CRUD

Scopul final

În acest laborator ne propunem să construim o aplicație CRUD simplă, cu PHP și MySQL, pentru gestiunea unor contacte. Vom face o validare simplă a informațiilor pe server, și vom încerca să folosim câmpul *adresă* pentru a traduce, prin geolocalizare adresa într-o pereche de coordonate geografice. Vom încerca apoi, să folosim API-ul Google Maps pentru a afișa această locație pe o hartă.

Concepte și tehnologii: baze de date, mysqli, clase PHP, template-uri (separarea codului HTML de PHP), geocoding și Google Maps API.

Surse de date în aplicații web

O aplicație web, de regulă, face *ceva interesant* cu un set de date - datele pot veni dintr-o verietate de surse, de la fișiere text la API-uri publice. Multe aplicații web comunică cu baze de date (relaționale sau de alt fel) pentru a citi / scrie.

Modul de funcționare al Sistemelor de Gestiune a Bazelor de Date nu este obiectul acestui laborator (plecăm de la presupunerea că ați acoperit bazele lor la Baze de Date), vom discuta doar accesarea lor printr-o interfață SQL.

Majoritatea SGBD-urilor cunoscute (MSSQL, Oracle, MySQL, Postgresql) oferă implementări sau biblioteci care permit conectarea la ele și interogarea lor din limbajele de programare importante, inclusiv PHP, dar și C, Java, python, Ruby. În PHP, conexiunea la aceste baze de date se face printr-o serie de clase din pachetul MySQLi - MySQL improved.

Pentru a testa aceste conexiuni și a vedea în practică cum se pot folosi bazele de date în aplicații web, vom folosi combinația "clasică" pentru multe aplicații web (de la cele mai simple la unele de succes, spre exemplu, Wordpress): PHP & MySQL. Totuși, toate conceptele cu care vom lucra funcționează ca și principiu la fel în oricare alt limbaj de programare și cu orice SGBD care suportă SQL.

Clase în PHP

Ca orice limbaj de nivel înalt, PHP are capabilități de programare orientată obiect (OOP). Documentația pentru OOP din PHP se găsește aici.

```php
<?php
class SimpleClass
{
    // property declaration
    public $var = 'a default value';
    // method declaration
    public function displayVar() {
        echo $this->var;
    }
}
?>
```

Câteva chestiuni de bază:

- un obiect nou (o instanță a unei clase) se creează folosind cuvântul cheie *new* (de exemplu, $obj = new SimpleClass();
- constructori și destructori - clasele PHP permit definirea funcțiilor __construct() și

__destruct(), care se apelează la creearea, respectiv la distrugerea obiectului - care permit inițializări.
- pentru a folosi atributul *atribut* al unui obiect, putem face așa: $obj->atribut
- pentru a apela o metoda numită *metoda*, $obj->metoda()
- in interiorul clasei, avem la dispozitie variabila $this, care este o referinta pentru instanta curenta.

Deși nu este o cerință a PHP, o regulă folosită în general de dezvoltatorii PHP este păstrarea convenției din Java, creearea unei singure clase intr-un fisier sursa (pentru curiosi, puteti citi mai multe despre unul din avantajele directe ale acestei metode aici). **Vom folosi si noi aceeasi conventie.**

Conexiunea la o bază de date cu PHP

PHP ne pune la dispoziție două metode de conectare la o bază de date - obiectual și procedural. Pentru că este mai aproape de obiectivul laboratorului nostru, vom folosi abordarea obiectuală. Există două obiecte care ne interesează în mod deosebit în acest caz, MySQLi și MySQLiResult. Exemplele de mai jos sunt luate, în principiu, din exemplele date în cele două pagini de mai sus.

