Laborator 01 - Introducere

- Repartizare teme
- Catalog PW

Laboratoare

- Laborator 01 - Introducere
- Laborator 02 - BD in PHP
- Laborator 03 - Arrays, Magic Methods
- Laborator 04 - (X)HTML, CSS
- Laborator 05 - Formulare HTML, Persistența Datelor
- Laborator 06 - Securitate I
- Laborator 07 - Securitate II
- Laborator 08 - JavaScript
- Laborator 09 - DOM scripting
- Laborator 10 - AJAX
- Laborator 11 - Web Frameworks

Teme

- Tema 01 - API pentru BD
- Tema 02 - Template System & Controller
- Tema 03 - Generator de formulare
- Tema 04 - Tree Web UI

Login

Obiective:

In urma parcurgerii acestui laborator studentul va fi capabil sa:

- cunoasca mecanismul de transmitere a informatiilor intre client si server, folosind protocolul HTTP
- inteleaga aspectele-cheie ale dezvoltarii unei aplicatii web cu PHP
- instaleze si foloseasca aplicatia WAMP(Windows-Apache-MySQL-PHP) si LAMP(Linux-Apache-MySQL-PHP)

Introducere

Protocolul **HTTP (HyperText Transfer Protocol)** este folosit pentru schimbul de informatii intre client si server in mediul Internet. HTTP este un **protocol "request-response"**. Astfel, clientul (de obicei un browser web), efectueaza o "cerere HTTP" catre un server, pentru a obtine **o resursa (de cele mai multe ori, o pagina web)**. Serverul trimite clientului un "raspuns HTTP" ce contine resursa ceruta, sau un mesaj de eroare, in cazul in care aceasta este inexistenta.

HTTP este un **protocol "stateless"**, in sensul ca fiecare cerere de la client este tratata independent de celelalte. Astfel, stabilirea unei "sesiuni de comunicare" persistente intre client si server nu poate fi realizata la nivelul HTTP. Pentru aceasta se vor folosi alte mecanisme, de exemplu sesiuni pe server sau cookie-uri in browser.

HTTP contine o serie de metode (comenzi care pot fi adresate serverului). Aceste metode indica actiunea de executat pentru resursa specificata. Metodele protocolului HTTP, utile pentru laboratorul de Programare Web sunt **GET** si **POST**.
Ele se clasifica in metode *safe* si metode generale. Metodele *safe* (cum este GET), pot fi folosite doar pentru obtinerea de informatii particulare aferente unei resurse; nu este recomandata folosirea metodelor *safe* pentru schimbarea starii interne a serverului (de ex, modificarea unui tabel din baza de date a serverului). Pentru astfel de actiuni, se recomanda folosirea metodei POST.
Modul de folosire a metodelor GET si POST va fi detaliat la laboratoarele viitoare. Detalii vizavi de protocolul HTTP

Urmatorii pasi descriu ce se intampla atunci cand doriti sa incarcati o pagina web in browser:

1. Browser-ul trimite o cerere HTTP catre server.
2. Serverul parseaza cererea si o proceseaza construind un raspuns (ce contine codul HTML aferent paginii) care este trimis inapoi browserului.
3. Browser-ul parseaza raspunsul primit si construieste o reprezentare in arbore DOM a HTML-ului primit. De asemenea, se construieste cate o cerere HTTP pentru fiecare dintre resursele referite din document (CSS, JavaScript, imagini).
4. Dupa descarcarea CSS-urilor, browserul le interpreteaza si le aplica DOM-ului.
5. Dupa descarcarea JavaScript-urilor, browserul le parseaza si le executa.

Elementele cheie din acest schimb de informatii sunt:

- Browser-ul: Firefox, Chrome, Safari, Opera, Internet Explorer
- Server-ul: Apache Httpd, Apache Tomcat, nginx, IIS, JBoss, WebLogic etc.

Din punct de vedere al browser-ului, recomandam insistent folosirea unuia care respecta standardele W3C (HTML, CSS, JavaScript). Emergenta standardelor este unul dintre cele mai mari avantaje pentru dezvoltatorii de azi, pentru ca astfel se pot scrie aplicatii cross-browser in mod consistent si nu prin aplicarea diverselor hack-uri. Internet Explorer 6 si 7 au ignorat in mod deliberat standardele si au tras inapoi dezvoltarea web pe tot parcursul perioadei in care au avut o cota de piata semnificativa. De la IE8, Microsoft a inceput sa se orienteze catre standarde, dar inca este in urma fata de principalii competitori (Chrome, Firefox, Safari).

