Ghid de dezvoltare

Creaza-ti propriile templeturi cu Bluo CMS

Introducere

In Bluo continutul este separat de grafica. Partea grafica a siteului este continuta in niste matrite numite templeturi. Acestea vor fi folosite ca matrite peste care vei aduga continut folosind sistemul de administrare pus la dispozitie. Bluo vine cu 5 templeturi customizabile dar citind acest document iti vei putea crea propriile tale templeturi.

Pasi in crearea unui template

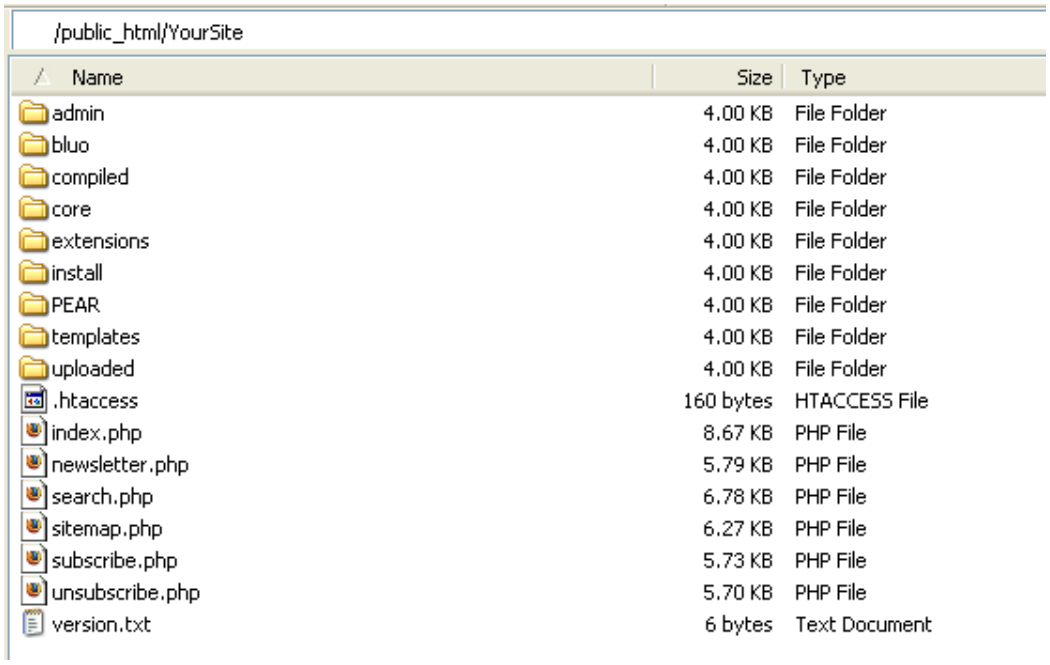
Pentru a construi un template Bluo este indicat sa urmezi urmatorii pasi:

1. Primul lucru de care ai nevoie este un design.

2. Dupa ce consideri ca designul pe care il ai este ceea ce iti doresti poti sa incepi sa creezi dupa acesta fisierul html. Este indicat ca in realizarea acestuia, pentru formatare, sa scrii toate clasele intr-un fisier de tip css.

3. In momentul in care sunt gata cele doua fisiere: fisierul cu codul html si fisierul de tip css poti incepe sa realizezi template-ul propriu-zis.

4. Directorul in care se afla site-ul tau[1] trebuie sa aiba structura din imaginea de mai jos, in caz contrar dearhiveaza din nou arhiva pe care ai downloadat-o de pe http://www.bluocms.com [2].

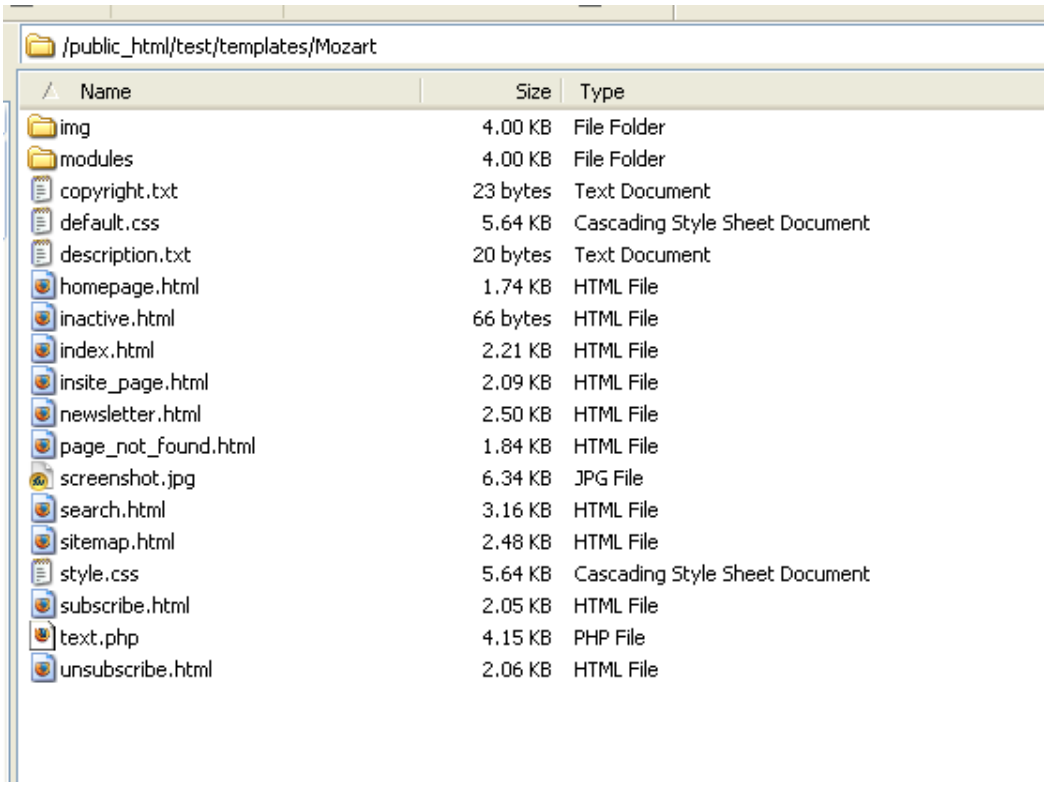


Dintre toate directoarele pe care le observi in imagine, cel care te intereseaza in mod direct este *templates*, adica directorul care contine toate template-urile disponibile pentru versiunea ta de Bluo.

5. Acest pas consta din crearea unui nou director in directorul templates al carui nume va da si numele template-ului tau si care sa aiba structura de fisiere si directoare de mai jos. Multe din acestea de mai jos sunt

[1] Daca nu ai Bluo intra pe http://www.bluocms.com si downloadeaza versiunea care te intereseaza.
[2] Daca nu o mai ai trebuie sa downloadezi din nou Bluo de pe http://www.bluocms.com

optionale dar le vom explica oricum in sectiunea urmatoare.



