The HTTP Protocol

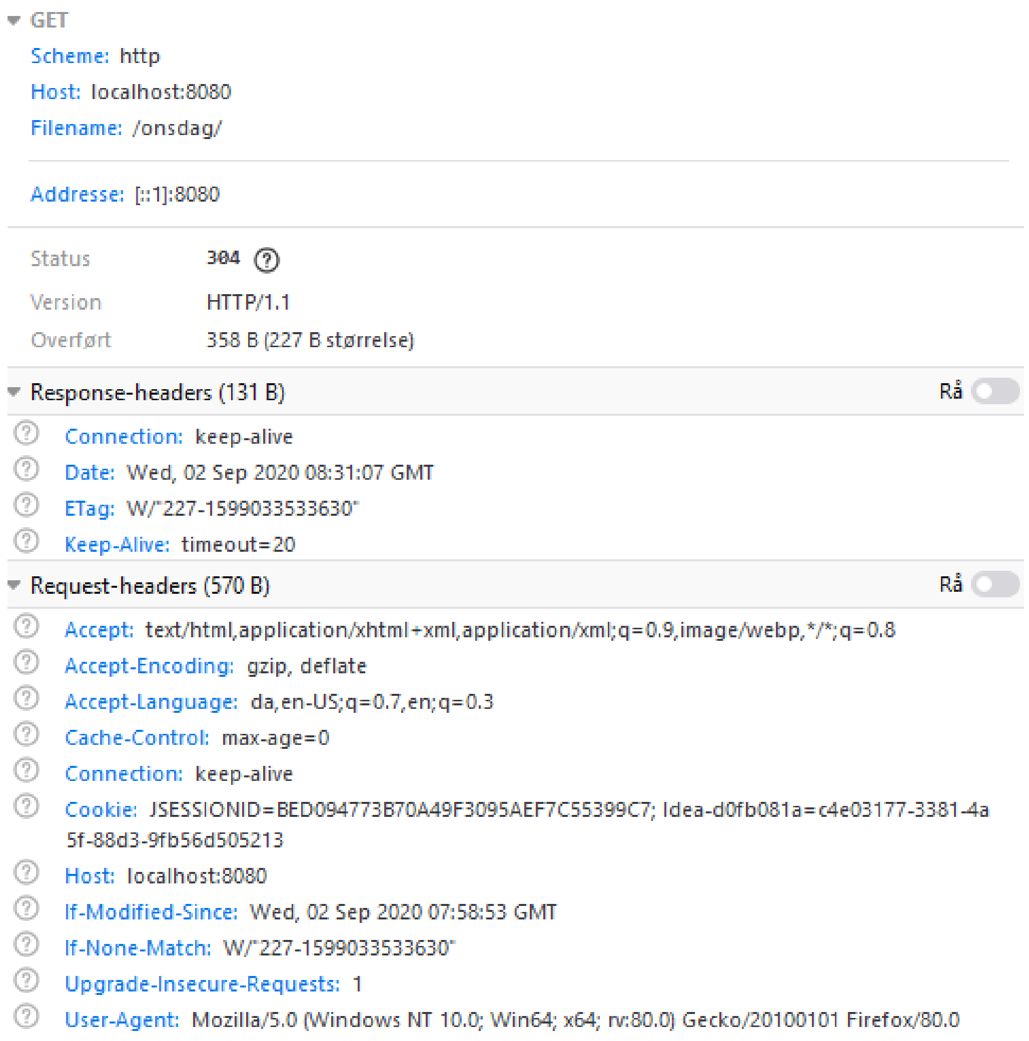
Before you start, you should create a document (or a just a piece of paper) where you should write down the Status Code generated by each of the following exercises (You need this for exercise 4-c)

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 (Windows: Ctrl-shift-j / Mac: Option-Command-i) and press F5

