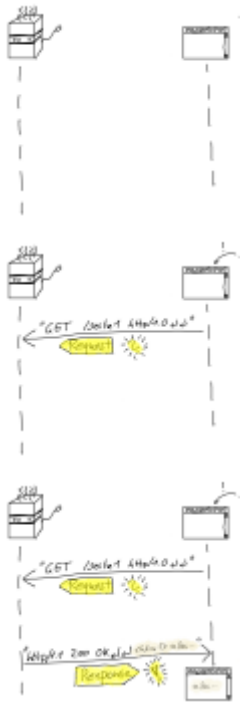
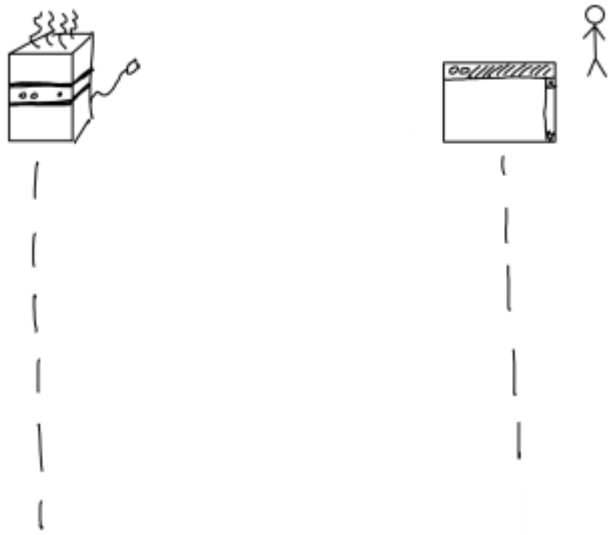
HTTP



Nachdem jetzt klar ist, wie grundsätzlich zwischen zwei Rechnern *gesprochen* wird, wird es etwas konkreter: Der Server ist jetzt der Web-Server und der Client ist der Web-Browser. Dafür müssen sie sich auf eine gemeinsame Sprache
verabreden, das ist das



Hyper**t**ext **T**ransfer **P**rotocol

Der Name scheint aus einer anderen Zeit zu stammen. Tatsächlich ist HTTP gar nicht so alt wie das Internet:

- RFC 1945 HTTP/1.0 (1996)
- RFC 2616 HTTP/1.1 (1999)
- RFC 7540 HTTP/2 (2015)

Zu den einzelnen Bestandteilen des Worts:

Was macht einen Text zu einem Hypertext?

Es geht nicht nur um Text im landläufigen Sinn. Der Text enthält selbst noch Metadaten, die Formatierung und Textsatz ermöglichen und (ganz wichtig) es sind Elemente mit Interaktion möglich. Zuerst nur *Links*, die zu einem anderen Dokument verweisen, aber auch Formularelemente o.ä..

Der Name steht für eine Gattung elektronischer Dokumente. Das prominenteste Format eines solchen Dokumententyps ist **HTML**:

Hyper**t**ext **M**arkup **L**anguage

Merke: Das ist kein Protokoll, das ist ein Dokumentenformat!

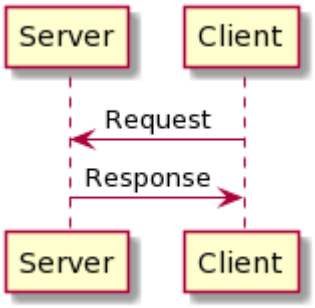
Das H in HTTP macht klar, dass das Protokoll unter anderem für den Zugriff auf Dokumente diesen Typs gebaut wurde. Es war und ist aber nicht die einzige Aufgabe geblieben.

Damit ist aber auch verständlich, wofür der Begriff *Transfer* steht: Der Zugriff und die Übertragung einzelner Dokumente, z.B. HTML Dokumente, die in sich ja Verweise (Links) auf andere Dokumente mittels HTTP Zugriff enthalten können.

Was macht es dann zu einem Protokoll?

