Univ. Babeș-Bolyai,

Facultatea de Matematică și Informatică

Lect. dr. Darius Bufnea

Notițe de curs Programare Web: PHP

Ce este PHP?

Este o tehnologie server side / limbaj de programare open-source de back-end – care se execută la nivelul serverului web. PHP este un limbaj de scripting (nu este compilat), este interpretat, numele său fiind abreviere de la "**P**HP: **H**ypertext **P**reprocessor". Ca să poată executa cod PHP, serverul web trebuie să fie dotat cu un modul care să permită execuția codului PHP. Apache (serverul web open source dominant pe platformele UNIX/Linux) suportă oarecum nativ execuția de cod PHP (dar și în cadrul Apache execuția de cod PHP poate fi dezactivată). Celalalt server web dominant din Internet , IIS de la Microsoft (Internet Information Service) suportă și el execuția de cod PHP prin intermediul unor module 3rd party (dar oarecum integrarea PHP-ului cu IIS-ul nu este la fel de naturală și firească precum legătura dintre Apache și PHP). Ca și concluzie, don't try to run PHP scripts using IIS…

Pe de altă parte, PHP-ul poate fi folosit și ca limbaj de scripting linia de comandă, existand un interpretor executabil numit chiar php (php.exe în Windows) care permite rularea de programe scris în PHP și în afara enviromement-ului oferit de un server Web.

AMP Stack sau stiva AMP

AMP este abreviere de la Apache, Mysql si PHP. De obicei o instalare de PHP nu vine de una singură, ea vine la pachet cu un server web, și după cum spuneam mai sus, cel mai natural se împacă cu Apache (open-source și el). Pe lângă Apache și PHP, în acest environment era nevoie și de un server de baze de date, MySQL fiind una dintre soluțiile open-source din domeniul serverelor de date care s-a bucurat de cel mai mare success, împreună cu serverul web Apache și cu PHP-ul formând o "stivă" de programare în domeniul aplicațiilor web (sau un ecosistem) numit AMP. Uneori îl veți regăsi referit și sub numele de LAMP stack (L venind de la Linux). Paranteză: MySQL dupa ce a ajuns în ograda Oracle a suferit un „fork", născându-se MariaDB (practic tot MySQL). PHP însa suportă și alte servere de date precum PostgreSQL (open source și el), SQL Server de la Microsoft sau Oracle.

Instalare

De ce avem nevoie ca să rulam cod PHP? În primul rând de un server web care să permită execuția de cod PHP, și după cum spuneam mai sus cel mai la îndemână este Apache. În funcție de sistemul de operare pe care vrem să facem instalarea:

Linux: majoritatea distribuțiilor Linux vin sau oferă spre download dintr-un repository atât serverul web Apache cât și modulele necesare ca acesta să fie capabil să ruleze cod PHP. De asemenea, cam toate distribuțiile Linux vin cu MySQL sau MariaDB ca server de baze de date implicit. În funcție de distribuție, aceste pachete se pot instala fie cu apt-get (Ubuntu based distributions) sau cu yum (Redhat/Fedora/CentOS based distributions).

MacOS: not a fan...

Windows: Windows nu are o tradiție prea bună / nu este cel mai bun enviroment posibil pentru ecosisteme open-source precum este stiva AMP. Cu toate acestea, există câteva distribuții bune ale acestei stive pe Windows, una dintre ele fiind XAMPP (Apache + MariaDB + PHP + Perl) pe care vă recomand să o instalați. Practic ce conține XAMPP? E un kit de instalare care instalează pe Windows un server web Apache cu suport de PHP integrat și un server de baze de date MySQL împreună cu o fereastră de control (Control Panel) care permite printre altele gestionarea facilă (oprirea / pornirea) acestor două servere.

Observație (pentru că am văzut mai mult de o dată această greșeală): nu căutați „XAMPP for Linux", nu există așa ceva! Distribuțiile Linux suportă nativ stiva AMP, având pachete dedicate pentru Apache, PHP și MySQL.

Înainte de a vă instala XAMPP

Este foarte posibil să aveți din alte surse gata instalate alte servere web sau să aveți deja MySQL instalat (l-ați instalat pentru MPP). Dacă instalați XAMPP, e posibil să vă loviți de tot felul de conflicte: MySQL din XAMPP nu poate ocupa portul 3306 (portul implicit al MySQL/MariaDB) pentru că acest port este deja ocupat, aveți deja un IIS care vă ocupa portul 80 pe care vrea sa îl ocupe și Apache-ul, portul 80 mai este ocupat uneori și de Skype sau diverși antiviruși.

Daca rulați într-un `cmd` un

```
netstat -aon
```

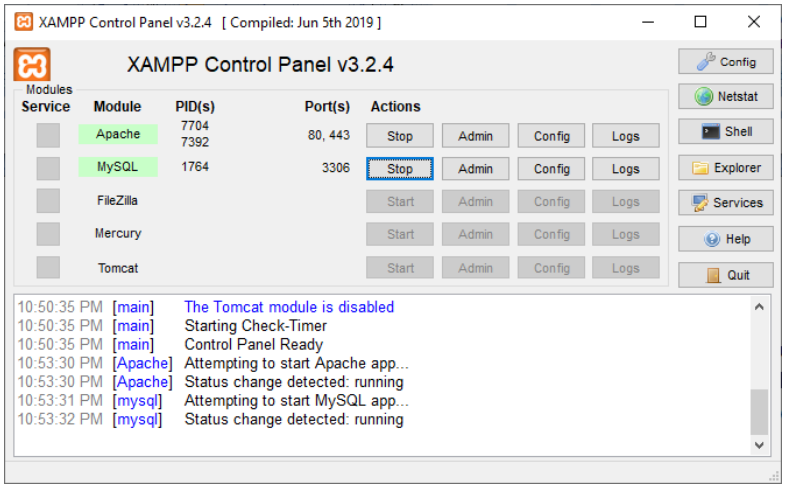
puteți vedea care porturi TCP sunt ocupate și de către ce proces (PID-ul procesului care ocupă un anumit port). Ulterior dacă va uitați în Task Managerul de Windows puteți vedea și identifica procesul cu un anumit PID.

O altă problema este ca multe IDE-uri (Integrated Development Environment - editoarele în care scrieți cod gen Eclipse, Netbeans, IteliJ, PHP Webstorm, Visual Studio) au „prostul obicei" - pentru a face munca web developer-ului mai ușoară - să pornească ele diverse servicii în spate cum ar fi un server web pentru a rula/depana un anumit proiect. Aceste servicii și porturile ocupate de aceste servicii pot intra iarăși în conflict cu diverse servere / pachete instalate de voi anterior.

Dupa instalarea XAMPP, ar trebui din Control Panel-ul acestuia să puteți porni cu succes atât Apache-ul cât și serverul MySQL/MariaDB. Dacă unul dintre ele nu pornește, trebuie să eliberați unul din porturile 80, 443 (HTTPS) sau 3306. Pentru aceasta:

- Verificați folosind comanda `netstat -aon` de mai sus dacă porturile sunt intr-adevăr ocupate și de către ce proces;

- Dacă puteți identifica ușor folosind comanda `netstat -aon` și Task Managerul de Windows procesul care ocupă portul/porturile problematice, opriți procesul respectiv.
- Verificați în Windows Services (click pe Start, scrieți Services, ENTER) că nu aveți un serviciu cu un nume „like" MySQL, IIS, Internet Information Server, Apache sau similar în starea Running (foarte probabil dintr-o instalare similară) și dacă identificați un astfel de proces opriți-l.



