

Projekt Labor

Fórum alkalmazás tervezése és implementálása

Technikai dokumentáció

Készítette: Lückl Roland (MSN: drunken_m@freemail.hu)
és Horváth Gergely (MSN: dzsoni8@freemail.hu)

Témavezető: Mógor Emil

A projekt: bemutatkozás

A feladatunk az volt, hogy készítsünk egy működőképes fórumot *Flash*-ben, amely *PHP*-t (vagy *Java*-t) használ a *Flash* és a *MySQL* adatbázis közötti kommunikációra (mi a *PHP* mellett döntöttünk). Ebben a dokumentumban a rendszer technikai megvalósításának pontos leírása található meg, a forráskód elemzésével együtt. Minden egyéb leírás a projektet leíró „Rendszer-dokumentáció”-ban van.

Az adatbázis: MySQL

A munka azzal kezdődött, hogy megterveztük az adatbázist, ami majd tárolni fogja a fórum adatait, úgymint a hozzászólásokat, a regisztrált felhasználókat és a fórumtémákat. Az adatbázist *phpMyAdmin*-nal készítettük el, mely egy nagyon egyszerű módot nyújt *MySQL* adatbázisok létrehozására és karbantartására. Adatbázisunk neve „**projekt**” lett, benne 3 tábla található:

- **hozzaszolások:** a felhasználók által írt hozzászólásokat tárolja, az alábbi mezőkkel:
 - *h_azonosito*: 5 bájtnál hosszabb integer, amely minden hozzászólást egyedileg azonosít
 - *h_datum*: a hozzászólás létrehozásának dátuma, másodpercre pontosan, a következő formátumban: ÉÉÉÉ-HH-NN ÓÓ:PP:MP
 - *h_szoveg*: a hozzászólás szövege, utf8_general_ci kódolást használva (mint az egész adatbázis) – itt meg kell jegyezni, hogy végig gondok voltak az ékezetes karakterek tárolásával és ez a kódolást bizonyult a legjobb megoldásnak, bár nem tökéletes (az „Á” és „Ő” karakterek helyett csak négyzetek jelennek meg)
 - *h_byuser*: annak a felhasználónak az azonosítója, aki a hozzászólást írta (egyenlő az *f_azonosito*-val)
 - *h_intopic*: annak a témának az azonosítója, amelybe a hozzászólást írták (egyenlő a *t_azonosito*-val)
- **temak:** a fórumban létrehozott témákat tárolja, mezői az alábbiak:
 - *t_azonosito*: a témát egyedileg azonosító 5 bájtnál hosszabb integer
 - *t_cim*: a téma neve, maximum 50 karakter hosszúságú lehet
 - *t_datum*: a téma létrehozásának dátuma, eredetileg úgy tűnt, hogy kelleni fog, de mégsem használjuk. Ugyanolyan formátumú, mint a *h_datum*.
- **felhasznalok:** a regisztrált felhasználókat itt tároljuk:
 - *f_azonosito*: szintén 5 bájtnál hosszabb integer, mely a felhasználókat egyedileg azonosítja
 - *f_teljesnev*: a felhasználó teljes neve, maximum 255 karakter hosszúságú szöveg lehet
 - *f_usernev*: a felhasználó user neve, mellyel beléphet a fórumba, szintén egyedi azonosító, nem lehet két egyforma felhasználónév a fórumban (max. 20 karakter hosszabb lehet)
 - *f_jelszo*: a felhasználó által regisztrációkor választott jelszó, MD5-el kódolva, mely azt jelenti, hogy még az adatbázis adminisztrátora sem tudja elolvasni senkinek a jelszavát, de egyben azt is jelenti, hogy jelszavak helyreállítására nincsen lehetőség (maximum 20 karakter hosszabb sztring lehet a jelszó)
 - *f_email*: a felhasználó e-mail címe, maximum 255 karakter hosszúságú sztring

Azt hiszem az adatbázisról többet nem nagyon lehet elmondani, ezért áttérünk a fórum grafikai megjelenésének leírására, utána pedig az *ActionScript* kód részletes leírására. Befejezésül a *PHP* kódot fogjuk kivesézni.

A fórum: *Flash*

A fórum annak ellenére, hogy *Flash*-ben lett elkészítve nem túl „dízajnos”. Animációkat nem használtunk, és a színvilág sem biztos, hogy mindenkinek ízlése. Ez azért alakult így, mert szeptemberben, amikor a projektet kezdtük még semmit nem tudtunk a *Flash* technológiáról. És ahelyett, hogy a grafikai nyalánkságokkal foglalkoznánk próbáltuk a „motort” minél jobban megírni. A fórum egy használat közbeni képét alább lehet látni:



Baloldalon található a menügombok, melyekkel az egyes funkciókat el lehet érni. Ezek sorban:

- **Regisztráció:** új felhasználók itt regisztrálhatják magukat
- **Belépés:** már létező felhasználók itt jelentkezhetnek be
- **Új téma:** új témát ezen menüpont segítségével lehet létrehozni (ld. kép fent)
- **Új hozzászólás:** ha a kiválasztott témához hozzá szeretnénk szólni, akkor itt tehetjük meg
- **Keresés:** a fórum teljes adatbázisában lehet szótöredék alapján keresni, eredményként a középső területen (az „olvasófelület”) megkapjuk azokat a hozzászólásokat, amelyekben megtalálható a megadott szótöredék.
- **Kijelentkezés:** ezzel a gombbal hagyhatjuk el a fórumot.

A Kijelentkezés gomb felett található még a bejelentkezett felhasználó neve... Ez valójában nem csinál semmit, csak ott van. Ha nincs felhasználó bejelentkezve, akkor „(nincs bejelentkezve)” szöveg található itt.

A felső legördülő menüből lehet kiválasztani azt a témát, amelyik tartalmát meg szeretnénk tekinteni. Ez alatt található a program „olvasófelülete”. Itt jelennek meg a kiválasztott témához tartozó hozzászólások és a keresés eredménye is itt lesz megjelenítve (a képen a „teszt1” téma tartalma található).

Jobboldalt legalul található a *Rendszerüzentek* panel, mely mindig az utolsóként elvégzett művelet állapotát írja ki. Pl.: új téma létrehozása esetén, hogy létrejött-e a téma vagy sem; keresésnél hogy hány találat volt.

