

# Perl AJAX MySQL HASZNÁLATA DINAMIKUS WEBLAPOKNÁL\*

SZIMKOVICS TAMÁS

Ungvári 10. Számú Dayka Gábor Középiskola, rendszergazda

Újabban már a weboldalak többsége dinamikus, tehát változó tartalmat megjelenítő. Ám létrehozása sokszor bonyolalmakkal jár, mivel kevés a részletes vagy magyar nyelvű leírás a témában. A munkámban leírt weblapmotorban egy kezdetleges dinamikus weblap-menü van leprogramozva. A munkám során a dinamikus tartalom változtatását AJAX-technológia segítségével valósítottam meg úgy, hogy a menüpontok tartalma egy MySQL adatbázisban találhatóak. A szükséges műveleteket pedig Perl-programnyelven vannak megírva.

## ABSTRACT

Останнім часом більшість інтернет-сторінок є динамічними, тобто відображають зміст, що змінюється. Створення таких сторінок є складним процесом через малу кількість детальних або узорськомовних описів за цією темою. У веб-двигуні, описаному в нашій роботі, запрограмоване початкове динамічне веб-меню. У процесі роботи зміну динамічного змісту реалізовано з допомогою технології AJAX таким чином, що зміст розділів меню знаходиться в табличних даних MySQL. Необхідні операції були написані мовою програмування Perl.

## FELHASZNÁLT ESZKÖZÖK RÖVID BEMUTATÁSA

A MySQL egy gyors, többszálú, több felhasználós robusztus SQL adatbázis-szerver. Ebben az adatbázisban fog tárolódni a weblap változó tartalma. Adatbázis létrehozása és kezelése grafikus (*phpMyAdmin*), illetve szöveges parancsértelmezővel (*Terminal* a Linux alapú rendszereknél) is megvalósítható.<sup>1</sup>

A Perl nyelv egy interpretált, azaz betöltéskor fordított nyelv. Eredetileg rendszeradminisztrációs feladatok megkönnyítésére írta Larry Wall, mert nem volt kedve a meglévő eszközök korlátaival bajlódni.

A nyelvet könnyű elsajátítani, mert:

1. ismert nyelvi elemeket tartalmaz
2. kis rész ismerete is elég a használatához

\* A tanulmányt Beregszászi István lektorálta.

<sup>1</sup> R4s: A MySQL 3, 1999.

<http://prog.hu/cikkek/3/A+MySQL.html> (Hozzáférés időpontja: 2014.02.18, 21:08)

3. mindent azonnal ki lehet próbálni, mert gyors<sup>2</sup>

A munkám során Perl-programon keresztül kapcsolódtam egy MySQL adatbázishoz, ezt a Perl egy moduljának segítségével valósítottam meg.

A Perl adatbázis-kezelésnek a magját képező modult *DBI*-nak (*DatabaseInterface*-nek) hívják, az egyes adatbázisokhoz pedig különböző *DBD* (*Database Driver*)-modulok segítségével kapcsolódhatunk. Ezeknek a moduloknak a célja egy alacsony szintű egységes interfész biztosítása a különböző gyártó specifikus adatbázis kapcsolati lehetőségekhez.<sup>3</sup>

Az AJAX nem egy új technológia, hanem több különböző, önállóan is remekül használható webtechnológia összessége:

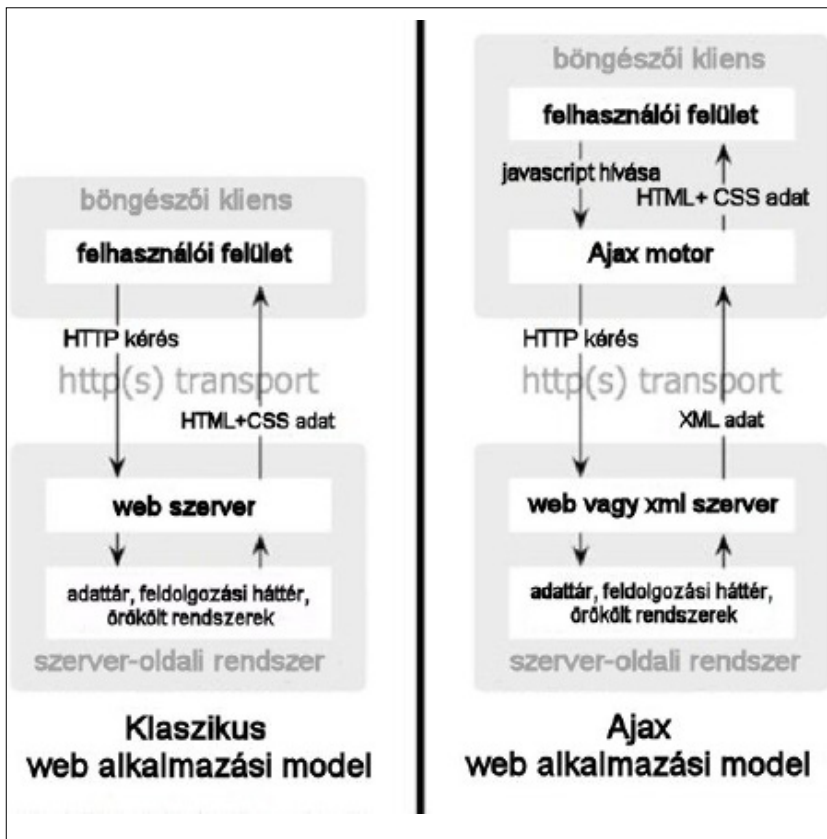
<sup>2</sup> Frohner Ákos: Perl.

<https://www.szabilinux.hu/ismerteto/perldoc.html> (Hozzáférés időpontja: 2014.02.18, 21:24)

<sup>3</sup> Bárházi András: Perl alapjai, • 2005.

<http://weblabor.hu/cikkek/perl alapjai> (Hozzáférés időpontja: 2014.02.18, 21:28)

- *XHTML* és *CSS* a szabványosított megjelenítése
- Dinamikus megjelenítés, struktúra-változtatás a *DOM* (DocumentObject-Model) segítségével
- Adatmanipuláció, módosítás az *XML* segítségével
- *XMLHttpRequest* objektum az aszinkron adatkommunikációért – ebben rejlik az *AJAX* lényege
- És *Java* szkript a kliens oldali adatfeldolgozásért, valamint az előző technológiák összefogásáért<sup>4</sup>



1. ábra. Szinkron vs. aszinkron kommunikáció (az eredeti kép URL címe : [http://www.jampaper.eu/Jampaper\\_H-ARC/2010.\\_V.\\_1.sz.\\_files/JAM100103h.pdf](http://www.jampaper.eu/Jampaper_H-ARC/2010._V._1.sz._files/JAM100103h.pdf), h. i.: 2014.02.18)

Így együtt igen erős csapatot alkotnak, innen a név *Asynchronous JavaScript And XML*

A technológia lényege, hogy az *AJAX* az *XMLHttpRequest* *Java* szkript objektumot használva úgy tud adatot cserélni a webszerverrel, hogy az adott weboldal nem töltődik újra (1. ábra). Az *AJAX* aszinkron adatszeret hajt végre a böngésző és a webszerver között (*HTTP request-ek* segítségével), ezáltal elérve azt, hogy a teljes weblap letöltése helyett, annak csak bizonyos részinformációit frissíti.

A klasszikus és az *AJAX* kommunikáció közötti különbséget az 1. ábra szemlélteti.

## PERL AJAX MYSQL A GYAKORLATBAN

Szükséges előkészületek:

- Egy telepített *Linux* alapú operációs rendszer, jelen esetben *Ubuntu*<sup>5</sup>
- Telepített programcsomagok: *Perl*, *MySQL*, *Apache 2* web-szerver (LAMP szerver), *Perl DBI*, *Perl DBD:MySQL*, *phpMyAdmin*.<sup>6</sup>

<sup>4</sup> Nagy Zsolt: *AJAX 2 in 1: interaktív oktatás és modern webtechnológia*. [http://www.jampaper.eu/Jampaper\\_H-ARC/2010.\\_V.\\_1.sz.\\_files/JAM100103h.pdf](http://www.jampaper.eu/Jampaper_H-ARC/2010._V._1.sz._files/JAM100103h.pdf) (Hozzáférés időpontja: 2014.02.18, 21:08)

<sup>5</sup> Dr. Blahota István: *Ubuntu Linux kezdőknek*. [http://zeus.nyf.hu/~blahota/ubuntu/Linux\\_11\\_10\\_06.pdf](http://zeus.nyf.hu/~blahota/ubuntu/Linux_11_10_06.pdf) (Hozzáférés időpontja: 2014.02.18, 21:08)