Das Zusammenspiel zwischen dem Browser und dem Server folgt dem allereinfachsten Muster bidirektionaler Kommunikation:

Frage / Antwort, das heißt im Fachjargon Request / Response

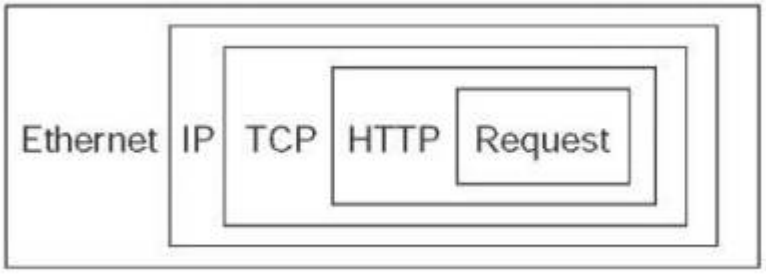


1. Der Browser (ein TCP-Client) stellt dem Server eine wohldefinierte Anfrage.
2. Der Server antwortet auf demselben TCP-Kommuniationskanal exakt auf diese Anfrage mit einem Status-Code (OK oder irgend ein Fehler) und in einem Rutsch mit dem angefragten Inhalt.
3. Danach wird der Kommunikationskanal wieder abgebaut (so war das zumindest ganz zu Beginn, mittlerweile wartet der Server auf der einmal aufgebauten TCP-Verbindung, ob der Browser nicht direkt mehrere Requests loswerden möchte, das ist effizienter)

> **Merke:** Die ganze Logik steckt eigentlich im Browser, der Server reagiert nur und gibt stumpf zu jeder einzelnen Anfrage das eine passende Dokument zurück (oder reagiert mit einer Fehlermeldung).

Ganz so einfach ist das heutzutage alles auch nicht mehr. Aber ein minimalistischer Webserver könnte genau so funktionieren. So ein Server ist beispielsweise ein sogenannter „statischer“ Webserver, der Anfragen auf fest vorgegebene Inhalte oder an fixen Orten abgelegte Dokumente

abbildet und mit diesen Inhalten antwortet. Ein solches Verhältnis zwischen Client und Server nennt man auch *zustandslos*, weil der *Response* völlig unabhängig von der Vorgeschichte der Kommunikation ausschließlich vom Inhalt des *Requests* abhängt: Gleiche Frage, gleiche Antwort!

> **Erinnerung:** Das Ganze hängt wie folgt mit dem eben beschriebnene TCP/IP zusammen. *http* ist ein Protokoll, das sich auf die unteren Protokollschichten (TCP und IP darunter) stützt. Analysiert man jedes Byte der Daten, die zwischen Browser und Server gesendet werden, dann wird man die „ineinander geschachtelten“ Protokollebenen wiedererkennen. Oder anders dargestellt: Jede Protokollebene verpackt die von den darüber liegenden Ebenen durchgereichten Daten in einen eigenen Datensatz und beim Empfang wird beim Durchlaufen der Protokollschichten wieder Schritt für Schritt „entpackt“:
>
> 
>

Aber wie sieht ein wohldefinierter Request aus?

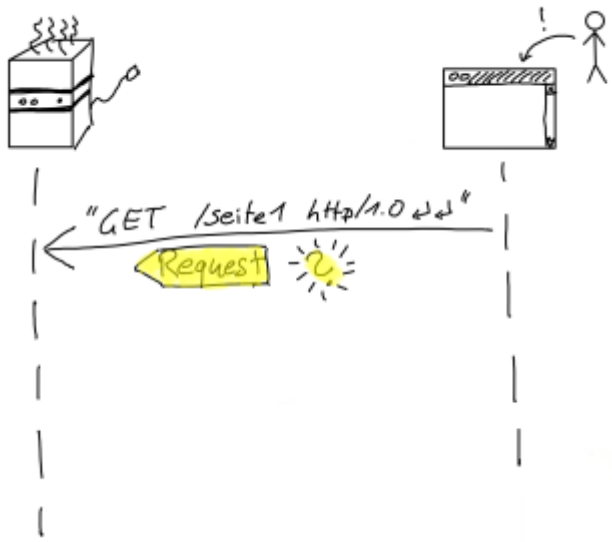
HTTP nutzt eine besonders einfache Art und Weise: Menschenlesbarer Text!

Das hat den Vorteil, dass es sich leicht realisieren lässt und weitgehend unabhängig von der zugrundeliegenden Technik ist.