A következőkben bemutatjuk az egyes panelek kinézetét, bármiféle szöveg nélkül, ugyanis minden funkció magától értetődő, aki tud olvasni, annak egyértelmű lesz:

Teljes név:

Felhasználónév:

Jelszó:

Jelszó még 1x:

E-Mail:

OK Mégse

Regisztráció

Felhasználónév:

Jelszó:

OK Mégse

Bejelentkezés

A keresendő szöveg:

OK Mégse

Keresés

Új téma címe:

OK Mégse

Új téma

Ide írd a hozzászólásod:

OK Mégse

Új hozzászólás

A „belső” kód: *ActionScript*

Ebben a fejezetben részletesen leírjuk az *ActionScript* kódot, ami a fórum egyik részét mozgatja, minden funkció részletes leírásával. Előre szeretnénk megjegyezni, hogy ha valaki nem ért egy függvényt vagy bármit, amit használunk, de nincsen elmagyarázva bátran fordulhat hozzánk, vagy keresse fel az alábbi helyek egyikét:

- Macromedia Flash 8 Professional HELP-je (a programon belül)
- www.actionscript.org

A *Flash* és *PHP* közötti kommunikációhoz egy *AMFphp* nevű „segédletet” használtunk, mely létrehoz egy „csővezeték” a *Flash* és a *PHP* között, melyben tömböket lehet egyszerűen átadni a két (egyébként teljesen különböző) nyelv között.

Akkor lássuk a kódot (elsőként mindig egy kódrészlet fog jönni – általában egy-egy függvény – majd utána annak a részletnek a pontos leírása):

```
import mx.remoting.*;
import mx.rpc.*;
import mx.remoting.debug.NetDebug;

var uid=-1; // User ID
var tid=-1; // Topic ID

var gatewayUrl:String = "http://localhost/amfphp/gateway.php"

NetDebug.initialize();
var _service:Service = new Service(gatewayUrl,null,'projekt',null,null);
```

A fenti részlet nem csinál mást, mint a szükséges dolgokat inicializálja, úgymint: a távoli eléréshez (*Flash-PHP* kommunikáció) szükséges függvénykönyvtárak betöltése (**import**), a felhasználó- és téma-azonosítók „-1”-re állítása (a felhasználó-azonosító azért -1, mert nincs senki bejelentkezve alapállapotban, a téma-azonosító pedig azért mert még nincs feltöltve a témákat tartalmazó legördülő menü). A következő 3 sor hozza létre az *AMFphp* segítségével való kommunikációhoz szükséges változókat és indítja a kapcsolatot. Ezek közül a legfontosabb talán a **_service** változó, ez később is elő fog kerülni.

```
var pc:PendingCall = _service.getTopics();
pc.responder = new RelayResponder(this, "handleGetTopics", "handleError");
```

És itt kezdődik az első tényleges kommunikáció a *PHP*-vel, majd a *MySQL* adatbázissal: a **PendingCall** típusú **pc** objektum a **_service**-en (a korábban nyitott csatorna) keresztül meghívja a *PHP*-nkben található **getTopics()** függvényt (leírása később a *PHP*-fejezetben). A *PHP*-ben található függvény egy asszociatív tömböt ad vissza a **pc** változó **responder** (válaszoló) metódusán keresztül. A **RelayResponder()**-nek mindig 3 paramétere van:

- az első jelenti, hogy hol akarjuk használni az eredményt (a lehetséges értékek közül néhány: **this**, **_parent**, **_root**)
- a második paraméter egy függvény neve, mely abban az esetben hívódik meg, ha a kérés sikeres volt és a *PHP*-től jött valami adat
- a harmadik paraméter szintén egy függvény, mely akkor aktivizálódik, ha probléma lépett fel a kommunikáció során (pl. az *AMFphp*-vel valami baj van)

A későbbiekben a fenti 2 sor elég sűrűn fog szerepelni, de többször nem írjuk le, hogy mit csinálnak. Általában a **RelayResponder** 2. paramétere lesz számunkra érdekes, ami a sikeres hívás esetén meghívott függvény.

```
textArea.text="<br><br><p align=\"center\"><i><b>Üdvözöllek a fórumban!  
      </b><br>Mielőtt bármit tudsz csinálni be kell  
      jelentkezned!</i><br>Ha még nem vagy regisztrálva,  
      először regisztrálj!</p>";
```

Az üdvözlőszöveg beállítása... A **textArea** változó jelképezi az „olvasófelületet”, a **.text** metódussal tudjuk módosítani, hogy mi kerüljön az „olvasófelületbe”.

```
topicList.addEventListener("change", topicListener);  
  
var topicListener:Object = new Object();  
topicListener.change = function(topic:Object) {  
    tid=topic.target.selectedItem.data;  
    var pc:PendingCall =  
        _service.getPosts(topic.target.selectedItem.data);  
    pc.responder =  
        new RelayResponder(_root, "handleGetPosts", "handleError");  
}
```

A **topicList** változó reprezentálja a felső legördülő menüt, ezen keresztül lehet elérni a témalistát. Ehhez adunk egy ún. esemény-figyelőt (**.addEventListener**, ld. később), mely az első paraméterként megadott esemény figyel (esetünkben a „változást”, tehát ha valaki egy másik témát választ ki). A 2. paraméter pedig megmondja, hogy a figyelt esemény bekövetkezésekor melyik függvény hívódjon meg. Tehát változáskor a **topicListener** függvény lesz meghívva, ami annyit csinál, hogy a **tid**-t (Topic ID) átállítja az újonnan kiválasztott téma azonosítójára, majd meghívja a **PHP getPosts()** függvényét, ami egy tömbben visszaadja az adott témához tartozó összes hozzászólás. Ezt kezeli majd a **handleGetPosts**.

```
function handleGetPosts(re:ResultEvent)  
{  
    if (uid===-1)  
    {  
        textArea.text="<br><br><p align=\"center\"><i>A témák  
            olvasásához először be kell jelentkezned!</i></p>"  
        errorBox_mc.errorBox="Először jelentkezz be!";  
    }  
    else  
    {  
        var i;  
        var e=Array();  
        e=re.result;  
        textArea.text="";  
        if (e.error==1)  
        {  
            textArea.text="<br><br><p align=\"center\"><i>A  
                kiválasztott témában nincsen hozzászólás!</i></p>";  
            errorBox_mc.errorBox="A téma üres...";  
        }  
        else {  
            errorBox_mc.errorBox=e.length + "db hozzászólás van  
                ebben a témában.";  
        }  
    }  
}
```

```

for (i=0;i<e.length;i++)
{
    textArea.text+="<b>" + e[i].f_usernev + "</b> írta <i>" +
        e[i].h_datum + "</i>-kor:";
    textArea.text+=e[i].h_szoveg + "<br><br>";
}
}
}