<sup>6</sup> LAMP szerver telepítésének leírása: [http://intermatrix.hu/lamp\\_server](http://intermatrix.hu/lamp_server) (Hozzáférés időpontja: 2014.02.18, 21:11)

A weblapmotor működésének rövid ismertetése: a weblapunkon rákattintunk a menü valamely pontjára, mindegyik menüpont rendelkezik valamilyen számértékkel. Ezt az értéket AJAX szkript és *form* tag segítségével, elküldjük egy Perl CGI szkriptnek. A Perl szkript csatolozik egy MySQL adattáblához. Majd a küldött értéknek megfelelően leolvassa a tábla megfelelő tartalmát (a weblap menüpontjának tartalmát). Továbbá a Perl szkript a leolvasott tartalmat kiírja (megjeleníti), amelyet majd az AJAX szkript beszúr a weblap megfelelő részébe.

Először is tisztázni kell a futatható állományok elérési útját. Alapértelmezés szerint

a CGI-fájlokat a következő könyvtárban futathatóak */usr/lib/cgi-bin*, a web-szerver gyökérkönyvtára viszont a */var/www*. Ezeknek a könyvtáraknak a helyét mi magunk is megszabhatjuk a */etc/apache2/sites-available/default* fájl segítségével<sup>7</sup>, így nálam a CGI-fájlok *home/tommy/public\_html/cgi-bin* futathatóak, a web-szerver gyökérkönyvtára pedig a *home/tommy/public\_html* lett (a saját könyvtárakban könnyebb dolgozni a jogosultságok miatt)

A főoldalt a következő Perl-kód fogja megjeleníteni:

```
#!/usr/bin/perl
use CGI;
use CGI::Carp qw/fatalsToBrowserwarningsToBrowser/;
$q = new CGI;
print<<end_html
<html>
<head>
<title>Weblap</title>
<script type = "text/javascript" language = "javascript"
src = " ../js/tartalom.js "> </script>
</head>
<body onload = "menu(this.name = '1')">
  <form>
    <ul>
      <li>
        <a href = "#" onclick = "menu(this.name)" name = "1">
          Főoldal
        </a>
      </li>
      <li>
        <a href = "#" onclick = "menu(this.name)" name = "2">
          Másodikmenüpont
        </a>
      </li>
      <li>
        <a href = "#" onclick = "menu(this.name)" name = "3">
          Harmadik menüpont
        </a>
      </li>
    </ul>
    <br>
    <div id = "tartalom">
    </div>
  </form>
</body>
</html>
end_html
```

### 1. forráskód. Az állandó változó tartalmat megjelenítő Perl-állomány (*index.pl*)

<sup>7</sup> VirtualHost-ok létrehozása, beállítása: <http://intermatrix.hu/virtualhost>

Vizsgáljuk meg a fenti kódot.

Az első sor megadja a kód lefutásához szükséges szoftver elérési útját, jelen esetben a *Perl*-program könyvtárát.

A második sorban felhasználásra kerül a *Perl CGI* könyvtára (általános átjáró felület): a *Perl* fő felhasználási területe, ez egy felület, amely összeköti a weblapot a *HTTP* szerverrel és ezen keresztül a felhasználóval.

A harmadik sor a szkript működéséhez nem szükséges, viszont sokat segít a kód írásában. A `USE CGI::Carp qw/fatalToBrowser warningsToBrowser` könyvtár, hiba esetén a böngészőben pontosan kiírja a hiba forrását, melyik sorban történt a hiba és mi okból. Így jóval kevesebb idő megy el a hiba keresgetésével és javításával.

A negyedik sorba deklarálódik a `$q` *CGI*-változó. Ezen a változón keresztül fog kommunikálni a felhasználó a szerverrel.

Majd jön a *HTML*-kód kiírása. Ezt többsoros kiíratással oldottam meg a

```
print<< karaktorsor
    tartalom
    karaktorsor
```

parancs segítségével.

A *HTML*-kódba a szokásos elemek tag-ek találhatóak: (`html`, `head`, `body`,...). A `body` tag-ben egy `onload` eseményre reagáló JavaScript található, amely az oldal betöltésekor a főoldal tartalmát fogja lekérni.