Observație: foarte mulți studenți își instalează pentru MPP un server MySQL care rulează tot timpul ca serviciu în background după pornirea laptopului consumând resurse inutile (la fel un SQL server pentru altă materie și un Oracle Express Edition pentru altă materie). Personal prefer serverul MySQL/MariaDB din XAMPP tocmai pentru flexibilitatea pe care mi-o oferă de al controla/porni/opri mai facilă și mai transparentă. Într-un scenariu în care aveti 2 servere MySQL instalate, trebuie să aveți grijă care server MySQL e pornit, sunt dese cazurile în care este pornit unul dintre ele și vreți să vă conectați la el cu anumite credențiale cu care de fapt ar trebuie să vă conectați la celalalt. Numele de utilizator implicit cu care vă conectați la serverul MySQL/MariaDB din XAMPP este root, fără parolă (parola vidă "").

Dacă instalarea XAMPP s-a făcut cu succes, ar trebui să puteți încarcă în browser următoarele URL-uri:

http://localhost

http://localhost/phpmyadmin - PhpMyAdmin este o aplicatie web scrisă în PHP care vă permite administrarea mai facilă a unui server de baze de date (MySQL/Maria DB în cazul nostru).

În caz că nu puteți elibera totuși porturile pentru a porni unul dintre servere, atât Apache cât și MySQL/MariaDB vă permit schimbarea porturilor pe care ascultă aceste servere. Schimbarea porturilor ar trebuie să fie însă ultima soluție pentru că ridică ulterior din nou alte probleme de conectare din partea clienților care vor să se conecteze la aceste servere ce folosesc porturi nestandard. Spre exemplu, conectarea din browser la un server web Apache care așteaptă pe portul 81 (nu pe portul standard 80 al protocolului HTTP) ar trebui să se facă cu un URL de forma: http://localhost:81. Schimbarea portului în Apache se face folosind directiva `Listen` din cadrul fișierului său de configurare `httpd.conf`.

Daca ați instalat XAMPP într-un director de forma C:\xampp, veți identifica următoarele două directoare:

```
c:\xampp\php\
```
și

`c:\xampp\mysql\bin\`

pe care vă recomand să le adăugați în variabila de mediu PATH pentru a avea acces la interpretorul php.exe și la clientul linie de comandă mysql.exe de oriunde (din orice director) din cadrul sistemului de fișiere în cadrul unui cmd.

Temă: vă recomand după parcurgerea acestui material să vă „jucați" în linia de comandă cu comenzile php.exe, mysql.exe si mysqldump.exe.

Apache-ul ca server web conține un director denumit `DocumentRoot` (acesta se specifică cu ajutorul unei directive denumite chiar `DocumentRoot` prezentă in fișierul de configurare `httpd.conf`) de unde vor fi „livrate" clienților (browserelor) toate fișierele cerute de aceștia. `DocumentRoot`-ul implicit pentru XAMPP este `c:\xampp\htdocs`, practic acest director corespunzând URL-ului rădăcină http://localhost, un fișier de `forma c:\xampp\htdocs\demo\lab1.php` fiind accesibil cu un URL de forma http://localhost/demo/lab1.php.

Personal recomand editarea și rularea fișierelor .php direct dintr-un subdirector al `DocumentRoot`-ului serverului web, după modificarea acestora fiind necesar un refresh / reload (CTRL-F5) în browser. Eventualele erori de execuție sunt raportare de serverul web într-un fișier denumit de obicei `error.log` sau `error_log` util a fi conspectat pentru un debugging mai facil. Ca alternativă, unele IDE-uri specializate oferă mecanisme avansate de debugging, depanare pas cu pas, la rularea de cod PHP într-un astfel de IDE fiind posibil ce acesta din urmă să își pornească propriul server web pe un port nestandard.

Observații:

Serverul vostru Apache din cadrul XAMPP este accesibil și de pe alte calculatoare din cadrul retelei locale cu un URL de forma: http://192.168.1.101 dacă acest lucru este permis de firewall-ul sistemului vostru de operare (a se înlocui cu IP-ul vostru privat din rețeaua locală) sau chiar de oriunde din Internet dacă faceți un DNAT la portul 80 de pe router (recapitulare de la Rețele...).

PHP

Codul php trebuie plasat în interiorul unui fișier cu extensia .php. Un astfel de fișier este executat la nivelul serverului web, execuția unui fișier php fiind în general invocată prin intermediul unui request HTTP sau de către un alt fișier PHP prin diferite mecanisme de includere.

Observație: Frecvent studenții deschid un fișier php direct in browser folosind un URL de tip file:// (spre exemplu file://d:/laboratoare/pw/php/pr1.php). ACEST LUCRU ESTE GREȘIT, într-o asemenea situație, fișierul php nu se execută – nu are cine să-l execute – pentru că serverul web care l-ar putea executa nu este specificat. Un fișier php trebuie întotdeauna „invocat" (apelat, referit) prin intermediul protocolului HTTP folosind un URL de forma http://numedeserver.com/demo.php. În acest exemplu, serverul web numedeserver.com este responsabil să găzduiască și să execute fișierul cu numele demo.php. Excepție de la această observație: rularea unui script php din linia de comandă folosind interpretorul php.

Un fișier .php poate conține fie exclusiv cod php, fie cod php imbricat cu cod html. O secvență de cod php din cadrul unui fișier .php poartă denumirea de scriptlet, fiind relativ frecvente situațiile în care în

cadrul unui fișier .php sunt prezente mai multe scriptlet-uri imbricate cu cod html (imbricate la rândul lor cu cod JavaScript care se execută pe client – și așa se ajunge la ceea ce se cheamă Spaghetti Code... de evitat ☺). Un scriptlet este delimitat de tag-ul `<?php ?>` (atenție, tag exclusiv PHP, interpretabil doar server side).

Un prim exemplu (accesibil aici):

```
<!DOCTYPE html>
<html>
<head>
    <title>Exemplul 1</title>
</head>
<body>
    Salut lume. Ora pe server este: <?php echo date('H:i', time()); ?>
</body>
</html>
```

Este util să vizualizați și codul sursă al paginii ce ajunge la browser (click dreapta, view source). Scriptlet-ul php plasat în interiorul tag-ului `<?php ?>` se execută pe back-end, iar la browser ajunge cod HTML „curat" – browser-ul nu este conștient de fapt că ceva se execută pe back-end, el cere prin intermediul protocolului HTTP și a unei metode HTTP precum GET sau POST o anumită resursă specificată prin intermediul URL-ului, resursa în cazul de față fiind fișierul .php a cărui execuție are loc la nivelul serverului web.

Observație: PHP-ul are un fișier de configurare denumit `php.ini` unde există diferite directive de configurare. Una dintre acestea, denumită `short_open_tag` (`= On` sau `= Off`) permite delimitarea scriptlet-urilor folosind așa numitul short tag `<? ?>`. Această abordare nu este recomandată, întrucât proiectul vostru php este posibil să ajungă să fie găzduit și rulat în cadrul unui server web care nu acceptă (și practic nu va recunoaște) short open tag-ul `<?`. În cazul modificării fișierelor `php.ini` sau `httpd.conf` trebuie sa reporniți serverul web Apache.

Daca un scriptlet se găsește la finalul unui fișier PHP, nemaiexistând cod HTML după acesta, marcajul de sfârșit de scriptlet (`?>`) poate să lipsească (altfel spus, dacă marcajul de sfârșit de scriptlet `?>` se regăsește la final în cadrul unui fișier PHP, acesta poate fi omis).

