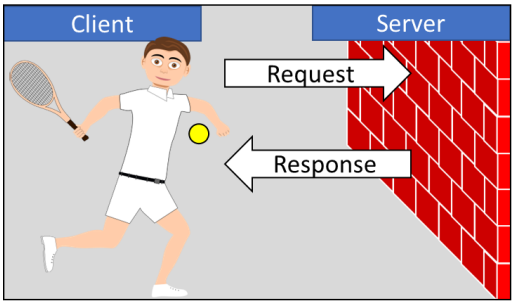
| | http - Protokoll | | AnPr | V 1.0 |
|---|---|---|---|---|
| | Name | Klasse | Datum | |

# 1 Grundsätzliches

Das „**H**yper**t**ext **T**ransport **P**rotocol“ wurde ursprünglich dazu geschaffen, Hypertext-Dokumente (also HTML Dateien) zwischen Computern auszutauschen. Heute ist es der Standard für die Kommunikation zwischen Webservern und Browsern. Das Protokoll wird entweder unverschlüsselt „http“ oder verschlüsselt „https“ genutzt. Eine Kommunikation wird immer vom Client aus gestartet (Request) und der Server antwortet darauf (Response). Dies bedeutet, dass ein Server nicht aktiv Daten zum Client senden kann[1]. dass es immer eine Request/Response Kommunikation ist – der Client schickt einen Request und der Server antwortet mit einem Response. Insofern unterscheiden wir auch die Request- und die Response Nachricht.




Der **Request** besteht wiederum aus drei Teilen:

1. Request Line – diese beinhaltet die Methode (also GET, PUT…) und die URL, gefolgt von weiteren allgemeinen Infos
2. Die Header Parameter dienen zur Klärung, was der Server mit den gesendeten Informationen (also über die URL und wenn vorhanden über den Body) machen kann und soll.
3. Den Body, wobei dieser optional ist. Der Body muss durch eine Leerzeile vom Header getrennt sein. Hier befinden sich zusätzliche Informationen, welche an den Server geschickt werden – meist bei PUT oder POST Anfragen

Hier eine Beispielanfrage:

```
POST  /   http/1.1
Host:  localhost:8080
User-Agent:  Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:103.0) Gecko/20100101 Firefox/103.0
Accept:  text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8,
Accept-Language:  de,en-US;q=0.7,en;q=0.3
Accept-Encoding:  gzip, deflate, br
Content-Type:  application/x-www-form-urlencoded
Content-Length:  26
Origin:  http://localhost:8080
Connection:  keep-alive
Referer:  http://localhost:8080/hello.html
Upgrade-Insecure-Requests:  1
Sec-Fetch-Dest:  document
Sec-Fetch-Mode:  navigate
Sec-Fetch-Site:  same-origin
Sec-Fetch-User:  ?1

fname=Max&lname=Mustermann
```

Da die Body Daten beliebig formatiert sein können, gibt uns der Header über „Content-Type“ einen Hinweis, wie wir die Daten interpretieren können. In unserem Fall liegen die Daten URL-Encoded vor – bspw. werden hier Name/Value Paare über „&“ getrennt.

Der **Response** besteht aus dem

1. Header, wo Kommunikationsinfos und Authentifizierungsdaten liegen …und dem
2. Body. Hier befinden sich die eigentlichen Daten, welcher der Server an den Client schickt. Beim Aufruf einer normalen Webadresse finden wir hier die HTML Informationen.