Ce trebuie să știm înainte să ne conectăm la o bază de date MySQL

Fiecare SGBD are o schema de conectare și de definire a resurselor (utilizatori, baze de date). În MySQL, avem nevoie de:

- adresa de rețea a server-ului care rulează MySQL (în cazul nostru, **localhost** sau 141.85.252.118)
- port-ul pe care răspunde server-ul (implicit **3306**)
- un utilizator și o parolă (**aw / test**)
- numele bazei de date (**aw_lab5**)

Conectarea efectivă

Pentru a ne conecta, trebuie să construim un obiect nou de tip MySQLi:

```
$connection
= new mysqli('localhost', 'utilizator', 'parola', 'nume_baza_de_date');
```

O bună practică este să verificăm dacă conexiunea s-a făcut corect. De asemenea, ne poate ajuta foarte mult la depanare (în cazul în care conexiunea eșuează) să știm ce s-a întâmplat. În cazul unei erori, aceasta este salvată în cadrul obiectului nou creeat.

```
if ($connection->connect_error) {
    die('Eroare de conectare (' . $connection->connect_errno . ') '
. $connection->connect_error);
}
```

În cazul în care totul a mers bine, în $mysqli vom avea o conexiune cu baza de date. Altfel, execuția script-ului va fi oprită (prin funcția PHP numită sugestiv die()).

Realizarea unei interogări (execuția unui query)

Există mai multe variante prin care putem să transmitem interogări la baza de date - fie direct, printr-un string, fie folosind un prepared statement (o variantă mai sigură, dar mai complexă).

```
$result = $connection->query("SELECT nume FROM contacte LIMIT 10");
```

Rezultatul metodei *query* este o instanță a clasei MySQLiResult, care conține informația obținută din baza de date (în cazul în care rezultatul query-ului este informație tabelară - cum avem în cazul

SELECT, dar nu și în cazul CREATE).

Spre exempu, pentru a obține numărul de linii întoarse de query-ul nostru, putem folosi:

```
$result->num_rows
```

Pentru a accesa informațiile din rezultat, avem la dispoziție mai multe metode. Rezultatul are un *cursor* intern, care este asemănător cu un cursor dintr-un fișier. Putem, spre exemplu, citi pe rând infromațiile din rezultat, linie cu linie. MySQLiResult ne pune la dispoziție mai multe forme în care putem primi informația dintr-o linie de tabel - spre exemplu, putem primi un dicționar (cheile = numele coloanelor definite în MySQL). O altă variantă este un obiect, care are ca atribute coloanele din tabel.

```
if ($result = $connection->query("SELECT nume FROM contacte LIMIT 10")) {
    printf("Select returned %d rows.\n", $result->num_rows);

    /* Ca și un fișier, rezultatele trebuie închise, pentur a elibera memoria
ocupată de ele */
    $result->close();
}
```

```
$query = "SELECT nume, prenume FROM contacte LIMIT 10";
if ($result = $mysqli->query($query)) {
    /* obtine un obiect cu informatia din linia urmatoare */
    while ($obj = $result->fetch_object()) {
        printf ("%s (%s)\n", $obj->nume, $obj->prenume);
    }

    $result->close();
}
```

Template-uri - soluția pentru separarea HTML de PHP

Până acum, am scris PHP printre HTML. Această practică, deși permisă de PHP, amestecă codul de prezentare (HTML) cu logica aplicației (PHP) și ne poate da mari bătăi de cap. De aceea, o buna practică este să le separăm. Totuși, de multe ori, obiectivul nostru este să aratăm utilizatorului ceva, în general cod HTML.

Pentru a separa codul HTML de PHP, putem folosi un motor (engine) de template-uri. Există o serie de template engine-uri bune (Smarty, altele), dar toate funcționează pe același principiu.

- pornim de la un template HTML, care conține codul HTML, fără conținutul dinamic (folosind exemplul laboratorului trecut, vom avea)
- se marchează cumva (depinde de template engine) zonele în care ne interesează să includem conținut dinamic ()
- în PHP, transmitem template engine-ului valorile pentru marcajele speciale (spre exemplu, într-un array - array("link_img" => $valoare_link, "img_src" => $valoare_src);
- in final, template engine-ul scoate HTML-ul complet, cu partea dinamică completată în locul marcajelor speciale

Pentru acest laborator, vom folosi un Template Engine extrem de simplu, inspirat de aici.