6. Dupa ce te-ai asigurat ca directorul cu numele template-ului tau are configuratia de mai sus trebuie sa copiezi imaginile incluse in fisierul html creat de tine dupa design in directorul img, dupa care sa copiezi continutul din fisierul care contine stilurile in *style.css*.

7. Scrii continutul fisierelor *copyright.txt* si *description.txt* si inlocuiesti imaginea *screenshot.jpg* cu imaginea pe care vrei sa o atribui template-ului tau. Ai grija insa ca numele acestei imagini sa fie tot *screenshot.jpg*[3].

8. Acum poti incepe realizarea propriu-zisa a template-ului prin completarea fisierelor de tip html. In continutul acestora poti sa folosesti o serie de module[4] care sa iti usureze considerabil munca.

9. Dupa crearea acestor template-uri poti sau nu sa modifici fisierul *style.css* dupa cum este mentionat in capitolul special dedicat acestui fisier.

10. Trebuie sa completezi fisierul *default.css* cu continutul din *style.css*, numai dupa ce esti sigur ca *style.css* contine stilurile default[5] pentru site-

[3] Este important ca extensia sa fie .jpg
[4] Iti voi prezenta aceste module in capitolul special dedicat acestora
[5] Stilurile custom se obtin in urma modificarilor din sectiunea Settings -> Design options din partea de administrare

ul tau.

11. Deschizi partea de administrare si selectezi din sectiunea *Settings -> Templates* templateul tau pentru a-l putea vizualiza.

12. Verifici daca template-ul este realizat bine prin vizualizarea front end-ului[1]. Corectezi eventualele probleme de formatare.

13. GATA!

[1] Aceasta verificare se poate face incepand cu pasul 8

Fisierele care compun un template

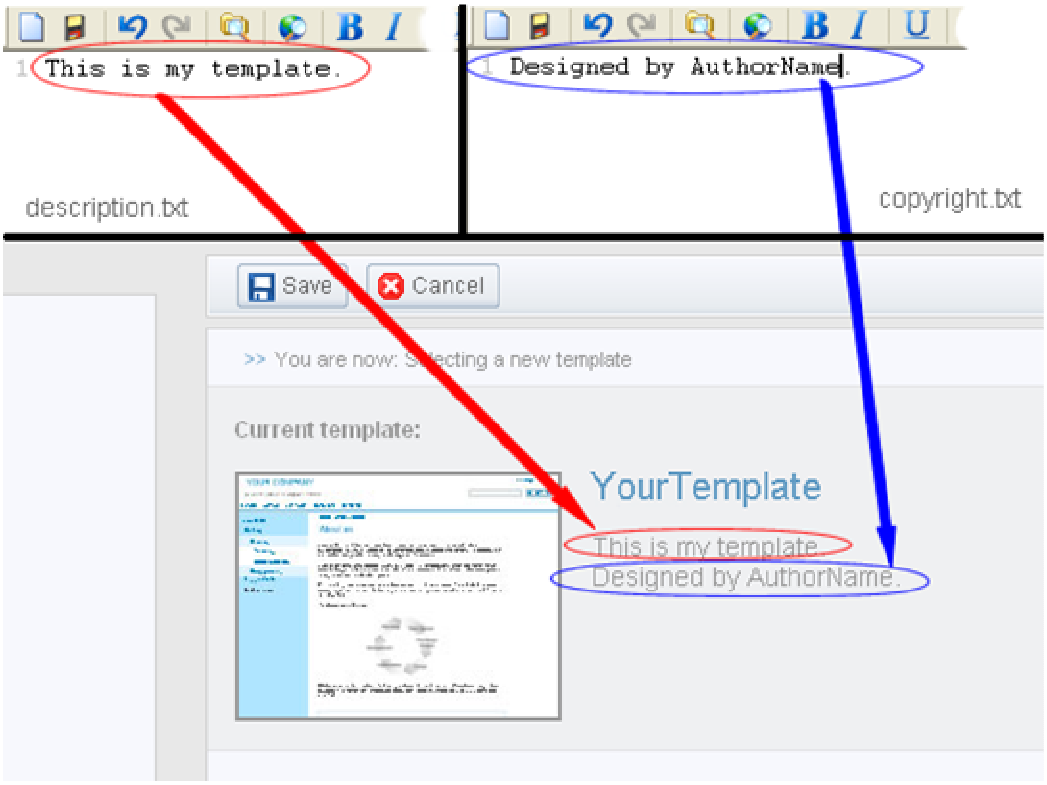
Haide sa vedem care este semnificatia fiecarui director/fisier din structura unui template.

Observatie: Importante si obligatorii sunt template-urile *homepage.html*, *index.html*, *inactive.html*, *insite_page.html* si *page_not_found.html*. Restul template-urilor nu sunt necesare daca site-ul tau nu impune acest lucru.

- **img** – este un director in care se stocheaza toate imaginile care apar pe site-ul tau, mai putin cele care apar in continutul unei pagini. Acestea din urma se stocheaza in directorul uploaded care apare in imaginea de la pasul 4. In legatura cu directorul *uploaded* trebuie mentionat faptul ca toate imaginile uploadate cu ajutorul editorului din partea de administrare sunt salvate in acest director.

- **modules** – in acest director se gasesc toate template-urile necesare pentru modulele *RenderHTML, RenderBreadcrumb* si *RenderMenu*. Mai multe despre continutul acestui director iti voi spune in capitolul dedicat explicarii modulelor Bluo.