```

Ennek a függvénynek van egy bemeneti paramétere, a **re**, ami **ResultEvent** típusú (a **ResultEvent** objektum leírása a *Flash Remoting Components* és *AMFphp* dokumentációjában megtalálható). Ebben van eltárolva a *PHP* visszatérési értéke (általában egy tömb, „**return \$arr;**” formában elküldve). Elsőként megnézzük, hogy be van-e jelentkezve valaki, ha nincs, hibaüzenetet írunk. Ha már be van jelentkezve valaki (tehát az **uid** \neq **-1**), akkor létrehozunk egy tömböt és belepakoljuk a *PHP*-től kapott adatokat. Ha a *PHP* csak egy „1”-est küldött válaszként, akkor az azt jelenti, hogy nincsen hozzászólás az adott témában, ezt kiírjuk, különben pedig azt hogy hány darab hozzászólás van. Majd kiírjuk a kapott hozzászólásokat az olvasófelületre, egymás után. Elsőként a felhasználó nevét (az **e[]** asszociatív tömb **f_usernev** mezője, **e[i].f_usernev**), aki a hozzászólást írta, majd hogy mikor írta (**e[i].h_datum**) és végül magát a hozzászólást (**e[i].h_szoveg**).

```

function handleGetTopics(re:ResultEvent)
{
    var i;
    var e=Array();
    e=re.result;

    for (i=0;i<e.length;i++)
    {
        topicList.addItem({data:e[i].t_azonosito,label:e[i].t_cim});
    }

    if (tid===-1) tid=topicList.selectedItem.data;
}

```

Valójában a **handleGetTopics** függvénynek a **handleGetPosts** előtt kellett volna elhelyezkednie, de az *ActionScript*-nek mindegy hogy hol van a kódban egy adott részlet, csak az számít, hogy meglegyen. Tehát ez a függvény, mint már a neve is mutatja, feltölti (**topicList.addItem**) a legördülő menünket a témák neveivel. Ami itt érdekességként elmondható, hogy a legördülő menübe „láthatatlan” adatok is kerülnek (**data:**), amelyeket a *Flash* nem jelenít meg, de a kódon belül hivatkozhatunk rá (íly módon: **topic.target.selectedItem.data** – s kiválasztott elem „**data:**” mezője). Így kapjuk meg a **handleGetPosts** függvényben a téma azonosítóját például.

```

function handleError(fe:FaultEvent)
{
    trace('There has been an error: ' + fe);
    errorBox_mc.errorBox="TheRe HaS BeEn aN R-oR :) " + fe;
}

```

Ez a függvény kezeli, ha a távoli hívás hibára jutott. Valójában csak annyit tudunk tenni, hogy kiíratjuk vele, hogy mi volt a hiba.

```

regArea_mc._visible=false;
loginArea_mc._visible=false;
postArea_mc._visible=false;
topicArea_mc._visible=false;
searchArea_mc._visible=false;

```

Itt kezdődnek a grafikus megjelenítéssel kapcsolatos dolgok. Elsőként az összes panelt elrejtjük, ez az alapállapot. A panelek egy-egy *MovieClip*-ben vannak (ezt jelzi a név után álló **_mc**), és ezeket lehet a **.visible** paraméterrel előhozni/elrejtteni.

```

function areaVisible(melyik)
{
    if (melyik!=aktualis)
    {
        most = eval(aktualis);
        most._visible=false;
        most = eval(melyik);
        most._visible=true;
        aktualis=melyik;
    }
    else
    {
        this[melyik]._visible = ++this[melyik]._visible%2;
    }

    // Ablak tartalmának törlése a panel bezárásakor
    ablak = eval(melyik);
    switch (ablak) {
        case searchArea_mc : ablak.newSearch.text=""; break
        case topicArea_mc  : ablak.newTopic.text=""; break
        case postArea_mc   : ablak.newPost.text=""; break
        case loginArea_mc  : ablak.userName.text="";
                           ablak.pass.text=""; break
        case regArea_mc    : ablak.fullName.text="";
                           ablak.userName.text="";
                           ablak.pass1.text="";
                           ablak.pass2.text="";
                           ablak.eMail.text=""; break
    }
}

```

Az **areaVisible** függvénnyel lehet előcsalogatni a rejtett paneleket, a bal oldali gombok nyomogatásával. Minden bal oldali gombhoz egy hasonló „figyelő” van rendelve, mint a **topicList**-hez (csak itt „**change**” helyett **onRelease** van, tehát akkor történik valami, amikor felengedjük a gombot, nem amikor változik a kijelölés – ld. később). Paraméterként azt adjuk át, hogy melyik panelt akarjuk előhozni. Az **if-else** lényegében annyit valósít meg, hogy ha a panel még nem látható, akkor tegyük láthatóvá, ha látható, akkor rejtjük el, és ha egy másik panel van megnyitva, akkor először azt rejtjük el és nyissuk meg azt amelyik gombra a felhasználó nyomott. Ezen felül még van egy többes választó (**switch**) is, ami pedig törli az egyes panelek beviteli mezőiben található szöveget, ugyanis ezt a *Flash* nem teszi meg automatikusan és zavaró lehet.

```

function cancelFunction(melyik)
{
    ablak = eval(melyik);
    ablak._visible=false;

    switch (ablak) {
        case searchArea_mc : ablak.newSearch.text=""; break
        case topicArea_mc  : ablak.newTopic.text="";  break
        case postArea_mc   : ablak.newPost.text="";   break
        case loginArea_mc  : ablak.userName.text="";  break
        case regArea_mc    : ablak.pass.text="";      break
        case regArea_mc    : ablak.fullName.text="";
                          ablak.userName.text="";
                          ablak.pass1.text="";
                          ablak.pass2.text="";
                          ablak.eMail.text="";      break
    }
}

```

Ez a függvény hasonló az előzőhöz, de nem a bal oldali gombokra reagál, hanem a panelekben elhelyezett *Mégse* gombok válthatják ki. Tevékenysége egyszerű: bezárja az adott panel és törli a benne lévő tartalmat.

```

function regCheck()
{
    var i=0;

    if (regArea_mc.eMail.text=="")
        {errorBox_mc.errorBox="Add meg az e-mail címed!"; i++;}
    if (regArea_mc.pass2.text=="")
        {errorBox_mc.errorBox="Add meg még egyszer a jelszavad!"; i++;}
    if (regArea_mc.pass1.text=="")
        {errorBox_mc.errorBox="Add meg a jelszavad!"; i++;}
    if (regArea_mc.userName.text=="")
        {errorBox_mc.errorBox="Adj meg egy felhasználónevet!"; i++;}
    if (regArea_mc.fullName.text=="")
        {errorBox_mc.errorBox="Add meg a teljes neved!"; i++;}
    if (regArea_mc.pass1.text!=regArea_mc.pass2.text)
        {errorBox_mc.errorBox="A jelszavak nem egyeznek!"; i++;}

    if (regArea_mc.eMail.text.length>255)
        {errorBox_mc.errorBox="Az e-mail cím max. 255 karakter
        hosszú lehet."; i++;}
    if (regArea_mc.pass2.text.length>20)
        {errorBox_mc.errorBox="A jelszó max. 20 karakter
        hosszú lehet."; i++;}
    if (regArea_mc.pass1.text.length>20)
        {errorBox_mc.errorBox="A jelszó max. 20 karakter
        hosszú lehet."; i++;}
    if (regArea_mc.userName.text.length>20)
        {errorBox_mc.errorBox="A felhasználónév max. 20 karakter
        hosszú lehet."; i++;}
    if (regArea_mc.fullName.text.length>255)
        {errorBox_mc.errorBox="A teljes név max. 255 karakter
        hosszú lehet."; i++;}

    if (regArea_mc.userName.text == regArea_mc.pass1.text)
        {errorBox_mc.errorBox="A felhasználónévnek és
        jelszónak különböznie kell!"; i++;}
}

