

Über dieses Skriptum

Der Zweck dieses Skriptums ist es, den TeilnehmerInnen der Lehrveranstaltung **Web Basics** einen grundlegenden Überblick über die in der Lehrveranstaltung behandelten Themen zu geben. Das Skriptum ist aber kein vollständiges Abbild der Lehrveranstaltungsinhalte, die Studierenden sind also dazu angehalten, sich wo nötig selbst Notizen zu machen.

Sollten Sie Fehler finden, Verständnisprobleme haben oder den Eindruck gewinnen, dass man sonst etwas am Skriptum verbessern kann, dann wenden Sie sich bitte an [den Lehrenden](#).

Nachdem die Lehrveranstaltung sich mit dem *World Wide Web* auseinandersetzt, ist es nur logisch, dass auch das Skriptum webbasiert ist. Das Skriptum wurde mit dem Framework [Docusaurus](#) erstellt und ist auf [Github](#) öffentlich zugänglich.

Interaktive Elemente

Das Skriptum beinhaltet auch interaktive Elemente, mit denen die Studierenden das eben gelernte direkt im Skriptum ausprobieren können:

[Edit in JSFiddle](#)

- [HTML](#)
- [Result](#)

```
<!DOCTYPE html>
<html>
<body>

<h1>Web Basics</h1>
<p>Sie können diesen Text im Reiter HTML editieren, im Reiter Result das Ergebnis ansehen!</p>

</body>
</html>
```

Im Reiter `HTML` (bzw. später in den Reitern `JS` und `CSS`) können Sie die jeweilige Datei, im Reiter `Result` das Ergebnis ansehen. Sie können die Beispiele natürlich auch in JSFiddle weiterbearbeiten und damit herumexperimentieren.

Die eine Sekunde...

Sie haben gerade diese Seite hier geöffnet, wahrscheinlich über einen Link. Die Zeitspanne zwischen dem Klick auf den Link und der Darstellung dieser Seite war wahrscheinlich kürzer als eine Sekunde. In dieser einen Sekunde sind aber sehr viele Dinge passiert: Ihr Rechner bzw. Ihr Browser hat:

- irgendwie herausgefunden, wie er den Rechner mit der Adresse dieser Seite erreicht
- eine Verbindung zu diesem Rechner aufgebaut
- diese Verbindung mittels moderner kryptographischer Methoden abgesichert
- einige Dateien von diesem Rechner heruntergeladen
- die in diesen Dateien enthaltenen Informationen für Ihr Ausgabegerät optimiert dargestellt

... und noch einiges mehr.

Was in dieser einen Sekunde vor sich gegangen ist, wird uns in diesem Semester beschäftigen. LeserInnen dieses Skriptums sollen am Ende einerseits verstehen, was in den oben genannten Schritten passiert, andererseits sollen sie auch Fähigkeiten erwerben, um selbst Webseiten zu bauen.

Das Web

Der Titel dieses Skriptums ist **Web Basics**. Was aber ist *das Web*? *Web* ist eine Kurzbezeichnung für das **World Wide Web**, also ein weltweites Netz. Im Gegensatz zum *Internet*, das ein Verbund von Rechnernetzen ist, ist das *World Wide Web* ein System von elektronischen Dokumenten, sogenannten *Webseiten*. Das *World Wide Web* ist über das Internet abrufbar, basiert also auf diesem.

Das World Wide Web basiert auf drei essenziellen Standards:

- einem System global eindeutiger Bezeichner für Ressourcen im Web, die sogenannte *URL* (uniform resource locator) bzw. *URI* (uniform resource identifier), siehe u.a. [RFC 3986](#), z.B. <https://michivo.github.io/webbasics/>
- einem Protokoll, mit dem Clients (i.A. Browser) Informationen von Webservern anfordern können: Dem HyperText Transfer Protocol HTTP
- einer Auszeichnungssprache, die festlegt, wie Informationen gegliedert und miteinander verknüpft sind: Der [HyperText Markup Language HTML](#)

Später kamen weitere Standards dazu:

- Cascading Style Sheets (CSS), die getrennt vom Inhalt einer Webseite ihr Aussehen beschreiben
- das Document Object Model (DOM) als Programmierschnittstelle, mit der z.B. Skripte (meist in JavaScript geschrieben) im Browser u.a. auf Webseiten zugreifen können
- HyperText Transfer Protocol Secure (HTTPS), mit dem der gesamte Datenverkehr verschlüsselt werden kann

Diese Standards werden größtenteils vom [World Wide Web Consortium](#) verwaltet.