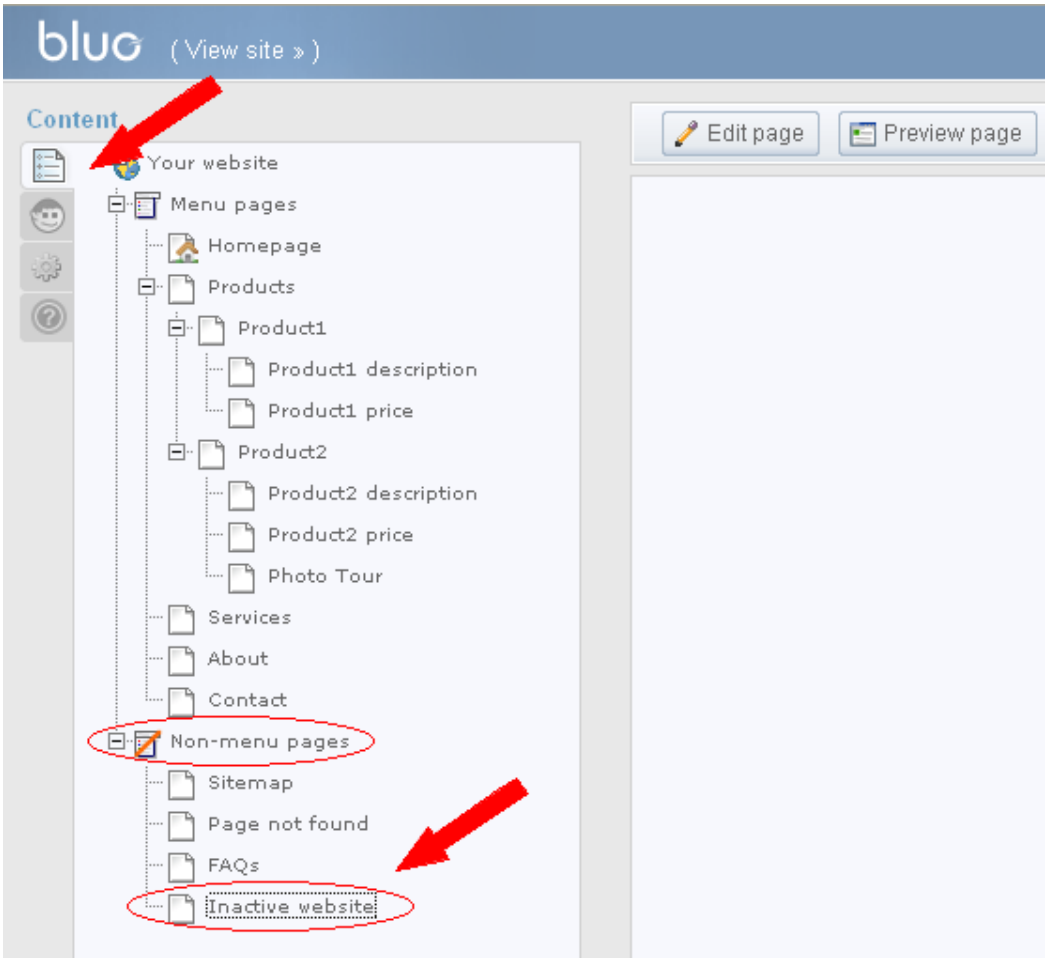
- **copyright.txt** si **description.txt** sunt doua fisiere complementare. *Description.txt* contine descrierea template-ului, iar *copyright.txt* contine numele celui care a realizat template-ul. Continutul acestor doua fisiere apare in partea de administrare in sectiunea *Settings>Templates*.



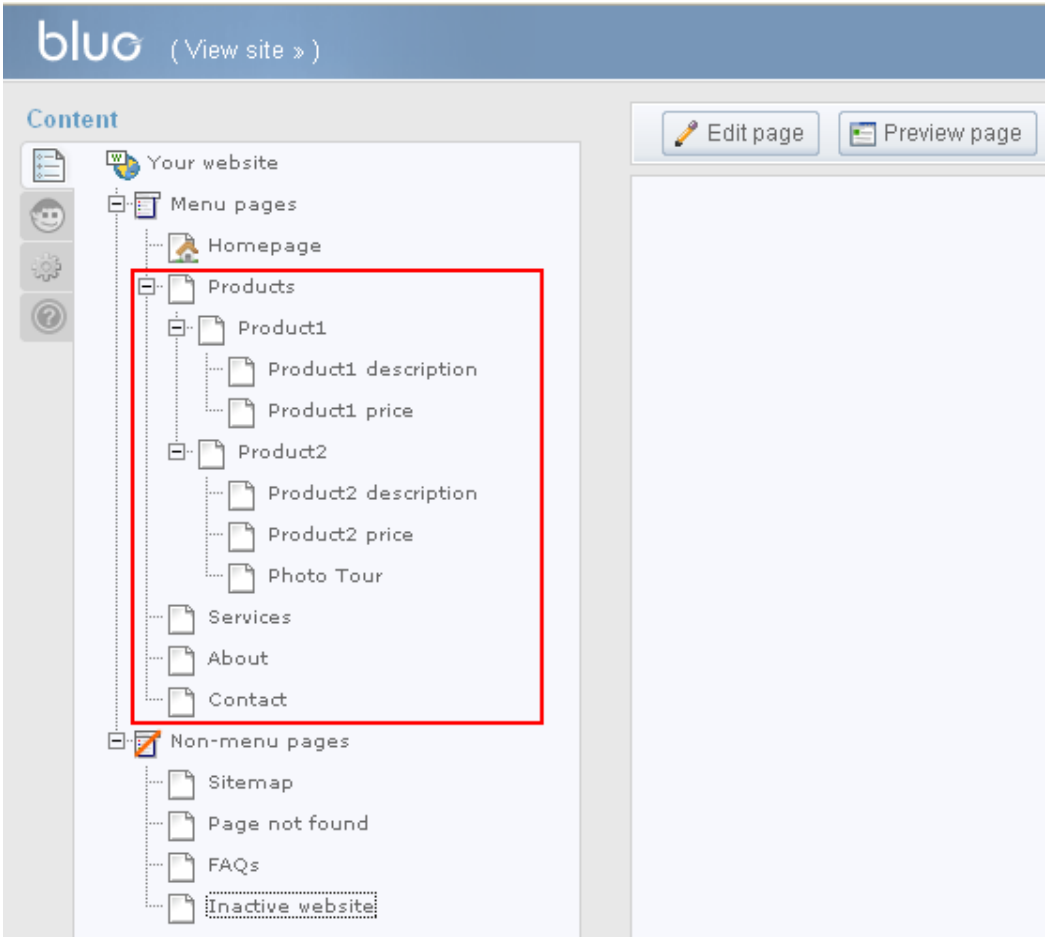
- **default.css** este fisierul care contine stilurile implicite (default)pentru template-ul tau. Acest lucru inseamna ca, in momentul in care in partea de administrare, in sectiunea *Settings -> Design options*, vei face click pe butonul *Restore to default*, continutul fisierului *style.css* va fi inlocuit automat de continutul din *default.css*. Nu trebuie sa confunzi *default.css* cu *style.css* despre care vom vorbi la momentul potrivit.

- **homepage.html** este template-ul pentru pagina de start a site-ului tau.

- **inactive.html** este template-ul pentru pagina Inactive Website care se afiseaza in cazul in care site-ul tau este dezactivat. Pagina *Inactive Website* se gaseste in tree-ul din partea de administrare in sectiunea *Non-Menu Pages*.