[1] Webapplikationen, welche eine bidirektionale Kommunikation benötigen, müssen auf ein anderes Protokoll ausweichen, wie Websockets

# 2 Die http Methoden

Obwohl bei Browsern fast nur GET und POST genutzt wird, bietet das http-Protokoll eine Vielzahl an möglichen Methoden an. Hier eine Auswahl der wichtigsten Methoden:

| **Befehl:** | **Bedeutung:** | **Body?** |
|---|---|---|
| GET | Datenanfrage an den Server. | Leer |
| POST | Senden von neuen Daten an den Server. Dies können auch ganze Files sein. | Daten |
| PUT | Senden von veränderten Datensätzen an den Server. Nur sinnvoll, wenn serverseitig die Daten einen definierten Status haben. | Daten |
| DELETE | Auftrag an den Server, dort Daten zu löschen. | ID |
| PATCH | Senden von veränderten (Teil-) Datensätzen n den Server. Im Gegensatz zu PUT wird also bspw. nicht ein kompletter Userdatensatz, sondern nur die ID und die zu ändernden Daten gesendet. | Daten +ID |
| HEAD | Bewirkt serverseitig das gleiche wie GET, jedoch werden bei der Repsonsenachricht keine Daten in den Body gelegt. | Leer |
| OPTIONS | Anfrage an den Server, welche Methoden er unterstützt | Leer |
| TRACE | Verfolgen der Route zum angefragten Server | Leer |

# 3 Response status Codes

Jeder Server liefert an den Client einen Code, über den er die serverseitige Verarbeitungsstatus mitteilt. Um die Interpretation zu vereinfachen, wurden sie in fünf Klassen eingeteilt:

| **Zahlenbereich:** | **Bedeutung:** |
|---|---|
| 100-199 | Informationen, die während der Verarbeitung zurückgegeben werden. Sie haben nur informativen Charakter. |
| 200-299 | Erfolgreiche Verarbeitung. 200 sollte der Normalfall sein. |
| 300-399 | Umgeleitete Anfrage. Meist müssen clientseitig weitere Aktivitäten erfolgen. |
| 400-499 | Clientseitige Fehler. Der Server geht von einer falschen Anfrage des Clients aus.<br>- 400: Die Anfrage folgt nicht dem geforderten Syntax<br>- 401: Es fehlt die Authentifizierung<br>- 403: Der Client hat nicht die Berechtigung, die Anfrage zu stellen<br>- 404: Die angefragte Ressource existiert nicht auf dem Server |
| 500-599 | Serverseitige Fehler. Meist ist es 500 (internal server error), was als allgemeiner Fehler zu interpretieren ist. |

Wer mehr Details benötigt, wird im Internet fündig. Neben Wikipedia hat Mozilla deine recht gute Aufstellung der möglichen Codes: https://developer.mozilla.org/de/docs/Web/HTTP/Status

# 4 Die URL

Initiale Anfragen an einen Server werden über die Eingabe der URL (Uniform Resource Locator) in der Adresszeile des Browsers durchgeführt. Da wir hier eine Ressource auf dem Server anfragen (und auch keinerlei Body-Informationen eingeben können), handelt es sich hier um eine GET Methode. URLs dürfen nur bestimmte Zeichen aufweisen – sie ist „URL-Encoded“. Infos hierzu finden sich bspw. hier: https://www.w3schools.com/tags/ref_urlencode.ASP . Der Aufbau einer URL ist wie folgt:

http://localhost:8081/MyAPI/myRessource?param1=wert1&param2=wert2

| | |
|---|---|
| http | Protokoll – für uns entweder http oder https |
| Localhost | Der Host – als IP Adresse, „localhost“ oder Webadresse |
| 8081 | Der Port, kann bei „80“ weggelassen werden |
| /MyAPI/myRessource | Der Pfad zur Ressource (je nach Ressourcenart mit oder ohne File Extension) |
| param1=wert1&param2=wert2 | Querystring für GET Parameter |

Vor allem die GET Parameter sind wichtig zu verstehen. Wir senden hiermit zwar keine „Daten“ an den Server, sehr wohl aber Informationen zur genaueren Spezifikation der Anfrage. Es sollten auf keinen Fall vertrauliche Informationen in den GET Parametern hinterlegt werden, da sie über die Browserhistorie klartextlich gespeichert werden!