The HTTP Protocol

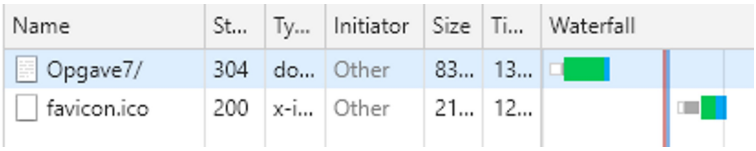
28. august 2019 10:50

1) Monitoring HTTP Headers 1

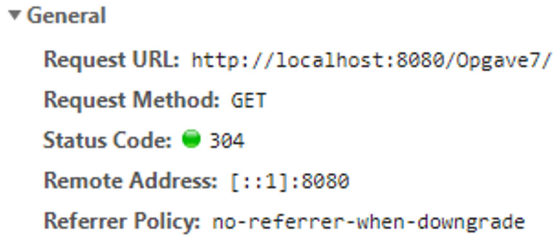
Create a new NetBeans Maven Web-project.
For this exercise, we will just use the default index.html generated by NetBeans.

Press the run button. When you see the file in the browser (Chrome), open the network tab in the developer window (Ctrl-shift-j) and press F5

Observe and explain each of the values monitored (use view source to see the plain messages).

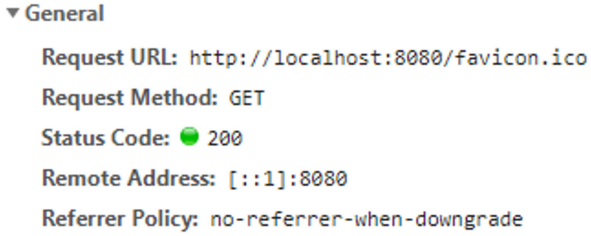


Der bliver GET'et to filer. Hvis man kigger på den første fil, **Opgave7/** kan man se hvilke headers der blev sendt med, og hvis man kigger på **General** får man følgende:



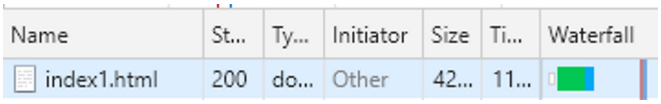
Man kan se request metoden var en GET, og at **Status Koden var 304**. At status koden er 304 i HTTP protocolen betyder at det var en redirect.

Den anden fil, favicon.ico er tomcat iconet, og har følgende header;



Her kan man igen se at request metoden var GET, men **Status Koden var 200** i dette tilfælde, hvilket blot betyder **ok**, altså at det var en succesfuld metode.

Go back to NetBeans and rename your file to index1.html
Go back to your browser and (while the developer window is open) change the url to point to the new file.
Observe the values
Press F5 and observe the values again.
Explain what you see.



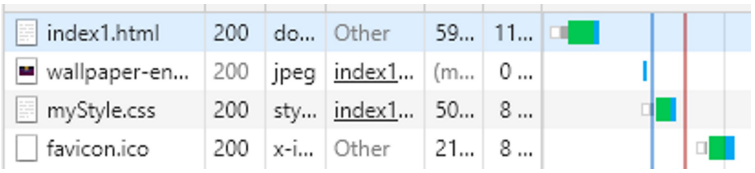
Man kan nu se at der kun bliver GET én fil, nemlig index1.html, som har status koden 200. Dette betyder at i stedet for at blive redirected direkte til index.html (som er den standard startside) går vi nu direkte ned i index1.html filen ved at tilføje noget til URL'en: http://localhost:8080/Opgave7/index1.html i stedet for http://localhost:8080/Opgave7/ .

2) Monitoring HTTP Headers 2

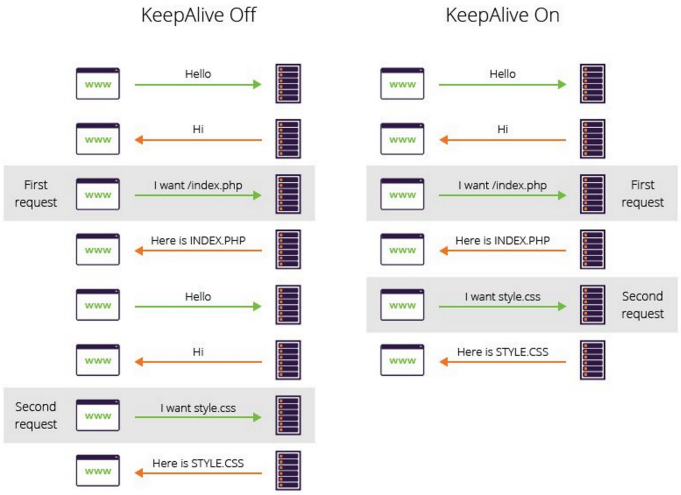
Add an image to the page

Add an external style sheet to the page `<link rel="stylesheet" type="text/css" href="myStyle.css">`

Reload the page again, observe the request(s) being made, and explain the purpose of the connection header.