Elemente de sintaxă

Limbajul PHP a împrumutat elemente de sintaxă atât de la limbajul C cât și de la limbajul shell al interpretorului de comenzi SH (sau BASH) din Unix/Linux. Nu vom insista foarte multe pe elementele de sintaxă ale limbajului, prezentul material dorind să insiste mai multe pe aspectele „web related" oferite de limbaj. Vă rog însă ferm ca pe lângă prezentul material să parcurgeți și secțiunea PHP Tutorial de pe W3Schools, întreg limbajul și API-ul oferit de acesta fiind acoperit și de documentația oficială disponibilă aici.

Câteva elemente de sintaxă pe scurt:

- Instrucțiunile de control (`if`, `while`, `for`) sunt identice cu cele din C/C++/Java
- Fiecare linie de cod PHP trebuie să se termine cu caracterul „;"
- Afișarea de cod HTML din interiorul unui scriptlet PHP se poate face cu print sau echo

- Elementele de sintaxă nu sunt case sensitive, dar numele variabilelor sunt;
- Comentariile se introduc în maniera C/C++/JAVA:
  - Single line: // sau # (prealuat din sh)
  - Multiline /* … */

Limbajul php suporta și un tag de scriptlet special `<?= ?>` pentru afișarea de pe back-end de valori „inline" în cadrul codului HTML ce se trimite spre front-end. Folosind acest tag, linia de cod anterioară:

```
Salut lume. Ora pe server este: <?php echo date('H:i', time()); ?>
```

poate fi rescrisă:

```
Salut lume. Ora pe server este: <?= date('H:i', time()) ?>
```

Variabile in PHP

- Toate variabilele în PHP încep cu simbolul $, exemplu: `$varsta = 18;` (precedarea variabilelor cu caracterul $ este un element de sintaxă preluat de la limbajul SHELL, numele variabilelor de sistem UNIX/Linux fiind precedate și ele de caracterul $).
- Variabilele nu trebuie declarate;
- Tipul acestora se deduce în funcție de valoarea asociată, PHP fiind un limbaj weakly typed;
- Numele variabilei (identificatorul acesteia) trebuie să urmeze aceleași reguli ca în limbajele de programare cunoscute.

Stringuri in PHP

- Pot fi delimitate atât de "" (ghilimele) cât și de '' (apostroafe);
- Pentru concatenarea acestora se foloseste operatorul "." Exemplu: `$suma = "In cont am " . 3 . " lei";`
- Pentru compararea șirurilor se poate folosi == dar și `strcmp`;
- Majoritatea funcțiilor de lucru cu șiruri sunt preluate din limbajul C: `strlen, strstr, strpos, strtok`;

Observație: PHP suporta și el operatorul `===` specific limbajelor weakly typed. Spre exemplu, funcția `strpos($sir, $subsir)` verifică apariția `$subsir`-ului în `$sir`, returnând în caz afirmativ indexul primei apariții (valoare numerică) sau `FALSE` (boolean în PHP) dacă `$subsir` nu se regăsește în `$sir`. O comparație de forma `if (strpos($sir, $subsir) == FALSE)` nu este corectă dacă se dorește verificarea apariției `$subsir`-ului în `$sir`, această condiție având valoarea de adevăr `TRUE` și dacă `$subsir`-ul nu se `regăsește` în `$sir` și dacă acesta se regăsește la începutul `$sir`-ului (index 0) – acest lucru pentru că boolean-ul `FALSE` se evaluează la valoarea 0. Corectă în cazul de față este folosirea operatorului `===` care compară și tipul valorii întoarse de funcția `strpos`.

Tablouri în PHP

PHP-ul suportă atât tablouri clasice cu indici numerici, cât și tablouri asociative (un fel de map-uri) cu indicii string-uri. Exemple:

Tablouri unidimensionale (tablouri clasice cu indici valori întregi):

```
<?php
$echipe = array("CFR", "FCSB", "Dinamo");
print $echipe[0];
print count($echipe);
?>
```

Tablouri cu indici de tip string (asociative):

```
<?php
$capitale["Romania"] = "Bucuresti";
$capitale["Ungaria"] = "Budapesta";
$capitale["Franta"] = "Paris";
print $capitale["Romania"];
?>
```

Pentru iterarea unui tablou se poate folosi instrucțiunea de control `foreach`. Exemple:

```
<?php
foreach ($capitale as $capitala)
       print $capitala . "<br>\n";
foreach ($capitale as $tara=>$capitala)
       print $tara . " are capitala " . $capitala . "<br>\n";
?>
```

Observație: în exemplu de mai sus delimitatorul `\n` a fost folosit pentru a delimita liniile de cod sursă HTML care se trimite browser-ului în urma execuției codului PHP sau într-un eventual output pe consolă la execuția codului folosind interpretorul php la linia de comanda, în timp ce tagul `<br>` a fost folosit pentru a delimita efectiv liniile de output afișate de browser utilizatorului.

Limbajul PHP suporta și tablouri multidimensionale:

```
<?php
   $continente = array(
      "Europa"=>array("Romania", "Franta", "Ungaria"),
      "Asia"=>array("China", "India", "Japonia"),
      "Africa"=>array("Egipt", "Maroc", "Nigeria")
   );
   print $continente["Asia"][2];
?>
```

Funcții PHP

Declararea funcțiilor în PHP se face folosind cuvântul rezervat `function` asemănător modului în care se face declararea acestora în JavaScript. Parametrii se pot transmite fie prin valoare (implicit) identic modului în care se transmit în C/C++/Java, fie prin referință - pentru transmiterea parametrilor prin referință parametri formali se prefixează cu operatorul & (asemănător modului în care se transmit prin referință în C++).

Exemplu:

```
function aduna($i, $j) {
  return $i + $j;
}
print aduna(4, 7);
```

Observație: Este deosebit de frecvent ca o funcție în php să returneze valori de tipuri diferite. Este recomandată întotdeauna și verificarea tipului valorii returnate de funcție nu doar a valorii. Spre exemplu, funcția `strpos` amintită mai sus poate întoarce atât un index numeric cât și valoarea de adevăr `FALSE`. Un alt exemplu, diferite funcții de lucru cu baze de date, întorc fie o resursă, fie boolean-ul FALSE în caz de eroare.

Alte elemente de limbaj

Regula "celor trei pahare " într-o singura linie de cod:

```
list($a, $b) = array($b, $a);
```

Observație: `list` nu este o funcție, ci un element oferit de limbajul PHP care permite atribuirea de valori mai multor variabile folosind un singur operator de atribuire.

Variabile globale

Variabilele declarate în afara corpului unei funcții nu sunt accesibile implicit în interiorul acesteia. Pentru a avea acces la ele, acestea trebuie declarate ca globale folosind cuvântul rezervat `global`. Exemplu:

```
<?php
$pi = 3.14;
function circleArea($radix) {
    global $pi;
    return $pi * $radix * $radix;
}
print circleArea(5);
?>
```

Variabile predefinite PHP

PHP-ul oferă din start câteva tablouri asociative predefinite, majoritatea fiind legate de contextul web în care se execută codul PHP. Aceste variabile se mai numesc și superglobale (superglobals), ele fiind accesibile oriunde la nivelul codului PHP fără a mai fi declarate ca globale. Acestea sunt:

- `$GLOBALS`
- `$_SERVER`
- `$_GET`
- `$_POST`
- `$_FILES`
- `$_COOKIE`

- $_SESSION
- $_REQUEST
- $_ENV

cele mai importante fiind $GLOBALS, $_GET, $_POST și $_SESSION. Observație: numele tablourilor sunt GLOBALS, _GET, _POST și _SESSION, ele fiind precedate de $ precum orice nume de variabila în PHP.