Aplicația noastră - un CRUD cu informații geografice

Google Maps API

Un API (Application Programming Interface) este o definitie a unei metode de acces programatic într-un sistem informatic. Definiția unui API depinde de aplicația de care ține (poate fi o listă de

metode, și valorile ce le întorc, pentru aplicații desktop, sau formatul și parametrii unei cereri HTTP pentru aplicații web, sau altele).

Google pune la dispoziție o serie întreagă de API-uri pentru produsul Maps, în diferite scopuri - pe noi ne interesează astăzi să producem o instanță a Google Map cu Javascript.

Sarcini:

1. **Folosind documentația de aici, realizați o pagină HTML care afișează o hartă centrată în București cu un zoom-level potrivit.**
2. **Folosind documentația de aici, adăugați un *marker* în punctul pe care ați centrat harta.**

Notă: Link-urile de mai sus vă duc direct la documentația necesară pentru a rezolva cele două sarcini. Desigur, Google Maps API are o grămadă de alte funcționalități, cu care sunteți încurajați să experimentați

Afișarea unor date dinamice, varianta clasică (PHP în HTML / JS)

Următorul pas este să preluăm puncte dintr-o bază de date și să le afișăm pe harta noastră. Avem la dispoziție o colecție de puncte de access WiFi, împreună cu poziția lor geografică (Long, Lat, SSID).

1. Creați un script .php în care să obțineți o parte (cu LIMIT) sau toate înregistrările din tabelul ***ssid_data***, folosind explicațiile de mai sus. Nu uitați să verificați dacă conexiunea a fost făcută cu succes, și, apoi, dacă resursa de tip MySQLiResult există (adică dacă query-ul vostru nu a returnat o eroare la execuția în SQL) - pentru acest caz puteți folosi atributul ***error*** al clasei MySQLi

Acum că avem informațiile (fie într-o serie de obiecte, fie într-un array de array-uri, în funcție de metoda MySQLiResult folosită), următorul pas este să le afișăm pe hartă. Într-o primă variantă, vom face acest lucru folosind aceeași metodă pe care am folosit-o și în celălalte laboratoare - inserarea de cod PHP în HTML și Javascript cu scopul de a genera anumite secțiuni de cod.

Dacă până acum am folosit ***echo*** pentru a scoate cod HTML către răspuns, în task-ul următor vom scrie cod Javascript. Țineți minte, în momentul execuției codului PHP, codul nostru javascript este doar un text, si nu este in executie, pentru că interpretorul Javascript este apelat doar DUPĂ ce răspunsul trimis de server ajunge la browser (și, deci, evident, execuția script-ului PHP s-a încheiat).

Ați văzut la o sarcină anterioară că adăugarea unui marker pe hartă este extrem de facilă:

```
var marker = new google.maps.Marker({
    position: new google.maps.LatLng(-25.363882,131.044922),
    map: map,
    title:"Hello World!"
});
```

1. Scrieți un script PHP care să genereze codul javascript care să afișeze puncte din baza de date. Nu uitați să schimbați extensia fișierului în .php și să-l încărcați pe un server pentru a fi executat de un interpretor php

Utilizarea unui template engine

În scheletul de laborator există un TemplateEngine extrem de simplu, a cărui parte de procesare este în esență o singură linie de cod PHP. Pentru a începe să folosim un template engine, vom avea nevoie să modificăm structura aplicației noastre. Există mai multe metode prin care putem face asta, iar în acest laborator vom experimenta cu utilizarea unui script *dispecer (dispatch)*.

Conceptul este următorul:

- avem un script dispecer, care în funcție de parametrii (GET, POST ...) apelează anumite funcții, care la rândul lor produc un răspuns HTTP. Avantajul aici este că avem un loc centralizat unde conectăm la aplicație toate funcțiile, și putem să facem o serie de verificări și de inițializări într-un singur loc.
- avem, în același script sau într-un alt script, definite o serie de metode care vor tipări ceva ca rezultat al cererii HTTP. Aceste metode pot fi simple sau parte din niște obiecte.
- în metodele care tratează diferitele acțiuni, folosim un TemplateEngine pentru a produce un rezultat conform unui șablon.

