FH | JOANNEUM
University of Applied Sciences

Systemnahe Programmierung

Grundlagen HTTP

Projekt

Webserver

Wir werden einen einfachen Webserver mit eingebauten „Spezialfunktionen“ entwickeln.

Der Webserver nimmt Anfragen – Requests aus dem Netzwerk entgegen, verarbeitet sie und sendet eine Antwort – Response zurück.

Im häufigsten Fall ist das eine HTML-Datei die zurückgesendet wird, es kann aber auch das Ergebnis einer Query usw. sein.

Die Funktionalität des Servers ist einfach gehalten:

- Abgespecktes HTTP 1.1 Protokoll
- Keine Authentifizierung
- Dateien senden
- Services ausführen

Das HTTP Protokoll

Basis für den Webserver ist das HTTP-Protokoll.

HTTP ist ein Client-Server Protokoll mit einem Server (Webserver) und einem User Agent (z.B.: Browser, curl …) als Kommunikationspartner.

User Agents senden **Requests** an den Server, dieser beantwortet sie mit einem **Response**.

Das HTTP Protokoll ist stateless, das heißt es werden zwischen den Requests keine Daten für einen Client im Server gespeichert (z.B.: Session).
Wir verwenden eine vereinfachte Version von HTTP 1.1.

Der Ablauf auf dem Server ist

- Request entgegennehmen
- Request bearbeiten
- Response senden
- Verbindung beenden

Siehe: https://de.wikipedia.org/wiki/Hypertext_Transfer_Protocol

Das HTTP Protokoll

HTTP Request

Ein HTTP Request ist im Wesentlichen eine Nachricht im Textformat, die über eine Netzwerkverbindung an einen Server gesendet wird.

Die wichtigsten Elemente eines Requests sind die Request-Methode, die Request-URI, die Request-Header und der Request-Body.

- Request-Methode: GET, POST, DELETE, PUT ….
- Request-URI: URI des Dienstes, der vom Webserver angefordert wird. Dies kann eine einfache Datei oder auch ein komplexes Service (z.B.: REST) sein. Die Request URI kann Parameter enthalten, die dem auszuführenden Service übergeben werden.
- Request-Header: enthalten Zusatzinformation/Metainformation für den Request.
- Request-Body: Enthält zusätzliche Daten für den Request (Parameter, JSON Daten ...)

Das HTTP Protokoll

HTTP Response

Ein HTTP Response ist die Antwort des Webservers auf einen Request. Sein Format ist auch textbasiert und ähnlich dem des Requests.

Ein Response enthält einen Statuscode über die Ausführung des Requests (erfolgreich, Fehler usw.), Response-Header und einen Response-Body.

Das Ergebnis eines Requests wird im Response-Body an den Aufrufer übermittelt.
Dies kann der Inhalt einer Datei (z.B.: HTML) sein, oder auch das Ergebnis eines REST-Aufrufes (z.B.: Json) oder auch ein beliebiges anders Resultat (pdf, zip …) sein.