Majd egy `form` tagben található egy lista, amely a menüket képzi, három menüpon-tunk van: Főoldal, Második menüpont, Harmadik menüpont. A `form` tag végén található a *tartalom* nevezetű `div` tag: a változó tartalom ide fog kerülni. A *HTML*-kód elején be van szűrve egy *tartalom.js* nevezetű java szkript. Ez egy *AJAX*-kód, amely fogadja az *index.pl* által küldött `q` paramétert és azt tovább adja egy másik feldolgozó szkriptnek. Az *AJAX*-kód a következő képen néz ki:

```
function menu(str) {
    if (str == "") {
        document.getElementById("tartalom").innerHTML = "";
        return;
    }
    if (window.XMLHttpRequest) {
        xmlhttp = newXMLHttpRequest();
    } else {
        xmlhttp = newActiveXObject("Microsoft.XMLHTTP");
    }
    xmlhttp.onreadystatechange = function() {
        if (xmlhttp.readyState == 4 &&xmlhttp.status == 200) {
            document.getElementById("tartalom").innerHTML = xmlhttp.responseText;
        }
    }
    xmlhttp.open("GET", "menu.cgi?q = "+str,true);
    xmlhttp.send();
}
```

## 2. forráskód. AJAX szkript menük váltására (tartalom.js)

A kódelejen definiálunk egy `menu(str)` függvényt, melyben egy eldöntési `if (str == "")` utasításnak köszönhetően üres paraméter megadásakor a függvény nem végez semmilyen műveltett sem.

Megadott paraméternél létrehozuk `XMLHttpRequest` kommunikációs objektumot a

legtöbb böngésző számára: `if (window.XMLHttpRequest) xmlhttp = newXMLHttpRequest();` Ha a felhasználó régebbi *Internet Explorer* böngészőt használ, akkor a következő képen jön létre az objektum: `else { xmlhttp = newActiveXObject("Microsoft.XMLHTTP"); }`

Vegyük most a lekérdező rész működését. Ha a CGI-állománytól érkezik a válasz, akkor az belekerül a *tartalom id-jű* elembe. Az `xmlhttp.onreadystatechange = function()` kódsor teszi lehetővé, hogy amikor a szerverről visszaérkezik a válasz, az irányítást `function()` függvény kapja meg. Ez a művelet még egy feltételhez van kötve, ami az `(xmlhttp.readyState == 4 && xmlhttp.status == 200)`. Ez azt jelenti, hogy ha a `readyState` egyenlő 4-el, tehát kész a kérés és a válasz is készen áll, és a `status`-nak az értéke 200 vagyis létezik az oldal, akkor hajtódik végre a `xmlhttp.onreadystatechange = function()` utasítás. A `function()` anonim függvény dolgozza fel a visszaérkező választ. Az `xmlhttp.open()` függvény a kérést hajtja végre a *GET*-metódussal és az elküldi azt a *cgi-bin* mappában található *menu.cgi* kódállományhoz a `q` paraméter `str` értékével. A `true`

paraméter pedig az aszinkron kommunikációt teszi lehetővé. Végül az `xmlhttp.send()` függvény végzi el a kérés tényleges elküldését.

Tekintsük meg a *menu.cgi* állományt:

```
#!/usr/bin/perl
use CGI;
use DBI;
use DBD::mysql;
local ($buffer, @pairs, $pair, $name, $value, %FORM);
# Küldött érték olvasása
$ENV{'REQUEST_METHOD'} =~ tr/a-z/A-Z/;
if ($ENV{'REQUEST_METHOD'} eq "GET") {
    $buffer = $ENV{'QUERY_STRING'};
}
# Név / érték válogatása
@pairs = split(/&/, $buffer);
foreach $pair (@pairs) {
    ($name, $value) = split(/=/, $pair);
    $value =~ tr/+//;
    $value =~ s/%(..)/pack("C", hex($1))/eg;
    $FORM{$name} = $value;
}
my $szam = $FORM{q};
# adatbázis leírása
my $adatbázis = "weblap";
my $kiszolgalo = "localhost";
my $port = "3306";
my $felhasznalo = "root";
my $jelszo = "1";
my $kapcsolodasiInfo = "DBI:mysql:database = $adatbázis;";
$kiszolgalo: $port;
# csatlakozás az adatbázishoz
my $kapcsolat = DBI->connect($kapcsolodasiInfo, $felhasznalo,
    $jelszo);
# megfelelő sor kiválasztása
my $sth = $kapcsolat->prepare("SELECT html FROM menu
    WHERE id = , $szam'");
$sth->execute();
$result = $sth->fetchrow_hashref();
print "Content-Type: text/html; CharSet = utf-8\n\n";
print "$result->{html}\n";
# befejezés
$parancs->finish();
# kapcsolat bontása az adatbázissal
$kapcsolat->disconnect;
```

3. forráskód. A menüpontok tartalmát kiírató CGI-állomány, *menu.cgi*

Vizsgáljuk meg a fenti kódot. A forráskód elején deklaráljuk a használni kívánt DBI, DBD : :mysql adatbázis kezelő könyvtárakat.