```

```

if (regArea_mc.userName.text == regArea_mc.pass2.text)
    {errorBox_mc.errorBox="A felhasználónévnek és
        jelszónak különböznie kell!"; i++;}

if (i>0)
{
    regArea_mc.gotoAndStop(1);
}
else
{
    var pc:PendingCall =
        _service.addUser(regArea_mc.fullName.text,regArea_mc.user
            Name.text,regArea_mc.pass1.text,regArea_mc.eMail.text);
    pc.responder =
        new RelayResponder(_root, "regResult", "handleError");
}
}
}

```

Itt kezdődnek az igazán izgalmas dolgok. Ez az óriási függvény kezeli, ha valaki regisztrálni szeretne a fórumba. Elsőként egy átfogó hibaellenőrzés veszi kezdetét, mely a következőkből áll:

- a regisztrációs ablakban egyik mező sem lehet üres, ez nézi az első 5 „**if**”
- a két megadott jelszónak egyeznie kell (+1 **if**)
- de nem csak azt nézzük, hogy üresek ne legyenek a mezők, hanem azt is, hogy egy bizonyos hossznál nagyobbak sem lehetnek (ld. az adatbázisban az egyes mezők hosszát) (+5 **if**)
- végül pedig leellenőrizzük, hogy a megadott jelszavak egyike sem egyezik-e a felhasználónévvel. Ide elég lenne egy **if** is, de biztos, ami biztos (+2 **if**)

Ezen feltételek mindegyike egy **i** változót növel ha probléma van. Amíg ez az **i** változó nagyobb mint 0 (tehát minimum egy hiba volt) nem enged továbblépni. Ha minden rendben van, akkor hívjuk a **PHP addUser()** függvényét a megfelelő paraméterekkel (a felhasználó adatai) és elkezdődhet az új felhasználó regisztrálása.

```

function regResult(re:ResultEvent)
{
    if (re.result==0)
    {
        errorBox_mc.errorBox="Sikereres regisztráció";
        regArea_mc._visible=false;
    }
    if (re.result==1)
        errorBox_mc.errorBox="Felhasználónév vagy e-mail
            cím már létezik";
}

```

A regisztráció kimenete kétféle lehet: sikeres (**result = 1**), tehát az új felhasználó adatai bekerültek az adatbázisba és innentől kezdve minden funkciót használhat. Vagy sikertelen (**result = 0**) a regisztráció, ez 2 ok miatt lehet: az adott felhasználónév vagy e-mail cím már létezik a rendszerben. Ebben az esetben más felhasználónevet kell megadni. Ha az sem megy, akkor pedig másik e-mail címet.

```

function loginFunction()
{
    var pc:PendingCall =
        _service.checkUserLogin(loginArea_mc.userName.text,
                                loginArea_mc.pass.text);
    pc.responder =
        new RelayResponder(_root, "handleCheckUserLogin",
                            "handleUserLoginError");
}

```

Mint a neve is mutatja, ez a függvény felelős a felhasználó bejelentkezéséért. Csak annyit csinál, hogy ha valaki a *Bejelentkezés*-panelen megadja a felhasználónevét és jelszavát, akkor azt elküldi a *PHP checkUserLogin()* (ld. később) függvényének, és ha megvan az eredmény, akkor meghívja a **handleCheckUserLogin** függvényt. A hibakezelés itt egy kicsit különbözik a többitől, mert itt nem a **handleError** függvény lesz meghívva (ami egy általános hibafüggvény), hanem a **handleUserLoginError**, amit később részletezünk.

```

function handleCheckUserLogin(re:ResultEvent)
{
    errorBox_mc.errorBox="Sikeres bejelentkezés";
    textArea.text="<br><br><p align=\"center\"><i>Gratulálok, bejutottál
                    a fórumba :)<br>Mostantól minden funkciót
                    használhatsz!</i></p>";
    loginArea_mc._visible=false;
    curuser=re.result[2];
    uid=re.result[0];
}

```

Ez a függvény kezeli le, ha sikeres volt a bejelentkezés, kiír egy üdvözlő üzenetet, elrejt a bejelentkező-ablakot, a bal oldalon a „*Felhasználónév*” alatt megjeleníti a bejelentkezett felhasználó nevét (**curuser**) és a User ID-t (**uid**) beállítja a bejelentkezett felhasználó azonosítójára (az adatbázisban: **f_azonosito**). Ez azért kell, hogy ha a felhasználó ír egy hozzászólást, akkor tudjuk hogy ki írta és később ennek megfelelően jelenítjük meg.

```

function handleUserLoginError(re:ResultEvent)
{
    errorBox_mc.errorBox="Hibás felhasználónév vagy jelszó";
    curuser="(nincs bejelentkezve)";
    uid=-1;
}

```

A **handleUserLoginError** mint a neve is mutatja azt az esetet kezeli, amikor a bejelentkezés sikertelen volt. Ez két esetben fordulhat elő: a felhasználó rossz felhasználónevet vagy rossz jelszót adott meg (esetleg mindkettő ☺). Ilyenkor a bal oldalon „*(nincs bejelentkezve)*” szöveg látható és a *User ID* „-1”-re lesz állítva, ezáltal tudatva a fórummal, hogy még nincsen bejelentkezve senki. Ha valaki elrontja a bejelentkezést, végtelen sokszor próbálkozhat újra, nem számoljuk, hogy hányszor rontotta el, és nem is lesz semmi következménye a sikertelen próbálkozásoknak.

```

function userLogout ()
{
    if (uid==-1) errorBox_mc.errorBox="Még be sem vagy jelentkezve... :)"
    else
    {
        uid=-1;
        curuser="(nincs bejelentkezve)";
        errorBox_mc.errorBox="Sikeres kijelentkezés";
        textArea.text="<br><br><p align=\"center\"><i>Köszönöm, hogy
                    a mi fórumunkat használtad!</i></p>";
    }
}