Eine kurze Geschichte des WWW

1989-1991: Die Geburtsstunde des World Wide Web

1989 skizzierte der englische Forscher Tim Berners-Lee die Idee einer großen Hypertext-Datenbank mit Verknüpfungen zwischen den darin enthaltenen Hypertext-Dokumenten.

NOTE

Was ist Hypertext?

- Hypertext ist nicht notwendigerweise linear
- Hypertext ist ein Text, der Links zu anderen Texten enthält
- Hypermedia ist ein Hypertext, der auch Grafiken, Videos oder Tonspuren enthalten kann
- Hypertext und Hypermedia sind Konzepte, keine Produkte

(siehe [1])

Berners-Lee arbeitete zu dieser Zeit beim CERN, bekam dort die notwendige Unterstützung für sein Projekt. Zu Weihnachten 1990 hatte Berners-Lee alle Teile implementiert, die für das Web benötigt wurden:

- HTTP und HTML waren ausgearbeitet
- der erste Web Browser (und Web Editor) war programmiert und in Betrieb
- die erste HTTP Server-Software lief auf dem ersten Web Server

Im August 1991 war die erste Webseite im Internet online, sie sind auch heute noch erreichbar:

<http://info.cern.ch/hypertext/WWW/TheProject.html>

1992-1995: Das Web wächst

Ursprünglich waren Berners-Lees Entwicklungen proprietär. 1993 beschloss das CERN, sämtliche Software und Protokolle, die mit dem WWW verbunden waren, frei und öffentlich zugänglich zu machen. Während das WWW anfangs nur an wissenschaftlichen Institutionen

zugänglich war, wuchs es in den nächsten Jahren rasch. Ursprünglich navigierte man ausschließlich über Verknüpfungen zwischen den Seiten, es gab noch keine Suchmaschinen im modernen Sinn. Die ersten Browser mit graphischer Benutzeroberfläche entstanden, so z.B. Mosaic (Grundlage für Netscape Navigator, viel später Grundlage für Mozilla Firefox).

Die ersten Internetunternehmen (Yahoo, amazon, ...) werden gegründet, Yahoo Directory wird zu einem der populärsten Webverzeichnisse.

Im Oktober 1994 wird (u.a. von Tim Berners-Lee) das **World Wide Web Consortium (W3C)** gegründet. Das Ziel des W3C ist es, einheitliche Standards für das WWW zu definieren.

1996-2003: Die Kommerzialisierung des Webs

Zahlreiche weitere Internetunternehmen werden gegründet (eBay 1995, Google 1998, ...), das Internet wird als Werbeplattform entdeckt. Microsoft entwickelt die erste Version des Internet Explorer (1995).

Das W3C entwickelt HTML kontinuierlich weiter (HTML 2 im Jahr 1995, HTML 3 und 4 im Jahr 1997), 1996 wird die erste Version der CSS-Spezifikation veröffentlicht. Trotz der Versuche, HTML und CSS zu standardisieren, kommt es zum Browserkrieg zwischen Netscape und Microsoft: Jedes der beiden Unternehmen versucht, die Standards eigens zu interpretieren und zu erweitern, um dem jeweiligen Konkurrenten das Leben schwer zu machen.

1996 wird JavaScript 1.0.0 von Netscape veröffentlicht, womit eine dynamische Manipulation von Seiten ermöglicht wurde.

2004-heute: Web 2.0, das Web wird allgegenwärtig

Soziale Netzwerke entstehen (MySpace, Facebook, YouTube, ...). Während Content im Web ursprünglich hauptsächlich von Seitenbetreibern generiert wurde, können nun auch Nutzer von Webseiten Inhalte ins Web stellen und über das Web miteinander interagieren und kommunizieren. Das Web wird allgegenwärtig: Jeder kann jederzeit von überall darauf zugreifen, darüber kommunizieren und Informationen teilen.

Führende Browserhersteller (Apple, Mozilla, Opera) gründen 2004 aus Frust über die langsame Entwicklung der Web-Standards durch das W3C die **Web Hypertext Application Technology Working Group (WHATWG)**. Heute besteht die Steuerungsgruppe der WHATWG aus Apple,

Mozilla, Google und Microsoft. Die WHATWG entwickelte einen auf HTML 4.01 aufbauenden neuen Standard namens HTML 5, der 2007 von der W3C übernommen wurde. Nach einigen Querelen um die Weiterentwicklung des HTML-Standards ist seit Mai 2019 offiziell die WHATWG für die Pflege und Weiterentwicklung des **HTML Living Standard** (früher HTML 5) verantwortlich.