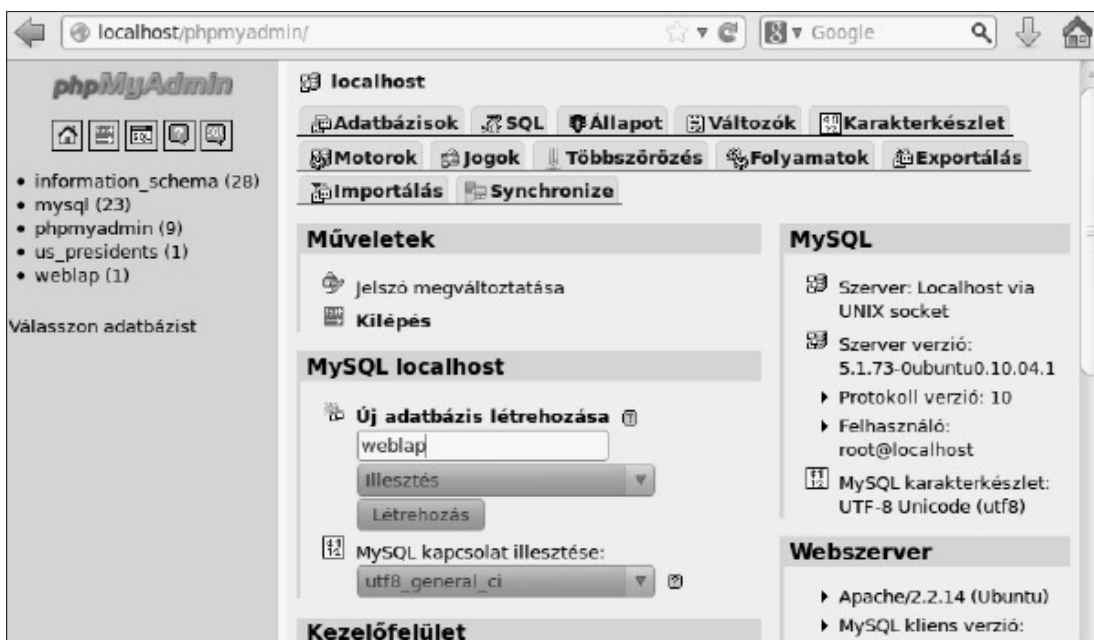
Utána jön egy hosszabb szkript, amely beolvassa a küldött \$q változót. Ezt következőképpen teszi: először is definiálva vannak, local változók, tömbök és egy asszociatív tömb (hash), a szkript elején megvizsgálja a küldés metódusát, esetünkben ez a GET, majd beolvassa azt egy sztringbe, egy \$buffer változóba. Utána jön a kulcs / értékpárok szétválogatása: először is az & (az & jelöli kulcs / értékpárok közti részt) jeleknél a szöveges tartalmat szétvágja és a @paris tömbbe teszi azt, majd egy foreach ciklusban sorra veszi kulcs / értékpárokat és az = (az = jel a kulcs és érték között helyezkedik el) jelnél különválasztja a kulcsoktól az értékeket. Mivel a küldés folyamán a speciális és ékezetes karakterek felcserélődnek, illetve a szóközők + karakterekkel helyettesítődnek ezért ezeket a beolvasás végén visszacserélődnek a tr parancs segítségével. A ciklus végén történik kulcs és érték összekapcsolása. A beolvasás

után a \$szam változóba helyezzük a küldött q változót.

Ezek után következik az adatbázis csatlakozásának előkészítése a szükséges adatok megadásával a megfelelő változókba: adatbázis neve, kiszolgáló, port, felhasználó neve, felhasználó jelszava. Majd a deklarált értékek segítségével a \$kapcsolat változóban megtörténik az adatbázishoz való kapcsolódás.

