

# Web Application Development

Client-Server, Web Apps, URI, HTTP

Martina Freundorfer

## Letztes Mal

- Scheinkriterien
- Einführung
- Frameworks
- Ihre Aufgaben bis heute:
  - 2-Teams organisieren
  - Github-Account besorgen
  - Privates Repo in github anlegen und martl einladen

# Semesterplan

#	K W	Datum	Vorlesung	Das Labor nach der Vorlesung findet für beide Gruppen wöchentlich statt.
1	14	3.4.	Einführung, Scheinkriterien	npm im Selbststudium, Anlegen eines Webprojektes, Installation von Live Server
2	16	17.4.	Client-Server, Web Apps, URI, HTTP	Aufgabe von Beleg 1: AdViz (nur HTML und CSS)
3	17	24.4.	HTML, CSS	Prototyp AdViz
4	20	15.5.	CSS, JavaScript allgemein	Abgabe: Prototyp AdViz
5	22	29.5.	JavaScript: DOM, JSON, AJAX	Aufgabe von Beleg 2: Adviz mit JS
6	23	5.6.	React Framework	AdViz mit JS
7	24	12.6.	React Framework	AdViz mit JS
8	25	19.6.	React Framework	Abgabe: AdViz mit JS
9	26	26.6.	React Framework / NodeJS	Aufgabe von Beleg 3: AdViz mit Backend
10	27	3.7.	NodeJS	AdViz mit Backend
11				AdViz mit Backend

## Heute

- Internet
- Client-Server-Modell
- Was ist überhaupt eine Webapplikation und was bedeutet “full-stack”?
- URI
- HTTP
- Lernziele:
  - Sie verstehen wie eine Webapplikation aufgebaut ist, und was die Buzz-Wörter “full-stack development” bedeuten.
  - Sie wissen was ein URI ist.
  - Sie kennen die Grundlagen des HTTP.

# Wie alt ist das Internet?

- SurveyMonkey-Umfrage: “Alter des Internets”
- <https://www.surveymonkey.de/r/FRCCNDP>

# Was ist das Internet?

- Internet, eigentlich “internetwork”, ist ein weltweiter Verbund von Rechner-netzwerken
- Da Rechnernetzwerke weltweit miteinander verbunden sind, können sich einzelne Rechner mit anderen Rechnern verbinden, und unter anderem Internetdienste wie
  - das World Wide Web
  - E-Mail
  - Telnet
  - FTPnutzen
- Für den Datenaustausch zwischen den Rechnern werden Internetprotokolle verwendet.
- Übertragung von Daten im Internet unabhängig von ihrem Inhalt, dem Absender und dem Empfänger wird als Netzneutralität bezeichnet.

# Was ist das Internet?

- 1969: Das Internet ging aus Arpanet hervor, welches im Jahr 1969 entstand.
- Arpanet war ein Projekt der Advanced Research Project Agency (ARPA), ARPA gehörte dem Department of Defense (DoD) an
- Arpanet sollte Universitäten und Forschungsinstitute vernetzen und die knappen Kapazitäten der teuren Großrechner besser nutzen
- Kommunikationsprotokolle waren ungeeignet für heterogene Umgebungen
- 1981: Entwicklung der TCP/IP-Netzwerkprotokolle in den 1980er
- 1984: Entwicklung des Domain Name System (DNS) ermöglichte die Adressierbarkeit von Rechnern mit Namen, die **man (Mensch)** sich merken konnte, z.B.: cern.ch
- 12.03.1989: Tim Berners-Lee (CERN) stellt die Idee des World Wide Web vor, ursprüngliches Ziel: Forschungsergebnisse mit Kollegen einfacher austauschen

# World Wide Web

- Tim Berners-Lee: “Das World Wide Web ist eine großräumige Hypermedia-Initiative zur Informationsbeschaffung mit dem Ziel, den allgemeinen Zugang zu einer großen Sammlung von Dokumenten zu erlauben.”
- Wikipedia: Das **World Wide Web** ist ein über das Internet abrufbares System von elektronischen Hypertext-Dokumenten, den Webseiten. Sie sind durch Hyperlinks untereinander verknüpft und werden im Internet über die Protokolle HTTP oder HTTPS übertragen. Die Webseiten enthalten meist Texte, oft mit Bildern und grafischen Elementen illustriert. Häufig sind auch Videos, Tondokumente und Musikstücke eingebettet.
- Umgangssprachlich wird WWW mit dem Internet gleichgesetzt, aber es ist jünger und eine von mehreren Nutzungen des Internets
- 1991: erste Webseite - <http://info.cern.ch/hypertext/WWW/TheProject.html>
- 1993: erster grafikfähige Browser “Mosaic”, der die Darstellung von Inhalten des WWW ermöglichte, kostenlos



# World Wide Web

- **Basiert auf drei Kernstandards:**

- **HTTP** = Hypertext Transfer Protocol als Protokoll, mit dem der Browser Informationen vom Webserver anfordern kann
- **HTML** als Auszeichnungssprache: Gliederung der Information, Verknüpfung von Dokumenten durch Hyperlinks
- **URLs** als eindeutige Bezeichnung einer Resource, die in Hyperlinks verwendet wird.

