CTIS 256 Web Technologies II

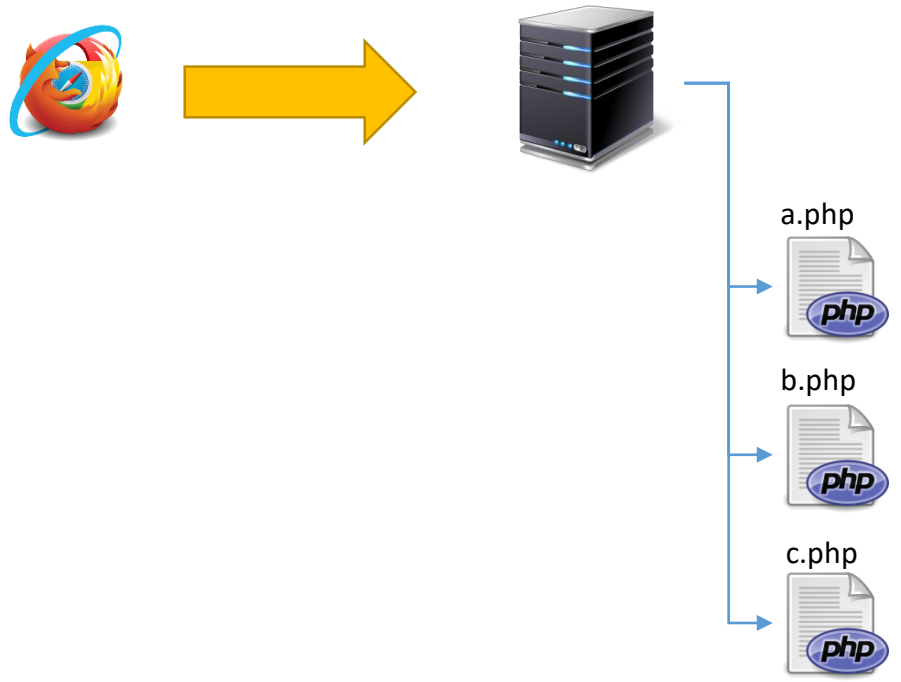
Notes # 9

Session Management

Serkan GENÇ

Problem

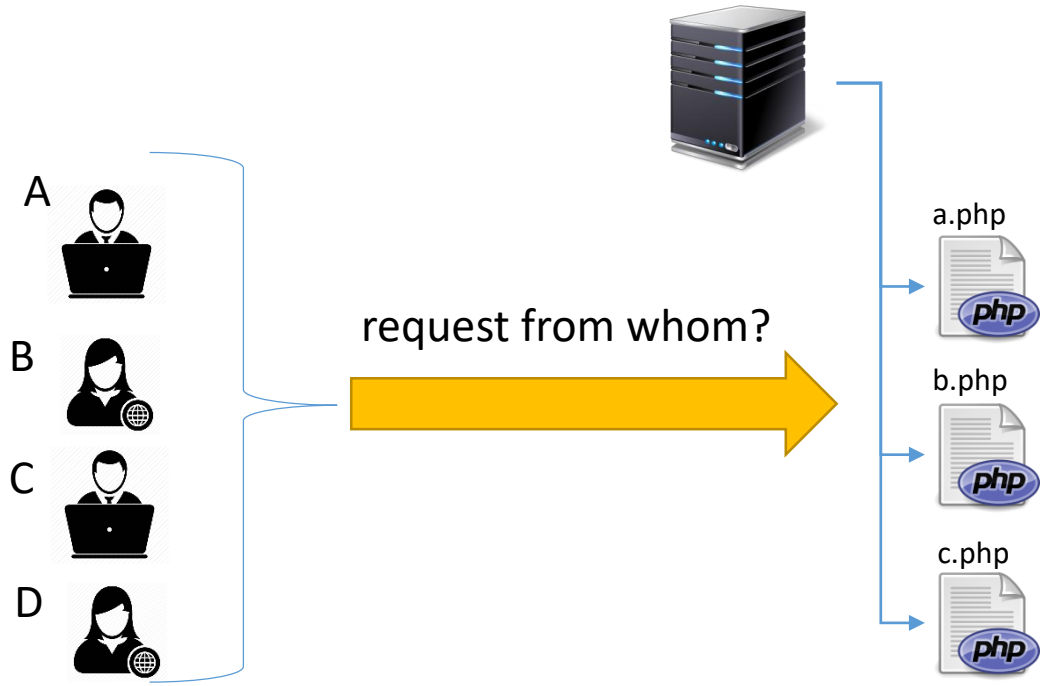
- HTTP is a stateless protocol.
- Requests are isolated and independent from each other.
- How does one request communicate with previous or next requests ?
- How does an application remember its previous states?
  - did the user log in before?
  - what is the content of the shopping cart?
- Question : *Assume that there are two scripts in our web application, namely, "a.php" and "b.php". How does "b.php" know that "a.php" was executed before by a specific user?*
  - *How does new request "b.php" know if there was a request "a.php" before?*
- Solution: a global shared area **per user**.
  - **HTTP Cookie**
  - **Session**



They work like three independent programs rather than as a one logical application. They don't communicate and share states/data for a common goal. Session management is all about making them work together for the logic of the application.

