Computers make very fast, very accurate mistakes.

Unknown

7 Cookies and Sessions

7.1 Introduction

As you already should know: PHP is a stateless scripting language. After a script is executed, everything is cleared from memory. This makes it rather lightweight, but raises a big problem: how can we achieve something like sessions? How can PHP know that a request coming from a client is coming from the same client as the previous request where that user authenticated himself? And how can we save some preferences a user has made (e.g. the language or the font-size) so that the next time that user visits, the PHP script can remember his settings and reload them?

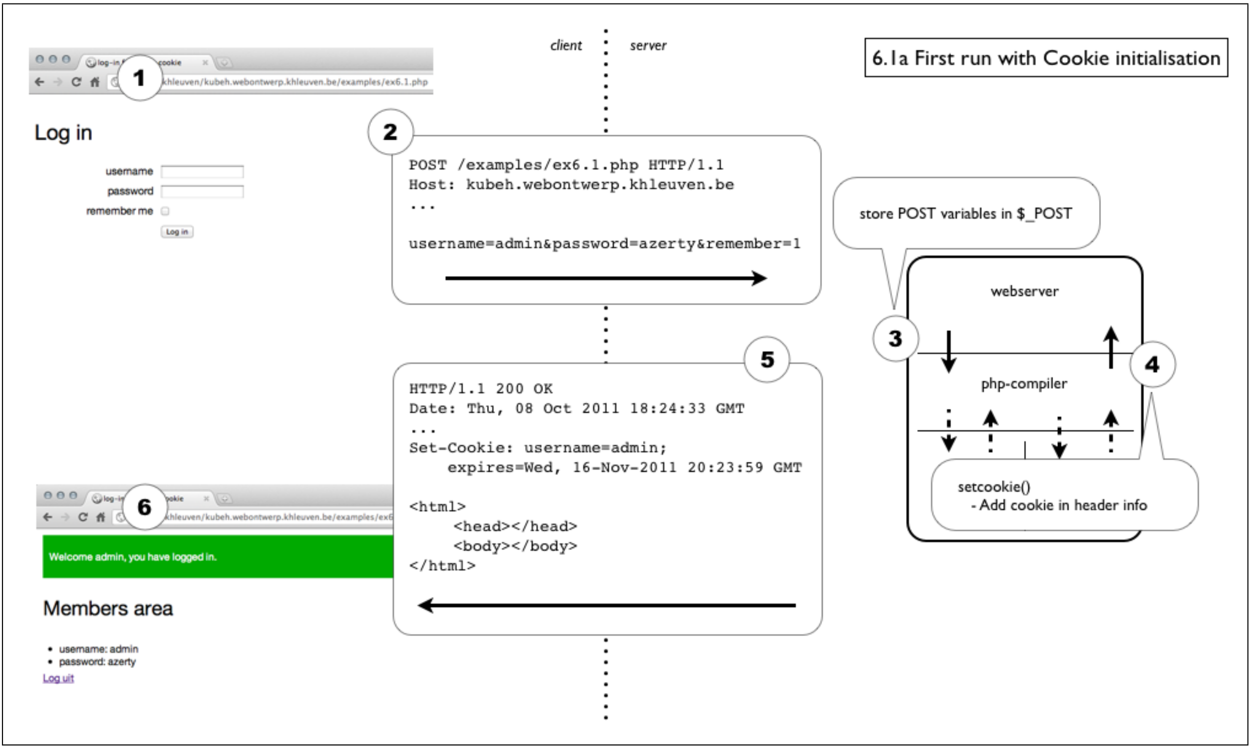
These storage of temporary data is achieved with the use of cookies.

Cookies are used client side as well as server side. They both are a piece of data (an array of variables with a name and a values) linked to a client. client side cookies are included in the header of a GET or POST request to the server.

7.2 Cookies

Cookies are created by a script and stored on the client side. They have an expliration time and are limited to a certain domain and/or path. The browser includes them in every request to a website as long as they are valid. If a cookie has no expiration time, the cookie expires when the browser is closed. All info on these parameters can be found at `www.php.net/manual/en/function.setcookie.php`.