Most az adatbázis megfelelő adattáblájához kell egy lekérdezést intézni. A menu adattábla kiválasztása után a \$szam id-jű record html a mező tartalmát, egy \$sth változóba írjuk ki. Az \$sth változóban lévő tartalom lesz a weblap változó tartalma, tehát HTML kódot tartalmaz. Majd csatlakozunk a HTML nyelvezethez és kiírjuk a változó tartalmát, amelyet az AJAX szkript visszaküld az index.pl állomány *tartalom* azonosítóval rendelkező div tag-be.

Nézzük meg, hogyan hozzuk létre és töltjük fel a weblap adatbázis menu adattábláját. A böngésző címsorába írjuk be a *localhost/phpmyadmin*



2. ábra. phpMyAdmin adatbázis létrehozása

címet. Belépéskor kérni fogja a felhasználónevet, illetve jelszavunkat. Ezek megadása után létrehozhatjuk az adatbázisunkat (2. ábra):

Első lépésként megadjuk az adatbázis nevét és rákattintunk a *Létrehozás* gombra, második lépésben kérni fogja az adattábla nevét és a mezők számát. Nálam az adattábla neve *menu*, a mezők (oszlopok) száma pedig kettő. Adatok bevitelével végezve rá kell kattintani az *Indítás* gombra. Következő lépésben be kell állítani a mezők típusát, nevét illetve tulajdonságait. Az első neve legyen *id*, a másodiké *html*. Sorban az első mező típusa legyen, *INT* (számtípus), a másodiké pedig *TEXT* (szövegtípus). Még az első mezőnél pipáljuk ki az *A\_I* részt (arra szolgál, hogy a mezőben újonnan létrehozott recordokat (sorokat) számozza meg 1-től kezdve) és állítsuk be elsődleges mezőnek azaz *PRIMARY*-nek (műveleteknél először ehhez a mezőhöz fog fordulni). Az így beállított adattáblát mentjük el. Most keressük ki és nyissuk meg a táblát. A felső sarokban lesz egy „*Beszúrás*” gomb kattintunk rá. Itt tölthetjük fel az adattáblánkat. Az *id* mezőt üresen hagyjuk mivel annak csak számláló funkciójára van szükség. A *html* mezőkben adjuk meg a weblap változó tartalmának a *HTML* kódját, az első *recordb* a főoldalét. Elegendő a `<body>` és `</body>` közötti rész a második és harmadik *recordb* pedig a „Második menüpont” és „Harmadik menüpont” tartalmát. A cellák feltöltése után kattintsunk az *Indítás* gombra.

Az adattáblánk fel lett töltve. Vázlatos alakban a következőképpen néz ki:

1. táblázat. A menü adattábla felépítése

| id | html                                |
|----|-------------------------------------|
| 1  | Főoldal <i>HTML</i> kódja           |
| 2  | Második menüpont <i>HTML</i> kódja  |
| 3  | Harmadik menüpont <i>HTML</i> kódja |

A szükséges programkódok megvannak már, csak egy kis magyarázatra szorulnak. Az első forráskódban a menüpontoknak van *name* paramétere, tehát nevük ezek sorban az 1, 2, 3 számok. Tehát amikor rákattintunk a menüpontunkra, elindul a `menu(This.name)` eseményfüggvény, amely a *tartalom.js* szkriptben található. Az ott lévő szkript elküldi a paramétert a *menu.cgi* állománynak, az csatlakozik a *weblap* adatbázis *menu* adattáblájához majd a *html* mezőjében a megfelelő recordot kikeresi és a benne lévő tartalmat visszaküldi az *index.pl* állomány *tartalom* azonosítóval rendelkező `div` tagbe. Az ilyen típusú szerkesztés nehézségeket okozhat. A változó tartalmat csak *phpMyAdmin* segítségével lehet szerkeszteni, amelyet problémás távolról elérni, tehát bonyodalmas a weblapfrissítése, szerkesztése. Hátrányból előny kovácsolható, ha a már ismert technológia segítségével létrehozunk egy adminisztrációs felületet. Természetesen a weblap szerkesztéséhez és az adminisztrációs felület megnyitásához egy beléptető rendszere is szükség van, de azt itt külön nem tárgyalom.