- **index.html** este template-ul pentru toate paginile din sectiunea *Menu Pages* diferite de Homepage. In imaginea de mai jos sunt indicate paginile pentru care se aplica template-ul *index.html*

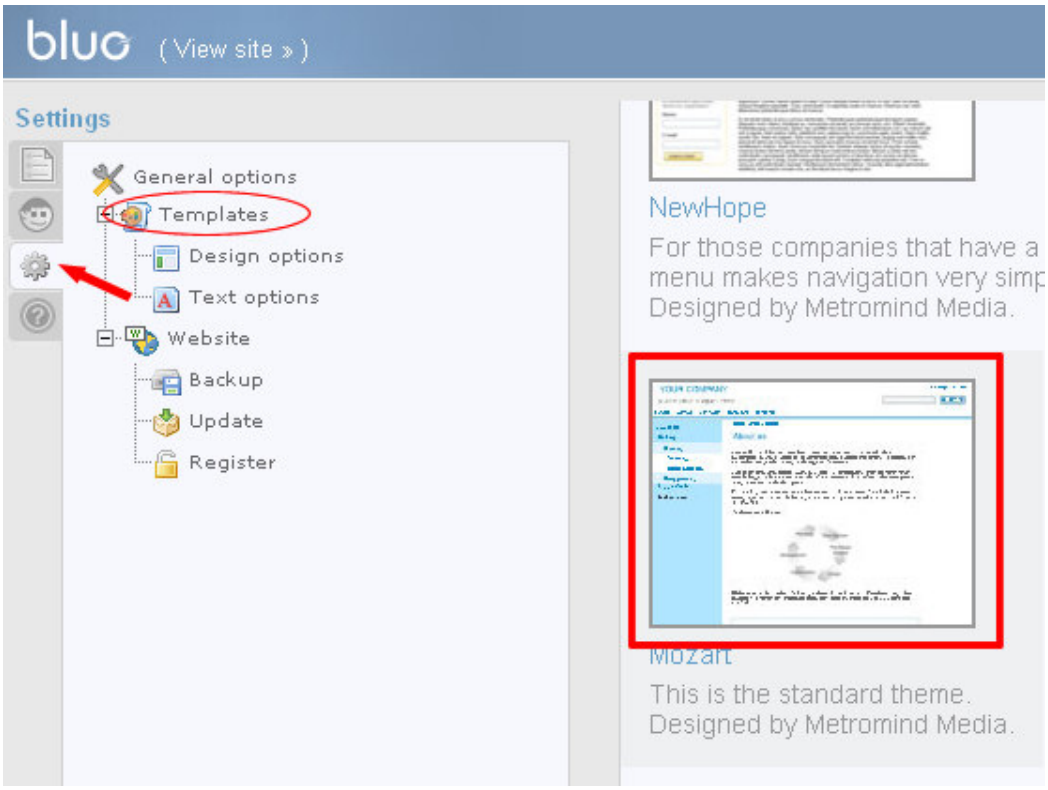


- **insite_page.html** este template-ul pentru toate paginile din sectiunea *Non-menu pages* diferite de Inactive Website, Page not found si Sitemap pentru care exista template-uri separate.

- **newsletter.html** este template-ul pentru pagina care se afiseaza in momentul in care un utilizator al site-ului tau isi lasa datele de contact in caseta special creata pentru el pe site.

- **page_not_found.html** este template-ul pentru pagina de Page not found care se gaseste in sectiunea *Non-menu pages* din partea de administrare.

- **screenshot.jpg** este imaginea care se atribuie template-ului tau si care apare in partea de administrare in sectiunea *Settings->Templates*. De exemplu, pentru template-ul Mozart, imaginea din screenshot.jpg este incadrata in patratul de culoare rosie de mai jos:



- **search.html** este template-ul pentru pagina Search Results, in care sunt afisate rezultatele unei cautari.

- **sitemap.html** este template-ul pentru pagina Sitemap care apare in sectiunea Non-menu pages.

- **style.css** este fisierul in care se afla stilurile pentru template-ul tau. Mai multe amanunte despre continutul acestui fisier iti voi prezenta in capitolul dedicat acestui fisier.

- **subscribe.html** este template-ul pentru pagina care se afiseaza in momentul in care un utilizator al site-ului tau confirma abonarea la newsletter in urma mailul primit dupa introducerea mailului.

- **text.php** este un fisier care contine doi vectori: *text* si *help* care se coompleteaza conform indicatiilor prezentate in modulul *RenderText*.

- **unsubscribe.html** este template-ul pentru pagina care se afiseaza in momentul in care un utilizator al site-ului tau se dezaboneaza de la newsletter.

Flexy

Flexy este un platforma de scripting din pachetul PEAR. Vom folosi acest limbaj pentru crearea templateurilor.

Iata cateva notiuni de flexy de care ne vom folosi in continuare:

`{numeVariabila}` – Afiseaza valoarea variabilei *numeVariabila*
`{numeVariabila:h}` – Afiseaza valoarea variabilei *numeVariabila* sub forma de cod HTML

`{numeFunctie(param1,param2,…)}` – Apeleaza functia *numeFunctie* cu parametri param1, …

`{runModule(#numeModul#,#param1#,#param2#,…..)}` – Apeleaza modulul numeModul, avand ca parametri param1, param2, …

Structura if

```
{if:variabila}
 Cod_HTML_Ramura_If
{else:}
 Cod_HTML_Ramura_Else
{end:}

{if:functieBooleana()}
 Cod_HTML_Ramura_If
{else:}
 Cod_HTML_Ramura_Else
{end:}
```

Structura foreach

```
{foreach:numeArray,cheie,valoare}
 Cod_HTML
{end:}
```

```
<tr flexy:foreach="numeArray,cheie,valoare"> <!—inserarea
structurii foreach in interiorul unui tag HTML -->
```

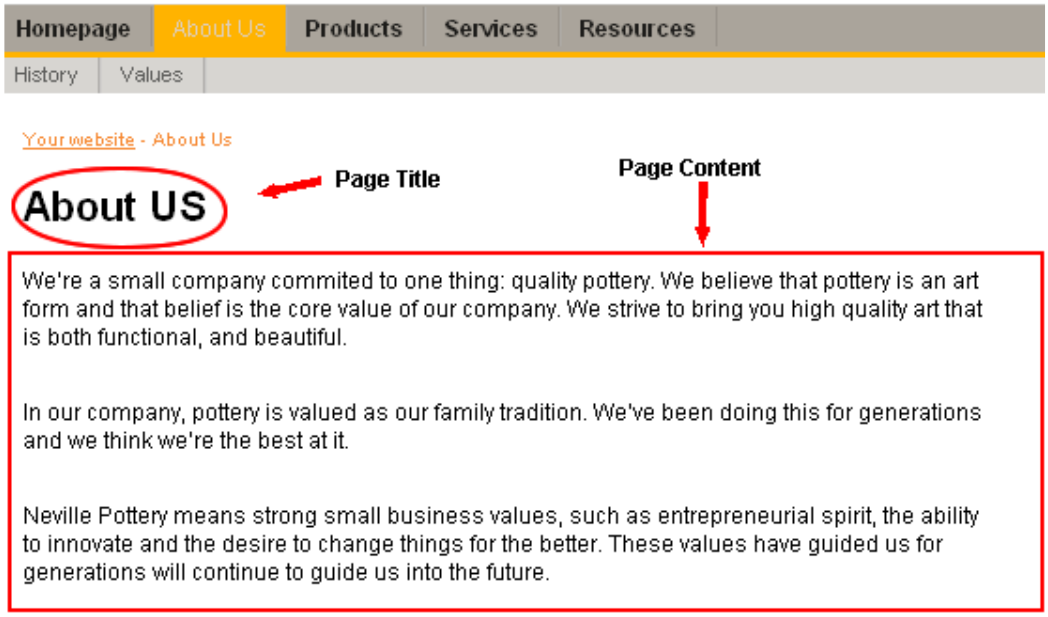
Mai multe la adresa: http://pear.php.net/package/HTML_Template_Flexy/

Structura unui template

Din punct de vedere structural putem sa vb despre partile constituente ale unui template care sunt vizibile in front end si despre modul in care acestea se genereaza.