Have a look at the drawings below. You can see the code in action at
`https://dynweb.webontwerp.khleuven.be/dynweb/examples/ex7.1.php`.
The sourcefile is available at `https://dynweb.webontwerp.khleuven.be/dynweb/examples/ex7.1.phps`.



```
POST /examples/ex6.1.php HTTP/1.1
Host: kubeh.webontwerp.khleuven.be
...

username=admin&password=azerty&remember=1
```

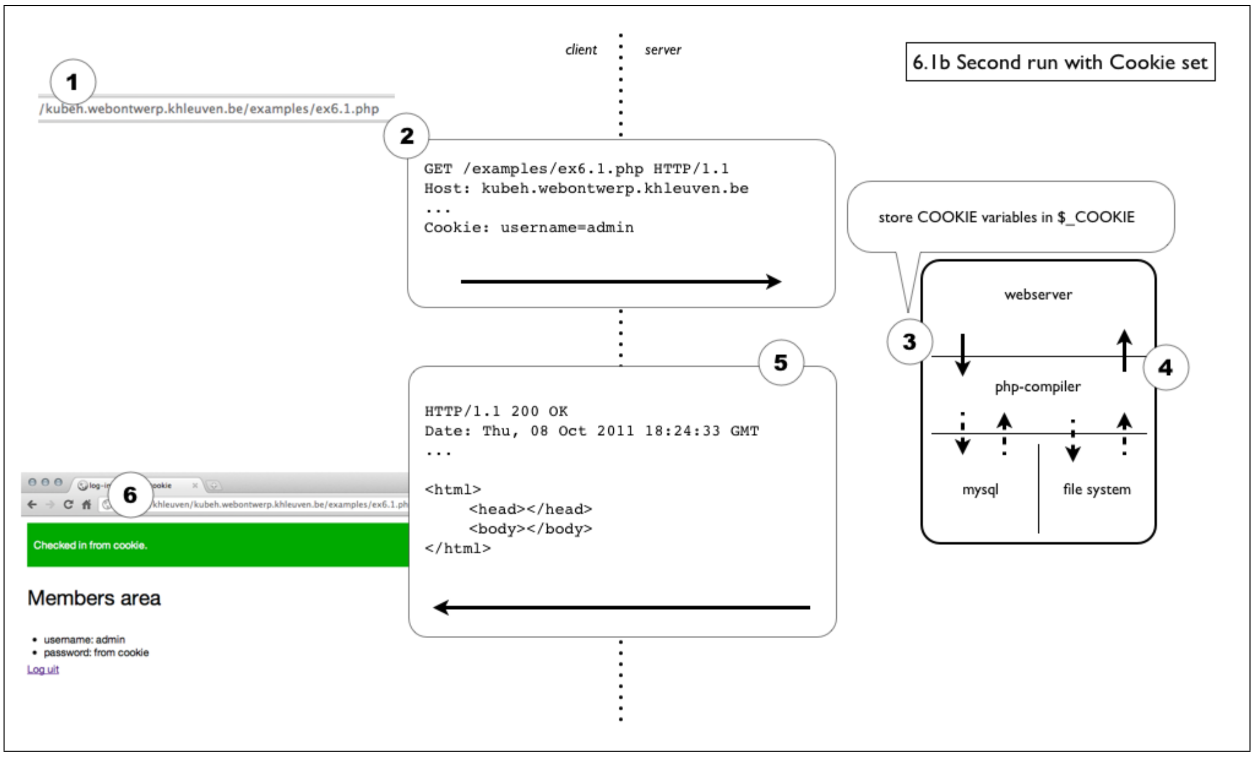
```
HTTP/1.1 200 OK
Date: Thu, 08 Oct 2011 18:24:33 GMT
...
Set-Cookie: username=admin;
    expires=Wed, 16-Nov-2011 20:23:59 GMT

<html>
     <head></head>
     <body></body>
</html>
```

Fig. 7.1 Client-Server model with initialisation of a cookie

1. A user sends a filled out form.
2. The browser translates this url in a POST request with a header and content.
3. The POST values are read by PHP and stored in $_POST.
4. In the PHP script a cookie is set using `setcookie()`. By doing this, some headers are set.
5. The webserver returns the message, with in the header a `Set-Cookie` instruction.
6. The browser reads the message, checks the headers and stores the cookie information, reads the code and interpretes the HTML inside to display the markup.