- Später kommen dazu:

- Cascading Style Sheets (CSS) legen das Aussehen der Elemente einer Webseite fest
- HTTPS = Hypertext Transfer Protocol Secure, eine Weiterentwicklung von HTTP, dient dem verschlüsselten Datentransfer
- Document Object Model (DOM) als Programmierschnittstelle für externe Programme oder Skriptsprachen von Webbrowsern.
- JavaScript - eine Skriptsprache mit Anweisungen für den Browser

# World Wide Web

- Erste persönliche Webseiten, mittlerweile über 19 Jahre alt:

<http://web.archive.org/web/20010929233155/http://martl.de.vu/>

# World Wide Web

- Dynamische Webseiten und Webanwendungen
- Können durch den Webserver oder/und im Browser erzeugt werden:
  - Server-seitig: Erzeugung des Inhalts der Webseite durch
    - CGI-Skripte (PHP, Perl)
    - kompilierte Anwendung (JSP, Servlets, .NET, Spring MVC)
  - Client-seitig: der Inhalt der Webseite wird mittels JavaScript erzeugt oder geändert
  - Gemischte Ausführung: AJAX (Asynchronous JavaScript and XML) – Browser sendet mittels JavaScript einen Request an den Webserver, Webserver schickt gewünschte Daten zurück, JavaScript erneuert Teile der HTML-Struktur mit diesen Daten
- Webapplikationen, an denen ich mitgewirkt habe: <http://firstsearch.oclc.org>, <https://www.mapquest.com>
- Mittlerweile sind Webanwendungen so interaktiv und allgegenwärtig, dass sie traditionelle Desktopapplikationen ersetzen: Mailclients, Google's Hangouts, Calendar, Drive, usw.. Solche Webapps werden auch als Rich Internet Applications = RIAs bezeichnet

# Wie sieht es mit Ihrer Erfahrung in der Webentwicklung aus?

- SurveyMonkey-Umfrage “Web Tech”:

<https://www.surveymonkey.de/r/FX7Y8KZ>

# Frameworks

- Welche Frameworks kennen Sie?
- Mit welchen Frameworks haben Sie schon gearbeitet?

# Frameworks

- Was genau ist ein Framework?

# Frameworks

Ralph E. Johnson, Brian Foote (1988):

- Ein Framework ist eine semi-vollständige Applikation.
- Es stellt für Applikationen eine wiederverwendbare, gemeinsame Struktur zur Verfügung.
- Die Entwickler bauen das Framework in ihre eigene Applikation ein, und erweitern es derart, dass es ihren spezifischen Anforderungen entspricht.
- Frameworks unterscheiden sich von Toolkits (oder Bibliotheken) dahingehend, dass sie eine zusammenhängende Struktur zur Verfügung stellen, anstatt einer einfachen Menge von Hilfsklassen.

# Frameworks

- Rahmen, innerhalb dessen der SW-Engineer eine Anwendung erstellt
- Struktur der individuellen Anwendung wird durch Framework beeinflusst
- Entwickler implementiert bzw. erweitert Schnittstellen oder Klassen des Frameworks, und registriert diese
- Framework steuert und ruft diese Implementierung auf -> Hollywood Principle: "Don't call us, we'll call you."
- Wiederverwendbare, gemeinsame Struktur
- Meist für ein bestimmtes Anwendungsgebiet
- Sind nicht nur Bibliotheken



# Frameworks

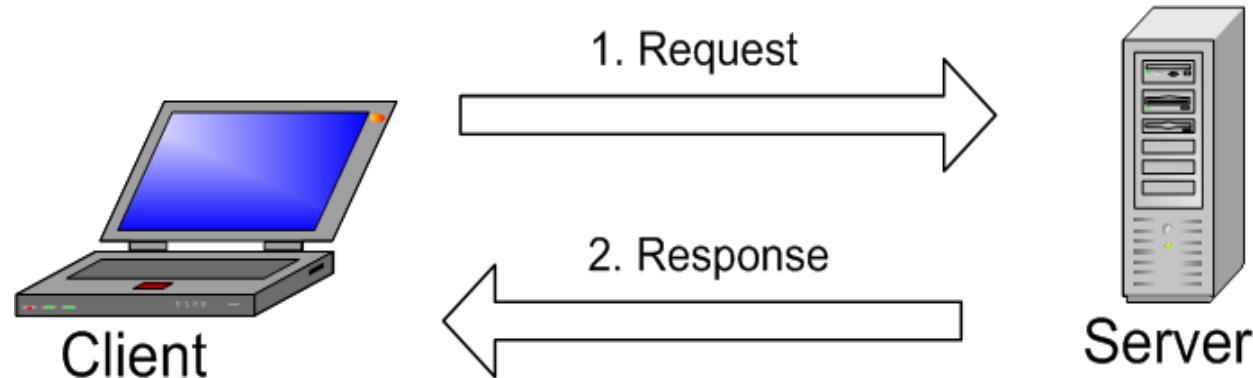
- Web Frameworks: **React Framework**, Spring MVC, Java Server Faces, etc.
- Testing Frameworks: JUnit, TestNG, Selenium
- Logging Frameworks: Apache Log4j, LogBack
- JSON Processing: Jackson (JSON = JavaScript Object Notation ist ein Datenformat)

