Sablonkezelés Smarty módra (2. rész)

Múlt hónapban, ha nem is alaposan, de megismerkedtünk a Smarty sablonrendszerrel. Ezt folytatva most egy kicsit mélyebbre ásunk…

Már egy hónapja, hogy tudjuk, miért is jó nekünk sablonokkal dolgozni. Tudással bírunk arról, hogy milyen módon kell különféle változóértékeket csempészni a sablonba, vagy akár mindjárt tömböket listázni. Azzal is tisztában vagyunk, hogyan is adjunk át adatokat a sablonjainknak PHP-ből. Ez az alaptudás az, amit egy sablonkezelőnek tudnia kell, ennyivel akár el is boldogulhatnánk. Valójában a Smarty még csak itt kezdődik. Lássuk, miféle hókuszpókuszokat tudunk még vele művelni!

Feltételkezelés

A feltételkezelés sablonmotorunknak az egyik olyan képessége, amit nem feltétlenül helyesel mindenki, hiszen a feltételkezelésnek igazából a logika részének kellene lennie, legalábbis ha összetett feltételek alapján való döntésről van szó. A már régóta jól ismert `if-elseif-else` a sablonokban történő hármas bevetése néhol valóban hasznos lehet. A legegyszerűbb esetben egy sima feltételről van szó:

```
{if $be_van_lepve == true}
<a href="kilep.php">KilØpØs</a>
{/if}
```

Ennél jóval bonyolultabb feltételek is megadhatók és zárójelek használata is lehetséges. Bár itt jegyezném meg, hogy a legtöbb esetben nem túl szerencsés a zárójelek alkalmazása. A bonyolultabb feltételek elemzése nem a sablon feladata, azt már logikai (PHP-kód) szinten el kell dönteni, és egy megfelelő változóban csak az eredményt közölni. Az egyes `if`, `elseif` és `else` ágak a következő ág megnyitásáig, avagy a `{/if}` címkéig tartanak.

Értékmódosítók

Sorozatunk előző részében bemutattam, hogyan lehet egyszerű értékbehelyettesítéseket végezni. Emlékeztetőül idézzük vissza, hogy ez milyen módon is zajlik:

```
Postafi kodban {$uzentek_szama}  zenet talÆlhat .
```

Előfordulhat, hogy sablonokkal dolgozó munkatársunkban felmerül az igény az iránt, hogy a kapott adatokat esetleg szeretné kicsit csinosítgatni, feljavítani. Vagy zavarja, hogy a szépen megtervezett üzenetdobozba olyan szöveg is kikerülhet, ami „szétnyomja” azt. Ilyenkor természetesen a programozót lehet nyaggatni, hogy márpedig ezeket a módosításokat kéri a kapott értékekben. Ám jobban belegondolva ezek a dolgok inkább a megjelenítési réteghez tartozó feladatok, semmint a programlogika részei. Már csak azért is, mivel egy oldalon ugyanaz az érték akár többször több formában is megjelenhet. Ráadásul nem tudni, hogy ami ma éppen csupa nagy betűvel tetszik a megrendelőnek, az másnap nem kellene-e mégis kisbetűkkel neki. A legjobb megoldás tehát a sablonban ezeket a megjelenítési formákat megadni (szenvedjen vele a grafikus, illetve az arculatfelelős). Lássuk, hogyan is oldható meg!
A helyzet kulcsa a | (csővezeték) jel, ezzel elválasztva lehet egy vagy akár több értékmódosítót is egy értékbehelyettesítéshez hozzárendelni. Lássuk példával is szemléltetve:

```
{$megnevezes|upper|spacify}
```