HTTP Cookies

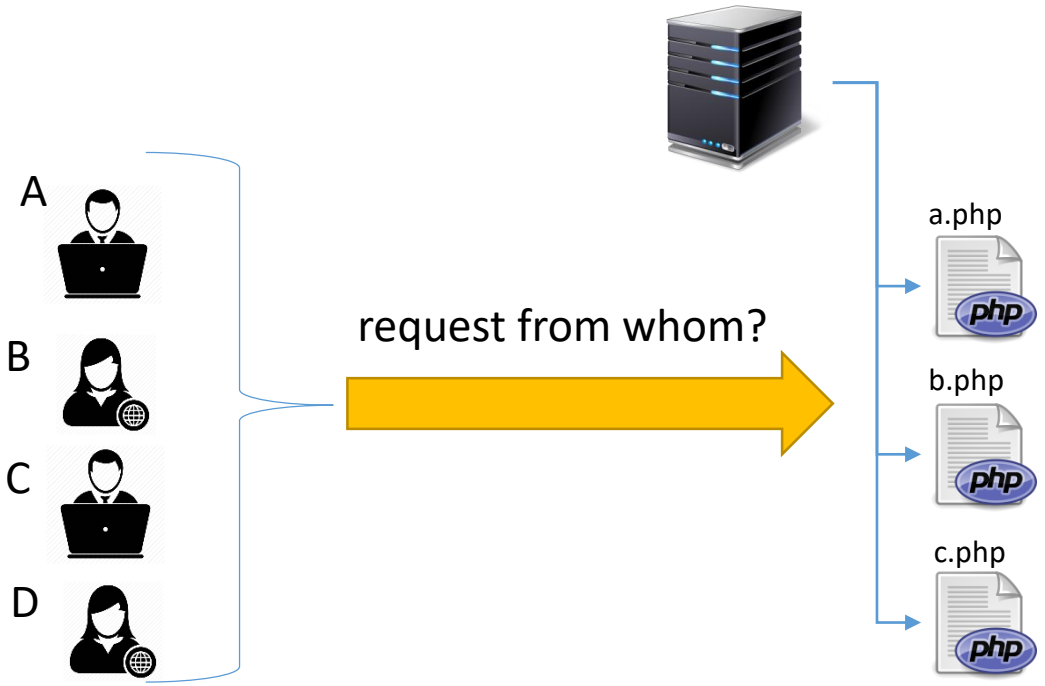
(tracking users)



Question : How does "*a.php*" find out if the request is coming from "user A"?

HTTP Cookies

(tracking users)

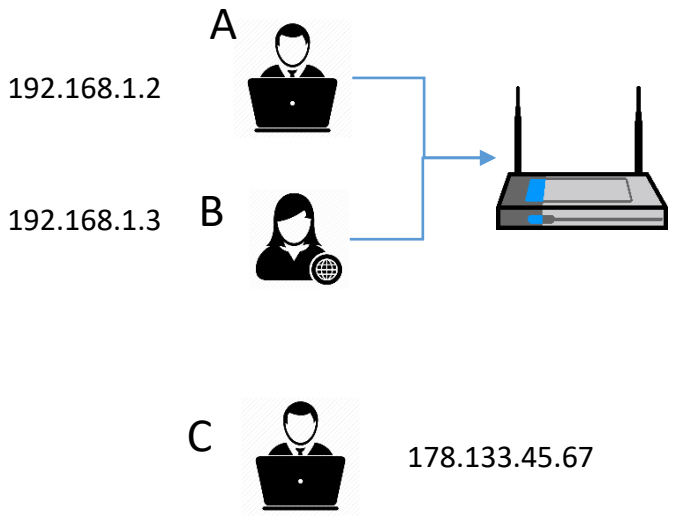


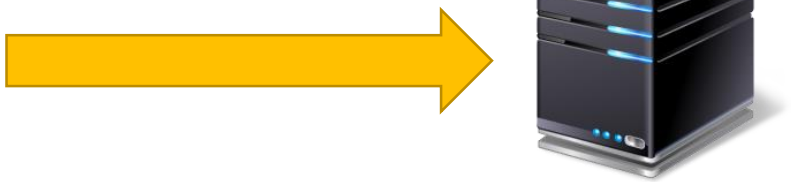
Question : How does "*a.php*" find out if the request is coming from "user A"?