```

A név itt is magáért beszél, a felhasználók kijelentkezése itt történik. Ha a *User ID* -1, akkor valaki poénos akart lenni és úgy nyomta meg a kijelentkezés gombot, hogy be sem volt jelentkezve, ezt egy smiley-val „megköszönjük”. Egyébként a *User ID*-t beállítjuk „-1”-re, a **curuser**-t, a rendszerüzenetet és olvasófelületet pedig a megfelelő szövegekkel látjuk el.

```

function postFunction()
{
    if (uid==-1)
    {
        postArea_mc._visible=false;
        textArea.text="<br><br><p align=\"center\"><i>Új hozzászólás
                    írásához először be kell jelentkezned!</i></p>"
        errorBox_mc.errorBox="Új hozzászólás írásához először be
                    kell jelentkezned!";
    }
    else if (postArea_mc.newPost.text=="")
    {
        errorBox_mc.errorBox="Hozzá akarsz szólni de nem írtál
                    semmit? Hmmm...";
        postArea_mc.gotoAndStop(1);
    }
    else
    {
        var pc:PendingCall =
            _service.post(uid,tid,postArea_mc.newPost.text);
        pc.responder =
            new RelayResponder(_root, "handleNewPost", "handleError");
    }
}

```

Ha valaki új hozzászólást akar írni, akkor ez a függvény kezeli, hogy a hozzászólás bekerüljön az adatbázisba (pontosabban, hogy eljusson a *PHP*-ig, ami majd beírja az adatbázisba). Itt szeretnénk megjegyezni, hogy új hozzászólást csak az aktuálisan kiválasztott témába lehet írni. Elsőként le kell ellenőriznünk, hogy be van-e jelentkezve egyáltalán valaki. Ha nincsen, akkor a megfelelő hibaüzenettel térünk vissza. Ha már van felhasználó bejelentkezve (tehát az **uid** ≠ -1), akkor megírhatja a hozzászólását. Még annyi hibakezelést vittünk be ide, hogy üres hozzászólást nem lehet elküldeni, minimum 1 karakter hosszú szöveget kell írni. Ha ez teljesül, akkor hívjuk a *PHP post()* függvényét, ami megfelelően paraméterezve elhelyezi a hozzászólást az adatbázisban. A paraméterek a következők:

- **uid**: *User ID* – ki írta a hozzászólást
- **tid**: *Topic ID* – melyik témába írta a hozzászólást
- **postArea_mc.newPost.text** – a hozzászólás szövege

Ezek közül az első kettőt a fórum automatikusan ismeri, csak a szöveget kell megírnia a felhasználónak, hogy hozzászólása megfelelően bekerüljön az adatbázisba. Ezek mellett még az aktuális dátum is bekerül a hozzászólás mellé, amit a *PHP* generál (ezt ld. később a **post()** függvény részletezésénél).

```
function handleNewPost(re:ResultEvent)
{
    if (re.result==0)
    {
        errorBox_mc.errorBox="Hozzászólás elküldve";
        postArea_mc._visible=false;
    }
}
```

Ha a hozzászólásunk sikeresen bekerült az adatbázisba, akkor a *PHP* egy 0-val tér vissza (**return 0**). Ebben az esetben tájékoztatjuk a felhasználót, hogy a hozzászólása sikeresen el lett küldve, majd elrejtjük a Hozzászólás-panelt.

```
function topicFunction()
{
    if (uid== -1)
    {
        topicArea_mc._visible=false;
        textArea.text="<br><br><p align=\"center\"><i>Új téma
            nyitásához először be kell jelentkezned!</i></p>"
        errorBox_mc.errorBox="Ha új témát akarsz nyitni,
            akkor jelentkezz be!";
    }
    else if (topicArea_mc.newTopic.text=="")
    {
        errorBox_mc.errorBox="Ha új témát akarsz nyitni
            legyen már neve..."
        topicArea_mc.gotoAndStop(1);
    }
    else if (topicArea_mc.newTopic.text.length>50)
    {
        errorBox_mc.errorBox="A téma neve max. 50 karakter
            hosszú lehet!";
        topicArea_mc.gotoAndStop(1);
    }
    else
    {
        var pc:PendingCall =
            _service.topic(topicArea_mc.newTopic.text);
        pc.responder =
            new RelayResponder(_root, "handleNewTopic", "handleError");
    }
}
```

Ha valaki új témát szeretne nyitni, akkor ennél a függvénynél köt ki az **OK** gomb megnyomására. A hibaellenőrzések sorban a következők:

- **uid = -1**: nincs felhasználó bejelentkezve, tehát nem is nyithat új témát senki.
- **topicArea_mc.newTopic.text == ""**: valaki új témát akar nyitni, de nem adott neki nevet. Ilyet nem szabad csinálni. Minimum 1 karaktert meg kell adni.

- **topicArea_mc.newTopic.text.length > 50**: sajnos nem elég annyi, hogy a téma nevének min. 1 karakternek kell lenni, még az is le van korlátozva, hogy maximum 50 karakter lehet, ugyanis ha túl hosszú nevet adnánk meg, akkor nem férne ki a legördülő menüben 1 sorba, vagy nem látszana a vége, ami meg ronda lenne. Ez a korlátozás opcionális (de szerintünk logikus), külön kérésre lehet törölni.

Ha a fenti 3 feltétel teljesül, akkor az **OK** gomb hatására a rendszer létrehozza a felhasználó által kívánt új témát. Ezt a *PHP*-ben a **post()** függvény valósítja meg. Igazából a téma nevének kívül semmi extra adatra nincsen szükség, a téma azonosítóját a *MySQL* generálja. A téma létrehozásának a dátuma szintén automatikusan kerül be az adatbázisba, bár használva nem lesz.

```
function handleNewTopic(re:ResultEvent)
{
    if (re.result==0)
    {
        errorBox_mc.errorBox="Az új téma létrejött";
        topicArea_mc._visible=false;
        var pc:PendingCall = _service.getTopics();
        pc.responder =
            new RelayResponder(this, "handleGetTopics",
                               "handleError");
    }
    if (re.result==1) errorBox_mc.errorBox="Ilyen nevű téma már létezik";
}
```

Miután elküldtük az új témát a *PHP*-nek, az ellenőrzi, hogy van-e már ilyen nevű téma, ha van, akkor hibüzenetet írunk ki. Különben beírja az adatbázisba és 0-val tér vissza, ami ebben az esetben azt jelenti, hogy minden rendben volt és létrejött az új témánk. Majd ezután frissítjük a témák listáját, hogy az újonnan létrejött téma is bekerüljön és látszódjon.

```
function searchFunction()
{
    if (searchArea_mc.newSearch.text=="")
    {
        errorBox_mc.errorBox="És pontosan mit is akartál keresni?";
        topicArea_mc.gotoAndStop(1);
    }
    else
    {
        var pc:PendingCall =
            _service.search(searchArea_mc.newSearch.text);
        pc.responder =
            new RelayResponder(_root, "handleNewSearch", "handleError");
    }
}
```

