an der technischen Weiterentwicklung des Internets. [↑](#endnote-ref-6)
7. Das Internet Architecture Board ist ein Komitee, welches den architekturellen Überblick über die Aktivitäten der IETF wahrt und die Internet Society unterstützt. [↑](#endnote-ref-7)
8. Eine Methode in einer Request definiert, welche Aktion ausgeführt wird. Je nachdem wird eine Datei erstellt, bearbeitet, gelöscht oder einfach gesendet. [↑](#endnote-ref-8)