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



```
GET
  Scheme: http
  Host: localhost:8080
  Filename: /onsdag/

  Address: [::1]:8080

  Status      304
  Version     HTTP/1.1
  Overført    358 B (227 B størrelse)

Response-headers (131 B)                                    Rå
  Connection: keep-alive
  Date: Wed, 02 Sep 2020 08:31:07 GMT
  ETag: W/"227-1599033533630"
  Keep-Alive: timeout=20

Request-headers (570 B)                                     Rå
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
  Accept-Encoding: gzip, deflate
  Accept-Language: da,en-US;q=0.7,en;q=0.3
  Cache-Control: max-age=0
  Connection: keep-alive
  Cookie: JSESSIONID=BED094773B70A49F3095AEF7C55399C7; Idea-d0fb081a=c4e03177-3381-4a
  5f-88d3-9fb56d505213
  Host: localhost:8080
  If-Modified-Since: Wed, 02 Sep 2020 07:58:53 GMT
  If-None-Match: W/"227-1599033533630"
  Upgrade-Insecure-Requests: 1
  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:80.0) Gecko/20100101 Firefox/80.0
```

Response-Headers:

Connection: Fortæller om forbindelsen mellem klient og server skal opretholdes.
Date: Fortæller dato og tid på hvornår responses blev gennemført.

Etag:

Keep-Alive: Fortæller hvor længe connectionen skal opretholdes i sekunder.

Request-headers:

Accept/Encoding/Langague: Fortæller typerne på det content clienten kan læse/forstå. Viser f.eks hvilket sprog som browseren bruger.

Cache-Control: Fortæller hvor længe en ressource er ”frisk”, og skal caches igen.

Cookie: Indeholder tidligere ”Cookies” som jeg som bruger har modtaget fra webserveren.

Host: Ip adressen på webserveren, i vores tilfælde vores localhost på port 8080

User-Agent: Fortæller information omkring vores client(browser), i mit tilfælde Mozilla Firefox

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.

Efter reload så bliver der lavet GET requests til at hente billedet og CCS.

Connection headeren: gør at forbindelsen bliver holdt i live, i vores tilfælde er der ikke en bestemt tid sat på om hvor længe forbindelsen skal opretholdes.
Keep-alive gør at vi kan sende flere requests uden den lukker forbindelsen mellem klient og server.

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**

Jeg ser 3 requests:
GET på redirect, som er min servlet.
GET på r.html som er min plain html side.
GET Favicon som er logoet fra tomcat serveren som vises i toppen i browseren.

Det første request jeg laver en en GET request som rammer min servlet. I Servleten er der metoden doGet som tager sig af GET requestene, den metoder sender så et response tilbage som redirecter mig til min r.html side.

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 requests monitored (notice where you requested to go, and where you actually ended).

Som jeg ser det laver jeg en GET request på studypoints.info, og får en tilbage melding om at det requestede adresse er blevet flyttet (Status 301) , i responsen bliver jeg redirected til https://studypoints.info/ som er HTTPS, og kører på port 443 i stedet for 80.

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

HTTP Status 500 – Internal Server Error

Type Exception Report

Message / by zero

Description The server encountered an unexpected condition that prevented it from fulfilling the request.

Exception

```
java.lang.ArithmeticException: / by zero
	ups.processRequest(ups.java:34)
	ups.doGet(ups.java:61)
	javax.servlet.http.HttpServlet.service(HttpServlet.java:626)
	javax.servlet.http.HttpServlet.service(HttpServlet.java:733)
	org.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.java:53)
```

Note The full stack trace of the root cause is available in the server logs.

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

HTTP Status 404 – Not Found

Type Status Report

Message The requested resource [/onsdag/i_dont_exist] is not available

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 = At requestet er blevet modtaget og bearbejdet.

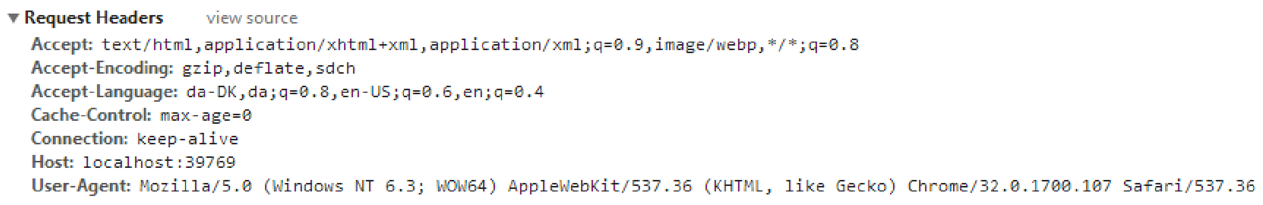
3XX = Redirection = Siden er blevet flyttet og serveren redirecter til den nye side.

4XX = Client = En client fejl, der er muligvis forkert syntaks i URL.

