Informatică Aplicată 1 – Îndrumar de Laborator

Capitolul 4 – noțiuni avansate de PHP

1) Includerea de fișiere în programe

Putem include în scripturi conținutul altor fișiere HTML sau PHP folosind comanda **include**.

Exercițiu

- Creați două fișiere, unul cu extensia .php și altul cu extensia .html. De exemplu: test.php și fisier.html.
- Editați fișierul ‚fisier.html' scriind în acesta un cod de HTML pur (de exemplu un paragraf scris cu marcaje <H2> sau <H1>, o linie cu <HR> etc. Salvați fișierul html.
- Editați fișierul test.php și scrieți în acesta un mic script PHP (încadrat de marcajele de început și sfârșit de program) care să conțină de exemplu o afișare cu funcția echo și apoi comanda:

  include 'fisier.html';
- Rulați scriptul PHP cu: localhost/test.php

Notă: în același mod se pot include în program și alte fișiere PHP.

Acest mecanism este foarte util de exemplu când avem de generat în mai multe pagini ale unui site partea de antet sau subsol a paginii.

2) Prelucrarea excepțiilor

Similar cu alte limbaje de programare avansate (Java, C++) și în PHP se pot defini excepții. Mecanismul excepțiilor oferă o modalitate elegantă de a prelucra erorile care pot apărea în program. Codul care poate să genereze o eroare este inclus în comanda „try" iar excepția (eroarea apărută) este prelucrată apoi în rubrica „catch". Mai jos exemplificăm cu un exemplu de funcție care poate genera o eroare de tip „împărțire la zero".

```
function impartire($x) {
    if ($x==0) {
        throw   new  Exception('Impartire la zero');
    }
    return 1/$x;
}

try {
    echo "1/3=".impartire(3) . "<br>";
    echo "1/0=".impartire(0) . "<br>";
} catch (Exception $e) {
    echo 'Eroarea aparuta: ',  $e->getMessage(), "<br>";
}
```

Rulați codul de mai sus și observați cum funcționează generarea și prelucrarea excepției.

3) Folosirea fișierelor de tip cookie

Un cookie [1] este o pereche nume-valoare asociată unui site web care este *stocată pe calculatorul clientului* de către browser-ul folosit de client pentru a accesa acel site. Odată ce un cookie a fost setat de către server la nivelul browserului clientului, acesta rămâne acolo și poate fi citit la alte accesări ulterioare, dar doar de către serverul (site-ul) care l-a setat, nu de către alte site-uri.

Exemplu de utilizare cookie:

Creați un fișier PHP care setează un cookie numit „nume1”:

```
echo "<br> Inceput program setare cookie<br>";

// setez un cookie sa expire in 30 sec:
setcookie('nume1', 'Popescu', time() + 30);

echo "<br> cookie setat <br>";
```

Creați apoi un alt fișier PHP care citește cookie-ul setat:

```
echo "<br> Inceput program citire cookie<br>";
if (isset($_COOKIE['nume1']))
{
   $nume1=$_COOKIE['nume1'];
   echo "<br> Valoare setata in cookie: $nume1<br>";
}
else
   echo "<br> cookie expirat";
```

Rulați alternativ aceste două programe pentru a observa cum se setează și când expira cookie-ul setat. Funcția time() returnează numărul de secunde trecut de la „epoca Unix”, adică 01 Ianuarie 1970. Astfel, time() + 30 înseamnă un interval până la expirare de 30 de secunde. Funcția isset() verifică dacă cookie-ul mai există și este activ.

Exercițiu

Creați un script care setează un cookie de tip număr întreg, inițializat cu zero. Apoi, la fiecare accesare a scriptului (paginii) incrementați valoarea setată în cookie, afișați-o și salvați în cookie valoarea incrementată. Astfel se va obține un contor de accesări ale paginii memorat în cookie.

4) Sesiuni PHP

Mecanismul cookie-urilor este absolut necesar dacă dorim ca să fie păstrate informații atunci când utilizatorul navighează în diversele pagini ale site-ului, și mai ales când revine pe site. Se pot stoca diverse informații precum preferințele utilizatorului, setări etc. Fără cookie-uri serverul nu poate recunoaște dacă un anumit utilizator revine pe site. Însă nu se pot folosi extensiv cookie-urile pentru toate variabilele pe care le-am dori păstrate, deoarece există limitări și în plus este ineficient să trimit de fiecare dată, pentru fiecare pagină accesată conținutul cookie-urilor prin rețea.