Un script dispecer se găsește în scheletul laboratorului, și constă în principiu în următoarea secvență:

```
switch ($action) {
    case "list":     lista_persoane();
                     break;
    default:         pagina_index();
}
```

Pentru a adăuga o funcționalitate nouă aplicației noastre web, este suficient să adăugăm o intrare în switch-ul acesta (pe care apoi o putem accesa folosind ceva de genul index.php?action=nume_actiune).

TemplateEngine-ul nostru are următoarele funcționalități:

Constructor: $t = new TemplateEngine($nume_fisier_html_cu_marcaje, $array_cu_elemente)

- ***$nume_fisier_html_cu_marcaje*** este un string care defineste calea absoluta sau relativa catre un fisier HTML cu marcaje de tipul {{nume_marcaj}}.
- ***$array_cu_elemente*** este un *array* (un dicționar) care pentru fiecare marcaj din fișierul anterior are câte o pereche cheie - valoare, în care cheia reprezintă numele marcajului și valoarea - valoarea cu care se dorește a fi înlocuit marcajul.

Metode

TemplateEngine-ul nostu ne pune la dispoziție două metode:

- TemplateEngine::**render_to_string()** - intoarce un string cu rezultatul HTML
- TemplateEngine::**output_to_response()** - nu intoarce nimic, dar tipareste catre raspuns rezultatul HTML

1. **Transformați HTML-ul inițial cu harta într-un template unde să puteți să introduceți codul Javascript pentru puncte.**
2. **Scrieți o funcție nouă care să încarce proaspătul vostru template, înregistrați-o în dispecer**
3. **Pentru a putea introduce mai multe puncte în template-ul cu harta, faceți un template intermediar (cu codul necesar pentru un singur punct) și exploatați metoda TemplateEngine::render_to_string() pentru a concatena rezultatele și a pregăti rezultatul de introdus în template-ul principal.**
4. **Folosiți metoda TemplateEngine::output_to_response() pentru a trimite rezultatul ca răspuns**

Partea de CRUD

CRUD vine de la CReate, Update, Delete - trei acțiuni elementare pe care le putem face într-o aplicație web care lucrează cu o bază de date (comenzile SQL corespondente sunt INSERT, UPDATE, și, respectiv, DELETE).

Ca să fim bine organizați, în primul rând vom avea nevoie de o clasă Punct(), care să aibă ca

membri cel puțin *id*, *long*, *lat* și *ssid* și care să aibă un constructor care să inițializeze aceste valori, în cazul în care sunt pasate ca parametri, dar și o funcție de utilitate - care să populeze câmpurile obiectului pornind de la un *id* (un număr unic fiecărei înregistrări).

1. **Scrieți clasa Punct, care să aibă metoda __construct($long = 0, $lat = 0, $ssid = ''), metoda get_from_id($id)**

Obiectivul nostru va fi sa adaugam, sa stergem si sa modificam anumite puncte din harta noastra.

1. **Scrieti prototipul (nume și parametrii) pentru cele trei funcții pentru create, update și delete - ce parametrii ar trebui să primească fiecare metodă și înregistrați metodele în dispecer.**
2. **Adăugați trei metode clasei *Punct*, care să primească ca parametru o conexiune *mysqli* și să realizeze query-urile SQL necesare acțiunilor respective**

Create și Update

Pentru Create și Update vom avea nevoie să folosim un formular - formularul trebuie să conțină trei input-uri - long, lat și ssid, și un buton de submit. Țineți minte că metoda HTTP pentru a modifica date pe server este POST. Adăugați clasei ***Punct*** o metodă statică care întoarce reprezentarea HTML a unui form gol pentru a crea un nou obiect Punct, și o metodă pentru a întoarce reprezentarea HTML a unui form populat cu informațiile curente ale obiectului.