Fiecare din servere poate executa programe scrise in unul sau mai multe limbaje. De exemplu, Apache Httpd poate rula PHP, Python, Ruby, ASP.NET. La acest laborator ne vom concentra asupra stivei Apache Httpd + PHP + MySQL deoarece este cea mai raspandita in mediile de hosting si este usor de abordat pentru programatorii aflati la inceput cu programarea web.

Limbajul de programare PHP

Un criteriu important pentru clasificarea limbajelor de programare este implementarea sistemului de tipuri. **In functie de verificarea tipurilor** (*type checking*) distingem:

- Limbaje cu verificare statica a tipurilor (verificare la compilare)
- Limbaje cu verificare dinamica a tipurilor (verificare la run-time)

In limbajele cu verificare statica (de exemplu: JAVA, C/C++) variabilele trebuie declarate explicit de catre programator (astfel, fiecarei variabile i se asociaza un tip: int, boolean, Object, etc). O variabila poate suferi un **cast** la alt tip, insa aceasta operatie **nu modifica tipul efectiv al variabilei**, ci doar

interpretarea contextuala a acestuia. Verificarea tipurilor se face la compilare, iar eventualele erori sunt semnalate inaintea rularii codului.

PHP este un limbaj cu **verificare dinamica** a tipurilor. In PHP, **nu este necesara** declararea explicita a variabilelor, inainte ca acestea sa fie folosite. PHP nu face nici o verificare asupra tipurilor la compilare. Verificarile se fac doar la rulare (*runtime*). Urmatoarele constructii sunt valide in PHP:

```
$var = 10; // variabila $var nu este declarata - nu i se asociaza un tip pre-definit; aceasta cont
$var = "String"; //aceeasi variabila a fost modificata astfel incat sa contina un sir de caractere
```

In functie de modul in care se realizeaza tiparea, limbajele pot avea:

- **tipare tare** (*strong typing*) (de exemplu: JAVA, C/C++)
- **tipare slaba** (*weak typing*)

PHP este un limbaj cu **tipare slaba**. Fie urmatorul exemplu:

```
$var = "x";
$var = $var + 2; // atribuire invalida intr-un limbaj tare-tipat
echo $var;
```

Ne-am astepta ca operatia `$var + 2` sa produca o eroare, din cauza diferentelor de tipuri intre `$var` si intregul `2`. Constructia de mai sus nu va produce erori in PHP; limbajul este *slab tipat* si ca o consecinta (in exemplul de mai sus), nu se fac verificari intre tipurile operanzilor. (O intrebare interesanta este: Ce va afisa codul de mai sus ? Testati.)

Caracteristicile limbajului PHP (*verificare dinamica* si *tipare slaba*) il fac foarte flexibil pentru scrierea de cod. Exista insa si dezavantaje. Fie urmatoarea constructie:

```
$value = 0;
$i = 0;
while ($i < 5) {
   $valu = $value + $i; //typo
   $i++;
}
echo $value;
```

Codul de mai sus isi propune sa calculeze suma primelor 5 numere naturale. Insa la linia 4, programatorul a facut o greseala lexicala (este folosita variabila `$valu` in loc de `$value`). In alte limbaje precum JAVA sau C/C++ se va semnala o eroare (motivele au fost explicate mai sus). In PHP insa, programul este functional; `$valu` este o variabila vizibila in blocul `while`,(nu trebuie declarata). (Care este rezultatul codului?)

Mai multe detalii despre *strong typing* si *weak typing*:

- http://en.wikipedia.org/wiki/Type_system
- http://articles.sitepoint.com/article/typing-versus-dynamic-typing

Aspecte cheie pentru dezvoltarea aplicatiilor web

Pe langa limbaj (PHP), vom pune accent la **Programare Web** pe *design patterns* si arhitecturi specifice pentru aplicatii web. Un *design pattern* des folosit in dezvoltarea de aplicatii web este **MVC (Model View Controller)**.
MVC este un *pattern* arhitectural care propune divizarea unei aplicatii in trei componente izolate:

- **Model** (defineste modalitatea de reprezentare a datelor)
- **View** (defineste modul in care sunt prezentate informatiile utilizatorului)
- **Controller** (defineste logica aplicatiei)

Modalitatea de definire a fiecarei componente in cadrul unei aplicatii se poate face in mai multe feluri.