Először is hozzunk létre egy negyedik menüpontot az *index.pl* állományban, amelynek a neve legyen *Szerkesztés* a *name* paramétere pedig 4-es:

```
<li>
    <a href = „#” onclick = “menu(this.
      name)” name = „4”>
      Szerkesztés
    </a>
</li>
```

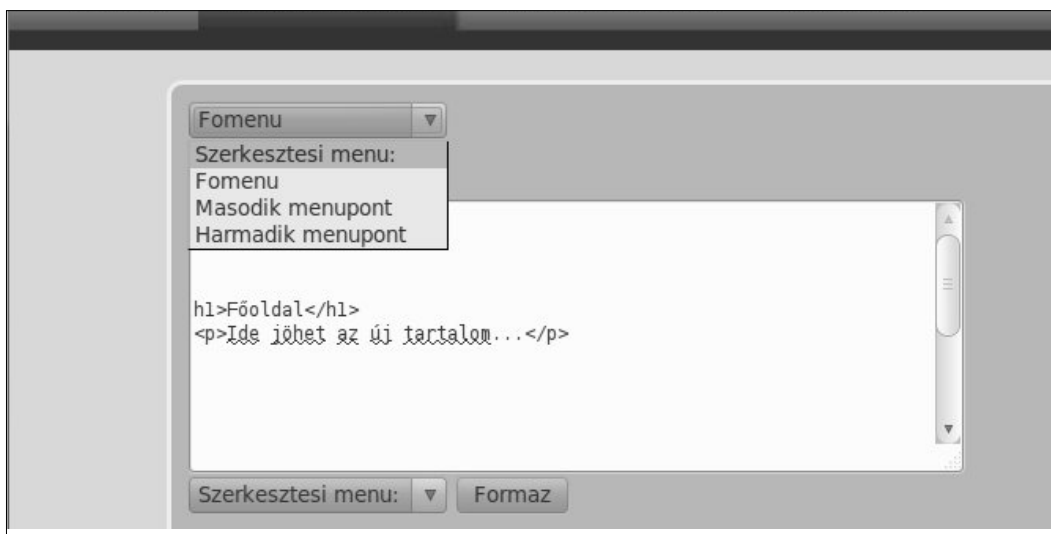
Majd töltsük fel a következő tartalommal az adatbázis *html* mező negyedik recordját:

```
<form>
<select name = „users” onchange = „szerk(this.value)”>
  <option value = „”>Szerkesztési menu:</option>
  <option value = „1”>Fomenu</option>
  <option value = „2”>Masodik menupont</option>
  <option value = „3”>Harmadik menupont</option>
</select>
</form>
<br>
<form action = „formaz.cgi”>
  <textarea id = „tartalom2” rows = „20” cols = „70”
name = „formaz”>
  </textarea>
<br>
  <select name = „mezo”>
    <option value = „”>Szerkesztési menu:</option>
    <option value = „1”>Fomenu</option>
    <option value = „2”>Masodik menupont</option>
    <option value = „3”>Harmadik menupont</option>
  </select>
  <br>
  <input type = „submit” value = „Formaz” />
</form>
```

#### 4. forráskód. A szerkeszt menüpont tartalma

Tehát a szerkeszt menüpontban lesz két legördülő menü, az elsőből kiválasztjuk a szerkeszteni kívánt menüpontot a másodikba megadjuk, hogy melyik menüpontban

történjen a frissítés. A két legördülő menü között található egy textarea tag ebben lehet szerkeszteni a HTML kódot (3. ábra).



3. ábra. Szerkesztési menüpont

Amikor kiválasztjuk a szerkeszteni kívánt menüpontot az első legördülő menüből, elindul egy onchange esemény, amelynek az eseményfüggvénye a szerk(this.value), ez a függvény kapja meg a választott menüpont számértéket. A függvény a szerkeszt.js állományban található, amelynek tartalma és működése megegyezik a tartalom.js szkriptel annyi különbséggel, hogy a függvénynek más nevet adunk, illetve a tartalom2 azonosítót kell megadni kommunikációnak, és természetesen más feldolgozó szkripthez fordul, a szerkeszt.cgi-hez amelynek a tartalma hasonló a menu.cgi állományhoz. Ezért azt csak vázlatosan mutatom be:

```
#!/usr/bin/perl
# könyvtárak beolvasása
# küldött értékek beolvasása
my $szam = $FORM{q};
# adatbázishoz való kapcsolódás
my $sth = $kapcsolat -> prepare("SELECT
html FROM menu
WHERE id = , $szam");
$sth -> execute();
$result = $sth->fetchrow_hashref();
print "Content-Type: text/html; CharSet
= utf-8\n\n";
print "$result -> {html}\n";
$sth->finish();
$kapcsolat -> disconnect;
```

#### 5. forráskód. A szerkesztendő

##### HTML-kódot kinyerő állomány, szerkeszt.cgi

A programkód hasonló a menu.cgi-hez, ugyanaz a funkciója is, csak annyi a különbség, hogy ebben az esetben a HTML-kód nem értelmezve jelenik meg a textarea tag-ban.