Funcțiile de acțiune (cele înregistrate în dispecer) pentru ***create*** și ***update*** folosesc următorul workflow:

- dacă metoda curentă a cererii este GET, atunci obțin și scriu intr-un template general uu formularul gol, respectiv unul populat cu informatiile obiectului
- daca metoda curenta a cererii este POST, valideaza informatiile din form, populeaza un obiect Punct cu ele, si il scrie in baza de date folosind metodele update sau insert, dupa caz.

Cum facem diferența între INSERT și UPDATE

Când primim un POST nou, trebuie să știm dacă datele sunt ale unui punct existent, sau ale unui punct modificat. O metodă simplă de a lua această decizie este de a adăuga formularului un câmp hidden, care să fie populat cu id-ul obiectului Punct, în cazul în care discutăm de modificare (în cazul în care discutăm de inserare, câmpul id va fi null).

În acest caz, workflow-ul nostru se modifica putin - în cazul POST, în cazul în care avem ceva de genul $_POST['id'] (sau cum ați denumit câmpul hidden), creeăm prima dată un obiect Punct, folosind metoda get_from_id(), și apoi îl populăm cu informațiile din form.

După ce salvăm modificarea, putem redirecta utilizatorul la o pagină (spre exemplu, la hartă), folosind metoda header("Location: ". $location) din PHP.

1. Implementati cele două metode (sau folosiți doar una, și o formulă de condiții pentru cele două cazuri)

Delete

Pentru delete ar fi suficient să apelăm funcția prin GET, dar acest lucru este nerecomandat. Bunele practici cer ca orice cerere care va face modificări în starea datelor să fie apelate prin POST. De aceea, în metoda de acțiune pentru delete, vom avea iarăși două cazuri:

- dacă cererea curentă este GET, atunci vom afișa un formular gol, care va contine doar un mesaj de confimrare
- daca cererea curenta este POST, atunci vom sterge Punct-ul, folosind metoda delete() a acestuia.

1. Implementați metoda delete

Utilizarea excepțiilor în PHP

Folosirea unei structuri cu dispecer ne dă și posibilitatea implementării unui mecanism simplu de tratare a erorilor. Toate aplicațiile, inclusiv aplicațiile web, din punct de vedere al user / interface design-ului trebuie să implementeze o formă de graceful failure - dacă are loc o eroare, aceasta să fie prezentată într-un mod în care utilizatorul să poată reveni la ce făcea înainte, sau, eventual, să poată cere ajutorul unui dezvoltator pentru a o rezolva.

Pentru asta, putem folosi sistemul de exception handling al PHP. Exceptiile sunt niste clase speciale, care moștenesc clasa de baza Exception, și care pot fi *ridicate (*thrown, în engleză) în anumite secțiuni ale codului. Toate API-urile și alte utilități PHP documentează în detaliu ce excepții se ridică și în ce condiții.

Excepțiile pot, desigur, să fie ridicate și în codul nostru. Spre exemplu, în cazul în care încercăm să scoatem un rând din tabel după id, și id-ul nu există, putem ridica o excepție. Ridicarea unei excepții se face cu throw:

```
if (!$result) {

  throw new Exception($mysqli->error);

}
```

În momentul în care o excepție este ridicată, execuția (interpretarea) codului este stopată, și excepția este propagată până la primul bloc try { ... } catch (Exception $e) { ... } intalnit. Dacă nu se întâlnește niciun bloc try .. catch, excepția este pasată ca răspuns utilizatorului (mai multe informații despre excepții în php, aici).

În tratarea unei excepții, aceasta poate fi pasată și nivelului superior. Pentru că, în cazul utilizării unui dispecer, toate apelurile de funcții au loc în aceeași secvență de cod, *îmbrăcarea* acelui switch cu o secvență de tip try ... catch ne poate permite să *prindem* orice Exceptie aruncată în cod.

Putem apoi, în blocul catch { ... } să construim un HTML dintr-un template de bază și codul erorii - pentru a fi prezentat utilizatorului, împreună, eventual, cu link-uri către alte locuri în aplicația noastră.