Itt tehát a „megnevezes” néven futó Smarty-érték kerül kiíratásra, ámde előtte két művelet is végrehajtódik rajta. Egyrészt egy `upper` értékmódosító hatására csupa nagybetűvel fog a szöveg kikerülni, a `spacify` hatására pedig szóközök kerülnek a betűk közé.
Az adatmódosító eszközök közül több értékekkel is ellátható. Ezek megadása közvetlenül az adatmódosító nevének megadása után lehetséges, kettősponttal elválasztva. Helyenként több érték is megadható, ilyenkor ezeket is kettősponttal kell egymástól elválasztani. Mindjárt ilyen lehetőségeket rejt maga a `spacify` is, ugyanis a szóközkarakterek beszúrása csak az alapértelmezett viselkedése. Ehelyett azonban bármilyen más karakter vagy karakterfüzér is beilleszthető a szöveg betűi közé, például az alábbi sabloncímkék alkalmazásával:

```
{$telepules}
{$telepules|spacify}
{$telepules|spacify:".."}
```

Ennek hatására ilyesféle eredményeket kapunk:

```
SzØkesfehØrvÆr
S z Ø k e s f e h Ø r v Æ r
S..z..Ø..k..e..s..f..e..h..Ø..r..v..Æ..r
```

A Smarty beépített módosítói

Három külön csoportra osztottam a Smarty beépített módosítóit. A legegyszerűbbek kerültek az egyik csoportba, amelyek apróbb formai átalakítást végeznek. Ezeknek paraméterezési lehetőségük sincs. A következő és egyben legnagyobb csoportba azok az értékmódosítók kerültek, amelyek különféle értékeket várnak. Ezek egy része kötelezően elvár bizonyos adatokat, mások nem – szükségességét mindenhol helyben jelzem. A harmadik csoport logikailag igazából nem tartozik az értékmódosítók közé, de megvalósításuk tekintetében ide sorolhatók. Ezek a kapott szöveg átalakítgatása helyett különféle, az adott szövegre jellemző számadatokra cserélik ki őket.

Egyszerű, érték nélküli módosítók

- `capitalize`: minden egyes szó kezdőbetűjét nagybetűssé alakítja.
- `lower`: a teljes szöveget kisbetűsíti.
- `upper`: a teljes szöveget nagybetűsíti.
- `nl2br`: a PHP `nl2br()` függvényének megfelelően a soremelés karaktereket `<br />` XHTML címkékre cseréli.

Paraméterezhető értékmódosítók

- `cat`: az egyetlen kötelező értékben megadott szöveget a behelyettesített szöveg végéhez fűzi. Elsőre értelmetlennek tűnik, hiszen ez a szöveg kerülhetne egyszerűen a sablon állandó részébe is. Így viszont további adatmódosítások is végezhetők a teljes összefűzött szövegen.
- `date_format`: a PHP `strftime()` függvényének megfelelő formában megadott formázás szerint jeleníti meg a dátumot, illetve az időt. Ez a forma az első értékben adható meg. Ha nincs megadva, az alapértéke: `%b %e, %Y`. Második értékként megadható egy szöveg, ami akkor jelenik meg, ha az eredeti érték üres volt. Nem kötelező élni vele.
- `default`: amennyiben a kérdéses érték üres, helyette a `default` egyetlen kötelezően megadott értékének szövege jelenik meg. Létező érték esetén azt érintetlenül hagyja.
- `escape`: egyetlen, egyben kötelezően megadandó értéke van, ami azt határozza meg, hogy milyen kódolással legyen az adott szöveg „levédve".