HTTP Keep-alive er en instruktion der giver en enkelt TCP forbindelse lov til forblive åben for flere HTTP Requests og Responses. Standard HTTP forbindelser lukker efter hver request, for eksempel hvis nogen besøger en side for første gang skal al JavaScript, Billeder og CSS filer hentes, men de skal hentes igen hvis brugeren genindlæser siden.



Pointen med HTTP Keep-Alive er at gemme filer så man ikke behøver at genindlæse alt der følger med af hvad siden har.

3) Monitoring HTTP Headers 3 (Response-codes 3xx)

In the Web-project, created for 1+2, add a new HTML-page called **r.html** and add this text in an h1-tag "**You got redirected to me**".

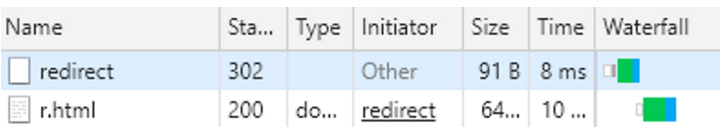
Use the Wizard to create a servlet called redirect

Remove the `processRequest` and the `doPost` method completely from the generated servlet-code.
In the `doGet(..)` method replace the call to `processRequest` with this line:
`response.sendRedirect("r.html");`

While your server is running, open a (Chrome) browser, and Developer Tools and the network tab.

Enter the address for the servlet (**http:localhost:8080/redirect)** into the browser and explain:

- The two HTTP-request you see
- How the browser knew where to go in the second request



Man får to filer tilsendt, hvor **redirect har status koden 302** som er en helt almindelig måde at beskrive en redirection. Den anden fil, r.html har status koden 200 og blev blot modtaget uden problemer.

Browseren kunne vide hvor den skulle hen i den anden request baseret på **Initiator** informationen, hvor man kan se at sourcen for r.html requesten kommer fra redirect filen. Dette giver jo også god mening siden redirect filen er en java servlet der har ét formål: at redirecte til r.html.

3a) Redirecting to HTTPs instead of HTTP

In Chrome enter this address (with the developer window + the network-tab open), and exactly as it is spelt: http://studypoints.info

Explain the first two request monitored (notice where you requested to go, and where you actually ended).

Important: Later this week, you will learn how to set up your own server to use https, and ONLY https.

Name	Sta...	Type	Initiator	Size	Time	Waterfall
studypoints.info	301	tex...	Other	20...	50 ...	
studypoints.info	200	do...	studyp...	3.5...	33 ...	

Dette minder meget om opgaven tidligere med redirect, men i stedet for status kode 302, så får man **status kode 301** på den første fil tilsendt. Status kode 301, også kendt som "301 Moved Permanently" . Dette er den bedste praksis for at redirecte fra HTTP til HTTPS.

4a) Status Codes (5xx)

Use the Wizard to create a servlet called **Ups**

In the `processRequest(..)` method, just before the try-statement add this code:

```
int result = 100/0;
```

While your server is running, open Chrome developer tools and the network tab and then call the servlet.

Write down the response code generated by the server as for the previous exercises

Name	Sta...	Type	Initiator	Size	Time	Waterfall
ups	500	do...	Other	1.7...	38 ...	
favicon.ico	200	x-i...	Other	21....	63 ...	

4b) Status Codes (4xx)

While your server is running, open Chrome developer tools and the network tab, and call this address: **`http://localhost:8080/i_dont_exist`**

Write down the response code generated by the server as for the previous exercises

Name	Sta...	Type	Initiator	Size	Time	Waterfall
idontexist	404	do...	Other	1.2...	11 ...	