Astfel, in cadrul unui template Bluo poti distinge urmatoarele sectiuni:

- Partile inserate cu ajutorul modulelor Bluo
- Continutul unei pagini care se introduce din partea de administrare
- Titlul unei pagini care deasemenea se introduce din partea de administrare.



Este foarte important modul in care introduci titlu si continutul unei pagini in codul html. Pentru acest lucru ai la dispozitie doua variabile:

pageContent – variabila ce are ca valoare continutul unei pagini introdus in partea de administrare in sectiunea Content.

Modul de apelare pentru aceasta variabila este:

```
{pageContent:h}
```

pageTitle – variabila ce are ca valoare titlul unei pagini introdus in partea de administrare in sectiunea Headline.

Modul de apelare pentru aceasta variabila este:

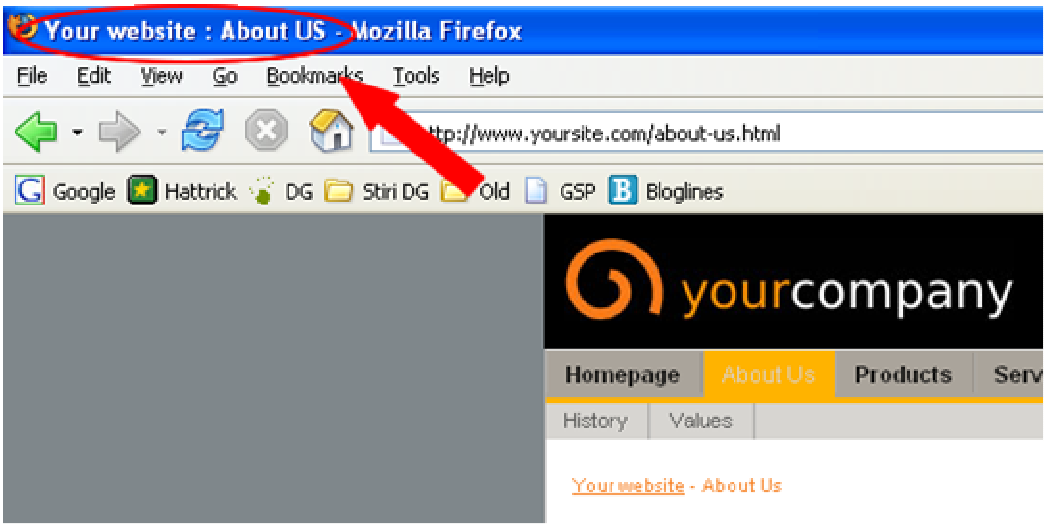
```
{pageTitle}
```

```
<td id="siteBorder">
    <div id="top">
        {runModule(#RenderMenu#,#false#,#1#,#1#,#all#,#TopMenu.html#)}
        {runModule(#RenderHTML#,#Header.html#)}
    </div>
    <div class="titleBox">
        <div class="leftTD">
            <h1>{pageTitle}</h1>
        </div>
    </div>
    <div class="topContent">
        {pageContent:h}
    </div>

    <div id="bottom"></div>
</td>
>

{runModule(#RenderMenu#,#true#,#1#,#2#,#all#,#BottomMenu.html#)}
```

Pentru ca am amintit de variabila care introduce titlul unei pagini in cadrul continutului ei, trebuie sa iti spun si despre variabila cu ajutorul careia se introduce titlul unei pagini afisata de browser.



Aceasta variabila se numeste head.

head – variabila care are ca valoare codul ce trebuie introdus intre tagurile *<head>* si respectiv *</head>.* Acest cod este necesar deoarece contine tagurile de tip *<meta>*[1] si *<title>,* cat si secventa de includere a fisierului *style.css*[2].

[1] As part of the HEAD of an HTML document, the <meta> tag provides information that describes the document in various ways. It contains valuable information for search robots to use in adding your web pages to their search indexes. A number of search engines use the information within tags as part of their algorithms. Infinity Web Technologies strongly recommends that you implement <META> tags that describe the content and keywords that describe your web site and its web pages.

[2] Despre acest fisier vom vorbi in capitolulele urmatoare

Apelarea acestei variabile intr-un fisier html se face astfel:

```
{head:h}
```

```
1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "ht
2 <html xmlns="http://www.w3.org/1999/xhtml">
3 <head>
4 <!-- HEAD -->
5 {htmlHead:h}
6 <!-- /HEAD -->
7 </head>
8
```

Iata cum arata continutul acestei variabile pentru o pagina dintr-un template Bluo:

```
1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtm
2 <html xmlns="http://www.w3.org/1999/xhtml">
3 <head>
4 <!-- HEAD -->
5         <title>Your website : About US</title>
6
7         <meta name="description" content="">
8         <meta name="keywords" content="">
9         <meta name="author" content="">
10        <meta name="rating" content="General">
11        <meta name="expires" content="never">
12
13        <meta name="language" content="english">
14        <meta name="charset" content="ISO-8859-1">
15        <meta name="robots" content="all">
16        <meta name="spiders" content="all">
17        <meta name="revisit-after" content="7 Days">
18        <meta name="email" content="contact@yoursite.ro">
19        <meta name="authors" content="yoursite.com">
20        <meta name="copyright" content="Copyright 2006 - yoursite.com">
21        <meta name="distribution" CONTENT="global">
22
23        <meta content="PHP" name="CODE_LANGUAGE">
24        <meta content="JavaScript" name="vs_defaultClientScript">
25        <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
26
27        <link href="http://www.metromind.ro/test/templates/BeautifulDawn/style.css"
28              rel="stylesheet" type="text/css">
29 <!-- /HEAD -->
30 </head>
```

Despre celelalte sectiuni, introduse cu ajutorul modulelor Bluo, vom vorbi in capitolele urmatoare.

Module

Modulele sunt niste unitati functionale care va permit adugarea de functionalitate fisierelor HTML care contin un template. De exemplu pentru a crea un meniu dinamic puteti apela un modul special creat pentru asta. Modulele sunt in general secvente de cod Flexy si cod HTML care sunt rulate in template cu functia `runModule`().