Quellen

[1] <https://www.w3.org/WhatIs.html>

Protokolle und das OSI-Modell

In diesem Skriptum wurde HTTP, das *Hypertext Transfer Protocol* bereits mehrfach erwähnt. Nachdem schon kurz umrissen wurde, was *Hypertext* ist und die Bedeutung von *Transfer* (also Übertragung) wohl keiner weiteren Erläuterung bedarf, bleibt noch der Begriff **Protocol**.

Ein Protokoll in der Informatik bzw. Telekommunikation ist eine Vereinbarung, nach der Datenübertragung zwischen zwei (oder mehreren) Parteien abläuft. Einige für das Internet wichtige Protokolle sind z.B. das Internet Protocol (IP) und das Transmission Control Protocol (TCP). Netzwerkprotokolle sind in der Regel aus Schichten aufgebaute Protokollstapel (d.h. HTTP basiert auf dem TCP, TCP setzt auf das IP, IP auf Ethernet). Das Standardmodell für solche Protokolle im Netzwerkbereich ist das 7-Schichten-OSI-Modell (auch ISO/OSI-Modell), für das Internet ist jedoch das 4-schichtige TCP/IP-Referenzmodell relevant.

Das TCP/IP-Referenzmodell

Wie bereits erwähnt, besteht das TCP/IP-Referenzmodell aus 4 Schichten. Auch wenn wir uns in diesem Skriptum ausschließlich mit Protokollen der 4. Schicht auseinandersetzen (DNS, HTTP, HTTPS), ist die grundlegende Kenntnis des Protokollstapels hilfreich für das Gesamtverständnis.

Netzzugangsschicht / Link Layer

Die Netzzugangsschicht beinhaltet Protokolle, die grundsätzlich einen Netzzugang bzw. eine Punkt-zu-Punkt-Datenübertragung ermöglichen. In den meisten Fällen kommen hier Protokolle wie Ethernet oder 802.11 (WLAN) zum Einsatz.

Internetschicht / Internet Layer

Die Internetschicht ist für die Weitervermittlung von Paketen und das Routing zuständig. Für den Internet Layer gibt es zwei Protokolle bzw. Protokollversionen: IPv4 und IPv6. Die Aufgabe dieser Schicht ist es, zu einem empfangenen Paket das nächste Zwischenziel (den nächsten "Hop") zu ermitteln und das Paket dorthin weiterzuleiten.

Transportschicht / Transport Layer

Die Transportschicht ermöglicht Ende-zu-Ende-Verbindungen, also z.B. von Ihrem Endgerät zu einem Server. Protokolle auf dieser Ebene ermöglichen den Austausch von Daten über eine Verbindung zwischen zwei Netzteilnehmer. Die wichtigsten Protokolle auf dieser Ebene sind das Transmission Control Protocol (TCP), Transport Layer Security (TLS) und das User Datagram Protocol (UDP).

Anwendungsschicht / Application Layer

Die Anwendungsschicht umfasst alle Protokolle, die von Anwendungsprogrammen (wie z.B. einem Webbrowser oder einem Webserver) für den Austausch anwendungsspezifischer Daten genutzt werden. Auf dieser Ebene gibt es eine große Anzahl unterschiedlicher standardisierter und nicht-standardisierter Protokolle, zum Beispiel DNS, FTP, HTTP, POP3, HTTPS, IMAP, SMTP, SSH, ...

DNS - Das Domain Name Service

Das Internet als IP-basiertes Netzwerk kennt *IP-Adressen*, keine *Domain-Namen* wie z.B. campus02.at. Wir als Benutzer arbeiten aber nur in seltenen Fällen direkt mit IP-Adressen, in der Regel kennen wir nur Domain-Namen wie z.B. campus02.at. Die Umwandlung eines Domain-Namens in eine IP-Adresse geschieht über das **Domain Name Service (DNS)**.


Der Domäne campus02.at ist zum Beispiel die IPv4-Adresse 149.154.100.47 zugeordnet (Stand 26.6.2020), der Domäne de.wikipedia.org die IPv4-Adresse 91.198.174.192 und die IPv6-Adresse 2620:0:862:ed1a::1.