## Client-Server-Modell

- Oder: Client-Server-Konzept, -Architektur, -System, -Prinzip
- Client-Server-Modell ist das Standardkonzept für die Verteilung von Aufgaben innerhalb eines Netzwerks
- Aufgaben werden von Programmen erledigt
- Bei diesen Programmen unterscheidet man zwischen “Clients” und “Servers”
- Servers bieten einen Service oder Dienst an
- Client kann einen Service/Dienst vom Server anfordern
- Server beantwortet die Anforderung
- **ein** Server gleichzeitig **für mehrere Clients** zuständig
- Server befindet sich meist auf einem anderen Rechner im Netzwerk, kann aber auch auf demselben Rechner wie der Client laufen

## Client-Server-Modell

- **Server** (*deutsch*: Bediener, Anbieter, Dienstleister, Bereitsteller) ist ein Programm (Prozess), das mit einem anderen Programm (Prozess), dem Client, kommuniziert, um ihm Zugang zu einem Dienst zu verschaffen. Bei dem Begriff "Server" handelt es sich um eine Rolle, nicht um einen Computer an sich.
- **Client** (*deutsch*: Kunde, Dienstinutzer) kann einen Service (Dienst) bei dem Server anfordern, welcher diesen Service bereitstellt.
- **Service** (*deutsch*: **Dienst**) Vereinbarung einer festgelegten Aufgabe, die der Server anbietet und der Client nutzen kann
- **Request** (*deutsch*: Anforderung, Anfrage) Anforderung eines Clients an den Server, dessen Dienst er benötigt
- **Response** (*deutsch*: Antwort) Antwort eines Servers auf eine Anforderung eines Clients



## Client-Server-Modell

- Die Kommunikation zwischen Client und Server ist abhängig vom Dienst
- Dienst bestimmt, welche Daten zwischen beiden ausgetauscht werden
- Regeln der Kommunikation für einen Dienst (Format, Aufruf des Servers, Bedeutung der zwischen Server und Client ausgetauschten Daten), werden durch ein für den jeweiligen Dienst spezifisches Protokoll festgelegt
- Der Server ist in Bereitschaft, um jederzeit auf die Kontaktaufnahme eines Clients reagieren zu können.
- Client, der *aktiv* einen Dienst anfordert
- Server ist *passiv* und wartet auf Anforderungen
- Clients und Server können als Programme auf verschiedenen Rechnern oder auf demselben Rechner ablaufen.

## Client-Server-Modell: Beispiele

- FTP-Server/-Client:

- File Transfer Protocol (FTP) von **Abhay Bhushan**
- [RFC 114](#) on 16 April 1971
- Beispiele für Server: Pure-FTPd (Unix), FileZilla-Server
- Beispiele für Client: FileZilla, ftp-Befehl in Kommandozeile

- Email-Server/-Client:

- Anfang der 1970er
- [RFC 561](#) (1973) u.a. auch von **Abhay Bhushan**
- Simple Mail Transfer Protocol (SMTP) zum Senden und Weiterleiten von Emails
- Post Office Protocol (POP) zum Herunterladen von Emails
- Internet Message Access Protocol (IMAP) als Netzwerkdateisystem
- Beispiele für Server: MS Exchange, Mercury MTS
- Beispiele für Client: Outlook, Thunderbird, etc.

- Datenbank-Server/-Client:

- Meist proprietäres Kommunikationsprotokoll
- Beispiele für Server: MySQL, Postgres, Oracle DB Server
- Beispiele für Client: MySQLWorkbench, mysql - Befehl in Kommandozeile

A **Request for Comments (RFC)**, in the context of Internet governance, is a type of publication from the Internet Engineering Task Force (IETF) and the Internet Society (ISOC), the principal technical development and standards-setting bodies for the Internet.



## Client-Server-Modell: Beispiele

- Webserver/-Client (Webserver und Webbrowser):
  - Programme für das World Wide Web (seit 1989)
  - Protokoll: Hypertext Transfer Protocol (HTTP)
  - HTTP entwickelt von **Tim Berners-Lee** u.a., der auch einen Browser und Server entwickelte





- Beispiele für Webserver: Apache, nginx, Tomcat, jetty, django,
- Beispiele für Clients: Mosaic, Mozilla, Chrome, Opera, Safari, Internet Explorer, ...