  A választási lehetőségek

  - `html`: a `htmlspecialchars()` PHP-függvénynek megfelelően alakítja át a szöveget. Mindezt bekapcsolt `ENT_QUOTES` értékkel, ami annyit tesz, hogy a `< (<) > (>) & (&)` és az idézőjel (") karakterek mellett az egyszeres idézőjelet (aposztrófot) is átkódolja ('). Jól használható HTML-forráslista megjelenítésére.
  - `htmlall`: az előzőhöz hasonlóan működik, de a fent felsoroltakon túl minden olyan karaktert átkódol, amiknek létezik HTML-egyed megfelelőjük.
  - `url`: a PHP `urlencode()` függvényét alkalmazza az adott szövegre.
  - `quotes`: az idézőjeleket visszaperjellel (\) védi le.
  - `hex`: a `%xx` formára kódol minden karaktert, ahol az „xx" az adott karakter kódja tizenhatos számrendszerben megadva.
  - `hexentity`: az előzőhöz hasonlóan a tizenhatos számrendszerben dolgozik, de szabványos HTML-hexaegyedeket hoz létre, például: `&#x6d;`
  - `javascript`: a következő cseréket hajtja végre: \-ből \\, '-ből \', a soremelés és kocsivissza karakterekből pedig egyöntetűen `\r` lesz.
- `indent`: a szövegek igazítására készült módosító. Minden sor elejére adott számú szóközt szúr be. A második értékben a szóközkarakter is lecserélhető.
- `regex_replace`: két kötelező értékkel bír, az első a keresési mintát, míg a második a csereszövegeket tartalmazza. Teljes mértékben megfelelnek a PHP `preg_replace()` függvénye első két értékének.
- `replace`: egyszerűbb cserékre való. Az `str_replace()` PHP-függvénynek felel meg.
- `spacify`: a példában már megismert módosító. Alapesetben minden két karakter közé egy szóközt helyez el, de megadható helyette más, akár többkarakteres szöveg is.
- `string_format`: egy értéke van, ez a PHP `sprintf()` függvényének megfelelő alakban megadott formázás.
- `strip`: minden többszörözött szóköz, tabulátor és újsor karaktersorokat egyetlen szóközre cserél. Értékként más karakter vagy karaktersor is megadható.
- `strip_tags`: az összes HTML-címkétől megszabadítja a szöveget. Egyetlen értékben megadható, hogy logikai értékként a HTML-címkék helyén hagyjon-e szóközt. Ha nem adjuk meg, hagy.
- `truncate`: hosszú szövegek csonkítására való. Az első értéke sem kötelező, de itt adható meg az a hossz, ami felett vágni szeretnénk. Alapértéke: 80. Második értékként a vágás helyén megjelenő „..." karaktersorozat cserélhető le esetlegesen másra. Harmadikként megadható, hogy a szavak közepén is vágjon-e. Ennek alapértéke: `false`.
- `wordwrap`: szövegtördelés. Három nem kötelező értéke van. Az elsőben azt módosíthatjuk, hogy hányadik karakteroszlopnál törje a sorokat, ennek alapértéke 80. Másodikként megadható, hogy milyen karakter vagy karaktersorozat legyen beszúrva sortörésként. Ha másképp nem rendelkezünk, ez a `"\n"` soremelés karakter lesz. Harmadikként egy logikai értéket vár, ami eredetileg hamisra (`false`) van állítva, ilyenkor csak szóhatárokon tör. Ha igaz (`true`) állapotba billentjük, pontosan az adott oszlopban fogja megtörni a sorokat, nem törődve azzal, hogy szó közepén jár-e, vagy sem. Példa:

```
{$hosszuszoveg|wordwrap:68:"<br>\n"}
```

Szövegelemző, számszerű adatokat közlő módosítók

- `count_characters`: a szöveg helyett annak karakterekben számolt hosszát adja vissza.
- `count paragraphs`: a szöveg helyett a benne található bekezdések számát adja vissza (a soremeléseket alapul véve).
- `count_sentences`: az előző kettőhöz hasonlatosan számot ad vissza, a szövegben található mondatok számát.
- `count_words`: megszámolja a szavakat, és a kapott számértéket adja vissza.