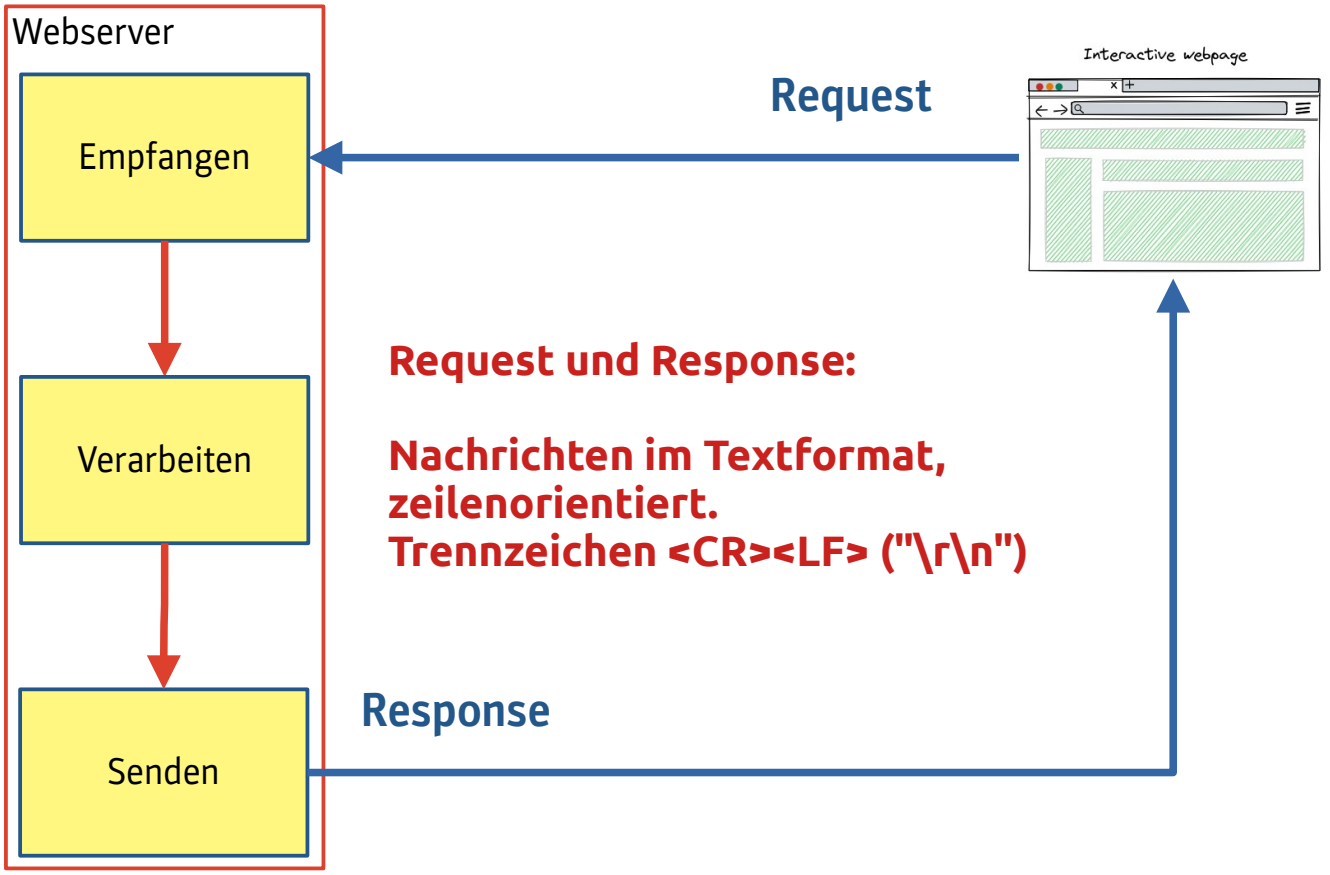
Response Statuscodes - Format: 3-stellige Zahl und ein Text (Reason)

- 1xx - Informational
- 2xx - Success
- 3xx - Redirection
- 4xx - Client Error
- 5xx - Server Error