## Was war WWW nochmal?

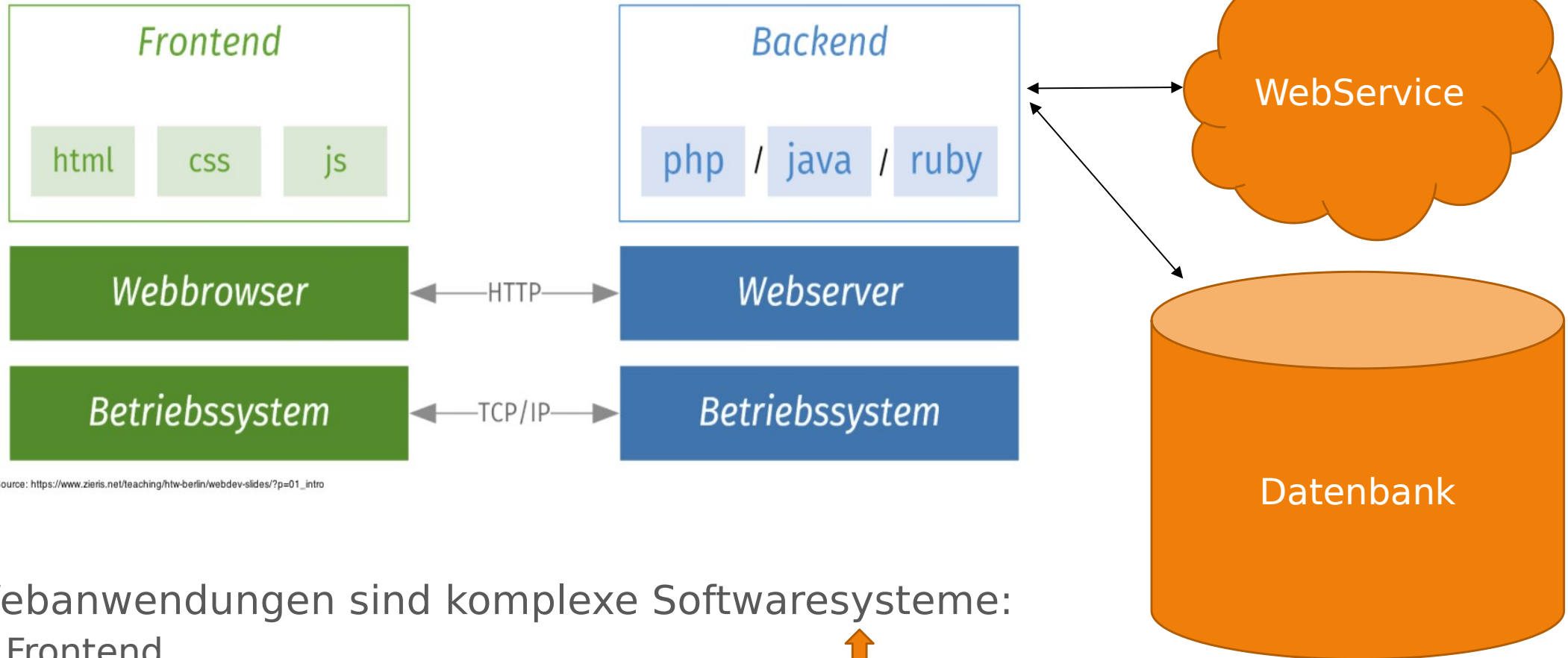
- **World Wide Web** (www):
  - 30 Jahre alt
  - über das Internet (50 Jahre) abrufbares System von elektronischen Hypertext-Dokumenten
  - die durch Hyperlinks untereinander verknüpft sind
  - und über das Protokoll HTTP (oder HTTPS) übertragen werden.
- Drei fundamentale Bestandteile:
  - HTML: Hypertext Markup Language
  - URI: Uniform Resource Identifier
  - HTTP: Hypertext Transfer Protocol
- **Browser** als übliches Betrachtungsprogramm
- Anfangs nur HTML-Seiten mit statischem Inhalt
- Später Entwicklung von Webanwendungen, die mittlerweile traditionelle Desktopapplikationen ersetzen.

# Webanwendungen

- “Webentwicklung” ist die Entwicklung von Webanwendungen:
  - ▢ Entwicklung von Anwendungen in Client-Server-Architektur
  - ▢ Client und Server kommunizieren über HTTP
- Webanwendungen bestehen aus einem **Frontend** (“Client”) und dem **Backend** (“Server”)
- **Frontend:**
  - ▢ Entwickelt von Developer
  - ▢ Besteht für uns aus HTML, CSS, JavaScript (“für uns“, d.h., keine Applets, Apache Flex (Flash), etc.)
  - ▢ Läuft im Browser
  -  ▢ Developer hat **keine** Kontrolle über die Client-Umgebung (welcher Browser, welche Version, usw.)
- **Backend:**
  - ▢ Entwickelt von Developer
  - ▢ Läuft im Webserver
  - ▢ Webserver stellt statische HTML-Seiten bereit
  - ▢ Webserver ruft Entwickler-Code für die dynamischen Inhalte von Seiten auf
  - ▢ Daten für diese Inhalte stammen aus Datenbanken oder von anderen Diensten (WebServices)
  -  ▢ Developer hat **volle** Kontrolle über die Server-Umgebung: welcher Server, Version, welche Programmiersprache setzt der Entwickler fest



# Webanwendungen



- Webanwendungen sind komplexe Softwaresysteme:

- Frontend
- Backend
- Database
- Integration with some other server-side system



## URIs

- **Definition** (Wikipedia):

Ein **Uniform Resource Identifier (URI)**, englisch für *einheitlicher Bezeichner für Ressourcen*) ist ein Identifikator und besteht aus einer Zeichenfolge, die zur Identifizierung einer abstrakten oder physischen Ressource dient. *URIs* werden zur Bezeichnung von Ressourcen im Internet und dort vor allem im WWW eingesetzt.