Ez a függvény kerül meghívásra, ha valaki keresni akar valamit. Elsőként ellenőrizzük, hogy van-e keresendő szöveg (**searchArea_mc.newSearch.text**). Ha nincsen, akkor egy hibüzenetet kap a felhasználó (lehetséges lenne, hogy nem ellenőrizzük le, hogy írtak-e be valamit, csak ebben az esetben visszakapnánk az összes hozzászólást, ami a fórumban van és ez akár nagyon nagy adatmennyiséget jelenthet). Gondolkodtunk rajta, hogy ne 1 karakter legyen a minimum, hanem 3 vagy 4, mint a legtöbb fórumban, de a jelenlegi hozzászólás-mennyiség mellett nem tűnt szükségesnek. Később ez természetesen módosítható. Miután a felhasználó beírt minimum 1 karaktert a kereséshez meghívjuk a *PHP search()* függvényét, ami elvégzi a keresést az adatbázisban.

```

function handleNewSearch(re:ResultEvent)
{
    if (uid===-1)
    {
        searchArea_mc._visible=false;
        textArea.text="<br><br><p align=\"center\"><i>Ha keresni
            akarsz, akkor be kell jelentkezned!</i></p>"
        errorBox_mc.errorBox="Először jelentkezz be!";
    }
    else
    {
        var i;
        var e=Array();
        e=re.result;

        searchArea_mc._visible=false;
        textArea.text="";

        if (e.error==1)
        {
            textArea.text="<br><br><p align=\"center\"><i>Nincsen
                találat!</i></p>";
            errorBox_mc.errorBox="0 találat...";
        }
        else
        {
            errorBox_mc.errorBox=e.length + "db találat...";
        }

        for (i=0;i<e.length;i++)
        {
            textArea.text+="<b>" + e[i].f_usernev + "</b> írta <i>" +
                e[i].h_datum + "</i>-kor <i>" + e[i].t_cim + "</i>
                témában:";
            textArea.text+=e[i].h_szoveg + "<br><br>";
        }
    }
}

```

Elsőként itt is leellenőrizzük, hogy be van-e jelentkezve valaki. Ha igen, akkor mehetünk tovább. Miután megérkeztek a keresési eredmények a *PHP*-től elsőként berakjuk őket egy tömbbe (**e = re.result**), majd elrejtjük a keresőpanelt, és az olvasófelületet kiürítjük (**textArea.text=""**). Ha a *PHP* egy „1”-es számot adott csak vissza a tömbben, akkor ez azt jelenti, hogy nem volt találat. Ezt a megfelelő üzenet kiírásával jelezzük. Ha volt találat, akkor a *Rendszerüzenetek*hez kiírjuk, hogy hány darab volt és az olvasófelületre pedig kiírjuk a találatokat a következő formában (ezt végzi a **for** ciklus): *FELHASZNÁLÓNÉV* írta *DÁTUM (ÉÉÉÉ-HH-NN ÓÓ:PP:MP)*-kor *TÉMANÉV* témában. A csupa nagybetűvel írt változókat dőlt betűvel jelenítjük meg. Ezek után új sorban kezdve kiírjuk a hozzászólás teljes szövegét, amiben a keresett szótöredék megtalálható.

```

regBtn.onRelease = function() { areaVisible("regArea_mc") }
loginBtn.onRelease = function() { areaVisible("loginArea_mc") }
postBtn.onRelease = function() { areaVisible("postArea_mc") }
topicBtn.onRelease = function() { areaVisible("topicArea_mc") }
searchBtn.onRelease = function() { areaVisible("searchArea_mc") }
regBtn.onRelease = function() { areaVisible("regArea_mc") }
logoutBtn.onRelease = function() { userLogout() }

```

Lassan elérkezünk az *ActionScript* kód végére, már csak a gombok kezelői vannak vissza. Elsőként a bal oldali 7 gomb kezelője látható, ezek annyit csinálnak, hogy amikor az egyik gomb felett felengedtük (!!!) a bal oldali egérgombot (azért a felengedést nézzük, mert ez a megszokás, tessék nyugodtan tesztelni Windows, Linux, stb... alatt hogy nem a gomb lenyomására reagál a rendszer, hanem a felengedésre!) akkor a zárójelben megtalálható függvényt hívja meg egy paraméterrel. Ez a paraméter pedig az, hogy melyik panelt akarjuk megnyitni („melyik” változó az **areaVisible** függvényénél – ld. korábban).

```
topicList.addEventListener("change", topicListener);
```

Ezt az 1 sort azért írtuk külön a többitől, mert nem „hagyományos” gomb-felengedéssel működik, hanem azt figyeli, hogy a felső legördülő menüben változott-e a kijelölt téma. Ha igen, akkor meghívja a **topicListener** függvényt (ld. az *ActionScript* leírás elején). Sajnos itt van egy kis „bug”, amit nem tudjuk, hogy hogyan lehetne kijavítani: mivel csak a változást figyeljük, ezért ha valaki újra kiválasztja ugyanazt a témát, ami utoljára aktív volt nem történik semmi (a logikus az lenne, hogy újbóli kiválasztáskor frissítse a téma tartalmát). Próbáltuk többféle módszerrel megvalósítani ezt a *listenert* (**.onClick**, **.onRelease**, **”click”**), de egyikkel sem működött.

```
regArea_mc.okButton.onRelease =
    function() { regCheck() }
regArea_mc.cancelButton.onRelease =
    function() { cancelFunction("regArea_mc") }
loginArea_mc.okButton.onRelease =
    function() { loginFunction() }
loginArea_mc.cancelButton.onRelease =
    function() { cancelFunction("loginArea_mc") }
postArea_mc.okButton.onRelease =
    function() { postFunction() }
postArea_mc.cancelButton.onRelease =
    function() { cancelFunction("postArea_mc") }
topicArea_mc.okButton.onRelease =
    function() { topicFunction() }
topicArea_mc.cancelButton.onRelease =
    function() { cancelFunction("topicArea_mc") }
searchArea_mc.okButton.onRelease =
    function() { searchFunction() }
searchArea_mc.cancelButton.onRelease =
    function() { cancelFunction("searchArea_mc") }
```

Ezek a *listenerek* az egyes paneleken lévő **OK (.okButton)** és **Mégse (.cancelButton)** gombokat figyelik. Ha bármelyik meg lett nyomva, akkor az adott funkciónak megfelelő függvényt hívják meg. A különlegesség az, hogy az összes **Mégse**-gomb ugyanazt az egy függvényt hívja, csak különböző paraméterrel. A **cancelFunction** függvény leírását a fejezet elején lehet megtalálni, közvetlenül az **areaVisible** függvény után.