Aber man muss sich an eine strikte Konvention halten, damit der Text der Anfrage/des Requests auch von jeder Maschine einfach interpretiert werden kann und nicht kompliziert analysiert werden muss. Folgende Regeln muss der Browser beim Zusammenbau des Request einhalten:

- Der Text beginnt mit der ersten Zeile und dem ersten Zeichen, Leerzeichen sind Trenner zwischen den einzelnen Bestandteilen.
- Das erste Wort ist ein Kommando, das beiden Seiten bekannt sein muss und im jeweils aktuellen RFC definiert wird: GET, POST, HEAD sind das bei Version 1.0.
- Danach folgt als zweites Wort die sogenannte *Ressource*, also das Dokument, dass auf diesem Server angefragt wird.
- Zum Schluss wird die HTTP-Version genannt: `HTTP/1.0` z.B.

Das ist der Mindestsatz an Informationen. Die Anfrage wird mit einer leeren Zeile beendet.



Je nach HTTP Version können zwischen der ersten Zeile und der Leerzeile noch weitere sogenannte HTTP-Header-Zeilen eingebaut sein. Die sind optional, erleichtern dem Server aber eventuell etwas die Arbeit. Bei HTTP/1.0 wären das u.a.:

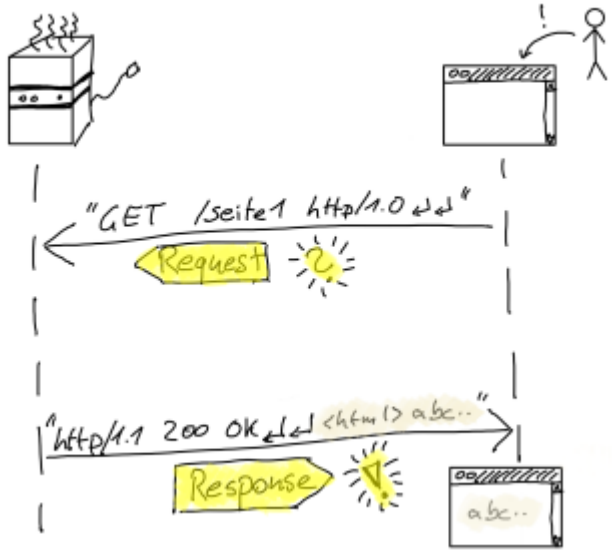
- `User-Agent`: Eine Bezeichnung für den Browser und die Version (z.B. `User-Agent: Mozilla/5.0`)
- `Referer`: Die URL der Webseite, die diesen Request ausgelöst hat

Ab HTTP 1.1 folgt nach der Zeile mit dem Kommando (also z.B. „GET /index.html HTTP/1.1“) immer eine Zeile, aus der der Name des Servers, so wie der Browser ihn nennt, ablesbar ist:

```
Host: www.qgelm.de
```

zum Beispiel. Es ist durchaus üblich, dass ein Server mit einer IP-Adresse verschiedene Namen haben kann und je nach Namen verschieden reagiert. Zum Zusammenhang des *Namens* mit der *IP-Adresse* gibt es hier einen kleinen Exkurs.

Der Server schaut dann, wie er den Inhalt der angeforderten Ressource bekommt und packt das wie folgt in seinen *Response*:



- Er nennt das verwendete Protokoll mit Version also z.B. `http/1.1`).

- Wenn alles glatt lief folgt dann der Statuscode mit Zahl und Kurzbeschreibung `200 OK`.
- Bis hier war alles in einer Zeile. Die nächsten Zeilen können dann weitere sogenannte *HTTP-Header* enthalten, sind aber optional.
- Die eigentlich übermittelte Ressource folgt dann nach einer Leerzeile und wird einfach Byte für Byte angehängt.

Der RFC für Version 1.0 definiert den übermittelten Statuscode übrigens so:

```
Status-Code    = "200"   ; OK
               | "201"   ; Created
               | "202"   ; Accepted
               | "204"   ; No Content
               | "301"   ; Moved Permanently
               | "302"   ; Moved Temporarily
               | "304"   ; Not Modified
               | "400"   ; Bad Request
               | "401"   ; Unauthorized
               | "403"   ; Forbidden
               | "404"   ; Not Found
               | "500"   ; Internal Server Error
               | "501"   ; Not Implemented
               | "502"   ; Bad Gateway
               | "503"   ; Service Unavailable
```

Wenn man nicht (unbemerkt) meistens den Status `200 OK` auslöst, dann ist wohl der am meisten beachtete Status `404 Not Found`. Das passiert immer, wenn die angeforderte Ressource für den Server nicht auffindbar ist, also z.B. der Link auf einer Webseite veraltet ist oder ein Vertipper passiert ist.

Kommen wir nach der Vorrede zu einem einfachen Beispiel eines statischen HTTP-Servers.