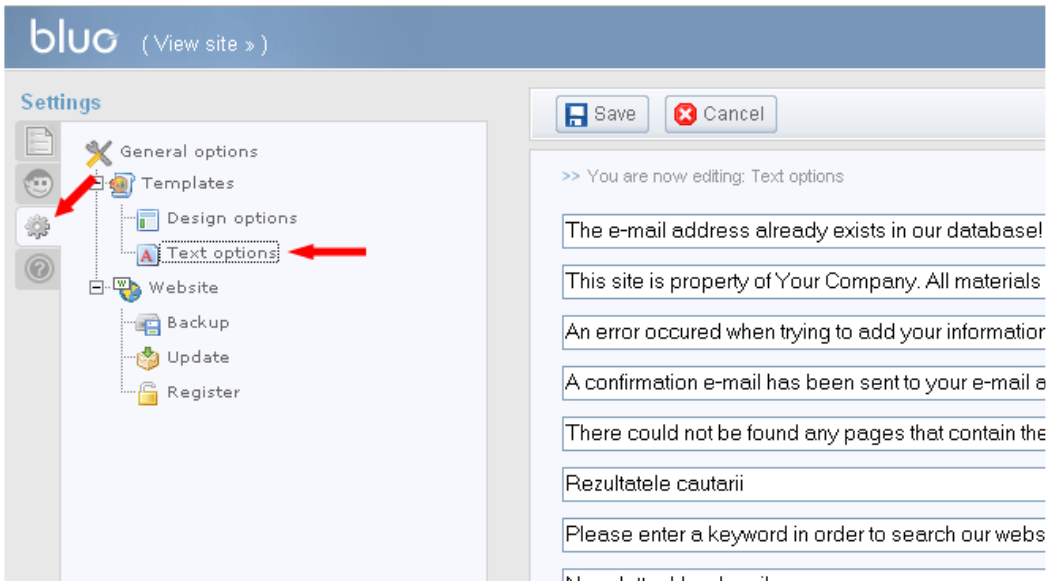
RenderText

Rol

Poti folosi modulul RenderText pentru introducerea in pagina a unor texte punctuale (de exmplu textul de pe un buton). Nu ne referim aici la partea de continut (numele paginii, continut etc).

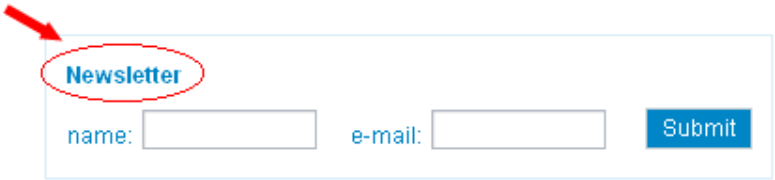
Vei putea modifica textul inserat cu ajutorul acestui modul din sectiunea *Settings->Text* options aflata in zona de administrare.

Daca nu ai reusit sa identifici inca acesta sectiune, imaginea urmatoare sigur iti va fi de folos.



Hai sa vedem un exemplu de text care arata utilitatea modulului *RenderText*:

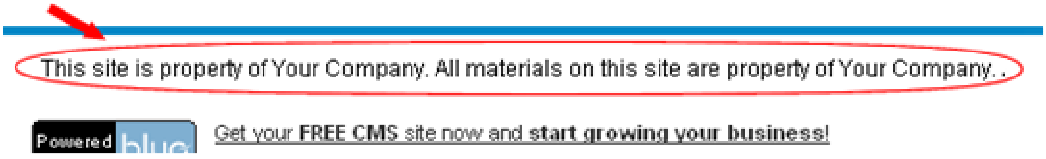
Sa presupunem ca in cadrul unei pagini din template-ul tau apare o sectiune de newsletter careia trebuie sa ii atribui un nume. Sa ii spunem Newsletter.



Dar poate dupa o perioada o sa vrei sa o denumesti Newsletter Subscription. Daca nu folosesti modulul *RenderText* pentru introducerea numelui, vei fi nevoit

sa modifici toate paginile-template unde apare acest text. Daca in schimb ai inserat textul folosind acest modul munca ta se reduce considerabil, fiind necesar doar sa faci cateva clickuri in partea de administrare.

Un alt exemplu de text care ar putea aparea pe siteul tau si pe care e indicat sa il introduci folosind *RenderText* este textul din footer in care sunt mentionate lucrurile legate de copyright si proprietate a siteului.



Mod de apelare

Tot ce trebuie sa stii pentru a introduce o secventa de text cu ajutorul modulului RenderText este modul de apelare al acestuia:

```
{runModule(#RenderText#,#TextName#)}
```

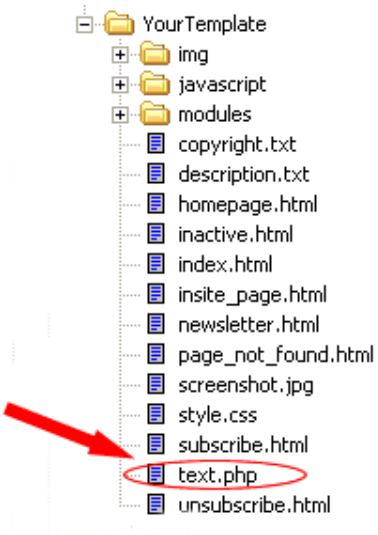
Aceasta linie de cod trebuie introdusa in pagina-template exact in locul textului pe care vrei sa il faci modificabil din partea de administrare.

Mod de lucru

Pentru a intelege mai bine modul in care opereaza acest modul trebuie sa plecam de la modul de apelare, adica de la:

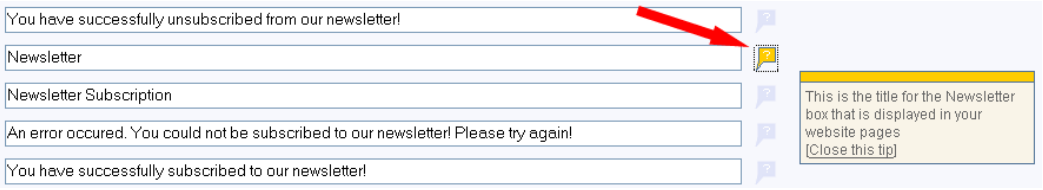
```
{runModule(#RenderText#,#TextName#)}
```

Cu aceasta linie de cod se ruleaza modulul *RenderText* si se trimite *TextName* ca parametru pentru acesta.
Parametrul setat este folosit ca si cheie pentru a cauta textul ce trebuie sa apara in locul respectiv. Unde se cauta acest text? Exista un fisier numit text.php care se gaseste in root-ul template-ului care contine un vector cu toatele textele introduse cu ajutorul modulului *RenderText*.



De ce este luat textul tocmai din acest fisier? Pentru ca asupra acestuia se poate interveni din partea de adminstrare modificandu-se textele.