- Aktuelle Spezifikation (2016): [RFC 3986](#)
- Begriff ursprünglich von Tim Berners-Lee 1994 als „*Universal Resource Identifier*“ eingeführt, im RFC 1630
- URIs sind Bezeichner für Ressourcen, wie:
  - Webseiten
  - Dateien
  - Webservices
- URIs als Zeichenfolge in digitalen Dokumenten, oder als Verweis von einer Webseite auf eine andere (Hyperlink)

## URI - Aufbau

- URI besteht aus 5 Teilen:
  - scheme (Schema oder Protokoll),
  - authority (Anbieter oder Server),
  - path (Pfad),
  - query (Abfrage) und
  - fragment (Teil)wovon nur scheme und path in jedem URI vorhanden sein müssen.
- Beispiel: `http://example.com:8042/over/there?name=ferret#nose`
  - scheme: `http`
  - authority: `example.com:8042`
  - path: `over/there`
  - query: `name=ferret`
  - fragment: `nose`
- Generische Syntax:  
`URI = scheme ":" hier-part [ "?" query ] [ "#" fragment ]`
- Keine Leerzeichen in der Zeichenfolge!

## URI: scheme

- URI = **scheme** ":" authority [path] ["?" query] ["#" fragment]
- scheme = Schema
- Eine Auswahl an Schemata:

Schema	Beschreibung	Beispiel
data	Data-URL: direkt eingebettete Daten	<b>data:</b> text/plain;charset=iso-8859-7,%be%fa%be
<b>file</b>	Dateien im lokalen <b>Dateisystem</b>	<b>file:</b> ///C:/Users/Benutzer/Desktop/Uniform%20Resource%20Identifier.html
<b>ftp</b>	<b>File Transfer Protocol</b>	<b>ftp:</b> //ftp.is.co.za/rfc/rfc1808.txt
geo	<b>Geografische Koordinaten</b>	<b>geo:</b> 48.33,14.122;u=22.5
gopher	<b>Gopher</b>	<b>gopher:</b> //gopher.floodgap.com
<b>http</b>	<b>Hypertext Transfer Protocol</b>	<b>http:</b> //de.wikipedia.org/wiki
ldap	<b>Lightweight Directory Access Proto</b>	<b>ldap:</b> //[2001:db8::7]/c=GB?objectClass?one
mailto	E-Mail-Adresse	<b>mailto:</b> John.Doe@example.com
news	Newsgroup oder Newsartikel	<b>news:</b> comp.infosystems.www.servers.unix
pop	Mailboxzugriff über <b>POP3</b>	<b>pop:</b> //<user>;auth=<auth>@<host>:<port>
rsync	Synchronisation von Daten mit <b>rsync</b>	<b>rsync:</b> //[user@]host[:PORT]/Source
ssh	<b>Secure Shell</b>	<b>ssh:</b> //[<user>[:fingerprint=<host-key fingerprint>]@]<host>[:<port>]
tel	<b>Telefonnummer</b>	<b>tel:</b> +1-816-555-1212
telnet	<b>Telnet</b>	<b>telnet:</b> //192.0.2.16:80/
urn	Uniform Resource Names (URNs)	<b>urn:</b> oasis:names:specification:docbook:dtd:xml:4.1.2 (eine Resource mit eindeutigem und dauerhaft gültigem Namen)

• Unterschied URI und URL: <https://danielmiessler.com/study/url-uri/>

## URI: authority

- URI = scheme “:” **authority** [path] [“?” query][“#” fragment]
- authority = Anbieter/Server
- Beispiel: `https://www.htw-berlin.de:8443/index.html`
- Verschiedene Domain Name Services (DNS) konvertieren die symbolische Domain `www.htw-berlin.de` in eine IP-Adresse:
  1. Hierbei wird zunächst die Top Level Domain (TLD) **.de** berücksichtigt, um einen DNS-Server für diese TLD zu ermitteln.
  2. Dieser wird befragt, um den DNS-Server der Domain **htw-berlin.de** zu ermitteln.
  3. Letztlich liefert der HTW-DNS für die Subdomain **www** eine IP-Adresse zurück.

Gemeinsam mit der optionalen Portangabe wird der Dienst auf dem Zielrechner exakt bestimmt.