Das DNS funktioniert also ähnlich einem Telefonbuch: Im Telefonbuch werden Unternehmen oder Personen Telefonnummern zugeordnet, im DNS werden Domain-Namen IP-Adressen zugeordnet. Im Falle des DNS werden die Domains in einem global verteilten System gespeichert, das den Namensraum des Internets verwaltet.

Auflösung eines DNS-Requests

Angenommen, Sie wollen von Ihrem Rechner eine Verbindung zu `www.campus02.at` aufbauen. Dazu brauchen Sie, wie bereits erwähnt, die *IP-Adresse*, die der Domain `www.campus02.at` zugeordnet ist. Wie wird nun so ein DNS-Request aufgelöst?

1. Ihr Rechner sucht in der Hosts-Datei bzw. in einem lokalen Cache, ob die IP-Adresse für `www.campus02.at` hinterlegt ist. Wenn nicht, ...
2. Ihr Rechner kontaktiert den DNS-Server, der in der Regel über DHCP automatisch zugewiesen wurde. Der DNS-Server wird in der Regel von Ihrem Internet Service Provider (ISP) betrieben. Der DNS-Server überprüft, ob er die IP-Adresse für `www.campus02.at` kennt. Wenn nicht, ...
3. Der DNS-Server fragt bei einem der weltweit 13 Root-Nameserver nach `www.campus02.at`. Der Root-Nameserver weiß, in welcher DNS-Zone `.at` verwaltet wird und sendet Namen und IP-Adressen der `.at`-Nameserver an den DNS-Server Ihres ISPs.
4. Der `.at`-Nameserver sendet dem DNS-Server die Namen und IP-Adressen der Nameserver für `campus02.at`.
5. Der DNS-Server fragt den Nameserver für `campus02.at` nach der IP-Adresse für `www.campus02.at`.

6. Der Nameserver antwortet dem DNS-Server mit der IP-Adresse. Der DNS-Server antwortet Ihrem Rechner ebenso mit dieser IP-Adresse.
7. Ihr Rechner kennt die IP-Adresse von `www.campus02.at`, Sie können eine Verbindung zu diesem Rechner aufbauen. 

Glossar

In diesem Skriptum werden Begriffe wie *Client*, *Server* oder *Browser* immer wieder vorkommen. Um sicherzustellen, dass es zu keinen Missverständnissen bei diesen Begriffen kommt, hier ein kurzer Überblick:

Client

Ein Programm, das Dienste von einem Server abrufen. Meist laufen Client und Server auf unterschiedlichen Rechnern, kommunizieren also meist über ein Netzwerk (wie das Internet) miteinander.

Browser

Ein Programm auf einem Rechner (PC, Tablet, Smartphone, ...) das in der Lage ist, Webseiten darzustellen. Der Browser ist ein *Client*-Programm. Beispiele für Browser sind Chrome, Firefox, Edge, Chromium, Internet Explorer, Safari und Opera.

Server

Ein Programm, das anderen Programmen (den Clients) Dienste anbietet. Wie bereits erwähnt wird die Bezeichnung Server in diesem Skriptum für *Software*, nicht für *Hardware* verwendet.

HTTP-Server oder Webserver

Ein Programm auf einem Rechner, das Dokumente, Dateien und weitere Daten meist über das Internet an Clients wie z.B. Browser überträgt. Es hat sich eingebürgert, auch die Rechner, auf denen solche Programme laufen, als Server zu bezeichnen, in diesem Skriptum ist mit HTTP-Server oder Webserver jedoch immer das Programm (die Software) und nicht der Rechner (die Hardware) gemeint. Die gängigsten HTTP-Server sind [Apache](#), [nginx](#), Microsoft Internet Information Services und <https://nodejs.org/en/>

Proxy-Server

Ein Programm, das Anfragen von einem Client entgegennimmt und an einen Server weiterleitet. Ein Client (wie z.B. ein Browser) stellt also keine direkte Verbindung zu einem Webserver her, sondern öffnet eine Verbindung zum Proxy-Server. Möchte der Client eine Webseite öffnen, schickt er also eine Anfrage an den Proxy-Server. Hat der Proxy-Server z.B. die angefragte Seite im Cache, kann er die Anfrage beantworten, ohne den Zielservice zu kontaktieren.

Proxies werden oft aus Sicherheitszwecken, zur Performanceoptimierung oder aus Datenschutzgründen eingesetzt.