Ezzel végig is értünk az *ActionScript* kód tárgyalásán, minden funkciót részletesen kifejtettünk. Ha valakinek még nem tiszta valami az bátran forduljon hozzánk kérdéseivel! („Don’t Google the question, Moss!” – The IT Crowd, Season 1)

A végére egy kis statisztikát szeretnénk írni:

- a teljes *ActionScript* kód **369 sor** lett, *comment*-ekkel együtt
- *comment*-ek nélkül **341 sor**
- összesen **18** saját függvényt használtunk (ezek valósítják meg az egyes funkciókat)
- **18db listenerre** volt szükség, hogy az összes gombot kezeljük
- a fórum használata során kb. **40db** változó lesz inicializálva

A „külső” kód: PHP – Hypertext Preprocessor

Az *ActionScript* kód után a *PHP*-nk kódját szeretnénk részletesen megtárgyalni, ugyanis elég sokszor hivatkoztunk rá és végül is ez valósítja meg a kommunikációt a *MySQL* adatbázissal. Valahányszor egy „**PendingCall**” típusú objektumot hozunk létre, az azt jelenti, hogy egy *PHP* függvény kerül meghívásra következő lépésként. Ezek a függvények az *AMFphp* „**services**” mappájában egyetlen fájlban található meg, melynek neve esetünkben: „**projekt.php**”. Ennek a fájlnek az elérési útja van definiálva az *ActionScript* kód legelején a **gatewayUrl** változóban expliciten. Ugyanis nem közvetlenül az van írva, hogy a **projekt.php** hol található, hanem azt hogy az *AMFphp* melyik mappában van elhelyezve, innentől kezdve pedig a „**services**” mappában keresi az *AMFphp* a függvényeinket. Ennyi bevezető után kezdjük a kód elemzését:

```
<?php
class projekt
{
    function projekt()
    {
        $kapcsolat = mysql_connect('localhost', 'root', '')
        or die('Nem tudok csatlakozni: ' . mysql_error());
        mysql_select_db('projekt') or die('Nem sikerült kiválasztanom
            az adatbázist');

        $this->methodTable = array
        (
            "getTopics" => array
            (
                "access" => "remote",
                "description" => "Get Topics",
                "returns"=>"array"
            ),
            "getPosts" => array
            (
                "access" => "remote",
                "description" => "Get Posts",
                "returns"=>"array"
            ),
            "addUser" => array
            (
                "access" => "remote",
                "description" => "Add User"
            ),
            "checkUserLogin"=>array
            (
                "access" => "remote",
                "description" => "Check User Login",
                "returns"=>"array"
            ),
        ),
    }
}
```

```

        "post"=>array
        (
            "access" => "remote",
            "description" => "Add Post"
        ),
        "topic"=>array
        (
            "access" => "remote",
            "description" => "Add Topic"
        ),
        "search"=>array
        (
            "access" => "remote",
            "description" => "Search"
        )
    );
}

// Itt vannak a függvények, melyeket majd ezután fogunk részletezni
}

?>

```

Aki látott már *PHP* kódot annak nem lesz meglepő a `<?php ?>` kezdése és lezárása a fájlnak. A többi szintén ismerősnek fog tűnni, de ettől függetlenül magyarázatra szorul. Elsőként csatlakozunk az adatbázishoz a **mysql_connect()** függvénnyel, melynek 3 paramétere van: a szerver címe, a felhasználónév és a jelszó. Esetünkben ez *localhost*, *root* és jelszó nincsen. Miután a csatlakozás sikerült kiválasztjuk az adatbázist, melynek neve „projekt”. Ha ez is sikerült, akkor először definiáljuk a függvények neveit, azon belül pedig, hogy a függvényt hogyan lehet elérni (esetünkben ez mindig **remote** – tehát távoli elérés). Opcionálisan lehet adni egy leírást is a függvénynek, hogy tudjuk, hogy melyik mit csinál. Végül pedig azt, hogy mit ad vissza, ami legtöbb esetben egy tömb.

```

function getPosts($tid)
{
    $query="SELECT h_szoveg,h_datum,f_usernev FROM hozzaszolasok h
            INNER JOIN felhasznalok f ON h.h_byuser=f.f_azonosito
            WHERE h_intopic=$tid";
    $res=mysql_query($query);
    while ($row=mysql_fetch_assoc($res))
        $arr[]=$row;

    if (!isset($arr)) return array('error'=>'1');
    else return $arr;
}

```

Az első függvényünk belevág a közepébe, ugyanis ez lesz a legtöbbet meghívva. Mint a név is mutatja, ez kezeli a hozzászólások lekérdezését, mindig akkor lesz meghívva, ha valaki egy másik témát választ ki a felső legördülő menüből. A paraméter az újonnan kiválasztott téma azonosítója. Elsőként lefut egy *MySQL* lekérdezés, mely kiválasztja a hozzászólás szövegét és dátumát a „**hozzaszolasok**” táblából, és azt hogy ki írta (**f_usernev**) a „**felhasznalok**” táblából. Feltételnek pedig a kiválasztott téma azonosítója lesz figyelve. Ha a lekérdezés megvan az eredményt egy tömbbe pakoljuk. Ha a tömbbe nem került semmi (azaz nem volt találat) akkor mi kézzel belerakunk egy 1-est. Ezt majd a **handleGetPosts** figyeli. Ha nem üres a tömb, akkor visszaküldjük magát a tömböt a belepakolt tartalommal együtt.

```

function getTopics()
{
    $query="SELECT * FROM temat";
    $res=mysql_query($query);
    while ($row=mysql_fetch_assoc($res))
        $sarr[]=$row;

    return $sarr;
}

```

A **getTopics()** függvény hívódik meg a második legtöbbször. Először is kapásból lefut a fórum megnyitásakor. Utána valahányszor egy új témát hozunk létre. A *MySQL* kérés kiválaszt mindent a „**temak**” táblából, habár elég lenne csak a téma nevét és azonosítóját lekérni (azaz itt van egy kis felesleges adatmozgatás, de így gyorsabban meg lehetett írni a lekérdezést). A létrehozás dátuma továbbra is kihasználatlan marad. Ha az adatbázisból megjötték a témák, bepakoljuk őket egy tömbbe és elküldjük a *Flash*-nek.

```

function addUser($full, $user, $pass, $email)
{
    $enc_pass=md5($pass);
    $query=" INSERT INTO felhasznalok
            (f_teljesnev,f_usernev,f_jelszo,f_email)
            VALUES ('$full','$user','$enc_pass','$email)";
    $res=mysql_query($query);
    if ($res===false)
    {
        $err=mysql_errno();
        if ($err==1062)
        {
            return 1;
        }
    }

    return 0;
}