Sesiunile vin să rezolve această problemă și să eficientizeze comunicația client-server. Astfel, prin sesiuni, variabilele pe care le dorim păstrate vor fi salvate (trimise) o singură dată către server și memorate de acesta într-un spațiu special, împreună cu un ID de sesiune generat în mod unic pentru fiecare utilizator (browser al utilizatorului de fapt). În comunicațiile ulterioare client-server, atunci când utilizatorul accesează același site, sau mai frecvent în aceeași „sesiune” navighează prin diverse pagini ale site-ului, serverul va ști după valoarea ID-ului de sesiune care sunt variabilele stocate și valorile lor. Am scris mai sus ca mai frecvent în aceeași sesiune deoarece mecanismul

sesiunilor este conceput în mod standard să dureze până la închiderea browserului. Acest comportament se poate însă modifica, după cum vom vedea mai jos.

În concluzie, *într-o sesiune clientul (browserul) şi serverul comunică folosind un ID de sesiune* (SID sau „session ID") unic care în mod prestabilit este salvat într-un cookie în browserul clientului. În timpul comunicației între client și server valorile variabilelor stocate în sesiune se trimit o singură dată și se păstrează pe server, iar la fiecare pagină accesată se va transmite doar ID-ul sesiunii păstrate în cookie. Prestabilit, atunci când închidem browserul, acel cookie va fi șters. Datele de pe server referitoare la sesiune vor fi șterse și ele într-un interval de timp prestabilit setat pe server. Acestea pot fi păstrate o perioadă mai lungă de timp prin anumite setări specifice.

Exercițiu

- Creați două fișiere PHP, primul de exemplu cu numele **test2.php** și următorul conținut:

```
<?php
//Pornirea sesiunii trebuie sa fie la inceputul fisierului (inainte de functii gen echo)
session_start();
echo "<br> Inceput program setare sesiune<br>";
// Setarea variabilelor salvate in sesiune
$_SESSION["prenume"] = "Mihai";
$_SESSION["oras"] = "Sibiu";
echo "<br> variabilele sesiunii au fost setate.";
print_r($_COOKIE); // pot vedea cookie-ul care pastrează SID-ul
?>
```

- Creați al doilea fișier cu numele de exemplu **test3.php** și următorul conținut:

```
<?php
// la inceput pornesc sesiunea
session_start();
$prenume = $_SESSION["prenume"];
$oras = $_SESSION["oras"];
// Am preluat variabilele setate anterior in sesiune si afisez
echo "Prenumele dvs. este: $prenume <br>";
echo "Orasul este: $oras <br>";
?>
```

- Încărcați întâi în browser fișierul: localhost/test2.php
- Rulați apoi fișierul localhost/test3.php și observați regăsirea variabilelor memorate în sesiune
- Închideți toate ferestrele browserului folosit anterior, și apoi încărcați din nou fișierul test3.php. Observați că în această situație variabilele nu mai se pot recupera (la închiderea browserului cookie-ul care păstra SID-ul a fost șters.
- Adăugați codul următor în fișierul test2.php înainte de apelul funcției session_start()

```
ini_set('session.cookie_lifetime', 60 * 60 * 24 * 1); // persistenta de o zi
```

- Rulați scriptul test2.php, apoi test3.php. Închideți apoi toate ferestrele browserului și rulați din nou fișierul test3.php. Ce observați de această dată?

Observații: variabilele din cadrul sesiunii sunt setate folosind vectorul $_SESSION care se va regăsi în orice script care utilizează sesiunea (are la început comanda *session_start()*). Pot face ca

sesiunea să persiste și după închiderea tuturor ferestrelor browserului prin comanda ini_set('session.cookie_lifetime'...) prezentată anterior.