Lernziele

Nach der Einführung sollten Sie folgende Fragen beantworten können:

- Was ist das World Wide Web?
- Was passiert zwischen dem Moment, in dem Sie eine URL in Ihrem Browser eingeben und jenem, in dem die Webseite in Ihrem Browser angezeigt wird?
- Welche sind die wichtigsten Standards, auf denen das WWW basiert?
- Was ist das TCP/IP-Referenzmodell?
- Wie funktioniert die Namensauflösung über das DNS?

HTTP - das HyperText Transfer Protocol

Das Hypertext Transfer Protocol (HTTP, englisch für Hypertext-Übertragungsprotokoll) ist ein zustandsloses Protokoll zur Übertragung von Daten auf der Anwendungsschicht über ein Rechnernetz. Es wird hauptsächlich eingesetzt, um Webseiten (Hypertext-Dokumente) aus dem World Wide Web (WWW) in einen Webbrowser zu laden. Es ist jedoch nicht prinzipiell darauf beschränkt und auch als allgemeines Dateiübertragungsprotokoll sehr verbreitet. (Quelle: https://de.wikipedia.org/wiki/Hypertext_Transfer_Protocol).

HTTP ist also

- ein zustandsloses Protokoll: Anfragen sind voneinander unabhängig. Anfragen haben keinen Bezug zu früheren Anfragen.
- ein Protokoll auf der Anwendungsschicht des [TCP/IP-Referenzmodells](#)
- ein Protokoll zur Übertragung von Daten über ein Rechnernetz, meist über das Internet

Die Kommunikation bei HTTP findet stets zwischen einem HTTP-Client (meist einem Browser, wie Chrome, Firefox, Edge, ...) und einem HTTP-Server (nginx, Apache, MS IIS, ...) statt. Der Client schickt **Anfragen/Requests** an den Server, der Server reagiert darauf mit **Antworten/Responses**. Anfragen und Antworten werden als **Nachrichten** bezeichnet, jede Nachricht besteht aus einem **Header** und optional einem **Body**. Der Header enthält Daten über den Body, damit dieser vom Empfänger der jeweiligen Nachricht richtig verarbeitet werden kann.

Beispiel-Anfrage

HTTP-Requests sind auch für Menschen lesbar. Will man zum Beispiel die Internetseite mit der URL <http://info.cern.ch/hypertext/WWW/TheProject.html> abrufen, wird zuerst (siehe [intro-dns](#)) der Domainname `info.cern.ch` aufgelöst. Der Domainname ist mit der IP-Adresse 188.184.100.82 verknüpft, zum Rechner mit dieser IP-Adresse wird dann folgende Anfrage geschickt:

```
GET /hypertext/WWW/TheProject.html HTTP/1.1
Host: info.cern.ch
```

- `GET` ist dabei das sogenannte **HTTP-Anfragemethode** und bezeichnet die Operation, die ausgeführt werden soll

- `/hypertext/WWW/TheProject.html` ist die Ressource, auf die zugegriffen werden soll
- `HTTP/1.1` ist der Protokollbezeichner - es wird also HTTP Version 1.1 für die Kommunikation verwendet
- `Host` ist ein **HTTP-Header**, der festlegt, für welchen DNS-Namen die Anfrage gedacht ist. So ist es möglich, mehrere Webseiten auf einem Rechner mit einer IP-Adresse zu betreiben. Der Wert dieses Headers ist der DNS-Name `info.cern.ch`.

Eine Anfrage *kann* im Allgemeinen auch einen Body haben, eine **GET**-Anfrage hat jedoch keinen Body, sie besteht nur aus einem Header.

```
HTTP/1.1 200 OK
Date: Tue, 30 Jun 2020 09:24:25 GMT
Server: Apache
Last-Modified: Thu, 03 Dec 1992 08:37:20 GMT
ETag: "40521e06-8a9-291e721905000"
Accept-Ranges: bytes
Content-Length: 2217
Connection: close
Content-Type: text/html

[Antwort-Body]
```

► Klicken Sie hier, um den gesamten Antwort-Body zu sehen (zur besseren Lesbarkeit formatiert).