This cookie 'remembers' the username that is filled out in the login-form. So, even when you close the browser, you still will be automatically log in. The drawings below explains what happens.

You can see the code in action if you reload the page (or close your browser and visits the website again) at
`https://dynweb.webontwerp.khleuven.be/dynweb/examples/ex7.1.php`.



```
GET /examples/ex6.1.php HTTP/1.1
Host: kubeh.webontwerp.khleuven.be
...
Cookie: username=admin
```

```
HTTP/1.1 200 OK
Date: Thu, 08 Oct 2011 18:24:33 GMT
...

<html>
     <head></head>
     <body></body>
</html>
```

Fig. 7.2 Client-Server model with use of a cookie

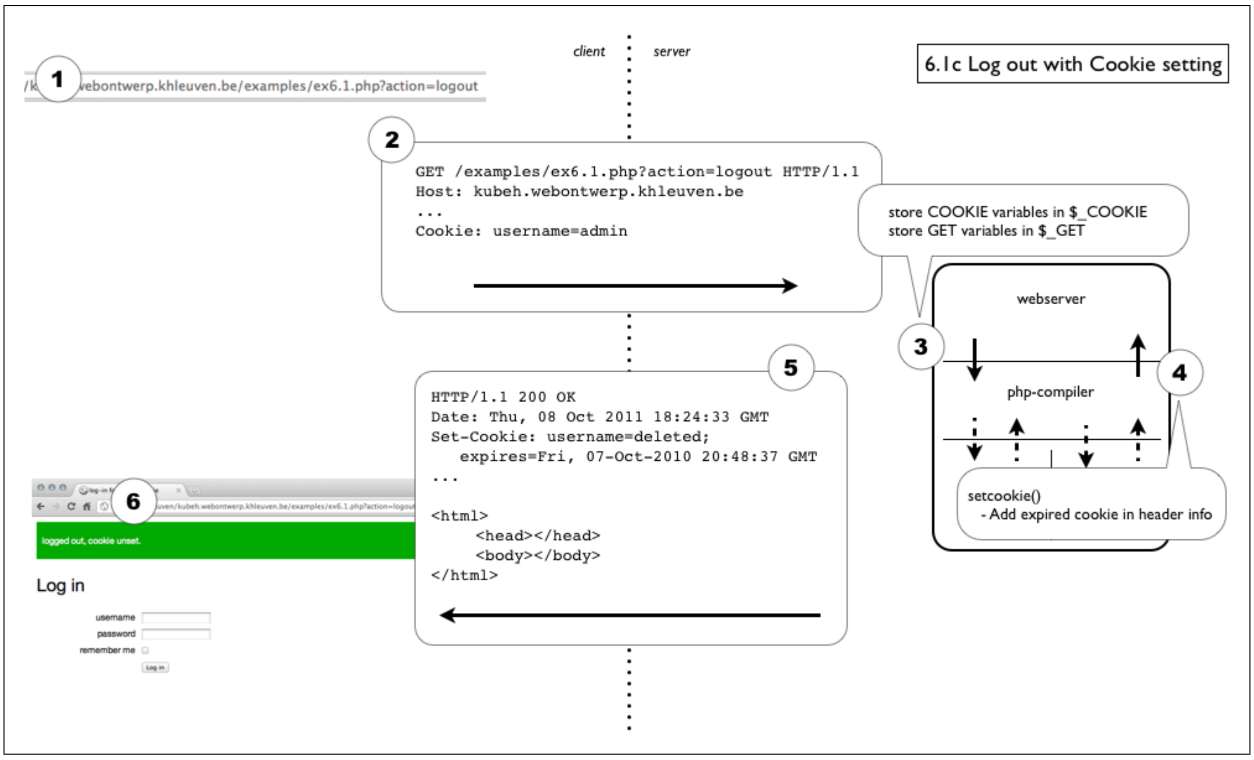
1. The user visits the site for the second time.
2. The browser checks if a valid cookie exists and adds the information to the header of the request.
3. The cookie values are read by PHP and stored in $_COOKIE.
4. The PHP script can use the cookie information and returns its output.
5. The webserver returns the message.
6. The browser reads the message, checks the headers, reads the code and interpretes the HTML inside to display the markup.