Egy értékbehelyettesítéshez több módosító is hozzárendelhető, ezeket további csővezeték (|) jelekkel lehet egymástól elválasztva kitenni. Kiértékelésük sorban balról jobbra történik. Például:

```
{$talalt_weboldal_szovege|strip_tags:false
➥|truncate:60}
```

Ilyen módon a keresőoldalon a talált oldalak szövegtartalmának első hatvan karakterét tudjuk kiíratni. Először a HTML-formázó címkéit távolítjuk el, majd a szöveget legfeljebb hatvan karakterre megvágjuk. Ha a két módosítót megcserélnénk, nem a hasznos szöveg hosszára vonatkozóan adnánk felső határt, hanem a nyers HTML-formázásokkal teli szövegre, ami jelen esetben kedvezőtlen.
Igaz, eddig úgy beszéltem az értékmódosító eszközökről, mintha azok csak változóbehelyettesítéskor működnének, ámde ezek más helyeken is hasznosíthatók:

```
{if $allat|upper eq "KACSA"}
```

Saját értékmódosítók készítése

A dolog sokkal kevésbé ördöngös, mint elsőre gondolnánk. Szerencsére a Smarty rendszert a könnyű bővíthetőség is jellemzi. Első lépésként nézzünk bele a Smarty ***plugins*** könyvtárába. Itt számos ***.php***-re végződő fájlt fogunk találni, de ezek közül is most a „modifier" karaktersorral kezdődők fognak minket érdekelni. Nézzünk is bele egybe, legyen ez például a ***modifier.capitalize.php***. A lényeg itt csupán ennyi:

```
function smarty_modifier_capitalize($string)
{
        return ucwords($string);
}
```

A PHP `ucwords()` függvénye pontosan azt teszi, amit a Smarty-féle `capitalize` értékmódosítónak is el kell az adott szövegen végeznie. Ez nem is csoda, hiszen a dolog megvalósítására is ezt a PHP-függvényt alkalmazza. Saját értékmódosító létrehozásához nem is kell más, minthogy létrehozzunk egy ***modifier.módosítónkneve.php*** állományt a ***plugins*** könyvtárban, amibe egy `smarty_modifier_m dos t nkneve` néven futó függvény teszünk. Az értékben kapott szöveggel ezután azt csinálunk, amit akarunk, csak végül ne felejtsük el egy `return`-nel visszaküldeni. Ennyi az egész.

Feldolgozástól védett sablonterületek, elkapható kimenet

Könnyen megeshet, hogy a sablonállományokba egy kis JavaScript- vagy stíluslapkódot tennénk. A gond ezekkel az, hogy sűrűn alkalmazzák a { } jelpárost. Smarty barátunk az ilyen jeleket látva azon nyomban megkísérli értelmezni az ott található karaktersort, de azonnal bele is bukik. Nem kívánatos, hogy ilyen helyeken kutakodjon, de ezt külön meg kell mondani neki. Ilyen kivételes sablonterületek körülhatárolására készült a `{literal}` `{/literal}` címkepáros. Ami ezek közé kerül, azt kedvenc sablonkezelőnk feldolgozás nélkül átengedi. Létrehozhatunk olyan területeket is a sablonon belül, amelyek ugyan feldolgozódnak, de ahelyett, hogy helyben megjelennének, egy sablonváltozóban landolnak. Ezeket a területeket a `{capture}` `{/capture}` címkékkel kell határolni. Megadható, bár nem kötelező, hogy mi legyen a Smarty-változó neve, ahol a feldolgozás után helyet kap a kimenet. Ez alapesetben a `$smarty.capture.default`. Ha a nyitócímkében megadunk `name` értéket, akkor a `default` erre cserélődik le. Egy példával élve: a `{capture name=elteszem}` nyitócímkével ellátott terület kimenete a `$smarty.capture.elteszem` változóban lesz ezekután elérhető.