Die Bedeutung der Response-Header ist wie folgt:

- `HTTP/1.1` - das Ergebnis entspricht HTTP Version 1.1
- `200 OK` - der Ergebnisstatuscode ist 200 OK, die Anfrage konnte erfolgreich bearbeitet werden
- `Date: Tue, 30 Jun 2020 09:24:25 GMT` - Datum der Response
- `Server: Apache` - die Response kommt von einem Apache-Server
- `Last-Modified: Thu, 03 Dec 1992 08:37:20 GMT` - das abgefragte Dokument wurde zuletzt 1992 modifiziert
- `ETag: "40521e06-8a9-291e721905000"` - Entity Tag, wird u.a. für Caching verwendet
- `Accept-Ranges: bytes` - Hiermit gibt der Server bekannt, dass er auch partielle Requests verarbeiten kann (z.B. wenn ein Download unterbrochen wurde)
- `Content-Length: 2217` - Länge des Response-Bodys
- `Connection: close` - Darunterliegende TCP-Verbindung soll geschlossen werden
- `Content-Type: text/html` - Das Ergebnis enthält HTML

HTTP für Webservices

HTTP wird nicht nur als Protokoll zur Übertragung von Webseiten verwendet, sondern auch für eigenständige Applikationen, sogenannte Webservices. Einige Beispiele dafür sind:

- Webservices zum Abfragen von statistischen Daten von Aktienkursen bis hin zu epidemiologischen Daten
- Webservices im Bereich *Open Data*. Behörden und Ämter stellen dabei Daten zur Verfügung, die im Interesse der Allgemeinheit sind (siehe <https://www.data.gv.at/>)
- Webservices zum Verschicken von Nachrichten als SMS, Mail durch Anbieter wie Twilio/Sendgrid
- Webservices im Betrieb verteilter Sensornetzwerke, wie z.B. Feinstaubsensoren in Graz (siehe <https://luftdaten.info/> bzw. <http://api.luftdaten.info/static/v1/data.json>)

Diese Webservices können zwar auch von Webanwendungen, die in einem Browser laufen, genutzt werden, können aber gleichermaßen von Desktopanwendungen oder mobilen Apps verwendet werden.

HTTP - Anfragemethoden

Wie bereits erwähnt, ist die **Anfragemethode** Teil jedes Requests. Es gibt folgende Anfragemethoden:

GET

Mit GET wird eine Ressource (also z.B. ein HTML-Dokument) abgefragt. GET ist die am häufigsten verwendete Methode. GET-Requests sollen keinen Body haben (teilweise werden GET-Requests mit Body von Servern verworfen, in der Regel wird der Body ignoriert). In vielen Fällen entspricht die Ressource, die abgefragt wird, einer Datei, das muss aber nicht sein - mit GET können beliebige Arten von Ressourcen abgefragt werden.

Antworten auf GET-Requests *können* gecacht werden.

HEAD

Die HEAD-Methode verhält sich mit einer Ausnahme gleich mit GET: HEAD liefert keinen Body. HEAD-Requests werden dafür verwendet, um zu überprüfen, ob eine Ressource erreichbar ist oder ob sie sich verändert hat.

POST

Mit der POST-Methode werden Daten an den Server zur weiteren Verarbeitung geschickt. Mit einer POST-Methode können neue Ressourcen am Server angelegt werden oder bestehende Ressourcen modifiziert werden. Mit POST-Requests kann man zum Beispiel

- Bilder oder andere Dateien auf eine Webseite hochladen
- Login-Daten (Benutzername & Passwort) sicher über eine verschlüsselte Verbindung an einen Server schicken
- Formulardaten an einen Server schicken

PUT

Mit der PUT-Methode kann man neue Ressourcen erzeugen oder eine bestehende Ressource durch eine neue ersetzen. Im Unterschied zur POST-Methode wird bei der PUT-Methode immer genau angegeben, welche Ressource neu angelegt bzw. ersetzt werden soll. Wird ein Bild mit POST hochgeladen, wäre die URI `userdata/images` (d.h. es wird eine neue Ressource innerhalb von `userdata/images` angelegt). Wird ein Bild mit PUT hochgeladen, würde man stattdessen die

URI `userdata/images/mylittlepony` angeben, d.h. man legt eine neues Bild unter dieser URI an oder ersetzt ein allenfalls vorhandenes Bild.

DELETE

Mit der DELETE-Methode wird die angegebene Ressource gelöscht.

PATCH

Mit PATCH kann eine bestehende Ressource geändert werden.