In this second run no extra cookie is set. The PHP script gave the cookie an expiration time that it hasn't expired yet. No password was set in the cookie so the PHP script just uses the username from the cookie.

So as long as the cookie doesn't expire or the cookie isn't manually deleted by the user, the cookie data is automatically sent every time the user visits the website.

A PHP script can also remove a cookie. Removing a cookie is done by making it expire by giving it an expiration date in the past. The browser will try to update the cookie's information and forcing it to expire.

This happens when a user logs out. You can see the code in action if you are logged in and click the logout link at
`https://dynweb.webontwerp.khleuven.be/dynweb/examples/ex7.1.php`.


client
server
6.1c Log out with Cookie setting
1
/kubeh.webontwerp.khleuven.be/examples/ex6.1.php?action=logout
2
store COOKIE variables in $_COOKIE
store GET variables in $_GET
3
webserver
php-compiler
4
setcookie()
- Add expired cookie in header info
5
6
logged out, cookie unset.
Log in
username
password
remember me
Log in

```
GET /examples/ex6.1.php?action=logout HTTP/1.1
Host: kubeh.webontwerp.khleuven.be
...
Cookie: username=admin
```

```
HTTP/1.1 200 OK
Date: Thu, 08 Oct 2011 18:24:33 GMT
Set-Cookie: username=deleted;
   expires=Fri, 07-Oct-2010 20:48:37 GMT
...

<html>
     <head></head>
     <body></body>
</html>
```

Fig. 7.3 Client-Server model destroying a cookie

1. The user visits the logout link.
2. The browser checks if a valid cookie exists and adds the information to the header of the request.
3. The get values are read, cookie values are read by PHP and stored in $_COOKIE.
4. The PHP script resets the value and expiration of the cookie.
5. The webserver returns the message, with in the header a `Set-Cookie` instruction with an expiration time in the past.
6. The browser reads the message, checks the headers and stores the cookie information (thus deleting the cookie), reads the code and interpretes the HTML inside to display the markup.

The cookie is now deleted. A next request for the webpage won't add a cookie to the header.

Some remarks about the use of cookies:

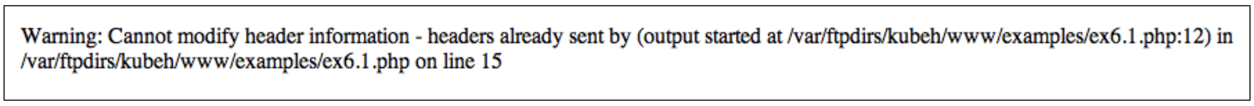
- You can't store large chunks of data in cookies: use a database / txt source for that.
- When developing PHP scripts with cookies, it's very handy to install a browser extension to manipulate/erase cookies manually. Have a look at `https://dynweb.webontwerp.khleuven.be/dynweb/home/resources`
- Data in cookies are not safe. It is very easy to manipulate the data. E.g. Try to change the username in the previous examples.

And finally: when using cookies (and sessions as we will see later on), it is important to not echo any output before you call functions that tamper the http header (e.g. `setcookie()`).

Listing 7.1 Trying to manipulate header information after generating output

```php
<?php
// echo something
echo ' ';

// this line will manipulate header info, which is not allowed after the output has been started.
setcookie ('username', '', time() - 3600);
?>
```

When you see an error like the following:



```
Warning: Cannot modify header information - headers already sent by (output started at /var/ftpdirs/kubeh/www/examples/ex6.1.php:12) in
/var/ftpdirs/kubeh/www/examples/ex6.1.php on line 15
```

Fig. 7.4 Error when trying to modify the headers after the output has started

You should check if there isn't an `echo` or any other output done before the function call to modify the headers.