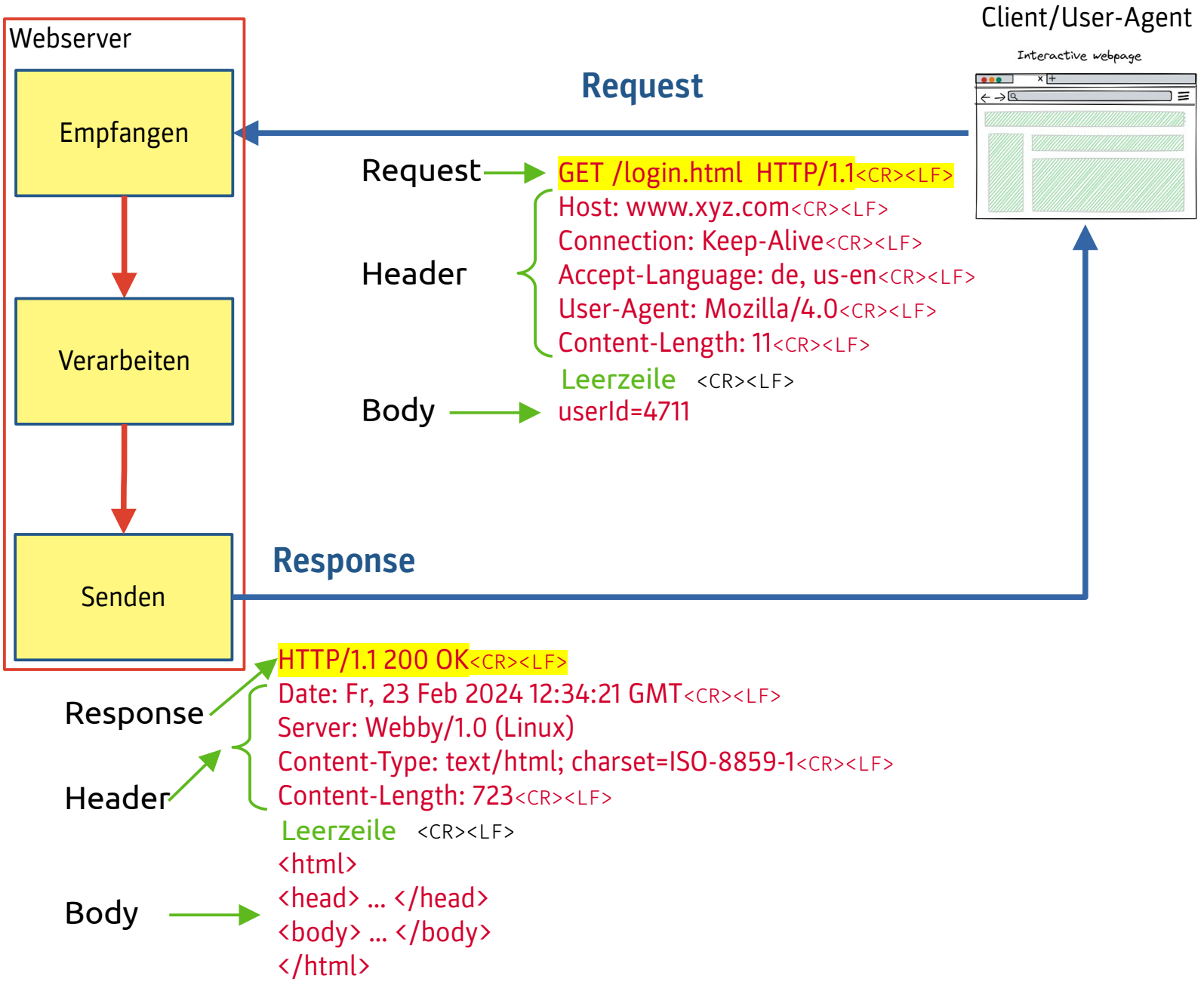
Häufige Werte:

- `200 OK`
- `400 Bad Request`
- `403 Forbidden`
- `404 Not Found`
- `500 Internal Server Error`

Das HTTP Protokoll



Das HTTP Protokoll



Das HTTP Protokoll

Request

Zeile	Inhalt	Format
1	Request-Line \r\n	Request-Method Request-URI HTTP-Version
2	Request-Header 1 \r\n	request-header-name ‘:‘ request-header-value
3	Request-Header 2 \r\n	
	...	
n-1	Blank Line \r\n	
n	Request-Body \r\n	optional – kann auch ein Binärformat sein

Request-Method: GET, POST, DELETE …
Request-URI: path?query_string#fragment_id

JEDE Zeile wird mit <CR><LF> ("\r\n") abgeschlossen!
Felder in der Request-Line werden mit Leerzeichen getrennt.
Mehrere Werte in einem Header werden durch ‘;‘getrennt.

Das HTTP Protokoll

Response

Zeile	Inhalt	Format
1	Response-Line \r\n	HTTP-Version Response-Statuscode [reason]
2	Response-Header 1 \r\n	response-header-name ‘:‘ response-header-value
3	Response-Header 2 \r\n	
	...	
n-1	Blank Line \r\n	
n	Response-Body \r\n	optional – kann auch ein Binärformat sein

Response-Statuscode: 200, 400, 500 ...
Reason: OK, Bad Request, Internal Server Error …
passend zu Response-Statuscode!

Siehe: https://de.wikipedia.org/wiki/HTTP-Statuscode

Das HTTP Protokoll

Beispiel

Request

```
GET /test/hallo.html HTTP/1.1\r\n
Host: localhost:17000\r\n
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:125.0)\r\n
Accept:text/html,application/xhtml+xml,*/*;q=0.8\r\n
\r\n
```

Response

```
HTTP/1.1 200 OK\r\n
Server: Webby/1.0\r\n
Date: Thu,11-Apr-24 11:07:32 +0200\r\n
Content-Length: 32\r\n
Content-Type: text/html\r\n
\r\n
<html><body>Hallo!</body></html>
```

Das HTTP Protokoll

Header

Header enthalten Metadaten für Requests und Responses.