Atentie! In partea de administrare nu este vizibila cheia dupa care se face cautarea in modulul *RenderText*. Motiv pentru care, daca privesti cu atentie, in fisierul text.php mai exista un vector denumit help cu ajutorul caruia se pot atribui indicii pentru fiecare text in parte aflat in vectorul text. Aceste indicii sunt vizibile in partea de administrare sub forma unor tooltip-uri, asa cum se observa in imaginea de mai jos:



Iata un exemplu de completare a vectorului help:

Sa presupunem ca vrei sa completezi in help un indiciu pentru textul *NewsletterTitle* despre care am vorbit in primul exemplu de la Modul de apelare. Astfel, fisierul *text.php* va arata in felul urmator:

```
$text = array(
                "NewsletterTitle" => "Newsletter",
                "OtherTextName1" => "text1",
                "OtherTextName2" => "text2",
        );

$help = array(
                "NewsletterTitle" => "Hint for NewsletterTitle text",
                "OtherTextName1" => "Hint for OtherTextName1 text",
                "OtherTextName2" => "Hint for OtherTextName2 text",
        );
```

Functii si variabile disponibile

Pentru acest modul nu este justificata punerea la dispozitie a vreunei functii sau variabile.

Exemple

Newsletter Title

Avem urmatoarea secventa de cod:

```
siteLink}/newsletter.php" method="POST" id="Newsletter" onSubmit="retu
 <div class="newsletter">
        <div class="newsletterTitle">Newsletter</div>

        <div class="newsletterForm">
                name:<input type="text" size="15" class="newsletterIn
                email:<input type="text" size="15" id="newsletterEmai
                <input type="submit" value="Submit" class="searchButt
        </div>

 </div>
```

Daca vrem sa introducem textul ‘Newsletter” cu modulul *RenderText*, atunci modificarea ce trebuie facuta in fisierul html este urmatoarea:

```
etter.php" method="POST" id="Newsletter" onSubmit="return newsletterV
wsletter">
lass="newsletterTitle">{runModule(#RenderText#,#NewsletterTitle#)}</d

lass="newsletterForm">
  name:<input type="text" size="15" class="newsletterInput" id="newsl
  email:<input type="text" size="15" id="newsletterEmail" class="news
  <input type="submit" value="Submit" class="searchButton Change_main
```

Dupa aceasta, deschidem fisierul *text.php*, care se afla in directorul cu numele templateului si care contine vectorul ce asociaza textul cu numele pe care l-ai setat in html. De exemplu, mai sus am ales numele NewsletterTitle pentru sirul de caractere “Newsletter”. In *text.php* va trebui sa indicam acest lucru astfel:

```
$text = array(
                "NewsletterTitle" => "Newsletter",
                "OtherTextName1" => "text1",
                "OtherTextName2" => "text2",
        );

$help = array(
                "NewsletterTitle" => "Hint for NewsletterTitle text"
                "OtherTextName1" => "Hint for OtherTextName1 text",
```

Submit Button Value
Avem urmatoarea secventa de cod intr-o pagina-template[1]:

```
<form action="{siteLink}/search.php" method='POST' onSubmit="return
        <input class="Change_mainFont" id="searchInput" type="text"
        <input type="submit" value="Search" class="searchButton" />
</form>
```

Ne propunem sa introducem textul afisat pe buton, adica valoarea inputului de tip submit, cu ajutorul modulului *RenderText*. Modificarea care trebuie facuta in fisierul html este urmatoarea:

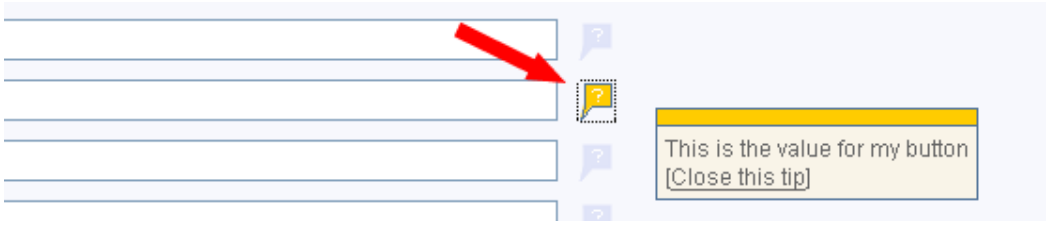
```
tion="{siteLink}/search.php" method='POST' onSubmit="return searchVal
<input class="Change_mainFont" id="searchInput" type="text" size="20"
<input type="submit" value="{runModule(#RenderText#,#ButtonValue#)}"
```

Dupa cum se observa, s-a atribuit sirului de caractere *Search* numele *ButtonValue*. In continuare, trebuie modificat fisierul *text.php*, care se afla in directorul root al template-ului, prin adaugarea unei noi:

```
$text = array(
                "ButtonValue" => "Search",
                "OtherTextName1" => "text1",
                "OtherTextName2" => "text2",
        );

$help = array(
                "ButtonValue" => "This is the value for my button",
                "OtherTextName1" => "Hint for OtherTextName1 text",
                "OtherTextName2" => "Hint for OtherTextName2 text",
        );
```

Faptul ca s-a completat si vectorul help cu linia indicata de cea de-a doua sageata face ca in partea de administare, in dreptul inputului in care se afiseaza textul *Search* sa apara un tooltip care contine explicatia *This is the value for my button*, asa cum este indicat in imaginea de mai jos:



[1] Iti reamintesc ca prin pagina template se intelege unul dintre fisierele html aflate in directorul root al template-ului

RenderHTML

Rol

Poti folosi acest modul pentru a insera o secventa de cod html care se repeta in mai multe fisiere din cadrul templateului tau.

Tot ce trebuie sa faci, practic, e sa iti creezi un fisier html in cadrul directorului modules[1] in care sa inserezi secventa de cod care se repeta, iar in locul acesteia sa apelezi modulul *RenderHTML* cu parametrii corespunzatori.

Probabil te intrebi de ce ai avea nevoie de un astfel de modul, cand poti pur si simplu sa lasi secventa aceea in fisierul initial, fara sa mai stai sa o muti. Iata si motivul:

Este foarte probabil ca dupa ce ai terminat template-ul sa vrei sa modifici o portiune dintr-o secventa de cod care apare in mai multe pagini-template. Astfel, daca nu ai folosit modulul *RenderHTML* va trebui sa iei fiecare fisier care contine secventa in cauza si sa faci aceleasi modificari PENTRU FIECARE. In schimb, daca ai inserat secventa respectiva folosind acest modul tot ce trebuie sa faci este sa deschizi fisierul creat de tine in directorul modules si care contine secventa de cod despre care vorbim si sa o modifici O SINGURA data.

Ca sa intelegi mai bine despre ce este vorba, iata cateva situatii in care este indicata folosirea acestui modul:

1. Inserarea headerului

De obicei, un site are un singur header, indiferent daca pagina vizitata este generata folosind template-ul homepage.html, index.html sau alta pagina-template. Acesta este un motiv suficient pentru a alege folosirea modului *RenderHTML* in locul alegerii variantei de a scrie in fiecare pagina-template secventa de cod pentru header.

2. Formularul de abonare la newsletter

Pentru a insera intr-o pagina-template formularul de abonare la newsletter este indicat sa fie folosit modulul RenderHTML, datorita faptului ca acesta este identic pe mai multe pagini-template.