Tabloul predefinit GLOBALS permite accesarea tuturor variabilelor globale. Astfel, folosind GLOBALS, funcția precedentă care calculează suprafața cercului poate fi rescrisa altfel:

```
<?php
$pi = 3.14;
function circleArea($radix) {
    return $GLOBALS["pi"] * $radix * $radix;
}
print circleArea(5);
?>
```

Despre $_GET, $_POST și $_SESSION vom vorbi în continuare în acest material.

Prelucrarea datelor din cadrul unui formular

Din cursurile precedente știm că datele din cadrul unui formular pot fi trimise pe back-end folosind una dintre metodele GET sau POST, iar script-ul de pe back-end care primește datele de la formular se specifică ca și valoare pentru atributul action al formularului. Elemente de tip input din cadrul unui formular (împreună cu elementele de tip textarea și option) vor fi accesibile în PHP fie prin intermediul tabloului asociativ $_GET fie $_POST în funcție de metoda prin care s-a făcut submit la formularul respectiv. În cadrul acestor tablouri, indecșii valorilor stocate vor fi numele inputurilor corespunzătoare din formular.

Prezentam mai jos un exemplu care se doreste a fi sugestiv în acest sens (disponibil aici). Front-end:

```
<form method="post" action="register.php">
Nume utilizator: <input type="text" name="username"><br>
E-mail: <input type="text" name="email"/><br>
Parola: <input type="password" name="pass"/><br>
Parola din nou: <input type="password" name="pass2"/><br>
<input type="Submit" value="Register">
</form>
```

Back-end (fișierul register.php):

```
Am primit de la browser:<br>
<?php
   print "Username: " . $_POST["username"] . "<br>\n";
   print "Email: " . $_POST["email"] . "<br>\n";
   print "Parola: " . $_POST["pass"] . "<br>\n";
   print "Parola2: " . $_POST["pass2"] . "<br>\n";
?>
```

```
</body>
```

Datele primite prin completarea formularului mai pot fi accesate și prin intermediul tabloului asociativ `$_REQUEST` (atât la submit-ul prin `GET` cat și la cel prin `POST`), iar `$_SERVER["REQUEST_METHOD"]` conține metoda HTTP prin care s-a realizat cererea.

IMPORTANT: Înainte de a fi folosite pe back-end, datele primite de la client, indiferent de metodă, trebuie VALIDATE OBLIGATORIU. Mai multe în acest sens mai târziu în acest material.

Accesarea bazelor de date din PHP

Din PHP pot fi accesate cele mai populare servere de date existente precum MySQL/MariaDB, SQL Server, Oracle sau PostgreSQL. Cum cea mai naturală „integrare" se face cu MySQL/MariaDB, exemplele din prezentul material se vor baza tot pe acest SGBD.

Există mai multe metode alternative de accesare din PHP a bazelor de date:

Metoda 1

Folosind familia de funcții prefixate de `mysql_` precum `mysql_connect` și `mysql_query`. Această metodă este DEPRECATED în PHP 7 și oricâte exemple ați găsi online folosind aceasta metodă vă rog sa nu o folosiți întrucât ridică serioase probleme de securitate (pe care le vom dezbate mai târziu în acest material când vom vorbi și de SQL Injection).

Metoda 2

Folosind noua familie de funcții prefixate de `mysqli_` (mysql "improved").

Pentru rularea exemplelor de față, este necesar să vă creați (folosind phpMyAdmin http://localhost/phpmyadmin/) o bază de date (spre exemplu „pw") și să importați în această bază de date tabela „trenuri" de la această adresă. Ca metodă alternativă la phpMyAdmin, dacă v-ați pus în variabila de mediu PATH directorul c:\xampp\mysql\bin\ ce conține clientul linie de comandă mysql.exe, atunci puteți face importul și în modul următor:

La linia de comanda cmd:

```
mysql.exe -u root -p
```

(se dă Enter, parola este vidă)

În cadrul liniei de comandă mysql:

```
create database pw;
exit
```

La linia de comanda cmd:

```
mysql -u root -p  < trenuri.sql
```

unde trenuri.sql e dump-ul SQL descărcat mai sus. Linia de mai sus rulează clientul mysql, intrarea standard pentru acesta fiind redirectată din fișierul `trenuri.sql` (comenzile SQL de executat sunt citite din fișierul `trenuri.sql`).

Pe această tabelă vom prezenta două exemple, unul va afișa toate trenurile, iar celălalt va adăuga un nou tren.

Fișierul viewTrains.php (disponibil în varianta online aici):

```
<table>
<tr><th>Plecare</th><th>Sosire</th><th>Ora</th><th>Minut</th></tr>
<?php

error_reporting(E_ERROR | E_PARSE);
$con= new mysqli('127.0.0.1', 'root', '', 'pw');
$con->set_charset("utf8");

$sql = "SELECT * FROM trenuri";
$result = $con->query($sql);
while ($row = $result->fetch_assoc()) {
    $plecare = $row['plecare'];
    $sosire = $row['sosire'];
    $ora = $row['ora'];
    $minut = $row['minut'];
    print
"\t<tr><td>$plecare</td><td>$sosire</td><td>$ora</td><td>$minut</td></tr>\n";
}

?>
</table>
```

Fișierul addTrain.html (disponibil în varianta online aici):

```
<form method="GET" action="insertTrain.php">
Plecare: <input type="text" name="plecare"><br>
Sosirel: <input type="text" name="sosire"/><br>
Ora: <input type="number" min="0" max="23" name="ora"/><br>
Minut: <input type="number" min="0" max="59" name="minut"/><br><br>
<input type="Submit" value="Adauga tren">
</form>
```

Fișierul insertTrain.php:

```
<?php
error_reporting(E_ERROR | E_PARSE);
$con = new mysqli('127.0.0.1', 'root', '', 'pw');
$con->set_charset("utf8");

$plecare = addslashes(htmlentities($_GET["plecare"], ENT_COMPAT, "UTF-8"));
$sosire = addslashes(htmlentities($_GET["sosire"], ENT_COMPAT, "UTF-8"));
$ora = addslashes(htmlentities($_GET["ora"], ENT_COMPAT, "UTF-8"));
$minut = addslashes(htmlentities($_GET["minut"], ENT_COMPAT, "UTF-8"));

$stmt = $con->prepare("INSERT INTO trenuri (plecare, sosire, ora, minut)
VALUES (?, ?, ?, ?)");

$stmt->bind_param("ssii", $plecare, $sosire, $ora, $minut);
if ($stmt->execute() === FALSE)
    print "Eroare!<br>";
else
```

```
    print "Trenul a fost adaugat cu succes.<br>";
?>
```

Observații:

- Pentru rularea exemplului pe laptopul vostru (http://localhost), inlocuiți în constructorul conexiunii parametrii cu propriile date de conectare la baza de date, parola voastră (pentru serverul mysql din XAMPP userul implicit este root, cu parola vida "") și numele bazei de date create ("pw" daca ați rulat pașii de mai sus). Pentru a rula exemplele pe serverul web de la facultate (http://www.scs.ubbcluj.ro), studenții au de obicei o bază de date cu numele numele de utilizator, userul de acces fiind tot numele de utilizator și parola fiind tot numele de utilizator.
- Serverul de baze de date din exemplu este localhost („127.0.0.1") – acest lucru semnificând că din perspectiva serverului web care realizează conectarea la serverul de date, acesta din urmă rulează pe aceeași mașină ca și serverul web. În producție, sunt frecvent întâlnite situațiile când serverul de baze de date rulează pe o mașină diferită de serverul web. Dacă rulați exemplul pe serverul web de la facultate (http://www.scs.ubbcluj.ro), serverul de baze de date la care vă conectați este tot localhost („127.0.0.1") – adică aceeași masină din perspectiva serverului web. Ce sa nu încercați pentru că nu o sa meargă: să nu încercați să rulați exemplele php găzduite pe serverul vostru web local din XAMPP (http://localhost), iar din cod php rulat la voi local să vă conectați la serverul de baze de date de la facultate "scs.ubbcluj.ro" sau "www.scs.ubbcluj.ro". Deși teoretic acest lucru este posibil, nu puteți să vă conectați din motive de securitate.
- În fișierul insertTrains.php am făcut niște validări minimale (s-ar fi putut face mai multe) pentru a preîntâmpina diverse probleme de securitate precum SQL injection sau JavaSript injection (XSS = Cross Site Scripting). Vom aborda aceste aspecte mai târziu în acest material. Tot din perspectiva securității, folosim un fel de „PreparedStatement" (în terminologie Java) pentru a executa query-ul SQL care face inserarea noului tren în baza de date. Vom vedea mai târziu în acest material ce se poate întâmpla „rău" dacă nu validam datele primite de la formular, nu folosim PreparedStatement și nu facem bind la parametrii query-ului.
- Vă rog sa nu inserații trenuri cu prostii și cu localități din Croația :), în caz contrar voi fi nevoit să dau jos exemplul online.
- Pentru a reduce din codul redundant, partea comună de cod care realizează conectare la serverul de date poate fi plasată într-un fișier extern PHP care se poate include cu require_once atât în viewTrains.php cât și în insertTrain.php.

Metoda 3

A treia variantă de a accesa servere / baze de date din PHP este folosind PDO (PHP Data Objects). Spre deosebire de varianta precedenta, accesarea serverelor de date folosind PDO asigură independența față de serverul de date folosit (nu mai sunt folosite funcții dependente de un anume server de date) - în caz că se dorește schimbarea serverului de date fiind necesare doar modificări minimale la nivelul unui string de conectare.

Mai jos prezentăm exemplul anterior, rescris folosind PDO. Fișierul viewTrainsUsingPDO.php:

```
<table>
<tr><th>Plecare</th><th>Sosire</th><th>Ora</th><th>Minut</th></tr>
<?php
error_reporting(E_ERROR | E_PARSE);
```

```
$dsn = "mysql:host=localhost;dbname=pw";
$user = "root";
$passwd = "";

$pdo = new PDO($dsn, $user, $passwd);
$stm = $pdo->query("SELECT * FROM trenuri");
$rows = $stm->fetchAll(PDO::FETCH_ASSOC);

foreach($rows as $row) {
    $plecare = $row['plecare'];
    $sosire = $row['sosire'];
    $ora = $row['ora'];
    $minut = $row['minut'];
    print
"\t<tr><td>$plecare</td><td>$sosire</td><td>$ora</td><td>$minut</td></tr>\n";
}

?>
</table>
```

Fișierul insertTrainUsingPDO.php:

```
<?php
$dsn = "mysql:host=localhost;dbname=pw";
$user = "root";
$passwd = "";

$pdo = new PDO($dsn, $user, $passwd);

$plecare = addslashes(htmlentities($_GET["plecare"], ENT_COMPAT, "UTF-8"));
$sosire = addslashes(htmlentities($_GET["sosire"], ENT_COMPAT, "UTF-8"));
$ora = addslashes(htmlentities($_GET["ora"], ENT_COMPAT, "UTF-8"));
$minut = addslashes(htmlentities($_GET["minut"], ENT_COMPAT, "UTF-8"));

$sql = "INSERT INTO trenuri (plecare, sosire, ora, minut) VALUES (:plecare,
:sosire, :ora, :minut)";
$stmt = $pdo->prepare($sql);
$stmt->bindParam(':plecare', $plecare);
$stmt->bindParam(':sosire', $sosire);
$stmt->bindParam(':ora', $ora);
$stmt->bindParam(':minut', $minut);
if (! $stmt->execute())
    print "Eroare!<br>";
else
    print "Trenul a fost adaugat cu succes.<br>";
?>
```

Pentru mai multe detalii legate de PDO, găsiți mai multa teorie / exemple în documentația oficială PHP legată de subiect.

Alte metode de a accesa baze de date din PHP

La categoria alte metode amintim pe scurt:

- MDB2 din PHP Pear. PEAR - PHP Extension and Application Repository este o colecție de librării semioficiale care extind funcționalitatea de baza a limbajului PHP oferind un API mai bogat la

dispoziția programatorului. O astfel de librărie este MDB2 care este focusată pe accesul la servere de date.
- Folosirea de diverse ORM-uri (librării Object-relational mapping) care asigură persistența în baze de date. Un exemplu în acest sens este Doctrine.

Includerea unui alt fișier PHP în cadrul unui fișier PHP

Pentru o mai bună structurare a codului, evitarea codului redundant și a „spaghetti code", PHP permite folosirea următoarelor declarații pentru a include un fișier PHP în cadrul altui fișier PHP:

- `require_once`
- `require`
- `include_once`
- `include`

Diferența dintre `include` si `require` stă ca cele două declarații `require` vor genera o eroare fatală dacă nu pot include fișierul, pe când cele două declarații `include` vor genera doar un warning. Din punct de vedere al securității, este recomandat să folosiți declarații `require`. Diferența dintre declarațiile postfixate cu `_once` și celelalte este că cele postfixate vor include un fișier doar dacă nu a mai fost inclus, pe când cele doua declarații care nu folosesc _once nu verifică acest lucru. Sunt dese situațiile când un fișier PHP conține definiții de funcții sau clase. Dacă se folosește `require` sau `include` simplu și fișierul este inclus de două ori, se va genera o eroare fatală cum că o anumită declarație a avut loc deja. Concluzionând, în majoritatea cazurilor este recomandată utilizarea declarației `require_once`.

Pentru ultimul exemplul de mai sus care realizează conectarea la o baze de date folosind PDO, codul de conectare poate fi plasat într-un fișier extern (numit spre exemplu connect.php) care să fie inclus folosind `require_once` de câte ori este nevoie în cadrul altor fișiere PHP. În plus, această abordare simplifică situațiile în care trebuie schimbate datele de conectare (server, baza de date, parola) la serverul de date.

Exemplu, fișierul connect.php:

```
<?php
$dsn = "mysql:host=localhost;dbname=pw";
$user = "root";
$passwd = "";
$pdo = new PDO($dsn, $user, $passwd);
?>
```

Fișierele viewTrainsUsingPDO.php și insertTrainUsingPDO.php pot include ambele acest fișier:

```
<?php
require_once "connect.php";
?>
```

Observații:

- Este posibil să găsiți exemple în care declațiile de mai sus sunt folosite sub forma unor apeluri de funcții: `require_once("some_other_file.php")` – nu este nicio diferență și nici foarte greșit;
- În funcție de cum este configurat PHP-ul, folosirea acestor funcții poate ridica anumite probleme de securitate. În fișierul php.ini există o declarație `allow_url_include = Off` care dacă este setata pe `On` permite includerea de cod PHP localizat la un URL remote (de obicei cu cod malitios controlat de un atacator), cod care odată inclus în fișierul vostru se va executa pe serverul vostru web.

În interiorul unui fișier inclus (fie cu `include` fie cu `require`) se poate folosi declarația `return` pentru a termina execuția fișierului inclus și redarea controlului execuției către fișierul care a făcut includerea (atenție, a nu se confunda cu folosirea `return` în cadrul unei funcții care asigura doar terminarea funcției respective).