Beállításfájlok kezelése

Bizony, ilyet is lehet. A Smarty beépített eszközeivel lehetővé teszi számunkra, hogy a projektjeinkhez a különféle beállításokat külön állományokban tároljunk, és ezekből adatokat olvassunk. Ezek a beállításállományok formailag igencsak hasonlatosak egy bizonyos redmondi cég ***.ini***-fájljaira. Minden adat külön sorban adható meg, név = érték formában. A sorokat csak a soremelés karakter zárja, nincs szükség pontosvesszőre vagy hasonló huncutságra. Az értékek idézőjelek (normál vagy egyszeres) közé tehetők, de nem feltétlenül szükséges.
Hogy ezek az adatok bekerüljenek sablonunk vérkeringésébe, a `{config_load}` címkét kell szorgosan alkalmaznunk.
A `file` értékben kell megadnunk, hogy melyik beállítófájl betöltését tűztük ki célul. Amint a beállítófájl betöltésre került, értékei máris felhasználhatók a következő formában:

```
{#ØrtØk_neve_a_beÆll tÆsfÆjlban#}
```

Az ebből a forrásból érkező értékek tehát nem keverednek a PHP-ból érkező adat-hozzárendeléseinkkel. A `{config_load}` további, nem kötelezően megadandó értékekkel vértezhető fel. Az egyik ilyen `section` néven fut, amivel beállítófájlunk meghatározott területeit tudjuk megcélozni. Miket is? Bizony, a beállítófájlok az egyszerű név–érték hozzárendelésen túl egyéb szolgáltatásokat is tudnak nyújtani, ha igény mutatkozik rá.
A beállításainkat tartalmazó állományaink ugyanis területekre (section) bonthatók. Tetszőleges számú terület nyitható egy ilyen fájlban, ehhez a terület elejére kell egy sort beszúrnunk, ahol kapcsos zárójelek között adandó meg a terület neve. Erre a névre hivatkozhatunk tehát a `{config_load} section`

1. lista site_config

```
# ez lesz az ÆltalÆnos ablakfejlØc
pageTitle = "Smarty dem oldal"

[Titok]
oageTitle = "Smarty dem  - titkos oldalak"

[Forum]
oageTitle = "Smarty dem  - f rum"
```

2. lista forum_header

```
{config_load file="site_config.conf"
section="Forum"}
<html>
<head>
  <title>{#pageTitle#}</title>
</head>
</html>
<body>
```

értékében. A terület ott kezdődik tehát, ahol ez a nyitósor áll, és a következő terület megnyitásáig ér a takarója, vagy ennek híján a fájl végéig. Mi lesz azokkal az érték-hozzárendelésekkel, amiket mindjárt a fájl elején, bárminemű területmegjelölés mellőzésével adunk meg? Ezek az értékek a `section` érték meglététől függetlenül minden esetben betöltődnek. Lehetőség nyílik továbbá arra is, hogy egy ilyen mindenható értéket egy területen belül más tartalommal írjunk felül. Lássunk erre egy élő példát, ahol is webes rendszerünk fejlécsorát fogjuk alakítgatni. Az egyes és kettes lista megtalálható a 47. CD Magazin/Smarty könyvtárában.
A fenti példa weboldalunk fórumterületének HTML-fejlécét állítja elő, ennek hatására a böngészőablak fejlécsorában a „Smarty demó – fórum” szöveg bukkan fel. Ha a `section` értéket „Titok”-ra változtatnánk, avagy teljesen elhagynánk, rendre a „Smarty demó – titkos oldalak”, illetve a „Smarty demó oldalak” szöveg kerülne ablakunk tetejére.
Tovább bonyolíthatjuk életünket a `scope` érték megismerésével. Ennek három különféle értéke is lehet: `local`, `parent`, `global`. A `local` annyit tesz, hogy az itt betöltött értékek csak ebben a sablonfájlban fognak hatni. A `parent` esetén nemcsak helyben, de az ezt meghívó (szülő) sablonállományban is hatással lesznek ezek a betöltött értékek. Amennyiben a `global`-t választjuk, mindegy, hol töltjük be adatainkat, azok mindenhol elérhetőek lesznek. De mi is ez a hierarchikus viszony és miféle más sablonokról is van szó? Mindjárt kiderül.