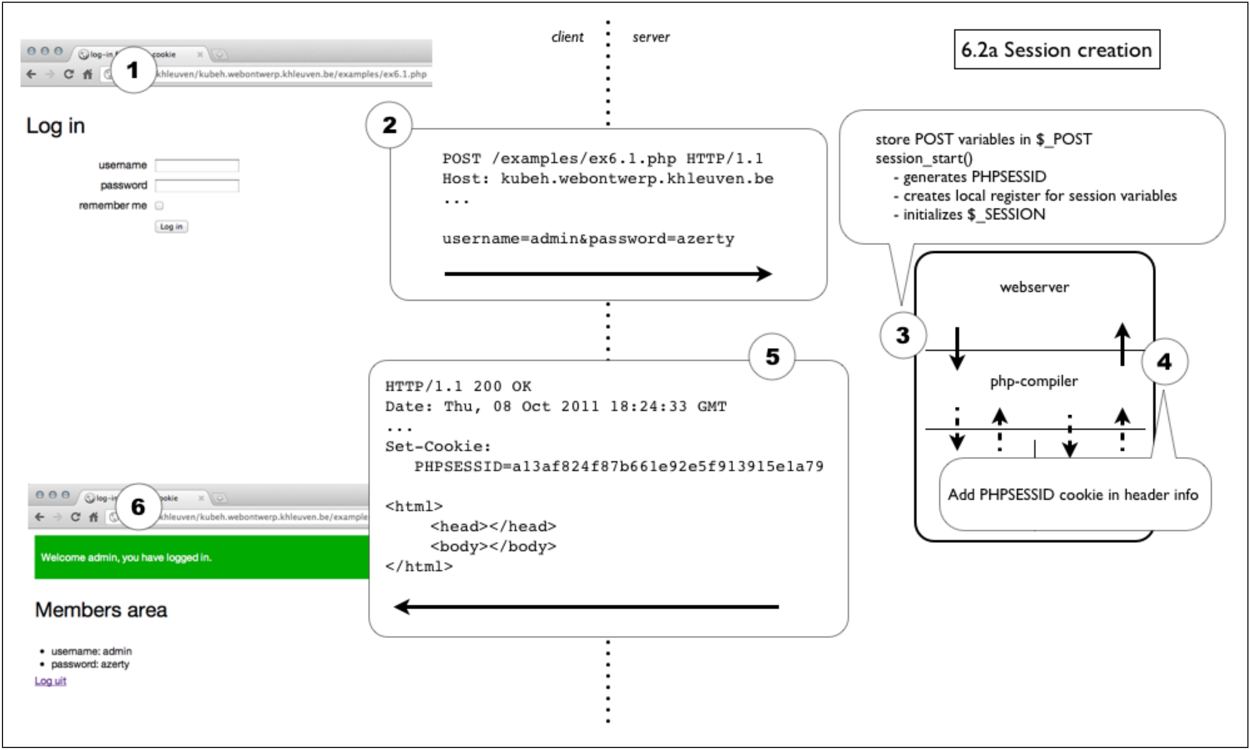
Exercise 7.1 - generate a counter for visits

- Download the source file from
  `http://dynweb.webontwerp.khleuven.be/exercises/7.1-source.txt`
- Create a PHP script that achieves the result you can see at
  `http://dynweb.webontwerp.khleuven.be/exercises/7.1.php`.
  Click on a link and see the result.
  Don't forget to also have a look at the source-code.
- Upload your file so that it is accessible at
  `http://<studentnr>.webontwerp.khleuven.be/exercises/7.1.php`
- Extra: Cheat on your own website by editing your cookie (try to display the 100000 message).

7.3 Sessions

When a user logs in at your website, you want te koop him logged in to display personal pages based on his profile. You surely don't want him to login every time over and over again.