What about storing user browser's IP address and username in a database table?
When a request comes in, it gets the IP address of the request, and finds out the name of the user. Does it work?

HTTP Cookies

(tracking users)





IP	User
144.173.25.77	?
178.133.45.67	C
164.13.34.13	D

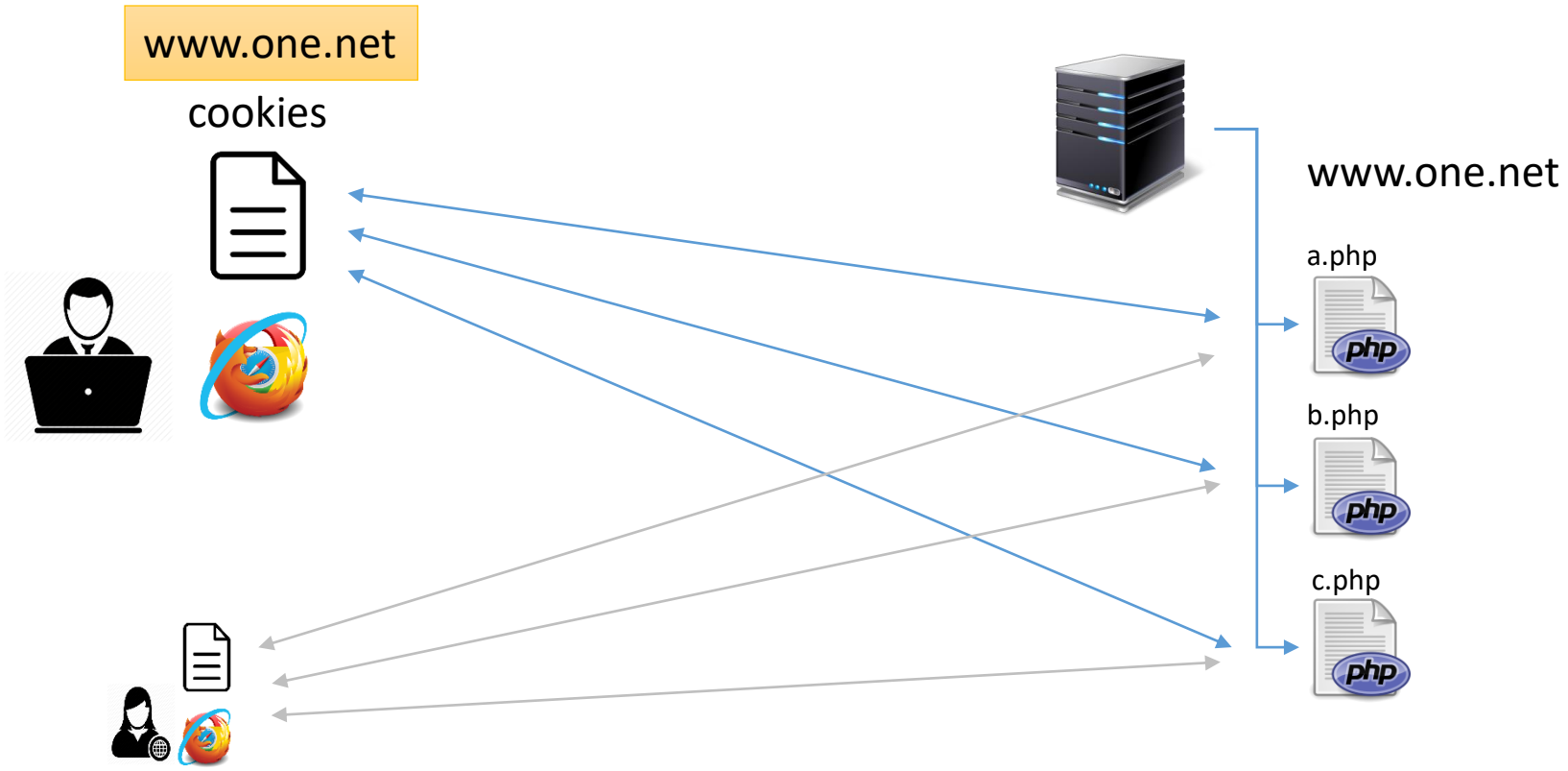
Question : How does "*a.php*" find out if the request is coming from "user A"?