```

Az új felhasználó létrehozását ez a függvény végzi. Paraméterként megkapjuk a *Flash*-től a felhasználó teljes nevét, kívánt felhasználónevét, jelszavát és e-mail címét. Elsőként a jelszót leködoljuk MD5-el, hogy visszafejthetetlen legyen. Majd végrehajtjuk az adatbázison a beillesztést. Ha a beillesztés sikertelen volt, akkor valószínűleg azért, mert már létezik az adott felhasználónév vagy e-mail cím az adatbázisban (mindkét mező „**unique**”, tehát csak 1db lehet belőle az adatbázisban). Az *1062*-es *MySQL* hibaszám jelzi a dupla bejegyzést. Ha ilyet kapunk, akkor 1-est küldünk a *Flash*-nek, amit majd a **regResult** kezel, különben pedig 0-t (azaz minden rendben volt).

```

function checkUserLogin($user, $pass)
{
    $enc_pass=md5($pass);
    $query="SELECT * FROM felhasznalok
            WHERE f_usernev='$user' AND f_jelszo='$enc_pass'";
    $res=mysql_query($query);
    if (!$res) return array();
    $row=mysql_fetch_row($res);
    if (count($row)) return $row;

    else return array();
}

```

Ha valaki be akar jelentkezni a fórumba, akkor a **checkUserLogin()** függvény nézi meg, hogy az adatbázisban van-e olyan felhasználónév és hozzá tartozó jelszó, amit paraméterként a *Flash*-tól kapott (tehát amit a felhasználó begépelte). Itt is elsőként elkódoljuk a jelszót MD5-el, majd lekérdezzük az adatbázisból az összes olyan sort, amiben az **f_usernev** megegyezik a megadott névvel és az **f_jelszo** megegyezik a lekódolt jelszóval. Ha ilyen nincs egy üres tömböt küldünk vissza. Ha van, akkor (csak 1 ilyen sor lehet, ezért elég a **mysql_fetch_row**) belerakjuk egy változóba (**\$row**). Ha az egy sor bekerült a **\$row** változóba, akkor azt visszaküldjük.

```
function post($user,$topic,$text)
{
    $query=" INSERT INTO hozzaszolasok
            (h_byuser,h_intopic,h_szoveg,h_datum)
            VALUES ($user,$topic, '".$text."',
                    '".date("Y-m-d H:i:s")."');"
    $res=mysql_query($query);

    return 0;
}
```

Az új hozzászólásokat szintén egy **INSERT** függvénnyel lehet belerakni az adatbázisba. Ezt végzi ez a függvény. Paraméterként megkapja a *Flash*-tól a bejelentkezett felhasználó azonosítóját (**\$user: uid – f_azonosito**), a téma azonosítóját, amibe a hozzászólást szeretnénk írni (**\$topic: tid – t_azonosito**) és természetesen a hozzászólás szövegét (**\$text: postArea_mc.newPost.text – h_szoveg**). Ezen 3 paraméter mellett még a hozzászólás létrejöttének dátuma is bekerül az adatbázisba, amit a *PHP date()* függvénye generál, az alábbi szabály szerint: **Y-m-d H:i:s**, ahol az **Y** jelenti az évet 4 karakteres formában (ÉÉÉÉ), **m** a hónapot 2 karakterben (HH) és **d** a napot szintén 2 karakterben (NN). A **H** jelenti az órát 24 órás formátumban (tehát 0-24-ig), **i** a percet és **s** a másodpercet. Ha a hozzáadás megtörtént, akkor 0-val tér vissza a függvény.

```
function topic($title)
{
    $query=" INSERT INTO temak (t_cim,t_datum)
            VALUES ('".$title."', '".date("Y-m-d H:i:s")."');"
    $res=mysql_query($query);
    if ($res===false)
    {
        $err=mysql_errno();
        if ($err==1062)
        {
            return 1;
        }
    }

    return 0;
}
```

A **topic()** függvény végzi az új téma felvitelét az adatbázisba. Ide elég egy paraméter, ami a létrehozandó téma nevét tartalmazza. A lekérdezés egy esetben futhat hibára, méghozzá ha már ilyen nevű téma létezik az adatbázisban (*1062-es MySQL hiba – a t_cim szintén unique* kell, hogy legyen). Ebben az esetben egy 1-est kap a *Flash*, különben az új téma sikeresen létrejött és 0-t küldünk.

```

function search($text)
{
    $query=" SELECT f_usernev,h_datum,h_szoveg,t_cim
            FROM hozzaszolasok,temak,felhasznalok
            WHERE h_intopic=t_azonosito
                  AND h_byuser=f_azonosito
                  AND h_szoveg LIKE '%" . $text . "%'";
    $res=mysql_query($query);
    while ($row=mysql_fetch_assoc($res))
        $arr[]=$row;

    if (!isset($arr)) return array('error'=>'1');
    else return $arr;
}

```

És végül de nem utolsó sorban: a keresés. Paraméterként a *Flash* a keresendő szöveget küldi. A lekérdezés kiválasztja az adatbázisból az összes olyan hozzászólást, amelyben megtalálható az adott szövegrészlet (**LIKE** parancs). De természetesen nem csak a hozzászólás szövege lesz elküldve a *Flash*-nek, hanem az is hogy ki írta, mikor és melyik témában. Ha a lekérdezés lefutott az eredményt berakjuk egy tömbbe. Ha a tömb üres maradt (azaz nem volt ilyen hozzászólás) akkor az **if(!isset(\$arr))** ezt felfedi, belerak egy 1-est a tömbbe és azt küldi el (a *Flash*-ben pedig figyeljük, hogy ha 1-est kapunk, akkor az azt jelenti, hogy nincsen találat), egyébként pedig magát a tömböt az összes megtalált hozzászólással.

Ezzel a *PHP* kód tárgyalásának végére is értünk. Reméljük, hogy minden érthető volt. Ha mégsem, akkor várjuk a kérdéseket! És végül itt is következzen egy kis statisztika:

- összesen **142** sor a projekt.php
- **7**db *MySQL* lekérdezés kezeli az összes akciónkat
- melyek **7+1** db függvényben foglalnak helyet
- és **41**db változó lesz inicializálva ezen függvények futása során

Végül

Összességében köszönetet mondunk mindenkinek, akik vették a fáradságot és magukat nem kímélve megismerték egy fórum készítésének alapjait. Reméljük sok hasznos tanáccsal és megvalósításra váró elképzeléssel szolgált a munkánk. Bízunk benne, hogy a projekt megértése során mindenki tud véleményt alkotni a témában, sőt megtisztelve éreznénk magunkat, ha tudunkra is adnák őket.

„Csak azért mert valami nem úgy működik, ahogy tervezted, még nem biztos, hogy haszontalan.”
- Thomas Edison