TRACE

Der Server antwortet mit der Anfrage, so wie er sie bekommen hat

OPTIONS

Der Server antwortet mit einer Liste der von ihm unterstützten Methoden und Merkmale

CONNECT

Wird in der Kommunikation über HTTP-Proxies benötigt, um gesicherte Verbindungen durch ein oder mehrere Proxies zu tunneln.

Im weiteren Verlauf der Vorlesung haben wir ausschließlich mit GET, POST, PUT und DELETE zu tun.

HTTP - Response Codes

Jede Response von einem HTTP-Server beginnt mit einem Response-Code, der dem Client Informationen darüber gibt, ob die von ihm gestellte Anfrage erfolgreich beantwortet werden konnte oder nicht. Wenn eine Anfrage nicht erfolgreich beantwortet werden konnte, enthält der Response Code Informationen darüber, was der Grund für den Fehler war.

Der Response Code ist eine dreistellige Dezimalzahl, deren erste Stelle die Statusklasse darstellt:

- 1xx -> Informationen - die Anfrage wurde entgegengenommen und ist in Bearbeitung
- 2xx -> Erfolgreiche Operationen - die Anfrage wurde entgegengenommen, verstanden und erfolgreich bearbeitet
- 3xx -> Umleitung - für die Bearbeitung der Anfrage muss der Client noch etwas machen
- 4xx -> Client-Fehler - die Anfrage ist ungültig bzw. kann nicht verarbeitet werden
- 5xx -> Server-Fehler - die Anfrage war zwar gültig, bei der Bearbeitung ist es am Server aber zu einem Fehler gekommen

Informationen

100 Continue

Eine Zwischenantwort, mit der der Server anzeigt, dass bisher alles in Ordnung ist und der Client mit seinem Request weitermachen soll. Möchte der Client z.B. große Mengen an Daten schicken, kann er zuerst einen Request ohne Body schicken, auf den der Server dann mit `100 Continue` oder einem entsprechenden Fehlercode antworten kann. Nur bei `100 Continue` würde der Client auch tatsächlich Daten schicken.

101 Switching protocols

Der Client hat beim Server um einen Protokollwechsel angefragt, der Server antwortet, dass das in Ordnung ist. Dieser Status Code kommt selten vor.

102 Processing

Der Server teilt dem Client mit, dass sein Request noch verarbeitet wird. Der Client weiß also, dass der Server noch am Request arbeitet und kein Timeout (oder ein anderer Fehler) aufgetreten ist.

Erfolg

200 OK

Der häufigste Response-Code. Die Anfrage wurde erfolgreich bearbeitet, der Body enthält die angeforderten Daten

201 Created

Die Anfrage wurde erfolgreich bearbeitet, eine oder mehrere Ressourcen wurden erfolgreich angelegt.

202 Accepted

Die Anfrage wurde akzeptiert, aber noch nicht vollständig verarbeitet.

203 Non-Authoritative Information

Die Antwort kommt von einem Proxy, der vom Zielserver eine Antwort mit Status `200 OK` bekommen hat, dem Client aber eine veränderte Version der Antwort weitergibt.

204 No Content

Der Server hat die Anfrage des Clients entgegengenommen und erfolgreich verarbeitet, es gibt aber keinen Body.

205 Reset Content

Der Server hat die Anfrage erfolgreich bearbeitet und teilt dem Client mit, dass er das aktive Dokument zurücksetzen bzw. neu laden soll.

206 Partial Content

Der Server liefert nur einen Teil der Antwort im Body, weil der Client das so angefordert hat (z.B. um einen zuvor abgebrochenen Download fortzusetzen).

Die weiteren 2xx-Codes haben kaum praktische Relevanz, werden hier daher nicht weiter behandelt.

Umleitung

300 Multiple Choices

Es gibt mehrere Optionen für die angefragte Ressource (z.B. mehrere unterschiedliche Videoformate, ...), der Client muss erst entscheiden, welches Format er bevorzugt.

301 Moved Permanently

Die Ressource, die unter dieser URI zu finden war, ist nun unter einer neuen URI zu finden. Der Client soll in Zukunft nur noch die neue URI verwenden.

302 Found

Die Ressource, die unter dieser URI zu finden war, ist temporär unter einer neuen URI zu finden. Diese Umleitung kann sich in Zukunft ändern, der Client sollte daher bei späteren Requests immer die ursprüngliche URI verwenden. Der Body der Antwort sollte einen kurzen Hinweis auf die Umleitung und einen Link zur neuen URI enthalten.