Using IP address to differentiate users does not work since **User A** and **B** have the same public IP addresses.

HTTP Cookies

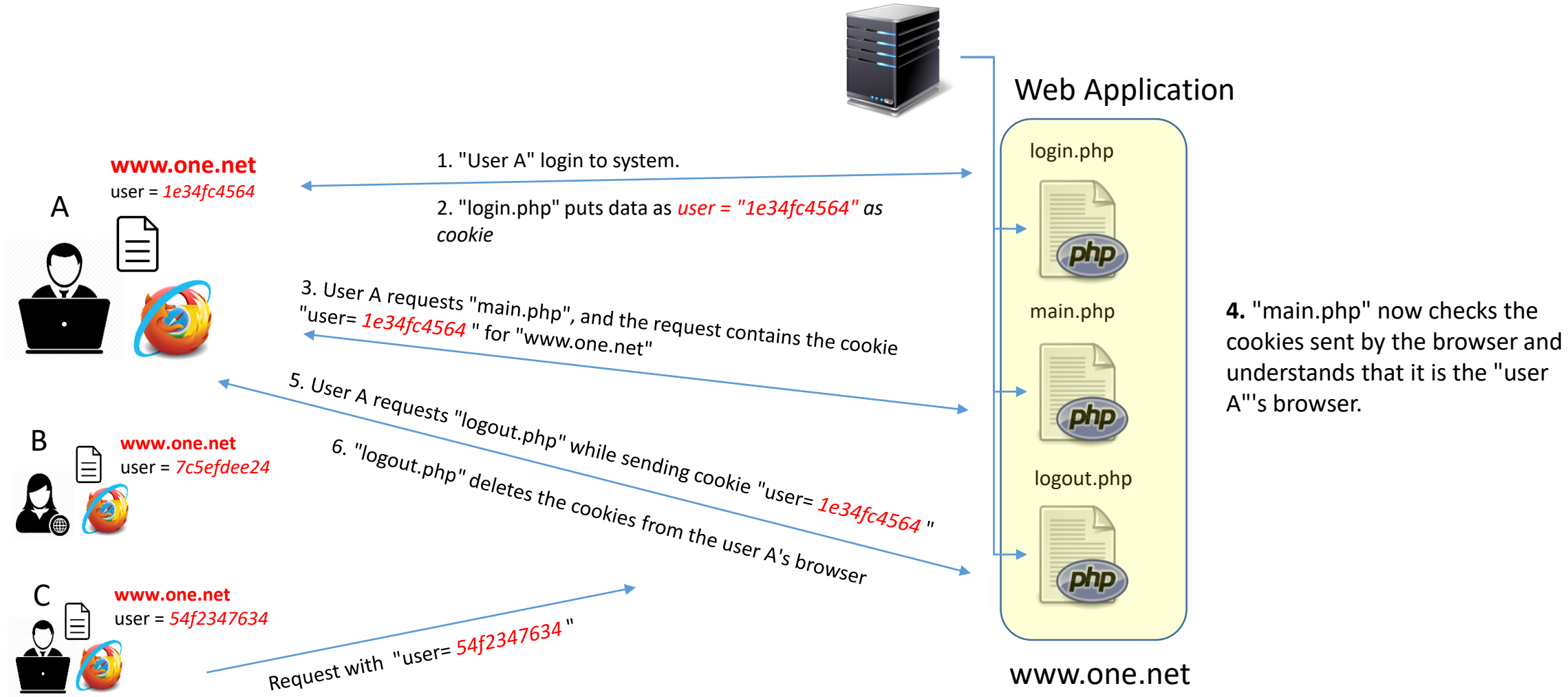
- An HTTP cookie is a small piece of data that a server-side program sends to store within the user's browser.
- Web browser sends cookies (data) back to the server in all subsequent requests.
- Php application can remember/track the user with cookie mechanism.
- Cookie data are transported along with HTTP request and response headers.
- **Used for**:
  - Session Management (user auto login, shopping cart)
  - Tracking (analyzing user behaviors)

Cookies as a Shared Storage



Logically, cookies are client-side shared area by all php files within the application (www.one.net). One program puts some data and others can read them any time. This is a way of communication between php programswithin the **same domain**. "a.php" puts a data and "b.php" can read it. In this way, all files can communicate with each other by the shared storage area (cookie)

HTTP Cookies Example



Cookies in HTTP

cookies

request login form

login.php

login.php gets username and password, checks them, and if they are correct, save a cookie in the user's browser

user=Mehmet

Response Headers view source
Connection: Keep-Alive
Content-Length: 0
Content-Type: text/html; charset=UTF-8
Date: Mon, 05 Dec 2016 13:02:15 GMT
Keep-Alive: timeout=5, max=100
Location: main.php
Server: Apache/2.4.23 (Win64) PHP/5.6.25
Set-Cookie: user=Mehmet; expires=Mon, 05-Dec-2016 13:18:55 GMT; Max-Age=1000