A session is a collection of data for a specific sessionid. When you initialise a session by calling `session_start()`, a session is generated at the server and a token (the sessionid) is send back to the client as a cookie.


client
server
6.2a Session creation
1
khleuven/kubeh.webontwerp.khleuven.be/examples/ex6.1.php
Log in
username
password
remember me
Log in
2
3
webserver
php-compiler
4
Add PHPSESSID cookie in header info
5
6
Welcome admin, you have logged in.
Members area
Log uit

```
POST /examples/ex6.1.php HTTP/1.1
Host: kubeh.webontwerp.khleuven.be
...

username=admin&password=azerty
```

store POST variables in $_POST
session_start()
- generates PHPSESSID
- creates local register for session variables
- initializes $_SESSION

```
HTTP/1.1 200 OK
Date: Thu, 08 Oct 2011 18:24:33 GMT
...
Set-Cookie:
   PHPSESSID=a13af824f87b661e92e5f913915e1a79

<html>
     <head></head>
     <body></body>
</html>
```

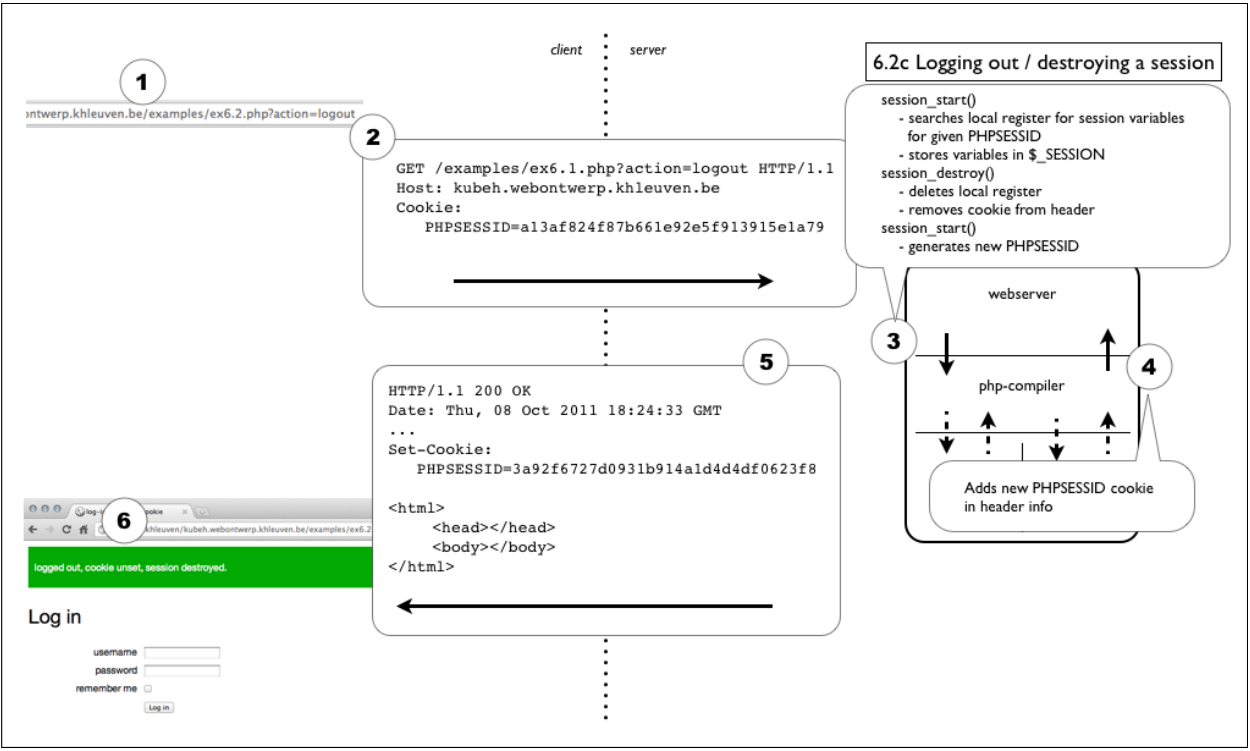
- username: admin
- password: azerty

Fig. 7.5 Client-Server model with session initialisation

1. A user sends a filled out form.
2. The browser translates this url in a POST request with a header and content.
3. The POST values are read by PHP and stored in $_POST. `session_start` is executed and a sessionid is generated.
4. At the end of the PHP script, the values in $_SESSION are saved to the server registry. The PHPSESSID is set as a cookie in the response header.
5. The webserver returns the message, with in the header a `Set-Cookie` instruction for the sessionid, with no expiration time, which means it will auto-expire when the browser is closed.
6. The browser reads the message, checks the headers and stores the cookie information, reads the code and interpretes the HTML inside to display the markup.