Pot elimina variabile dintr-o sesiune folosind comanda unset(). De exemplu daca doresc eliminarea orașului, la sfârșitul scriptului test3.php pot adăuga:

```
unset($_SESSION['oras']);
```

Dacă doresc eliminarea completă a sesiunii pot rula comanda: *session_destroy()*;

De asemenea, valorile stocare în variabilele sesiunii pot fi oricând modificate. De exemplu putem modifica numele scriind:

```
$_SESSION["prenume"] = "Ion";
```

Adăugați această linie la sfârșitul scriptului test3.php și observați efectul.

5) Stocarea unei parole cu ajutorul PHP

În general toate sistemele online care necesită autentificare prin nume de utilizator și parolă nu stochează parolele utilizatorilor sub formă de text simplu (‚plaintext'). Din motive de securitate (acces neautorizat asupra bazei de date care conține parolele) se stochează nu parolele în sine ci un cod ‚hash' al parolelor. Codul hash este un șir de biți sau caractere care este generat printr-o funcție hash. Aceste funcții se mai numesc în română funcții de dispersie sau funcții rezumat.

O funcție hash aplicată unui șir de caractere de lungime oarecare va produce un șir de caractere de lungime fixă unic pentru șirul de caractere de la intrarea funcției. Șiruri de caractere care diferă doar printr-un singur bit vor duce la coduri hash diferite. Astfel, putem avea un cod hash unic pentru fiecare parolă a utilizatorului. O proprietate importantă a unei funcții hash este aceea că, știind un anumit cod hash, nu se poate afla șirul de caractere din care a provenit (parola).

Există mai mulți algoritmi siguri pentru generarea codurilor hash, de exemplu: SHA256 și Bcrypt.

De exemplu, SHA256 primește la intrare un șir de caractere de lungime variabilă și generează un cod hash de lungime fixă, 256 biți sau 64 de caractere hexazecimale (numerele hexazecimale pe 4 biți de la 0 la f). Dacă aplicăm textul ‚1234' la intrarea **SHA256**, vom obține:

‚03ac674216f3e15c761ee1a5e255f067953623c8b388b4459e13f978d7c846f4'

Totuși, în cazul sistemelor moderne, nu ajunge să stocăm în baza de date codurile hash, deoarece în cazul în care un atacator ar obține lista de coduri hash a utilizatorilor, și având posibilitatea de a genera coduri hash pentru parole uzuale, ar putea printr-un algoritm simplu să afle parolele unor utilizatori, deoarece funcția SHA256 este publică și oricine o poate folosi.

Algoritmul Bcrypt

Versiunile de PHP începând cu varianta 5.5 implementează pentru a face „rezumatul" unei parole (a genera codul hash) algoritmul Bcrypt. Acest algoritm se dorește a fi mai sigur decât SHA256 din două perspective:

- include un parametru de ‚cost' computațional, care face ca generarea să ia mai mult timp;
- atașează un șir de 16 octeți parolei, șir denumit în criptografie ‚sare' sau ‚salt'.

Prin atașarea șirului de ‚sare', codul hash generat nu va fi același pentru o anumită parolă, ci va diferi în funcție de acești biți atașați. Acest mecanism împiedică un posibil atacator, care ar avea la dispoziție o listă de coduri hash, să ghicească parolele, deoarece chiar parole identice din baza de date vor avea coduri hash diferite. În plus, algoritmul adaugă și un parametru de cost care complică generarea și duce, desigur la coduri hash diferite.

Exemplu de cod PHP pentru verificarea unei parole:

```
$parola = "parola1234";
$cod_hash = password_hash($parola, PASSWORD_DEFAULT);
echo "<br> cod hash: $cod_hash";

echo "<br> Acum verific parola<br>";

if (password_verify($parola, $cod_hash))
    echo 'Parola este corecta';
else
    echo 'Parola gresita';
```

Probleme propuse

1) Realizați un formular care să preia următoarele date: nume_utilizator și parola. Scrieți apoi codul sursă al unui fișier PHP care preia datele din formular și verifică parola introdusă, folosind un cod hash stocat în fișier.
2) Folosind funcțiile file_put_contents() și file_get_contents() implementați un mecanism prin care codul hash este întâi scris într-un fișier .txt pe server, apoi citit spre verificarea parolei.

Referințe

[1] Kevin Yank, PHP & MySQL novice to ninja, 5th edition, 2012 SitePoint Pty. Ltd