## Standardisierte Serverports

- Ports 0 bis 1.023 werden auch well-known oder standardisierte Ports genannt
- Diese Ports sind für "weitverbreitete" Dienste reserviert

Beispiele:

- HTTP-Server @ Port 80
  - Eigentlich: `http://www.google.de:80`
  - 80 ist der HTTP-Standard-Port: URI ohne `:80` , also `http://www.google.de` ist ausreichend
  - Tomcat's HTTP-Port ist 8080, kein Standard-Port, dann Port in URI angeben, z.B. `http://localhost:8080/`
- HTTPS-Server @ Port 443
  - `https://www.dkb.de/ueber_uns` (same as `https://www.dkb.de:443/ueber_uns`)
  - `https://www.htw-berlin.de:8443/index.html`, 8443 ist kein Standardport
- [https://de.wikipedia.org/wiki/Liste\\_der\\_standardisierten\\_Ports](https://de.wikipedia.org/wiki/Liste_der_standardisierten_Ports)

## URI: path

- URI = scheme “:” authority [path] [“?” query][“#” fragment]
- path = Pfad
- Beispiel:  
`http://www.htw-berlin.de/hochschule/aktuelles.html`
- Durch den Pfad wird das Dokument-Exemplar eindeutig bestimmt

## URI: query, fragment

- URI = scheme “:” authority [path] [“?” **query**][“#” **fragment**]
- query = Abfrage
- fragment = Fragment
- Beispiel für query:

<https://www.wetteronline.de/regenradar/brandenburg?gid=10382>

- Bei http- bzw. https-URIs werden diese als HTTP-Parameter bezeichnet:
  - gid – ist HTTP-Parametername
  - 10382 – ist der HTTP-Parameterwert
- Query kann mehrere Parameter enthalten, dann durch “&” trennen:

[https://www.wetteronline.de/regenradar/brandenburg?  
\*\*gid=10382&sid=5\*\*](https://www.wetteronline.de/regenradar/brandenburg?gid=10382&sid=5)



## URI: fragment

- URI = scheme “:” authority [path] [“?” query][“#” fragment]
- fragment = Fragment
- Beispiel für fragment:

<https://de.wikipedia.org/wiki/URL-Encoding#Prozentdarstellung>

- Referenzierung der gewünschten Position innerhalb des Dokuments
- Dokumentanker names “Prozentdarstellung”

## URI-Encoding

- Folgende Zeichen haben eine spezifische Bedeutungen im Dokumentenpfad und gelten als reserviert:  
: / ? # [ ] @ \$ & ' ( ) \* + , ; =
- Diese reservierten Zeichen dürfen also so nicht im Pfad vorkommen und müssen kodiert werden. D.h., sie werden umgewandelt ("verharmlost") und durch prozentkodierte Zeichen ersetzt
- Auswahl an ASCII-Zeichen in Prozentdarstellung:

␣	!	"	#	\$	%	&	'	(	)	...	=	>	?	@	[	\	]	{		}
%20	%21	%22	%23	%24	%25	%26	%27	%28	%29	...	%3D	%3E	%3F	%40	%5B	%5C	%5D	%7B	%7C	%7D

- <https://de.wikipedia.org/wiki/URL-Encoding>
- Beispiel:

`http://www.example.net/index.html?session=A54C6FE2=xyz`

“A54C6FE2=xyz” soll eine Session-Id sein

‘=’ muss prozent-kodiert werden: ‘=’ ist in ASCII-Code 061 □ 61 = 0x3D □ %3D

`http://www.example.net/index.html?session=A54C6FE2%3Dxyz`

## URI-scheme: HTTP

Beispiele:

- <http://firstsearch.oclc.org>
- <http://htw-berlin.de>

usw.

# HTTP

- [Wikipedia](#): as **Hypertext Transfer Protocol (HTTP)** ist ein zustandsloses Protokoll zur Übertragung von Daten auf der Anwendungsschicht über ein Rechnernetz. Es wird hauptsächlich eingesetzt, um Webseiten 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.
- HTTP beruht auf einer Client-Server-Kommunikation mit Request-Response-Mechanismus
- Synchrone Kommunikation: während der Abarbeitungszeit der Anfrage durch den Server wartet der Client auf die Antwort
- Jede HTTP-Nachricht besteht aus zwei Teilen:
  - ▢ Nachrichtenkopf (Message Headers oder auch HTTP-Headers)
  - ▢ Nachrichtenrumpf (Message Body, kurz: Body)
- HTTP-Header enthalten u.a. Informationen über den Nachrichtenrumpf, z.B., Inhaltstyp, Kodierung, damit der Body vom Empfänger richtig interpretiert werden kann.
- Der Body oder Nachrichtenrumpf enthält die Nutzdaten.