Browser sees a cookie in response header, and save it in its storage.

user=Mehmet

Request Headers view source
Accept: text/html,application/xhtml+xml,a
Accept-Encoding: gzip, deflate, sdch, br
Accept-Language: en-US,en;q=0.8,tr;q=0.6
Cache-Control: max-age=0
Connection: keep-alive
Cookie: user=Mehmet

main.php

main.php checks the cookies and use them appropriately. It depends on application logic.

Browser sends cookie data to all files in the same domain through Request header.

Creating/Updating Cookies

- ***bool setcookie(name, value [, expire date][,path][,domain][,security][,httpOnly]);***
- **name:** cookie name, **value**: cookie value that is stored in URLEncoded format.
- **expire date** : the time the cookie expires. This is a Unix timestamp so is in number of seconds since the epoch. If set to 0,or omitted, the cookie will expire at the end of the session (when the browser closes). Note that, although some browsers close, they restore session cookies as if they were not closed.
- **path** : the path on the server in which the cookie will be available on. If set to '/', the cookie will be available within the entire domain. If set to '/foo/', the cookie will only be available withing the /foo/ directory and all sub-directories such as /foo/bar/ of domain. The default value is the current directory that the cookie is being set.
- **domain**: the (sub)domain that the cookie is available to. Setting this to a subdomain such as *www.one.net*, it is valid only for *www.one.net* and its subdomains *test.www.one.net* but not *docs.one.net*. However, if you set to *one.net*, it means all subdomain of one.net.
- **secure**: indicates that the cookie should only be transmitted over a secure HTTPS connection from the client.
- **httponly**: when TRUE, it won't be accessible by scripting languages such as javascript. This can effectively reduce XSS attacks.
- **Important Note:** setcookie modifies response header like header() command, remember that header() and setcookie() must be called before any actual output is sent. Otherwise, you get "Warning: Cannot modify header Information" error. However, in php.ini file, if you set "output_buffering = On", PHP does not send any output before script finishes. You can use setcookie anywhere in your script.

```php
setcookie("email", "ali@hotmail.com") ; // Session Cookie
// Persistent Cookie, it lasts two hours in the browser.
setcookie("email", "ali@hotmail.com", time() + 60*60*2) ;
//Cookie with a path, sent only to files under app folder.
setcookie("email", "ali@hotmail.com", time()+7200, "/php/");
//Sent only to files under "www.one.net" domain but not "app.one.net"
setcookie("email", "ali@hotmail.com", time()+7200, "/php/", "www.one.net");
// Sent only with HTTPS protocol but not with HTTP
setcookie("email", "ali@hotmail.com", time()+7200, "/php/", "www.one.net", TRUE);
// Javascript cannot access the cookie using "document.cookie"
setcookie("email", "ali@hotmail.com", time()+7200, "/php/", "www.one.net", TRUE, TRUE);
```

Deleting and Reading Cookies

- If the expire date of a cookie is a date from the past, the browser deletes the cookie.
- The php interpreter automatically gets the cookies from the request HTTP header and populates within $_COOKIE array.
- Use $_**COOKIE** array to **read** the cookies stored in the browser. $_COOKIE array can not be used to update a cookie, it is just for reading. To set or modify a cookie, use **setcookie** function.
- Session cookies (memory cookies) don't have expire dates, therefore, to delete a session cookie, use setcookie function together with a passed time and the path of the cookie. For example, setcookie('id', '', 1, '/')

Deleting a cookie :

```php
// Delete a cookie
setcookie('email', '' , time() - 3600 ) ;
// OR
setcookie('email', '' , 1 ) ;
```

Reading a cookie :

```php
// Read email cookie if it is available.
$email = '';
if ( isset( $_COOKIE['email'])) {
    $email = $_COOKIE['email'] ;
}
```

Reading all cookies :

```php
// Traverse cookies
foreach( $_COOKIE as $name => $value) {
    print "<p>Cookie : $name = $value</p>";
}

// Get the number of cookies
$num = count( $_COOKIE ) ;
```

addCookies.php

```php
<?php

  $product = array("name" => "Sony TV", "price" => 1200) ;
  setcookie("product", serialize($product) , time() + 60 * 60* 24 );
  header("Location: index.php?add") ;
```

deleteCookies.php

```php
<?php

  if ( isset($_COOKIE["product"])) {
      setcookie("product", '', 1);
  }
  header("Location: index.php") ;
```

showCookies.php

```php
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title></title>
    </head>
    <body>
        <?php

        if ( isset($_COOKIE["product"])) {

            $product = unserialize($_COOKIE["product"]) ;

            foreach( $product as $k => $v) {
                print  "<p> $k : $v</p>" ;
            }
        } else {
            print "<p>No Cookies to view yet</p>";
        }

        ?>
        <p><a href="index.php">Back to Main</a></p>
    </body>
</html>
```

serialize() converts an object to string representation.

unserialize() reconstruct the object from the string.