Manipularea antetelor HTTP din cadrul PHP

Sunt frecvente situațiile în cazul folosirii tehnologiilor server-side (PHP, ASP, JSP, servlet, etc), când de pe back-end trebuie trimise/setate anumite antete HTTP. Acest lucru se realizează în PHP cu funcția `header()`. IMPORTANT: folosirea acestei funcții, precum și a oricărei alte funcții care este posibil să seteze antete (headere) HTTP, trebuie să se facă înainte ca scriptul PHP să trimită spre client (browser) orice alt conținut HTML (sau de alt tip) care în mod normal succede new-line-ului de după antetele HTTP (recapitulare protocolul HTTP). Dacă back-endul a trimis deja conținut HTML spre browser, acesta succede în mod normal new-line-ului de după headere, lucru care înseamnă că și head-erele HTTP au fost trimise (se mai spune și că s-a făcut „commit” la acestea). Într-o asemenea situație, trimiterea de noi antete nu mai este posibilă.

Exemplu (fișierul redirectare.php disponibil aici ☺):

```
<?php
header("Location: http://www.google.ro");
?>
```

Exemplu de utilizare greșită a funcției `header`:

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Contra exemplu: asa nu! E gresit</title>
<style>

body, input {
    font-family: Tahoma, sans-serif;
    font-size: 12px
}
</style>
<body>
<?php
if ($ceva_eroare) {
    header("Location: tratare_ceva_eroare.php");
```

```
}
else {
    // All ok
}
?>
</body>
</html>
```

În exemplul de mai sus funcția `header` este folosită incorect. Pentru a putea seta headerul "Location", este nevoie ca antetele HTTP să nu fie trimise de pe back-end pe client, dar acestea au fost trimise pentru ca deja back-end-ul a trimis și codul HTML ce precede scriptlet-ul PHP. Pentru a o utiliza corect, funcția `header` trebuie folosită într-un scriptlet care să fie poziționat la începutul fișierului PHP (teoretic nici măcar o linie vidă nu este permisă înaintea acestui scriptlet), iar în cadrul scriptlet-ului respectiv nu trebuie afișat niciun fel de output către front-end.

Un alt exemplu de utilizare greșita a funcției `header` (disponibil aici):