5XX = Serverfejl = Serveren f.eks ikke kan sende et brugbart request tilbage. Eller ikke svarer.

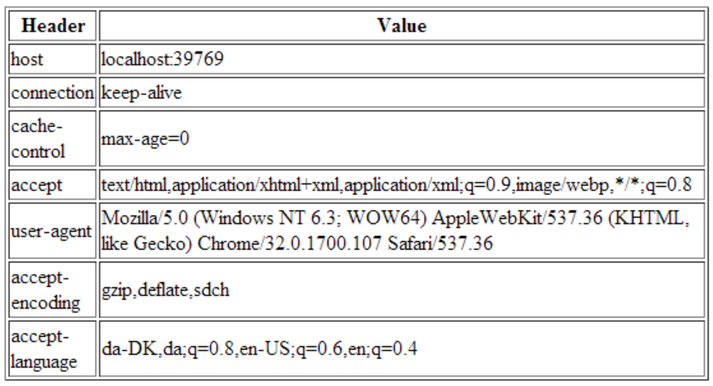
5) Get HTTP Request Headers on the Server

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



```
▼ Request Headers     view source
   Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
   Accept-Encoding: gzip,deflate,sdch
   Accept-Language: da-DK,da;q=0.8,en-US;q=0.6,en;q=0.4
   Cache-Control: max-age=0
   Connection: keep-alive
   Host: localhost:39769
   User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/32.0.1700.107 Safari/537.36
```

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 <u>don't focus on presentation</u>).

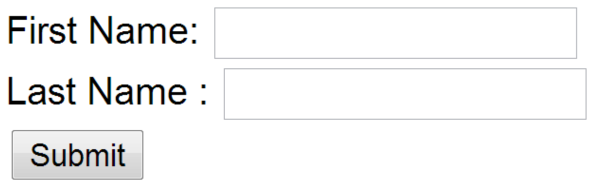


Header	Value
host	localhost:39769
connection	keep-alive
cache-control	max-age=0
accept	text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
user-agent	Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/32.0.1700.107 Safari/537.36
accept-encoding	gzip,deflate,sdch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

Hints: Use the request objects getHeaderXXX methods.

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.

http://localhost:8080/onsdag/form.html?fname=&lname=&hidden=123456#

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.

http://localhost:8080/onsdag/form.html#

I GET metoden bliver form parametrene vist oppe i URL’en, Hvorimod på GET kan man kun se dem ved at åbne developer tools og kigge under request parameters.

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)

d. Enter your name and press submit, copy the URL in the browser into your clipboard, close the tab (but not the browser) and load the page again in a new tab using the URL in the clipboard.

e. While doing the things in step d, you should monitor the content of your local cookies and the HTTP requests being sent, using the development tools in Chrome.

f. Most import part of this exercise:

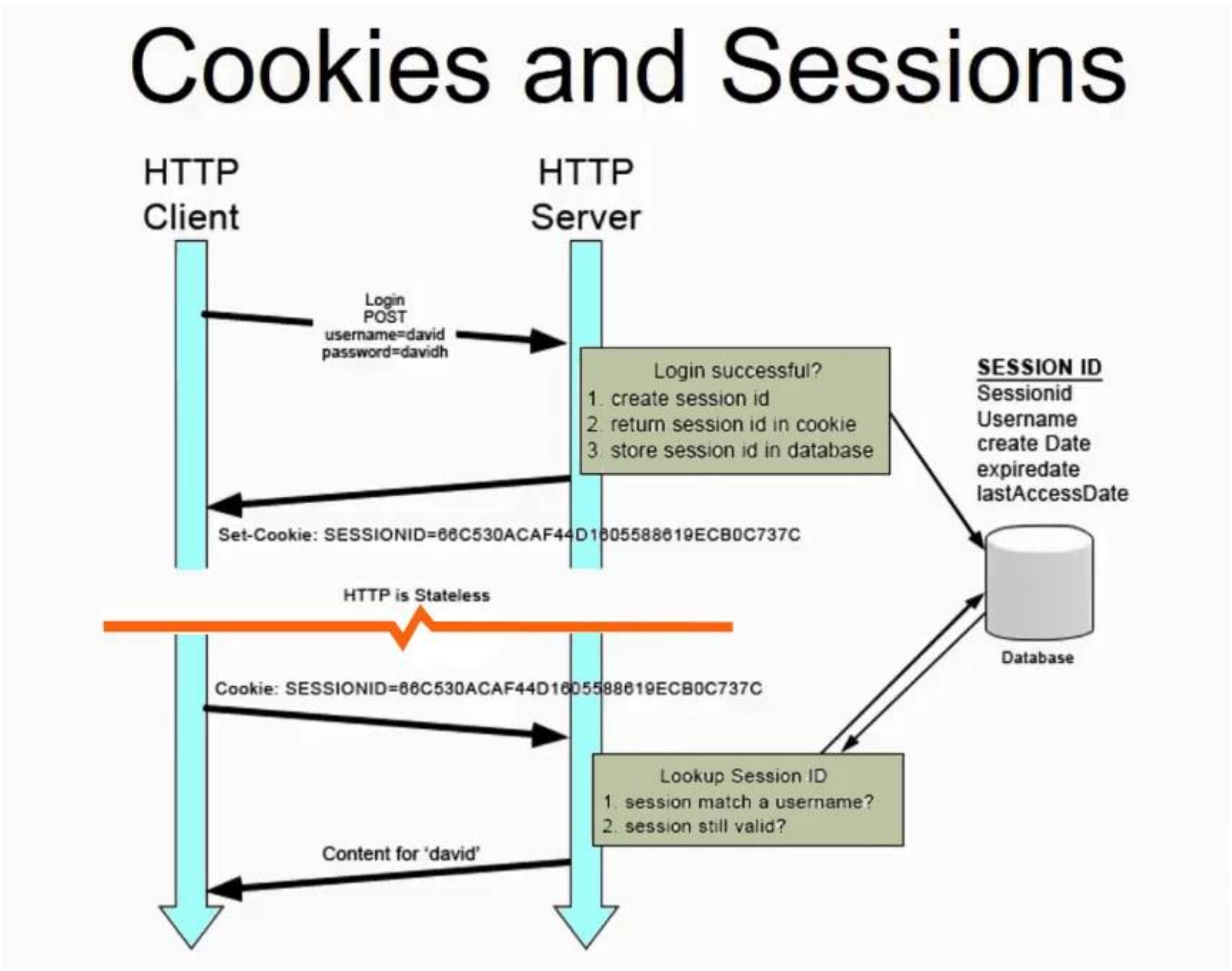
Explain (on paper) using both words and images how the Server can maintain state between subsequent calls even when the state is not transmitted from the client to server.

Forklarer længere nede omkring begge som jeg forstår det.

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.

d. Enter your name and press submit, copy the URL in the browser into your clipboard, close the tab (but not the browser) and load the page again in a new tab using the URL in the clipboard.
e. Now close your browser (you could even close your laptop, but don't ;-) , open it again and load the page again using the URL in the clipboard
f. While doing the things in step e, you should monitor the content of your local cookies and the HTTP requests being sent, using the development tools in Chrome.
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.


Cookies and Sessions
HTTP Client
HTTP Server
Login POST username=david password=davidh
Login successful?
1. create session id
2. return session id in cookie
3. store session id in database
SESSION ID
Sessionid
Username
create Date
expiredate
lastAccessDate
Set-Cookie: SESSIONID=66C530ACAF44D1605588619ECB0C737C
HTTP is Stateless
Database
Cookie: SESSIONID=66C530ACAF44D1605588619ECB0C737C
Lookup Session ID
1. session match a username?
2. session still valid?
Content for 'david'

Session cookies forstår jeg på den måde at når vi som client laver et request til webserveren får vi tilbage i vores response en cookie, som er et slags finder aftryk af denne ”session”, da http er stateless er dette nødvendigt for at kunne ”huske” nogle af de informationer vi ønsker.

I eksemplet ovenfor laver vi et login POST request med username og password, hvis username og password passer, laver httpserveren et ”session id” som bliver returneret til clienten. Det går at næste gang vi requester noget fra serveren, så sender vi vores cookie session id med i requesten, så kan serveren genkende hvem vi er og returnere det pågældende indhold som relatere til i dette eksempel ”david”

Session Cookien bliver gemt indtil vi lukker browseren, eller efter cirka 30 min.

I mens Persistant cookies bliver gemt så længe som de er sat til. F.eks i vores øvelse er den sat til 1 år (cookie.setMaxAge(60 * 60 * 24 * 365);)

Man kunne også se i developertools under ”cookies” se vores parameter ”username” og value.