A formaz gombra kattintva a menüpont száma és új tartalma továbbítódik a formaz.cgi állományhoz. A forráskódnak csak a fontosabb részleteit mutatom be:

```
#!/usr/bin/perl
# könyvtárak beolvasása
local ($buffer, @pairs, $pair, $name,
$value, %FORM);
$ENV{REQUEST_METHOD} =~ tr/a-z/A-Z/;
if ($ENV{REQUEST_METHOD} eq
„POST”){
    read(STDIN, $buffer, $ENV{CONTENT_
LENGTH});
} else {
    $buffer = $ENV{QUERY_STRING};
}
@pairs = split(/&/, $buffer);
foreach $pair (@pairs) {
    ($name, $value) = split(/=/,
$pair);
    $value =~ tr/+//;
    $value =~ s/%(..)/pack(„C”,
hex($1))/eg;
    $FORM{$name} = $value;
}
my $formaz = $FORM{formaz};
my $mezo = $FORM{mezo};
# adatbázishoz való kapcsolódás
print „Content-Type: text/html; CharSet
= utf-8\n\n”;
my $sth = $kapcsolat -> prepare(„UPDATE
menu SET html = , formaz’
WHERE id = , $mezo ,”);
$sth->execute();
$sth->finish();
$kapcsolat->disconnect;
my $url = „index.pl”;
my $t = 1;
print <<end
<html>
<head>
<META HTTP-EQUIV = refresh CONTENT
= \"$t; URL = $url\">
</head>
<body>
<p>Sikeres szerkesztés!</p>
</body>
</html>
end
```

#### 6. forráskód. A változó tartalmat frissítő szkript, formaz.cgi

Tehát a frissítet menüpont tartalmát ez a szkript kezeli, először is elolvassa a küldött értékeket POST metódussal, mivel a küldött tartalom terjedelmesebb ezért ez a metódus célszerűbb. Majd a `$formaz` és `$mezo` változóban tárolódnak a küldött értékek.

Ezek után történik az adatbázishoz való csatlakozás. Majd a *menu* adattábla frissítése (UPDATE) a `$mezo id`-jű recordnak megfelelő *html* mezőjében. Az ott lévő tartalom frissül a `$formaz` változóban lévő tartalommal egyidőben. Frissíti, tehát felülírja azt, ezért a szerkesztésnél óvatosan kell dolgozni, mivel a frissítendő tartalom törlődik. A kapcsolat bontása után deklarálva van két változó a `$url` és a `$t`. Mivel a *formaz.cgi* állomány nem AJAX-módszerrel kommunikál ezért az adatok küldésénél a feldolgozó szkript jelenik meg a böngészőben, vagyis a *formaz.cgi* erről az oldalról kell viszalépni az *index.pl* főoldalához. Ez megoldható lenne egy link elhelyezésével, de praktikusabbnak találtam a folyamatos automatizálni. Ezt a műveletet egy *META* tag segítségével valósítottam meg, amelyben a korábban deklarált két változó található. Az

`$url`, vagyis milyen legyen az url címe annak a weboldalnak ahová lépni szeretnénk, illetve a `$t` változó, amelyben azt adtam meg, hogy hány másodperc múlva történjen a weboldal megnyitása. A *META* tagnak köszönhetően szerkesztés után egy másodperccel újra a weblapunk főoldala jelenik meg. Így már teljes a dinamikus tartalmat megjelenítő weblap motor.

A munkámban bemutattam, hogy az AJAX-technológia a *Perl*-program nyelvvel karöltve hogyan használható fel dinamikus weblap menü előállításához. A technológia kisebb változtatásával bemutattam egy kezdetleges adminisztrációs felület létrehozásának lehetőségét. Az *AJAX Perl* párbeszédi kommunikáció felhasználható különböző webalkalmazások írására is. Illetve mivel a változó tartalom adatbázisban található, hasonlóképpen tárolhatóak és szerkeszthetőek benne a beléptető rendszerhez szükséges jelszavak, felhasználó nevek is. A munkám továbbfejleszthető felhasználási területe a weblapok és webalkalmazások szerkesztésénél tág. *PHP*-alapú weblapoknak lehet egy alternatívája.