303 See Other

Der Server leitet die Anfrage des Clients zu einer anderen Ressource um, die eine indirekte Antwort auf die ursprüngliche Anfrage des Clients gibt.

307 Temporary Redirect

Temporary Redirect verhält sich bis auf ein Detail gleich wie `302 Found`. Der einzige Unterschied ist, dass `302 Found` aus historischen Gründen eine Änderung der Methode von `POST` zu `GET` zulässt.

Client-Fehler

400 Bad Request

Der Server kann die Anfrage nicht beantworten, weil die Anfrage ungültig ist (Fehlende Daten, ungültige Syntax, ...)

401 Unauthorized

Für die Anfrage muss sich der Client erst gegenüber dem Server authentifizieren.

403 Forbidden

Der Server hat die Anfrage verstanden, der Client ist aber nicht dazu berechtigt, auf die angefragte Ressource zuzugreifen. Der Client sollte nicht versuchen, die Anfrage noch einmal gleich zu stellen, kann sie aber mit neuen Zugriffsinformationen erneut senden. Server können statt mit `403 Forbidden` auch mit `404 Not Found` antworten, wenn sie aus Sicherheitsgründen dem Client nicht mitteilen wollen, dass die angefragte Ressource überhaupt existiert.

404 Not Found

Der Server kann die angefragte Ressource nicht finden oder will dem Client aus Sicherheitsgründen nicht mitteilen, dass die Ressource existiert. Dieser Statuscode sagt nichts darüber aus, ob dieser Zustand temporär oder permanent ist.

405 Method Not Allowed

Die angefragte Ressource existiert zwar, ein Zugriff mit der vom Client verwendeten Anfragemethode (GET, POST, ...) ist aber nicht möglich. In den Response-Headern teilt der Server dem Client die unterstützten Methoden mit.

406 Not Acceptable

Die angefragte Ressource steht nicht in der angefragten Form zur Verfügung. Der Server kann dem Client mitteilen, in welcher Form die Ressource angefragt werden kann. Diese *Form* ist üblicherweise ein Format (z.B. Datenformat XML, Videoformat, ...).

407 Proxy Authentication Required

Vergleichbar mit `401 Unauthorized`, jedoch muss sich der Client hier nicht gegenüber einem Server, sondern gegenüber einem Proxy authentifizieren.

408 Request Timeout

Innerhalb der vom Server erlaubten Zeitspanne wurde keine vollständige Anfrage des Clients empfangen.

409 Conflict

Die Anfrage kann nicht verarbeitet werden, weil die Zielressource in einem Zustand ist, der diese Anfrage nicht zulässt.

410 Gone

Die Zielressource ist am Server nicht mehr vorhanden, daran wird sich auch voraussichtlich nichts ändern.

Das sind die wichtigsten Fehlercodes für Client-Fehler, eine umfassendere Liste findet sich in den in den Quellenangaben verlinkten Dokumenten. Wie der Name schon sagt, ist (zumindest aus Sicht des Servers) der **Client** verantwortlich für die 4xx-er Fehler, während der Server für die 5xx-er-Fehler verantwortlich ist. Die Abgrenzung zwischen Client- und Serverfehlern ist jedoch nicht immer ganz klar.

Server-Fehler

500 Internal Server Error

Generischer Fehlercode für unerwartete Serverfehler.

501 Not Implemented

Die Funktionalität, um eine Anfrage zu bearbeiten, wurde noch nicht implementiert.

502 Bad Gateway

Diese Antwort kommt üblicherweise von einem Proxy, wenn er wiederum eine ungültige Antwort bekommen hat.

503 Service Unavailable

Der Server steht gerade nicht zur Verfügung, z.B. weil er gerade neu gestartet wird oder wegen Wartungsarbeiten.

504 Gateway Timeout

Diese Antwort kommt üblicherweise von einem Proxy, wenn er wiederum keine Antwort innerhalb der erwarteten Zeit bekommen hat.

505 HTTP Version not supported

Die im Request angegebene HTTP-Version wird nicht unterstützt.

Auch hier gibt es weitere Fehler-Codes, die oben genannten sind allerdings die häufigsten serverseitigen Fehler.

Quellen

- <https://tools.ietf.org/html/rfc7231#section-6>
- <https://developer.mozilla.org/de/docs/Web/HTTP/Status>
- <https://de.wikipedia.org/wiki/HTTP-Statuscode>
- https://en.wikipedia.org/wiki/List_of_HTTP_status_codes

HTTP Header

...