# HTTP - Aufbau der Nachrichten

Befehl:

```
> telnet checkip.dyndns.org 80
Trying 216.146.43.70...
Connected to checkip.dyndns.com.
Escape character is '^]'.
```

```
GET /index.html HTTP/1.1
Host: checkip.dyndns.org
```

Aufbau der Anfrage:  
**VERB URL HTTP/Version**  
**Headers**

**Body**

```
HTTP/1.1 200 OK
Content-Type: text/html
Server: DynDNS-CheckIP/1.0.1
Connection: close
Cache-Control: no-cache
Pragma: no-cache
Content-Length: 104
```

Aufbau der Antwort:  
**HTTP/Version Statuscode Statusmeldung**  
**Headers**

**Body**

```
<html><head><title>Current IP
Check</title></head><body>Curren
t IP Address: 95.91.247.96</body></
html>
```

# HTTP - Aufbau der Nachrichten

Es geht natürlich auch einfacher:

- **BROWSER (Chrome) öffnen und folgende URL eingeben:  
`http://localhost:8080/ersteswebproj/`**
- **Chrome's "Developer Tools" öffnen**
- **Gucken was im "Network" -Tab des Tools passiert**

# Auswahl von HTTP-Befehlen/Methoden/Verben

Verb	Bedeutung
<b>GET</b>	eine Ressource (z.B., eine Datei) vom Server anfordern: GET /webapp?parameter1=value1&parameter2=value2 "key=value"-Paare sind die HTTP-Parameter
<b>POST</b>	Daten zur weiteren Verarbeitung zum Server senden Daten befinden sich im Body des Requests
<b>PUT</b>	dient dazu, eine Ressource unter Angabe des Ziel-URLs auf einen Webserver hochzuladen, muss nicht durch den Server verarbeitet werden (Ressource befindet sich im Body des Requests)
<b>DELETE</b>	löscht die angegebene Ressource auf dem Server
HEAD	weist den Server an, die gleichen HTTP-Header wie bei GET, nicht jedoch den Nachrichtenrumpf zu senden
TRACE	liefert die Anfrage so zurück, wie der Server sie empfangen hat
OPTIONS	liefert eine Liste der vom Server unterstützten Methoden
PATCH	ändert ein bestehendes Dokument ohne dieses wie bei PUT vollständig zu ersetzen

# HTTP-Status Codes

- <https://wiki.selfhtml.org/wiki/HTTP/Statuscodes>
  - ▮ **1xx: Informativ (100 - 102):** Die Bearbeitung der Anfrage dauert noch an.
  - ▮ **2xx: Erfolg:** Die Anfrage war erfolgreich, die Antwort kann verwertet werden.
  - ▮ **3xx: Umleitung:** Um eine erfolgreiche Bearbeitung der Anfrage sicherzustellen, sind weitere Schritte seitens des Clients erforderlich.
  - ▮ **4xx: Client-Fehler:** Die Ursache des Scheiterns der Anfrage liegt (eher) im Verantwortungsbereich des Clients.
  - ▮ **5xx: Server-Fehler:** Nicht klar von den so genannten Client-Fehlern abzugrenzen. Die Ursache des Scheiterns der Anfrage liegt jedoch eher im Verantwortungsbereich des Servers.



# HTTP-Status Codes

- Die wichtigsten 2xx-Status Codes:

Code	Nachricht	Bedeutung
200	<i>OK</i>	Die Anfrage wurde erfolgreich bearbeitet und das Ergebnis der Anfrage wird in der Antwort übertragen.
201	<i>Created</i>	Die Anfrage wurde erfolgreich bearbeitet. Die angeforderte Ressource wurde vor dem Senden der Antwort erstellt. Das „Location“-Header-Feld enthält eventuell die Adresse der erstellten Ressource.
202	<i>Accepted</i>	Die Anfrage wurde akzeptiert, wird aber zu einem späteren Zeitpunkt ausgeführt. Das Gelingen der Anfrage kann nicht garantiert werden.
204	<i>No Content</i>	Die Anfrage wurde erfolgreich durchgeführt, die Antwort enthält jedoch bewusst keine Daten.

# HTTP-Status Codes

- Die wichtigsten 4xx-Status Codes:

Code	Nachricht	Bedeutung
400	<i>Bad Request</i>	Die Anfrage-Nachricht war fehlerhaft aufgebaut.
401	<i>Unauthorized</i>	Die Anfrage kann nicht ohne gültige Authentifizierung durchgeführt werden. Wie die Authentifizierung durchgeführt werden soll, wird im „WWW-Authenticate“-Header-Feld der Antwort übermittelt.
402	<i>Payment Required</i>	Übersetzt: Bezahlung benötigt. Dieser Status ist für <b>zukünftige</b> HTTP-Protokolle <b>reserviert</b> .
403	<i>Forbidden</i>	Die Anfrage wurde mangels Berechtigung des Clients nicht durchgeführt, bspw. weil der authentifizierte Benutzer nicht berechtigt ist, oder eine als HTTPS konfigurierte URL nur mit HTTP aufgerufen wurde.
404	<i>Not Found</i>	Die angeforderte Ressource wurde nicht gefunden. Dieser Statuscode kann ebenfalls verwendet werden, um eine Anfrage ohne näheren Grund abzuweisen. Links, welche auf solche Fehlerseiten verweisen, werden auch als <u>Tote Links</u> bezeichnet.
405	<i>Method Not Allowed</i>	Die Anfrage darf nur mit anderen HTTP-Methoden (zum Beispiel GET statt POST) gestellt werden. Gültige Methoden für die betreffende Ressource werden im „Allow“-Header-Feld der Antwort übermittelt.
406	<i>Not Acceptable</i>	Die angeforderte Ressource steht nicht in der gewünschten Form zur Verfügung. Gültige „Content-Type“-Werte können in der Antwort übermittelt werden.