Részsablonok beillesztése

Nem kell sokat dolgozni a Smartyval, hogy az ember felismerje, jó lenne, ha oldalainak azonos (vagy közel azonos) fejléceit és lábléceit nem kellene minden sablonállományban módosítani, ha valami változás áll be, például egy minden oldalon jelen levő menü esetén. A megoldást `{include}`-nak hívják, ez, gondolom, nem nagy meglepetés. Ami mindenképpen elhagyhatatlan számára, az a `file` érték, amelyben a beilleszteni kívánt sablonállomány nevét kell megadni. A beillesztett sablonok további sablonokat húzhatnak be maguk alá és

így tovább. Az így kialakuló szülő-gyermek viszonyok azok, amelyekről a beállítófálj értékek hatókörének taglalásakor értekeztem. A beillesztett sablon szülőjétől minden változóértéket átvesz. Bár a beillesztő utasításunk természetesen {capture} {/capture} címkékkel körülvéve megjelenítés helyett Smarty változóba kényszeríthető, van erre egy rövidebb mód is, mégpedig az {include} assign értékének szorgos alkalmazása. Ekkor ugyanis az itt megjelölt nevű változóba kerül a beillesztett sablon kimenete, a megjelenítés helyett.

Gyorstárazás

Már az nagy sebességnövekedést jelent, hogy a Smarty nem értelmezi minden egyes alkalommal a sablonokat, csak ha azok változtak. Ámde minden egyes lekéréskor lefut az a teljes PHP-kód, amit sablonfordításkor a Smarty hoz létre. Érdemes minden esetben a Smarty gyorstárazási lehetőségeinek kihasználásában gondolkodni, hiszen ezzel jelentős processzoridőt takaríthatunk meg. Igaz, a Smartyval dinamikus tartalmak létrehozása a szokványos feladat, de ezeknek az oldalaknak nem feltétlenül kell percre kész adatot szolgáltatniuk. Esetleg elégséges, ha fél- negyedóránként frissül a tartalmuk, vagy akár még ritkábban. Alkalmazzunk tehát gyorstárat! Mi kell hozzá? Kell egy könyvtár valahol a gépen, aminek nem feltétlenül kell – sőt! – a nyilvános webkönyvtárban lennie, de fontos, hogy a webkiszolgáló mint felhasználó könyvtárakat, fájlokat tudjon benne létrehozni. Ennek a könyvtárnak az elérhetősége adható meg a Smarty objektum $cache_dir változójában. A gyorstárazás bekapcsolása ugyanitt a $caching = true érték hozzárendelésével tehető meg. Ekkor a Smarty a sablon megjelenítésekor csak akkor fogja a sablonból fordított PHP-kódot futtatni, ha nincs hozzá megfelelő tárolt változat a gyorstárban. Ha futtatnia kell, természetesen egyből tárolja is az adott kimenetet. Alapesetben a gyorstár elévülési ideje egy óra. Ezt a $cache_lifetime Smarty objektumváltozón keresztül igazíthatjuk az igényeinkhez. Az időt másodpercekben kell megadni.

Elő- és utószűrők

Két helyen is alakíthatunk még programunkon. Egyrészt a sablonállományok feldolgozásának megkezdése előtt végezhetünk rajtuk szűréseket, másrészt az előállított PHP-kódon is faraghatunk a saját függvényeinkkel. Ehhez mindkét esetben olyan függvényt kell létrehoznunk, ami a szűrendő adatokat kapja első értékében, és az átalakítások után a módosult „szöveget” adja vissza. A függvény meghatározásában második értéknek mindig biggyesszünk oda egy hivatkozott értéket: &$smarty. Természetesen bármiféle más neve is lehet, de ez a „szabvány”. Ha nincs is szükség a szűrőn belül a Smarty objektum elérésére, ezt akkor is oda kell tenni, ugyanis a Smarty objektum meghívásakor feltétlenül átadja.