Das allgemeine Format eines Headers ist:

`<Headername> : <Header Values><CR><LF>`

`<Header Values>` : mehrere Werte werden durch ‘ , ‘ getrennt (Liste)

`<Header Values>` können zusätzliche Parameter haben, diese werden durch ‘ ; ‘ getrennt.

Beispiele:

```
Content-Length: 723
Accept: text/html, application/xhtml+xml, application/xml
Content-Type: text/html; charset=ISO-8859-1
```

Das HTTP Protokoll

Request-Header

Folgende Request-Header (`<Headername>`) sind im Projekt zu implementieren:

- `Host` - Hostname des Clients
- `User-Agent` - Bezeichnung des Client-Programms (z.B.: `curl/7.81.0`)
- `Accept` - Welche Mime-Typen der Client akzeptiert (normalerweise `*/*`)
- `Content-Length` - Länge der Daten im Body-Abschnitt in Bytes (0 falls nicht vorhanden)
- `Content-Type` - Mime-Type des Inhalts im Body

Implementieren heißt: erkennen, den Wert auslesen, im Logging ausgeben und anwenden falls notwendig (Content-Length).
Beachten sie auch, dass der Client keine Header mitsenden muss!

Das HTTP Protokoll

Response-Header

Folgende Response-Header müssen einem Response mitgegeben werden (ab Stufe 3):

- `Server` - Bezeichnung des Servers mit Versionsnummer (Beispiel: `Webby/1.0`)
- `Date` - Datum und Zeit zu dem der Response gesendet wurde.
- `Content-Length` - Länge des gesendeten Body-Inhalts in Bytes.
- `Content-Type` - Mime-Type des gesendeten Inhalts

Mime-Types

Ein Mime-Type spezifiziert die in einem Body enthaltenen Daten (Request und Response).
Eine Mime-Type besteht aus einem Medientyp und einem Subtyp, getrennt durch ‘`/`‘.
Beispiele:

- `text/plain`
- `text/html`
- `image/png`
- `application/pdf`

Das HTTP Protokoll

URI/URL Handling

```
schema://[username:password@]domain[:port]/path[?query_string]
[#fragment_id]
```

- Schema: `http, https, ftp` ...
- Domain: Netzwerkadresse
- Port: optional, default `80`
- Path: Pfad zur angeforderten Resssource
- Query-String: `<name>=<wert>` getrennt durch ‚`&`‘
- Fragment-Id: Verweis innerhalb der Ressource (z.b.: Html Anchor)

Wie kommt das beim Server an?
Die Request-Line enthält diese Komponenten:

```
<Method> path[?query_string][#fragment_id] HTTP/<Version>
```

Beispiel:

```
https://www.google.com/search?client=firefox-b-lm&q=programmierung
```

Ergibt beim Google Server folgenden Request:

```
GET /search?client=firefox-b-lm&q=programmierung HTTP/2
```

Das HTTP Protokoll

URI Encoding

Die URI/URL darf nur darstellbare ASCII Zeichen enthalten, deshalb müssen Sonderzeichen, Umlaute usw. codiert werden.
URI Encoding wandelt reservierte, unsichere, und Nicht-ASCII Zeichen in eine HTTP konforme Darstellung um .
Dafür wird jedes Byte des Zeichencodes in hexadezimale Darstellung übertragen und ein `'%'` Zeichen vorangestellt.

Beispiele: `ä` wird zu `%C3%A4`
Leerzeichen wird zu `%20` oder `'+'`

Uncodiert: `index.htmäl “`
Wird zu: `index.htm%C3%A4l%20%22`

Unsichere Zeichen:
`„ < > # % { } \ | ^ [ ] ` Leerzeichen`

Reservierte Zeichen:
`! # $ % & ' ( ) * + , / : ; = ? @ [ ]`

Das HTTP Protokoll

URI Encoding

Der Webserver muss diese Codierung decodieren können, dafür ist im Template eine Funktion inkludiert (`url.c`). Decodieren sie die erhaltene URI mithilfe dieser Funktion:

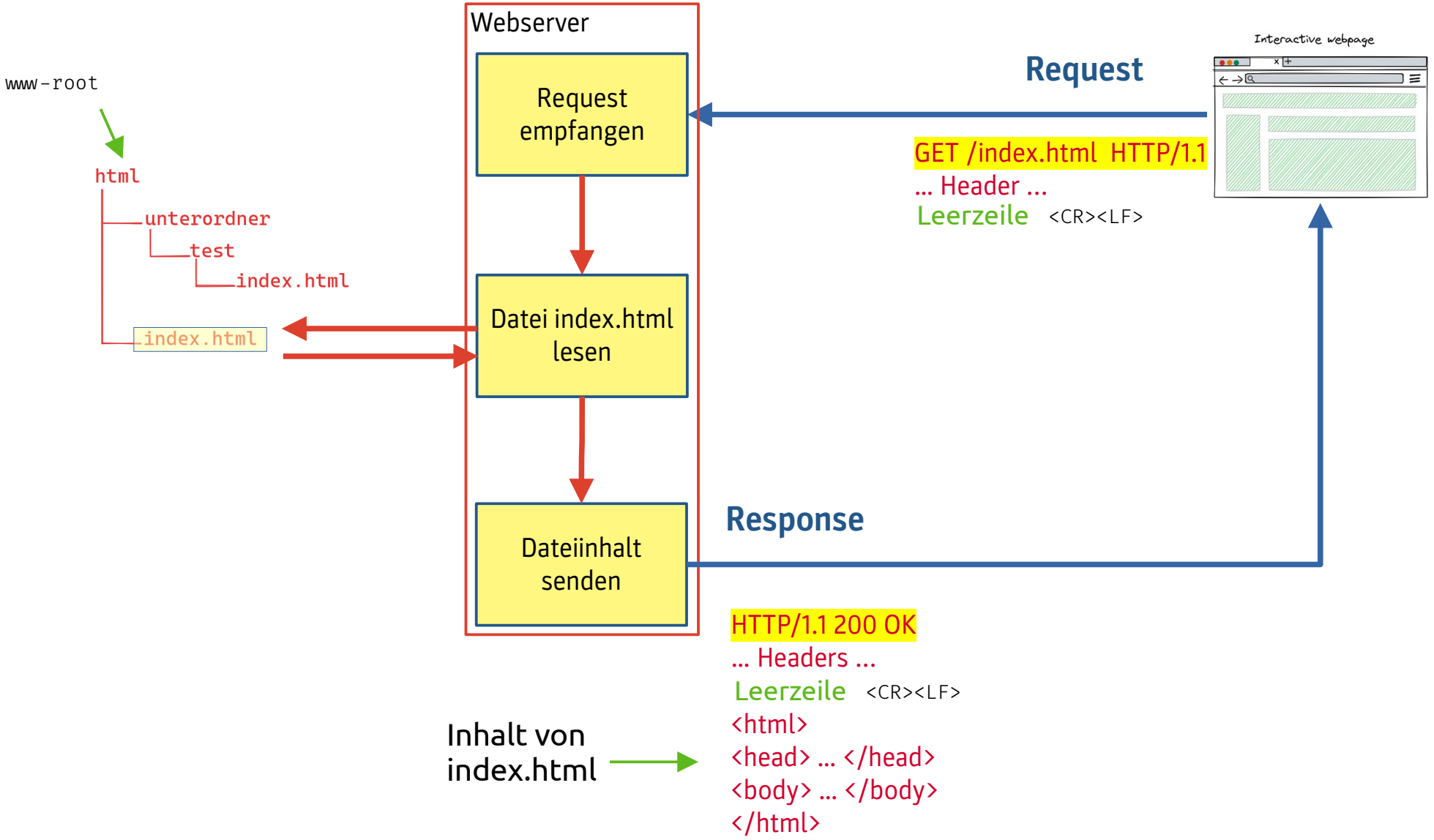
```
#include <url.h>

char *decodeUri(char *uri);
```

Gibt einen Zeiger zur dekodierten Version von `uri` zurück oder `NULL` im Fehlerfall. Wichtig: der durch die Funktion allozierte Speicher ist nach Verwendung freizugeben.