Model

Putem considera baze de date aleasa (MySQL spre exemplu), ca fiind componenta **Model** a aplicatiei. Alegerea nu este gresita din punct de vedere al arhitecturii MVC, insa ajuta foarte putin la dezvoltarii aplicatiei. Fie urmatorul exemplu de query:

```
SELECT name FROM users_table WHERE id = 5
```

Pentru conectarea la baza de date, executia *query*-ului si obtinerea rezultatului, trebuie rulata o secventa de aprox. 3-4 instructiuni in PHP. Un astfel de *query* poate aparea foarte des in dezvoltarea unei aplicatii web, iar rescrierea secventei pentru fiecare *query* este greoaie si ineficienta. De asemenea, daca din diferite motive, se doreste folosirea altei baze de date, tot codul trebuie re-scris.

Fie un **API (Application Programming Interface)** in PHP, care permite obtinerea de informatii din baza de date astfel:

```
$object = new Record(5,"users_table");
echo $object->name;
```

Se observa usurinta de extragere si prelucrare a informatiilor din baza de date. In plus, utilizatorul API-ului nu este afectat de alegerea sau modificarea bazei de date folosite. Putem considera un

astfel de API ca fiind o componenta **Model** eficienta, pentru o aplicatie web MVC.

View

Componenta View defineste interfata-utilizator a aplicatiei web. Asemanator cu exemplul de mai sus, putem considera limbajul HTML ca fiind componenta *View* a aplicatiei. Din nou, alegerea este formal-corecta, insa foarte putin eficienta. Pe de alta parte, folosirea unui *template-system*, precum Smarty este mult mai eficienta. **Smarty** permite definirea de componente HTML re-utilizabile, si permite izolarea codului functional de interfata-utilizator. Aspecte legate de *template-systems* vor fi discutate in laboratoarele viitoare.

Controller

Componenta *Controller* defineste intreaga logica de functionare a aplicatiei web. *Controller*-ul primeste comenzi de la utilizator, si obtine informatii sau modifica starea aplicatiei, in concordanta.
Comenzile pot fi:

- formulare trimise prin metodele GET/POST (despre care vom discuta in laboratoarele viitoare)
- Script-uri rulate sub forma accesarii de link-uri (ex: http://www.aplicatie.com/script.php)

Efectele comenzilor pot fi:

- Extragerea de informatii (ex: afisarea listelor de produse, detalii-produs, sortarea acestora, etc)
- Modificarea starii aplicatiei (ex: introducerea unui utilizator nou, modificarea drepturilor unui utilizator, adaugarea unui produs nou, stergerea unui produs, etc)

Comenzile pot fi:

- accesate/rulate de orice utilizator
- accesate/rulate de utilizatori inregistrati
- accesate/rulate de utilizatori inregistrati, avand anumite drepturi

Este sarcina Controller-ului sa stabileasca si decida daca o comanda poate fi executata sau nu.

Putem considera *Controller*-ul ca fiind partea centrala a aplicatiei. Urmatoarele elemente sunt esentiale:

- decuplarea intre **Controller** si **Model**: logica aplicatiei nu trebuie sa depinda direct de modul de stocare/reprezentare a datelor
- decuplarea intre **Controller** si **View**: logica aplicatiei nu trebuie sa depinda de interfata-utilizator (aceasta din urma trebuie sa poata fi modificata fara afecta codul asociat *Controller*-ului.

In cadrul laboratorului vom cauta sa definim elemente specifice pentru *Model*, *View* si *Controller*, respectand observatiile de mai sus.

WAMP

WampServer este un mediu de dezvoltare in Windows pentru aplicatii web folosind ApacheServer, PHP si MySQL. De asemenea el contine utilitarele SQLite Manager si PHPMyAdmin utile pentru gestionarea bazei de date.
Pentru detalii si download consultati:
http://www.wampserver.com/

Pentru administrarea bazelor de date mai puteti folosi pachetul de aplicatii "MySQL Administrator - MySQL Query Browser - MySQL Migration Toolkit", ce poate fi downloadat de aici:

http://dev.mysql.com/downloads/gui-tools/5.0.html

LAMP

Pentru a instala Apache, MySQL si PHP pe un sistem Linux, va propunem urmatorul tutorial:
http://www.howtoforge.com/ubuntu_lamp_for_newbies