Tricks of cookies

```php
<?php
// IMPORTANT :
// Cookie value is created/updated in Response packet.
// While php is executing,
// the cookie value in the browser does not change at once.
// After execution finishes, browser gets cookie values.

// Question :
// What is the output of "echo $counter" at the first run?

$counter = isset($_COOKIE['counter']) ? $_COOKIE['counter'] : 0 ;

setcookie('counter' , $counter + 1 ) ;

$counter = isset($_COOKIE['counter']) ? $_COOKIE['counter'] : 0 ;

echo $counter ;

// The result is 0. Because, the value in setcookie function is sent after
// echo $counter, i.e. after the script execution finishes.
```

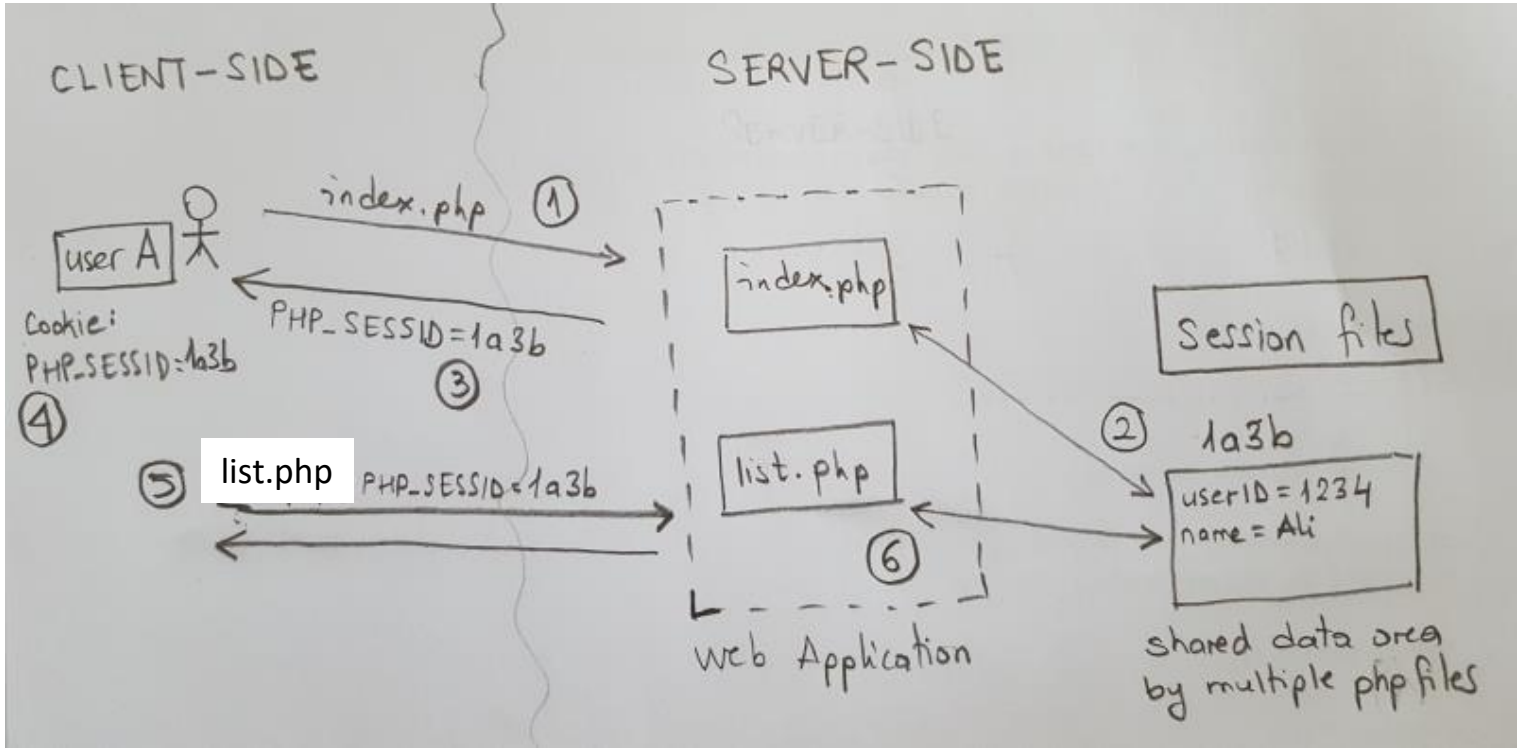
About Session

- A session is the time period during the visit of a user. Session starts when the user visits the application and ends when she closes the browser.
- The application can store data in this time period and multiple php pages in the same application can communicate each other by reading and writing data to/from a persistent storage.
- Session also refers the name for the storage area shared by all php programs within the application. It is a persistent storage for multiple pages during session time period.
- Session data are stored in a temporary file on the server-side. However, cookie data are stored in the client-side. "session.save_path" in "php.ini" file sets the directory of session files.
- Session data are more secure than cookie since users can not access session data directly.
- Session data are limited by the lifetime of the browser unlike cookie data.

Session Mechanism

1. "**User A**" requests "*index.php*"
2. "*index.php*" starts a new session, that is, it randomly generates a 32 digits hexadecimal number (7eu31i6fsci9dgihqcflf0ko6tkar909) called session id to differentiate users, creates a file with the same name, sess_7eu31i6fsci9dgihqcflf0ko6tkar909.
3. "index.php" sends the session id as a cookie called **PHPSESSID** to store in client side.
4. Browser stores a memory cookie (**PHPSESSID**) in its memory.
5. In all subsequent requests to the php files in the same domain, Browser sends **PHPSESSID** to the php file.
6. Php file in the same domain gets the session id as a cookie data, and accesses the file related to the "user A".

Session file is a way to share data among php files in the application. After the user exits from the browser, all memory cookies (**PHPSESSID**) are removed from the client's browser memory. Logically, previous session data are invalidated. Server deletes session files that are not accessed in a certain period of time, i.e, 30 minutes or one month.