# HTTP-Headers

- <https://wiki.selfhtml.org/wiki/HTTP/Header>

Name	Content-Type
HTTP-Version	1.1
Beschreibung	Der <a href="#">MIME-Typ</a> der Daten im Body.
Erlaubte Werte	Medien- ( <a href="#">MIME</a> -)Typ
Spezifikation	<u>RFC 2616 Kapitel 14.17</u>
Beispiel	Content-Type: text/html; charset=utf-8

# HTTP-Headers

- <https://wiki.selfhtml.org/wiki/HTTP/Header>

Name	<b>Accept</b>
HTTP-Version	1.1
Beschreibung	Welche Inhaltstypen der Client verarbeiten kann.
Erlaubte Werte	Medien- ( <a href="#">MIME</a> -)Typ ( <a href="#">q-Liste</a> )
Spezifikation	<u><a href="#">RFC 2616 Kapitel 14.2</a></u>
Beispiel	Accept:text/html,application/xhtml+xml,application/xml  Ist es dem Server nicht möglich, einen Inhaltstyp bereitzustellen, der vom Client akzeptiert wird, kann er entweder den HTTP-Statuscode <a href="#">406 Not acceptable</a> senden oder einen beliebigen Inhaltstyp zum Kodieren der angeforderten Informationen verwenden.
Beachten Sie	Fehlt das Accept-Feld, so bedeutet dies, dass der Client alle Inhaltstypen akzeptiert. Kann der Server in diesem Beispiel den Inhalt der angeforderten Ressource sowohl als HTML als auch als Bild im GIF-Format an den Client senden, führt der Accept-Header der Anfrage dazu, dass als Inhaltstyp der Antwort HTML gewählt wird.

# HTTP-Requests ausprobieren

- Im Browser können wir nur HTTP-GETs ausprobieren
- Browser öffnen und dazu Webdeveloper Console (Firefox, Chrome) zum Angucken, welche Anfragen tatsächlich an den Server geschickt werden, wie die Header, Bodies der Anfragen und Antworten aussehen
- HTTP-POST in Postman:
  - ▢ Postman installieren
  - ▢ POST-Request ausprobieren: "POST http://postman-echo.com/post"

Oder mit:

- cURL – Kommandozeilen-Tool zum Verschicken von Anfragen, sehr nützlich  
▢

# HTTP - Geschichtliches

- 1989: Roy Fielding, Tim Berners-Lee und andere am CERN beginnen das Hypertext Transfer Protocol zu entwickeln. Zusammen mit den Konzepten URI und HTML, werden die Grundlagen des World Wide Web geschaffen.
- 1991: HTTP/0.9
- Mai 1996: HTTP/1.0
  - Für jede Anfrage eine neue TCP-Verbindung, wird vom Server geschlossen. Für ein HTML-Dokument mit 5 eingebetteten Bilder, falls dieses in einem Browser richtig angezeigt werden soll, müssen 6 TCP-Verbindungen hergestellt werden.
  - GET HEAD POST
- 1999: HTTP/1.1
  - Mit HTTP-Pipelining mehrere Anfragen pro TCP-Verbindung. Für HTML-Dokument mit 5 Bildern kann ich für alle 6 Requests eine TCP-Verbindung nutzen.
  - Neue Methoden: PUT, DELETE, OPTIONS, etc
- Mai 2015: HTTP/2.0
  - die Beschleunigung durch Zusammenfassen mehrerer Anfragen
  - weitergehende Datenkompressionsmöglichkeiten auch von Headern
  - die binär kodierte Übertragung von Inhalten
  - Server-initiierte Datenübertragungen (push-Verfahren).

# The End

- Questions?
- References:
  - ▢ Dane Cameron: “A Software Engineer Learns HTML5, JavaScript & jQuery”, Cisdal Publishings, 2015
  - ▢ <https://webfoundation.org/2018/03/web-birthday-29/>
  - ▢ <https://www.reporter-ohne-grenzen.de/pressemitteilungen/meldung/uncensored-playlist-mit-pop-songs-die-zensur-umgehen/>
  - ▢ Franz Zieris “Webentwicklung WiSe 2017/18 HTW Berlin” - Folien : <https://www.zieris.net/teaching/htw-berlin/webdev/>
  - ▢ Max Beier/Thomas Ziemer “Webentwicklung WiSe 2017/18 HTW Berlin” - Folien: <https://beier.f4.htw-berlin.de>
  - ▢ HTTP-Reference: <https://wiki.selfhtml.org/wiki/HTTP>
  - ▢ Michael Inden: “Der Java-Profi: Persistenzlösungen und REST-Services”, dpunkt.verlag, 2016