```
<?php
    for ($i = 0; $i < 10000; $i++)
       echo $i;
    header("Location: http://www.google.ro");
?>
```

În exemplul de mai sus, a fost trimis intenționat în bucla `for` foarte mult conținut de pe back-end spre browser. Acest conținut este trimis pe conexiunea HTTP după trimiterea head-erelor, un eventual header trimis mai târziu (Location în cazul de față), nemaiputând fi luat în considerare de către browser, iar back-endul afișând un mesaj de avertizare în acest sens: „Warning: Cannot modify header information - headers already sent”.

Observație: pentru că pe back-end se păstrează un „cache” al output-ului înainte ca acesta să fie trimis efectiv spre browser, este posibil ca în unele situații antetele să poată fi setate/adăugate și în situația în care există un output (minimal) trimis spre browser (câteva linii de cod HTML ce preced scriptlet-ul sau puțin output afișat în cadrul scriptlet-urilor PHP înainte de folosirea funcției `header`).

Noțiunea de sesiune Web

Unul dintre aspectele cu care ar fi trebuit să rămâneți după parcurgerea referințelor bibliografice legate de protocolul HTTP și a problemelor de la laboratorul de HTTP este că acest protocol are un caracter „stateless”. Acest lucru se datorează faptului că este posibil ca fiecare cerere pe care o realizează un client (browser) spre serverul web să fie făcută pe o conexiune TCP independentă și separată de restul conexiunilor. De asemenea, inclusiv pentru încărcarea unei aceleași pagini și a tuturor resurselor necesare pentru randarea acesteia (imagini, fișiere CSS, fișiere JavaScript), browser-ul este posibil să inițieze conexiuni independente în paralel pe care să realizeze cererile către aceste resurse (conexiuni/cereri pe care le puteți vizualiza în tab-ul de Network din Developer Tools). Multitudinea cererilor pe care un client web (browser) le realizează către un server/aplicație web în scopul rezolvării unei probleme sau realizării unei anumite activități poartă denumirea de sesiune web sau sesiune HTTP. Cât de mult durează o sesiunea web, depinde atât de tehnologie folosită pe back-end dar și de logica/semantica activităților și cererilor pe care le face clientul. Este posibil ca după un anumit număr

de secunde de inactivitate în care clientul (browserul) nu mai realizează cereri, serverul să considere sesiunea expirată si să considere toate cererile HTTP ulterioare ca aparținând unei noi sesiuni. Dar tot o sesiune web poate fi considerată ca fiind alcătuită din multitudinea cererilor HTTP realizate pentru citirea e-mail-urilor de la login și până la logout sau multitudinea cererilor realizate în cadrul unei sesiuni de cumpărături online începută seara cu adăugarea câtorva produse în cos și continuată a doua zi dimineața cu finalizarea comenzii. Deși unele tehnologii de backend sugerează un anumit număr de secunde de inactivitate pentru păstrarea în viața a sesiunii de la ultima cerere făcută (spre exemplu în PHP 1440 secunde = 24 minute, valoare ce poate fi modificată din php.ini), această valoare poate fi modificată și depinde până la urmă de semantica cererilor și acțiunilor întreprinse cumulat de către aceste cereri la nivelul aplicației web/back-end-ului. Noțiunea de sesiune web nu trebuie gândită ca fiind asociata existenței unui mecanism de login/logout pe un anumit site. Exemple: există site-uri pe care puteți face cumpărături și fără a vă crea cont. Sau o „vizita" de câțiva „clicks" în cadrul paginii cursului de Programare Web în care vă informați despre ce a mai adăugat profesorul la curs sau despre cerințele unor laboratoare constitute tot o sesiunea web (ați rezolvat / realizat o activitate, toate cererile HTTP făcute de browser-ul vostru fiind realizate în același scop).

Pe partea de back-end pentru asocierea tuturor cererilor HTTP care vin de la browser la aceeași sesiune web este nevoie de un element comun de identificare a tuturor acestor cereri. Cum aceste cereri HTTP se realizează pe conexiuni TCP diferite este nevoie de un „ACEL CEVA" comun tuturor acestor conexiuni care să permite gruparea lor comună în cadrul aceleași sesiuni. Spre exemplu, un server TCP – la nivel transport în stiva TCP/IP - asociază toate pachetele care vinde la același ip_sursă, port_sursă, către același ip_destinație, port_destinație ca aparținând aceleași conexiuni (bazându-se pe proprietăți ale nivelelor rețea și transport). Datorită faptului că cererile HTTP se realizează pe conexiuni TCP diferite, nu ne mai putem baza pe adresa IP sau pe portul clientului, pentru a asocia aceste cereri ca aparținând aceleași sesiuni web pentru că:

- Cererile realizate de către același client sunt făcute pe conexiuni TCP diferite, conexiunile independente fiind realizate cu porturi sursă diferite, chiar dacă adresa IP a clientului este aceeași;
- De la aceeași adresa IP pot ajunge la serverul web cereri de la clienți web/utilizatori diferiți (spre exemplu toți studenții din căminul 16 fac cereri în exterior cu aceeași adresă IP – adresa IP reală a routerului din cămin care face (S)NAT) – în acest caz trebuie să existe o posibilitate ca la nivelul back-endului să diferențiem/grupa aceste cereri pe utilizatori;
- Inclusiv cererile realizate de un același client/browser/utilizator în cadrul unei sesiuni web pot fi făcute de la adrese IP diferite: gândiți-vă la situația în care navigați de pe mobil folosind o sesiune de date mobile, site-ul vizitat vede cererile voastre ca fiind realizate de la o adresa IP din spațiul de adrese a operatorului de telefonie mobilă, după care faceți switch pe Wi-Fi, site-ul vizitat văzând de data aceasta că cererile vin „din alta parte", de la o altă adresă IP.

Este practic nevoie de un „ACEL CEVA" comun tuturor acestor conexiuni care să permită gruparea lor comuna in cadrul aceleași sesiuni web. „ACEL CEVA" poate fi:

- Un cookie (numit și cookie de sesiune) setat de back-end prin intermediul header-ului `Set-cookie` și retrimis ulterior de client în fiecare cerere realizată prin intermediul header-ului `Cookie` (aceasta fiind și cea mai des întâlnită formă de persistare a sesiunii). Numele cookie-ului implicit de sesiune din PHP este PHPSESSID (dar numele acestuia poate fi modificat din php.ini);

- Un atribut comun care se găsește în QUERY_STRING-ul sau în body-ul tuturor request-urilor realizate;
- În cazul navigării prin protocolul HTTPS, „ACEL CEVA” poate fi legat și de cheia secretă de sesiune care e negociată de browser cu serverul web pentru realizarea criptării;
- Un mecanism la latitudinea programatorului, spre exemplul în cazul exemplului 8 de la laboratorul de HTTP, în cadrul unei sesiuni trebuia ghicit un număr. Asocierea tuturor cererilor făcute de un client/utilizator unei aceleași sesiuni (ghicirea aceluiași număr) se făcea prin intermediul unui `input` de tip `hidden` a cărui valoare era pasată de la front-ed spre back-end și înapoi;
- JSON Web Token - relatively new and hot :)

Este importat ca front-end-ul (clientul) și back-end-ul (serverul/aplicația web) să-și facă „ping-pong” și să-și trimită unul altuia „ACEL CEVA” (spre exemplu cookie-ul de sesiune) ca informație comună despre cererile HTTP pentru ca acestea să poate fi asociate cu succes ca aparținând aceleași sesiuni.

Stocarea datelor pe sesiune

Ținând cont că cererile HTTP se realizează prin intermediul unor conexiuni independente, toate tehnologiile de back-end oferă un mecanism de comunicare între „paginile” care se execută pe back-end la momente de timp diferite. Sunt frecvente situațiile în care o anumită valoare primită de la browser de către o pagina dezvoltată într-o tehnologie de back-end trebuie să păstreze „UNDEVA” această valoare pentru a fi vizibilă și de către o alta pagină ceruta în cadrul aceleași sesiuni web. Exemplu clasic: o pagina web din cadrul unei magazin online vă permite adăugarea unui produs în cos – este foarte posibil să realizați două cereri/conexiuni în acest sens – una pentru vizualizarea paginii care descrie produsul și una pentru adăugarea produsului în cos. Ulterior, pagina de checkout care va fi invocată printr-o cerere HTTP separată trebuie să aibă acces la informația adăugată de requestul precedent = coșul de cumpărături. „UNDEVA”-ul amintit mai sus, este de obicei un obiect/variabilă disponibilă la nivelul back-end-ului și care depinde de tehnologia folosită pe back-end. Spre exemplu, în PHP este un tablou asociativ cu numele `$_SESSION`, în cadrul tehnologiilor Java de back-end este un obiect (interfață mai degrabă) de tipul `HttpSession`. Cum persistă fiecare tehnologie valorile acestor obiecte/variabile între cererile realizate de către client nu este important - spre exemplu PHP-ul memorează sesiunile și datele salvate în cadrul unei sesiuni prin intermediul unor fișiere temporare, alte tehnologii de back-end pot păstra informațiile în memoria server-erelor de aplicații sau chiar în baze de date.

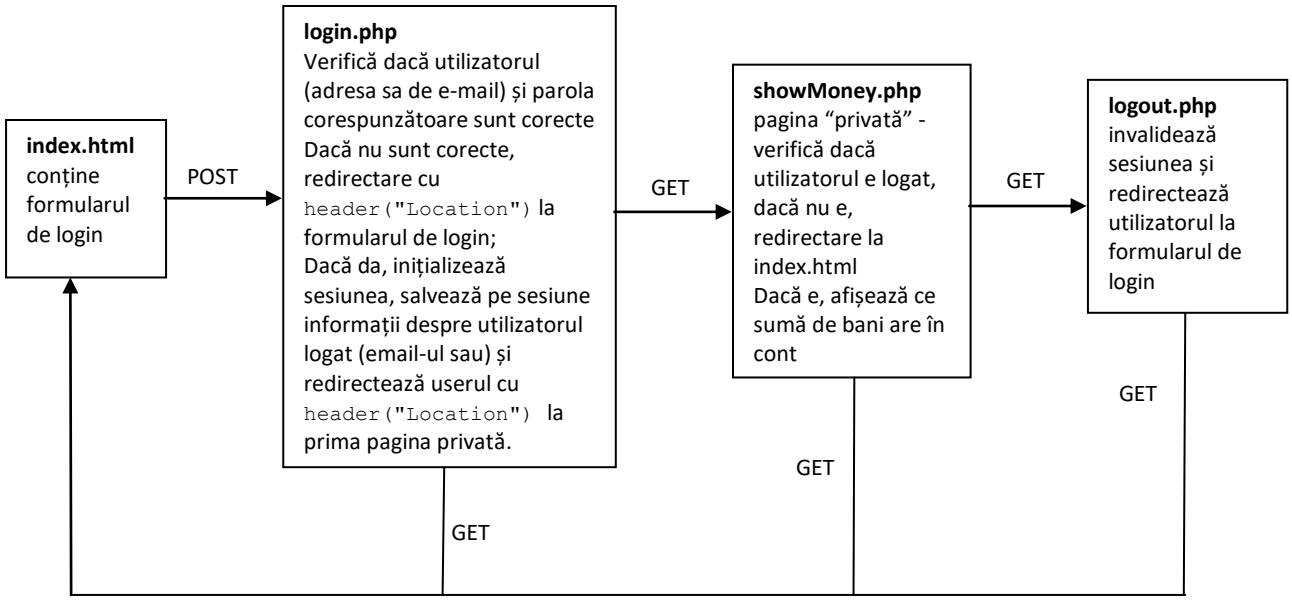
Observație pentru fixarea cunoștințelor: dacă reparcurgeți exemplul 8 de la laboratorul HTTP, unde în cadrul unei sesiuni aveați de ghicit un număr, am păstrat ca informație pe sesiune (informație care persista între toate cererile realizate în cadrul aceleași sesiuni web = sesiuni de joc) două valori: numărul care trebuia ghicit și numărul de încercări. Ca modalitate de stocare a acestora (de persistare a sesiuni) am ales tot fișiere temporare.

Lucrul cu sesiunea HTTP in PHP

În cele ce urmează prezentăm un exemplu clasic: Un utilizator se autentifică, accesează în urma autentificării una sau mai multe pagini private (cu informație privată, spre exemplu cu ce sumă de bani are în cont) care nu ar putea fi accesate dacă utilizatorul nu ar fi autentificat, după care se deloghează

(termenul este „își invalidează sesiunea”). După delogare, evident paginile private ce conțin informații private (precum e-mail-urie utilizatorului sau suma de bani disponibilă din cont) nu mai pot fi accesate.

Exemplu de față este disponibil în variantă online aici. Pentru a-l înțelege mai bine prezentăm flow-ul de execuție a acestuia:



Pentru a-l putea rula mai aveți nevoie de dump-ul tabelei useri (fișierul useri.sql). Tot în acest fișier găsiți și datele cu care vă puteți conecta (adresele de e-mail și parolele utilizatorilor cu care vă puteți loga în aplicație).

Formularul de login (fișierul index.html):