További tippek és trükkök

Minden assign() hívás egy-egy másolatot készít azokról a változókról, amiben az értéket biztosítjuk számára. Nagyobb tömbök, objektumok esetén ez jelentős memóriaigény-többletet jelenthet. Márpedig majdnem minden sablonhoz rendelt értékre igaz, hogy azt egy helyen egyszer fogjuk felhasználni, onnantól fogva pedig nincs rá szükségünk. Jobb lenne tán csak a PHP-ból jól ismert változóhivatkozást (nem ténylegesen új változó, csak adott változó memóriaterületére mutató álnév) alkalmazni. Minden eldobható adatnál érdemes ezt az értékátadási módot megfontolni. Ha mellette döntünk, csupán az assign() eljárást kell assign_by_ref() testvérére lecserélni, és máris megtakarítottunk egy kis memóriát!

A {section} és {foreach} listákban nem feltétlenül kell a táblázatsorok háttérszínét a PHP-től kérni, ha annak sorszíne váltakozó. Bár ez megoldható , ha az átküldött tömbben a sorok színét is tároltatjuk, de elegánsabb megoldás a sablonban jelezni (és igény esetén módosítani), hogy váltakozó sorszínt szeretnénk. Ehhez a {cycle} címkét kell alkalmazni:

```
{section name=i loop=$nevek}
<tr class="{cycle values="sotet,vilagos"}">
  <td>{$nevek[i]}</td>
</tr>
{/section}
```

Kimenő elektronikus levelek megformálására is alkalmazható ez a sablonrendszer. Ehhez a Smarty objektum fetch() eljárása ad segítséget. Használata megegyezik a display()-ével, de az eredmény megjelenítése helyett azt visszaadja. Ezt a visszakapott értéket azután a mail() PHP-függvényünk rendelkezésére bocsáthatjuk.
Alapesetben a template és *template_c* könyvtárakat a Smarty ott keresi, ahol a hívó-PHP található. Az alapbeállítások tehát feltételezik, hogy ezek a könyvtárak nyilvánosan is elérhető területen vannak. Ez nem feltétlenül kívánatos tulajdonság. Szerencsére mint minden, ez is átállítható a Smarty objektum $template_dir és $compile_dir változóinak felülírásával. Egyiküknek sem kell tehát nyilvános területen lennie. Amennyiben nem teljes a bizalmunk a sablonokat szerkesztő személyekkel szemben, érdemes a Smarty objektum $security változóját true állapotba billenteni. Ha ezt nem tesszük meg, a sablonba mindenféle PHP-kód is beilleszthető lesz a {php} {/php} címkék között. Nem hiszem, hogy taglalnom kellene, miért is veszélyes ez. Emellett ilyenkor csak az általunk meghatározott könyvtárakból tudnak {include}-dal sablonállományokat beilleszteni. Hogy melyek ezek a könyvtárak, a $secure_dir Smarty objektumváltozóban adható meg. Az említett változó egy olyan tömb, amelyben akárhány ilyen könyvtárat felsorolhatunk.

Végszó

Véleményem szerint a Smarty megismerése a PHP-ben járatosak számára nem megterhelő feladat. A nyelvezet alapjait egy hagyományos weblaptervezéssel foglalkozó munkatársnak sem nehéz elsajátítania. A sablonelvű tervezés által több időnk marad a program jó megtervezésére, és mivel leíráskészítésre is rákényszerülünk (a sablonban mit min keresztül adunk át), a rendszerünk is átgondoltabb lesz.

Heiglig (Cece) Szabolcs (cece@phphost.hu)
Veszprémben él. Hobbija, kedvenc időtöltése és munkája is a programozás. Szabadidejében hajlamos kerékpárra pattanni, vagy baráti társaságban szerepjátékokkal foglalkozni.

KAPCSOLÓDÓ CÍMEK

A Smarty hivatalos oldala
➲ http://smarty.php.net
Szabó Dénes Smarty témájú előadásának anyaga
➲ http://www.phpconf.hu/slides/smarty/