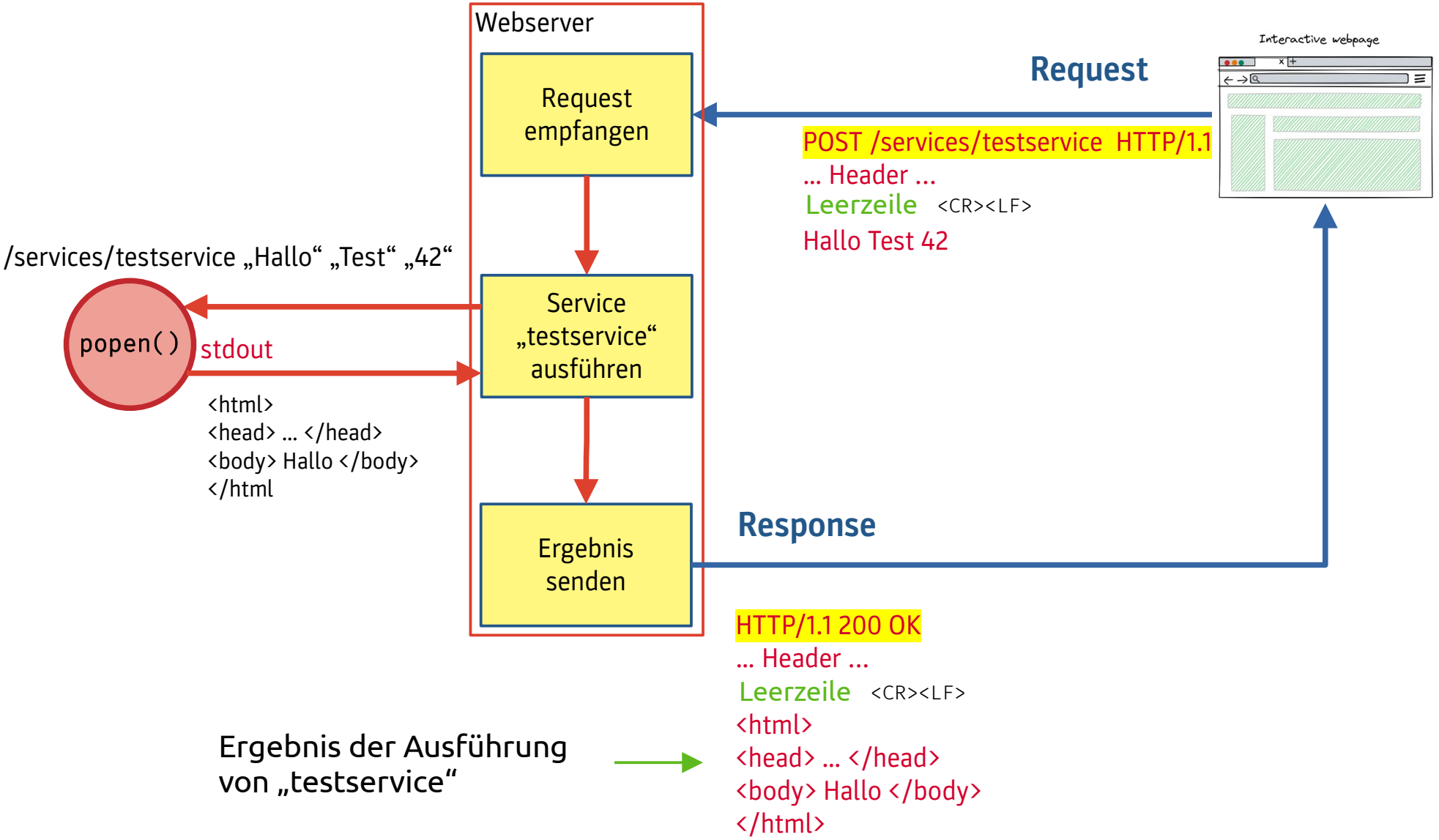
Das HTTP Protokoll

File Request



Das HTTP Protokoll

Service Request



Projekt Webserver

Anforderungen

Wir verwenden ein vereinfachtes HTTP Protokoll nach folgender Spezifikation:

Request

Methoden: `GET, POST`
Request-Header: `Content-Length, Host, User-Agent, Accept, Content-Type`
URI Format: `path[?parameter]`
- Path: durch `"/"` getrennte Verzeichnisse bzw. Dateinamen
- Parameter: `name=wert` getrennt durch `"&"`
- Beispiel:

```
/verzeichnis/datei.html?id=3&vorname=ida
```

Response

Response-Header: `Server, Date, Content-Length, Content-Type`
Response-Statuscodes: `200, 400, 403, 404, 500`