As long as you keep the browser open until you hit the log out link, your session cookie is kept alive. But if you don't make a new request to the server (clicking a link, reloading a page, ...) within the session timeout configured at the server, your session register will be erased.

Everytime you make a new request with your PHPSESSID as a cookie, the expiration time at serverside is extended by the timeout. At standard this will be 20 minutes.


client
server
6.2c Logging out / destroying a session
1
ıntwerp.khleuven.be/examples/ex6.2.php?action=logout
2
3
4
5
6
webserver
php-compiler
Adds new PHPSESSID cookie in header info
log-i… …ookie
…khleuven/kubeh.webontwerp.khleuven.be/examples/ex6.2
Log in
username
password
remember me
Log in

```
GET /examples/ex6.1.php?action=logout HTTP/1.1
Host: kubeh.webontwerp.khleuven.be
Cookie:
   PHPSESSID=a13af824f87b661e92e5f913915e1a79
```

```
session_start()
  - searches local register for session variables
    for given PHPSESSID
  - stores variables in $_SESSION
session_destroy()
  - deletes local register
  - removes cookie from header
session_start()
  - generates new PHPSESSID
```

```
HTTP/1.1 200 OK
Date: Thu, 08 Oct 2011 18:24:33 GMT
...
Set-Cookie:
   PHPSESSID=3a92f6727d0931b914a1d4d4df0623f8

<html>
     <head></head>
     <body></body>
</html>
```

logged out, cookie unset, session destroyed.

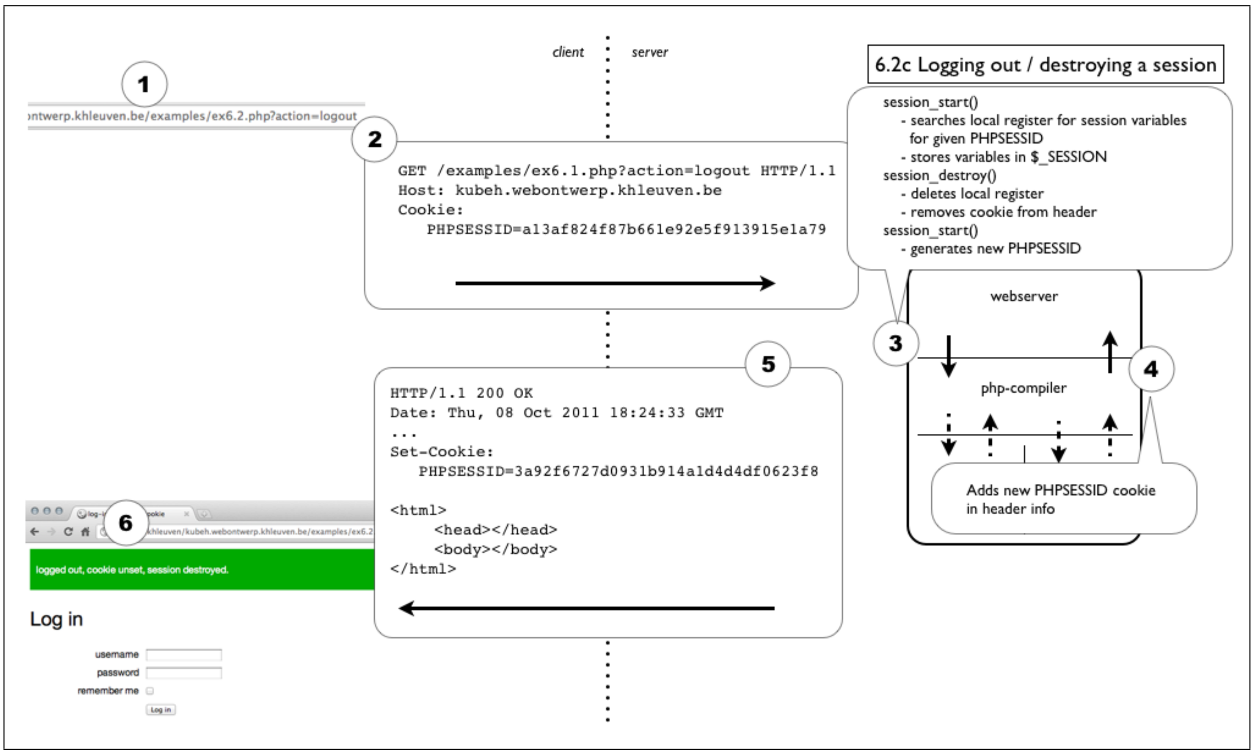
Fig. 7.6 Client-Server model reloading a page with active session