```
<form method="POST" action="login.php">
E-mail: <input type="text" name="email"><br>
Parola: <input type="password" name="parola"><br>
<input type="submit" value="Login">
</form>
```

Observați submitul prin metoda POST făcut datorită faptului că formularul conține un `input` de tip `password`.

Fișierul login.php

```
<?php
$formEmail = addslashes(htmlentities($_POST['email'], ENT_COMPAT, "UTF-8"));
$formPassword = addslashes(htmlentities($_POST['parola'], ENT_COMPAT, "UTF-
8"));

require_once "connection.php";

$stmt = $pdo->prepare("SELECT email FROM useri WHERE email=:email and
parola=:parola");
// 1 record will be selected if the e-mail and password ar ok
```

```
$stmt->bindParam(':email', $formEmail);
$stmt->bindParam(':parola', $formPassword);
$stmt->execute();
$result = $stmt->fetchAll();

if (count($result) != 1) {
    // invalid login
    header("Location: index.html");
    return;
}

// all ok
session_start();
$_SESSION['email'] = $formEmail;
header("Location: showMoney.php");
?>
```

Verifică dacă utilizatorul (adresa sa de e-mail) și parola corespunzătoare sunt corecte. Dacă nu sunt corecte, realizează o redirectare a clientului trimițându-i un header "Location" la formularul de login. Dacă datele de autentificare sunt corecte, inițializează sesiunea folosind `session_start()`, salvează pe sesiune (în cadrul tabloului asociativ `$_SESSION`) informații despre utilizatorul logat (email-ul său) și redirectează utilizatorul cu `header("Location")` la prima pagina privată. Funcția `session_start()` inițializează fie o sesiune nouă (generând un cookie de sesiune aleator și trimițând acest cookie browserului prin intermediul headerului `Set-cookie`) fie asociază requestul curent unei sesiuni anterioare create deja pe baza valorii cookie-ului de sesiune pe care browserul îl trimite prin intermediul header-ului `Cookie` pe request (în antetele cererii). Puteți observa în Developer Tools, pe antetele răspunsului headerul `Set-cookie` cu valoarea corespunzătoare (cookie-ul de sesiune PHPSESSID și valoarea asociată acestuia). Pentru că funcția `session_start()` este posibil să seteze un header HTTP (în caz de sesiune nouă), constrângerile de folosire ale acesteia sunt identice cu cele ale funcției `header()` de care am amintit anterior.

Fișierul connection.php

```
<?php
$dsn = "mysql:host=localhost;dbname=pw"; // abir1234 pe serverul de la
                                         // facultate
$user = "root"; // abir1234 pe serverul de la facultate
$passwd = ""; // abir1234 pe serverul de la facultate
$pdo = new PDO($dsn, $user, $passwd);
?>
```

Fișierul showMoney.php

```
<?php
require_once "checkSession.php";
require_once "connection.php";

$email = $_SESSION['email'];

$stmt = $pdo->prepare("SELECT * FROM useri WHERE email=:email");
$stmt->bindParam(':email', $email);
$stmt->execute();
$result = $stmt->fetch();
```

```
$suma = $result["suma"];

print "Sunteti autentificat ca $email<br/>";
print "In depozitul dumneavoastra se gasesc $suma euro.<br/>";
<br/>
<a href="logout.php">Log out</a>
```

Afișează câți bani are utilizatorul logat în cont. Utilizatorul căruia îi afișăm suma de bani disponibilă (adresa sa de e-mail) este preluat de pe sesiune dacă există setat pe sesiune acest lucru. Acest lucru este verificat în fișierul checkSession.php. De fapt acest fișier verifică dacă este cineva logat și dacă nu, face redirectare la formularul de login. Funcția `session_start()` din fișier-ul checkSession.php (deja suntem la al doilea apel al funcției `session_start()`) se uită la cookie-ul de sesiune primit pe request prin intermediul header-ului `Cookie`, și pe baza valorii acestui cookie populează tabloul asociativ `$_SESSION` cu valorile salvate anterior de către alte pagini accesate (login.php) în sesiunea curentă. Puteți observa în Developer Tools în antetele cererii făcute către fișierul showMoney.php headerul `Cookie` trimis de browser împreună cu valoarea asociată acestuia (cookie-ul de sesiune PHPSESSID și valoarea acestuia).

Observație: valorile stocate pe sesiune nu vor fi niciodată disponibile clientului (nu sunt vizibile in Developer Tools), sesiunea persistându-se exclusiv pe back-end.

Fișierul checkSession.php

```
<?php
error_reporting(0);
session_start();

// daca pe sesiune nu e setat un e-mail de utilizator autentificat
// inseamna ca nu s-a trecut prin formularul de login sau
// login.php nu a setat pe sesiune un e-mail de utilizator autentificat
if (! isset($_SESSION['email']) || ($_SESSION['email'] == ""))
    header("Location: index.html");
```

Fișierul logout.php

```
<?php
session_start();
session_destroy();
header("Location: index.html");
```

Distruge sesiunea curentă folosind `session_destroy()`, dar pentru a ști ce sesiunea trebuie invalidată apelează mai întâi `session_start()` (care din nou se „uită” la cookie-ul de sesiune primit pe cererea curentă).

Observații:

- Valoarea cookie-ului de sesiune este considerată informație „sensibilă” din punct de vedere al securității, dacă aceasta informație este furată/expusă unui terț (atacator, alt utilizator, etc.) acesta va putea face cereri în numele utilizatorului logat de la care s-a furat cookie-ul, fără ca atacatorul să mai fie nevoit să treacă prin formularul de login (orice cerere făcută de un client cu un cookie valid de sesiune, este asociat sesiuni respective). Altfel spus, atacatorul nu trebuie să știe user-ul și parola dacă „fura” cookie-ul de sesiune.

- După ce se face logout și se invalidează sesiunea folosind `session_destroy()`, browser-ul va continua să trimită cereri în care sa tot seteze antet-ul `Cookie` la valoarea cookie-ului de sesiune tocmai invalidate. Acest cookie de sesiune va fi ignorat de back-end ne mai fiind asociat unei sesiuni active. Este posibil că la o posibila relogare viitoare a clientului, back-end-ul sa „recicleze” la rularea funcției `session_start()` cookie-ul de sesiune folosit într-o sesiune anterioară și „neuitat” de client asociindu-i din nou o sesiune activă.

Va urma cu câteva discuții pe teme de securitate web

Observație: În mod normal fișierele php rulează sub același utilizator (cu privilegiile utilizatorului) sub care rulează și serverul web. Serverul web de pe www.scs.ubbcluj.ro rulează sub un utilizator fictiv numit apache. Serverul de la facultate este configurat mai special, php-urile rulate de către un student la un URL de forma www.scs.ubbcluj.ro/~abir1234 rulează cu privilegiile studentului respectiv (abir1234). Pentru a putea rula php-uri pe serverul de la școală trebuie să le setați ca fișiere executabile (+x pe fișierele php).

Ca de obicei sunt deschis la orice sugestii de îmbunătățire a acestui material și observații privind eventuale scăpări / greșeli (acord bonusuri recompensă ☺). Mulțumesc.