Start/Resume a Session

- session_start() : to create or resume a session with a session id passed by via GET, POST or **Cookie**

function session_start() : (default execution)

```
    if ( !isset($_COOKIE['PHPSESSID']) ) {
        $sid = create a new unique session id for the user
        send $sid with a memory cookie PHPSESSID to browser
        set $_SESSION to an empty array
    } else {
        $sid = $_COOKIE['PHPSESSID']
        $_SESSION = session_load()  // load from session file
    }
```

```
function session_load() {
    $sessid = $_COOKIE["PHPSESSID"] ;
    $fp = fopen( "sess_$sessid", "r") ;
    $data = fgets($fp) ;
    $_SESSION = unserialize($data) ;
}
```

registers a callback function (**session_save**) to save $_SESSION into the session file.

At the end of the php execution, **php engine calls save_session()**

```
function session_save() {
    $data = serialize( $_SESSION) ;
    $sessid = $_COOKIE["PHPSESSID"] ;
    $fp = fopen("sess_$sessid", "w") ;
    fwrite($fp, $data) ;
}
```

CREATE SESSION DATA

```php
<?php

session_start() ;

// $_SESSION array created.

$_SESSION["login"] = ["name" => "ali", "surname" => "gul", "id" => 1232124] ;

// Shopping Cart
$_SESSION["shopcart"][34523] = ["product" => "USB Storage" ,  "price" => 50] ;
$_SESSION["shopcart"][65345] = ["product" => "HP Printer" ,  "price" => 280] ;

// redirect to index.php
header("Location: index.php") ;
```

READ SESSION DATA

```php
<?php
session_start();

// All session data are stored in $_SESSION super global array

if ( isset($_SESSION["login"])) {
    print "<p>Name : {$_SESSION["login"]["name"]}";
    print "Surname : {$_SESSION["login"]["surname"]}";
    print "id : {$_SESSION["login"]["id"]} </p>" ;
}

// display all data withing shoppingcart
if ( isset($_SESSION["shopcart"])) {
    echo "<p>Total product : " . count($_SESSION["shopcart"]) . "</p>";
    foreach( $_SESSION["shopcart"] as $id => $item) {
      print "<p>ID : $id ";
      print "Product : {$item["product"]} ";
      print "Price : {$item["price"]}</p>";
    }
}

print "<p><a href='index.php'>Back to Main</a></p>";
```

UPDATE & **DELETE**

```php
<?php

session_start() ;

// Update a session data.
if ( isset($_SESSION["login"])) {
  $_SESSION["login"]["id"] = 999999 ;
}

// Delete session data. (delete all shopping cart data)
if ( isset($_SESSION['shopcart'])) {
    unset($_SESSION['shopcart']);
}

header("Location: index.php") ;
```

Attention

$_SESSION[index]
index cannot be numeric.
$_SESSION[0] not valid.
$_SESSION["car"][0] is ok.