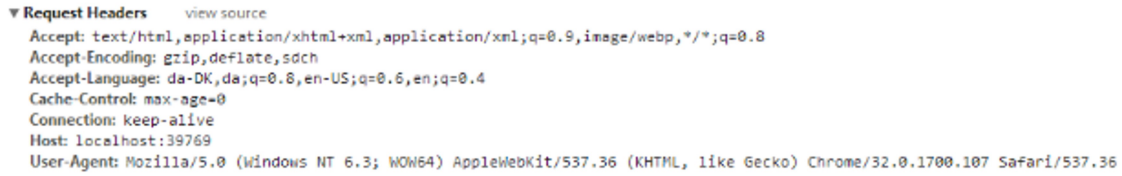
4c) Status Codes - Ranges

Your document, containing the Status Codes for all the exercises done so far, should now contain codes like 2xx, 3xx, 4xx and 5xx.
Explain (write down your answer so you won't forget) the meaning of the first digit in the 3-digit Status Codes you have seen so far.

- 2xx: Betyder at alt gik ok og både request og response kom frem uden problemer.
- 3xx: Betyder at man blev redirected et sted hen.
- 4xx: Filer kunne ikke findes.
- 5xx: En exception / fejl i koden opstod.

5) Get HTTP Request Headers on the Server

We have seen that an HTTP request from a Browser typically includes a lot of headers with information related to the client.



This information is available to a servlet (actually to any web-server technology) via the request object. Create a Servlet, which should output this information in a table as sketched in this figure (or in any way you like, but don't focus on presentation).

Header	Value
host	localhost:39769
connection	keep-alive
cache-control	max-age=0
accept	text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
user-agent	Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/32.0.1700.107 Safari/537.36
accept-encoding	gzip,deflate,sdch
accept-language	da-DK,da;q=0.8,en-US;q=0.6,en;q=0.4

Hints: Use the request objects getHeaderXXX methods.

Løsningen til dette er java servletten kaldet GetHttPHeaders. Så man kan køre projektet og så gå ind på Localhost:8080/Opgave7/GetHttPHeader og se de listede headers.

6) Get/Post-parameters

Create a new HTML-file in the web-project made in exercise 1.
Add a form to the file, including two text input boxes and a submit button as sketched below:

Add an extra input field to the form with type="hidden", name="hidden" and value=12345678.

Add the value "#" for the forms action attribute.

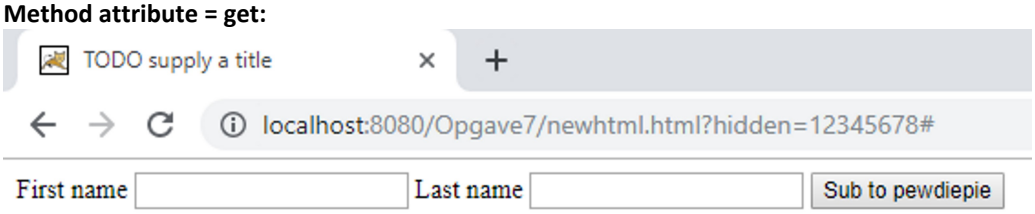
Set the forms method-attribute to the value "GET" (actually the default value) and test the form. Observe what happens in your browser's address field.

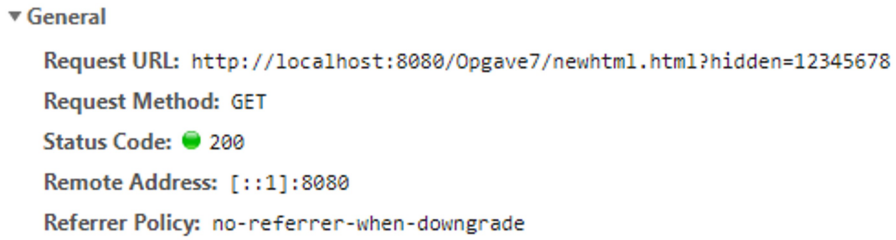
Change the forms method-attribute to the value "POST" and test the form. Observe the change in your browsers address field. Figure out (using Chrome Developer Tools), how parameters are passed in, for a POST request.

Change the forms method-attribute to the value "POST" and test the form. Observe the change in your browsers address field. Figure out (using Chrome Developer Tools), how parameters are passed in, for a POST request.