1. A user reloads the page (or visits another page within the same website that has `session_start` active).
2. The browser checks if a valid cookie exists and adds the information to the header of the request (PHPSESSID is valid).
3. Values from the server registry for the given PHPSESSID are loaded into $_SESSION.
4. The output of the script is returned. $_SESSION values are saved and PHPSESSID is put back as a cookie in the header.
5. The webserver returns the message, including the cookie.
6. The browser reads the message, checks the headers, reads the code and interpretes the HTML inside to display the markup.

The PHPSESSID is a unique id for a users' session. It is the login and password in one phrase. So if you can grab the session id from somebody else, you can 'be' him at the website he visits. *Google for 'firesheep' for an exploit at stealing someone else's facebook session.*

You can see the code in action if you log in and reload the page by clicking at the link at the bottom at
`https://dynweb.webontwerp.khleuven.be/dynweb/examples/ex7.2.php`.
View the source by visiting `https://dynweb.webontwerp.khleuven.be/dynweb/examples/ex7.2.phps`.

A way to manually erase the serverside session is by executing `session_destroy()`. This deletes the PHPSESSID and the sessiondata and removes the PHPSESSID cookie from the header.


client
server
6.2c Logging out / destroying a session
1
ntwerp.khleuven.be/examples/ex6.2.php?action=logout
2
session_start()
- searches local register for session variables for given PHPSESSID
- stores variables in $_SESSION
session_destroy()
- deletes local register
- removes cookie from header
session_start()
- generates new PHPSESSID
webserver
3
php-compiler
4
Adds new PHPSESSID cookie in header info
5
6
logged out, cookie unset, session destroyed.
Log in
username
password
remember me
Log in

```
GET /examples/ex6.1.php?action=logout HTTP/1.1
Host: kubeh.webontwerp.khleuven.be
Cookie:
   PHPSESSID=a13af824f87b661e92e5f913915e1a79
```

```
HTTP/1.1 200 OK
Date: Thu, 08 Oct 2011 18:24:33 GMT
...
Set-Cookie:
   PHPSESSID=3a92f6727d0931b914a1d4d4df0623f8

<html>
     <head></head>
     <body></body>
</html>
```

Fig. 7.7 Client-Server model destroying a session

1. A user reloads the page (or visits another page within the same website that has `session_start` active).
2. The browser checks if a valid cookie exists and adds the information to the header of the request (PHPSESSID is valid).
3. Values from the server registry for the given PHPSESSID are loaded into $_SESSION. The session is destroyed and a new PHPSESSID is generated.
4. The output of the script is returned. $_SESSION values are saved and the new PHPSESSID is put back as a cookie in the header.
5. The webserver returns the message, including the cookie.
6. The browser reads the message, checks the headers, reads the code and interpretes the HTML inside to display the markup.

Some remarks about the use of sessions:

- You can't store large chunks of data in sessions: use a database / txt source for that.
- Data in cookies are kept safe at server side. A user cannot manually tamper with this data.
- A PHPSESSID can be 'stolen'. This is called 'session-hijacking'.

The remark made for cookies about altering header info after outputting some data also applies for sessions. A session is represented by a cookie and therefore must be initialised before any output is generated.