DESTROY/LOGOUT

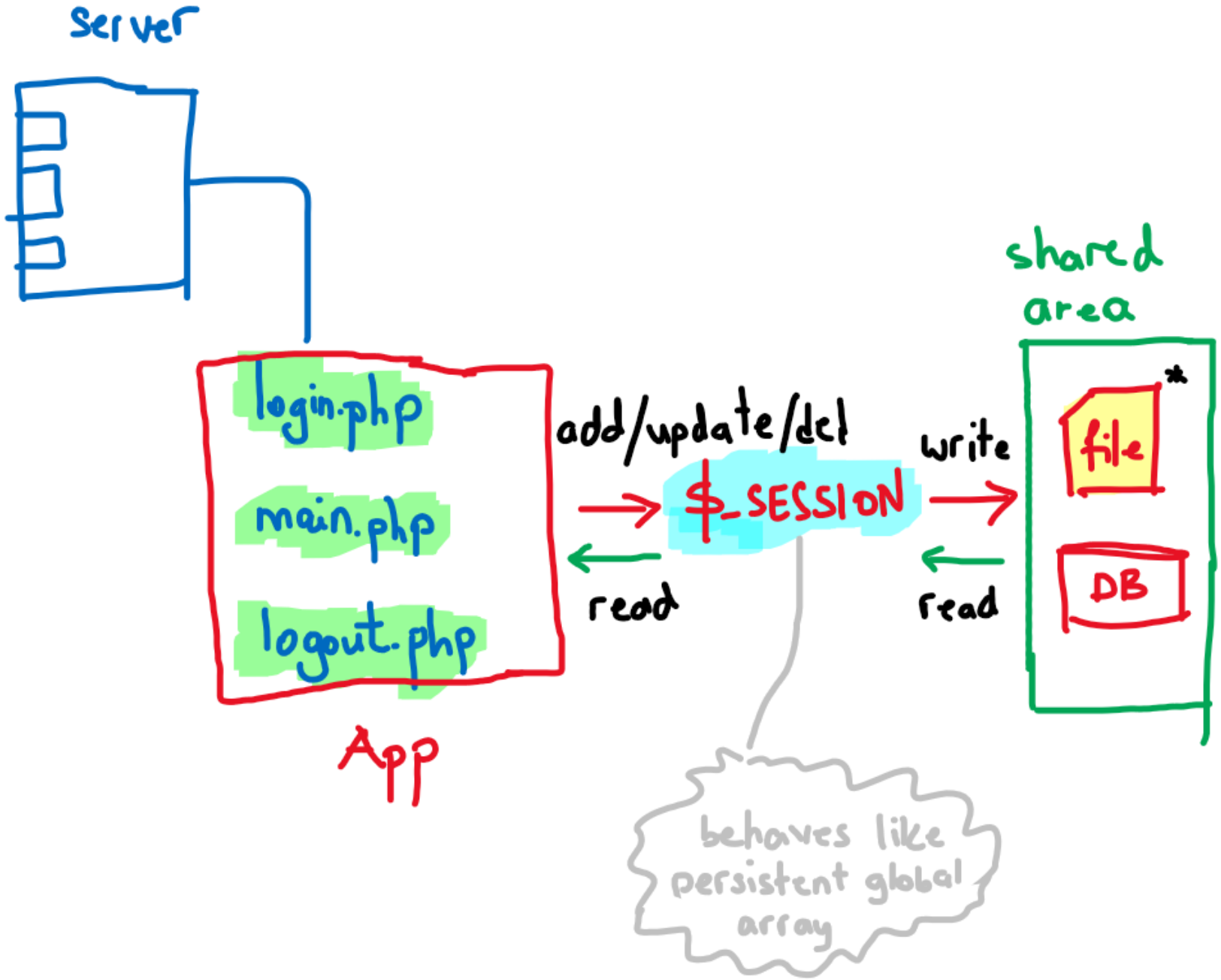
```php
<?php

session_start();

// delete all data in the session array.
// session id still valid.
$_SESSION = [] ;

// delete PHPSESSID memory cookie
// session_name() returns the cookie name; by default it is PHPSESSID
setcookie(session_name(), '', 1, '/');  // delete PHPSESSID Cookie
session_destroy();  // delete session file

header("Location: index.php") ;
```


server
login.php
main.php
logout.php
App
add/update/del
$_SESSION
read
write
read
shared area
file
*
DB
behaves like persistent global array