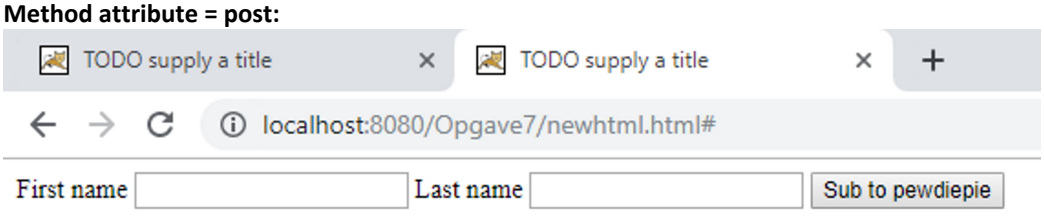
Write down your observations

Method attribute = get:

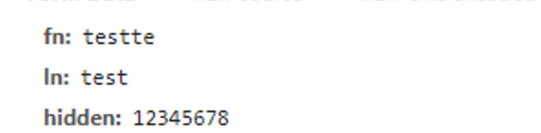




Method attribute = post:



Parameter er passed gennem requesten i følgende:



Session and Cookies

For the next two exercises/demos you should create a new Maven web-project. Both the demos use a Servlet.

7) Sessions (Session Cookies)

f) Most import part of this exercise:
Explain (on paper) using both words and images how the Server can maintain state between subsequent calls even when the state is not transmitted from the client to server.

f) **Most import part of this exercise:**
Explain (on paper) using both words and images how the Server can maintain state between subsequent calls even when the state is not transmitted from the client to server.

```
String name = request.getParameter("name");
if (name != null) {
    request.getSession().setAttribute("name", name);
} else {
    name = (String) request.getSession().getAttribute("name");
}
```

Det første som servletten gør, er at køre denne kode igennem. Denne kode tjekker for en parameter ved navn "**name**" og tilknytter den til objekt refferencen "name". Hvis dette objekt ikke er null, bliver den sat som en Attribut på sessionen med navnet "name" igen.

Hvis den derimod er null, og der ikke blev sendt en parameter med requesten, så vil den prøve at trække den ud som en attribut fra sessionen (castet som en string). **(Bemærk at name stadig kan være null hvis der ikke eksisterer en attribut ved navn name)**.

Det næste der sker i koden er dette stykke:

```
if (name != null) {
    name = (String) request.getSession().getAttribute("name");
    out.println("<p> Welcome " + name + " !</p>");
} else {
    out.println("<h2>Please enter your name, and submit</h2>");
    out.println("<form action='SessionDemo'>");
    out.println("<input type='input' name='name'>");
    out.println("<input type='submit'></form>");
}
```

Dette tjekker nu om name objektet der blev lavet tidligere er null. Hvis dette ikke er tilfældet, betyder det at der endten fuldte parameter med i requesten, eller at attributen lå på sessionen og blev succesfuldt hentet ned som String objekt.

Hvis den derimod er null, bliver der lagt to input tags op, ét som et tekst input der har navnet "name" samt en submit knap, begge to i en form der har actionen at sende en til servletten "SessionDemo", altså den man sidder på.

Når man indtaster en tekst i tekst inputtet og submitter, vil siden nu blive refreshed, men der vil blive tilføjet en parameter sammen med. Dette betyder at al koden kører igen, men denne gang vil der være en parameter, og der vil derfor være et andet udfald af koden.

8) Persistent Cookies

a) In your web project, use the wizard to generate a new servlet
b) Enter *CookieDemo* as the name of the Servlet and *servlets* as package name
c) Change the generated processRequest(..) method as sketched below.
g) The most import part of this exercise:
Explain (on paper) how Cookies can be used to maintain "state" on the client between subsequent calls to a server, even when a browser has been closed down.

```
String name = request.getParameter("name");
if (name != null) {
    Cookie cookie = new Cookie("username", name);
    cookie.setMaxAge(60 * 60 * 24 * 365);
    response.addCookie(cookie);
}
Cookie[] cookies = request.getCookies();
if (cookies != null) {
    for (Cookie cookie : request.getCookies()) {
        if (cookie.getName().equals("username")) {
            name = cookie.getValue();
        }
    }
}
```

Dette er det samme princip som i opgaven tidligere. Dog bliver der benyttet Cookies i stedet for Session Attributter.

Cookies ligger dog lokalt på ens computer, og kan dermed tilgås gennem browseren fra filer der allerede ligger på computeren. Dette betyder at selvom forbindelse OG at sessionen ville blive brudt, så vil der stadig være filer gemt fra websiten som bliver loadet når man tilgår den.