Mod de apelare

Apelarea modului RenderHTML intr-o pagina-template se face folosind urmatorul cod:

```
{runModule(#RenderHTML#,#NumeFisier.html#)}
```

unde *NumeFisier* este numele fisierului creat de tine in directorul modules.

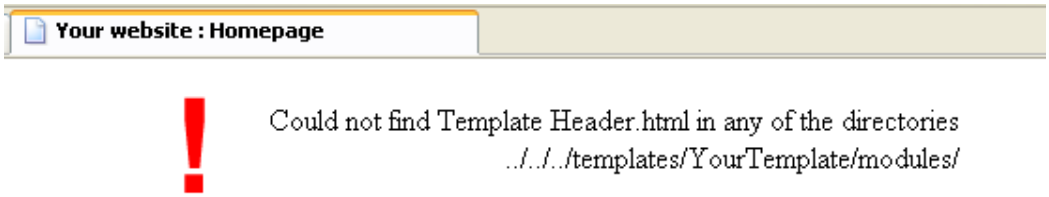
[1] Directorul modules se afla in directorul root al templateului.

Atentie! Acest cod se introduce exact in locul secventei de cod care a fost inserata in fisierul *NumeFisier.html* din directorul modules.

Mod de lucru

Pentru a intelege mai bine felul in care acest modul opereaza haide sa plecam de la secventa de apelare scrisa mai sus. Acea linie de cod poate fi interpretata astfel: se ruleaza modulul *RenderHTML* si se trimite ca parametru sirul de caractere *NumeFisier.html*. Acest parametru este folosit pentru a identifica fisierul html din directorul modules care contine codul necesar.

Foarte importanta este localizarea fisierului *NumeFisier.html* care trebuie NEAPARAT sa se gaseasca in directorul modules, deoarece in cadrul modulului *RenderHTML* se cauta fisierul html indicat de parametru NUMAI in acest director (modules) aflat in root-ul template-ului. In cazul in care ai gresit adresa de creare a fisierului ti se va returna in front end o eroare similara cu cea din imaginea urmatoare:



In schimb, daca ai creat fisierul html in locul potrivit, modulul *RenderHTML* iti va insera in pagina-template codul html din acesta exact in locul liniei de apelare. In imaginea de mai jos este prezentata pagina-template cu si, respectiv, fara secventa de cod care apare in fisierul *NumeFisier.html*, precum si continutul acestuia din urma.

La un nivel simplificat, putem spune ca tot ceea ce face *RenderHTML* este de fapt sa includa fisierul al carui nume este specificat ca parametru in pagina-template.

Functii si constante

Pentru acest modul sunt puse la dispozitie o serie de functii si variabile pe care le poti utiliza in fisierul *NumeFisier.html*[1].

Este de mentionat faptul ca toate functiile si variabilele se pot folosi numai utilizand limbajul Flexy.

[1] Am considerat ca in pagina-template s-a folosit urmatoarea secventa pentru introducerea modului RenderHTML: {runModule(#RenderHTML#,#NumeFisier.html#)}

siteLink – constanta ce contine linkul catre pagina de start a siteului dumneavoastra.

De exemplu:
siteLink = http://www.exemplu.com

Daca vrei sa vezi exact cum arata aceasta constanta nu trebuie decat sa deschizi fisierul settings.php care se afla la adresa *root/core/settings/settings.php*. Valoarea acestei constante este indicata de SITE_LINK dupa cum se poate vedea in imaginea de mai jos:

settings.php

```
define('SITE_NAME','Your Site Name');
define('SITE_ALIAS','Your Site Alias');
define('SITE_LINK','http://www.yoursite.com');
define('BUILD_PAGES_MODE','write_mode');
```

Modul de apelare este usor de intuit daca ai citit in prealabil introducerea in limbajul flexy prezentat mai sus. Astfel, in cadrul codului html, constanta siteLink se introduce folosind urmatorul cod:

```
{siteLink}
```

Daca vei incerca sa impletezi un template pentru Bluo CMS, vei observa cat de importanta este aceasta constanta si cat de mult te ajuta in scrierea codului. Iata cateva exemple unde se foloseste:

1. Scrierea sursei unei imagini:

```
{else:}
        <img src="{siteLink}/templates/YourTemplate/imagesDirectory/imageName.jpg" id='logo' alt='' />
{end:}
```

2. Inserarea unui link catre pagina de start

---

bluo_template – variabila ce contine numele template-ului curent folosit de Bluo CMS. Prin template curent se intelege ultimul template care a fost setat din partea de administrare ca fiind activ.

De exemplu:
`bluo_template= YourTemplate`

Te intrebi probabil cum iti dai seama care este denumirea unui template. Foarte simplu! Numele unui template este dat atunci cand se alege o denumire pentru directorul root al templateului. Astfel, daca template-ul tau se gaseste in directorul *MyTemplate* atunci *MyTemplate* va fi si numele templateului. In momentul in care acesta devine activ, adica este template-ul selectat in partea de administrare in sectiunea *Settings->Templates*, variabila *bluo_template* va prelua numele acestuia, adica va contine valoarea “MyTemplate”.

Daca vrei sa vezi exact cum arata aceasta variabila la un moment dat, nu trebuie decat sa deschizi fisierul settings.php care se afla la adresa *root/core/settings/settings.php*. Valoarea acestei variabile este indicata asa cum apare in imaginea de mai jos:

```
//page default options
define('DEFAULT_AUTHOR','');
define('DEFAULT_DESCRIPTION','');
define('DEFAULT_KEYWORDS','');

define('TEMPLATE','YourTemplate');

define('LOGO','');
```

settings.php

Mod de apelare pentru aceasta variabila este:

```
{bluo_template}
```

Ca si *siteLink*, aceasta variabila este importanta in scrierea adresei catre diverse fisiere din interiorul directorului root al templateului. Iata cateva exemple unde se foloseste:

1. Scrierea sursei unei imagini din interiorul directorului root al templateului:

2. Scrierea adresei catre un anumit fisier din directorul root al templateului

---

checkLogo() – este o functie cu ajutorul careia se verifica daca a fost uploadat un nou logo din partea de administrare, sectiunea *Settings-> General Options*. Functia returneaza valoarea true daca sunt indeplinite simultan urmatoarele doua conditii:

1. in fisierul settings[1], constanta LOGO este setata, adica nu este nula

2. daca LOGO nu e nula, atunci exista imaginea aflata la adresa root/uploaded/LOGO[2]

Daca una dintre acestea nu este adevarata atunci functia *checkLogo()* returneaza valoarea false si in acest caz trebuie afisat logo-ul default, adica imaginea asociata ca logo pentru care se stie adresa unde se afla.

Trebuie mentionat faptul ca, in momentul in care se selecteaza un nou logo din partea de administrare, automat constanta LOGO din settings.php ia ca valoare

[1] Aflat la adresa MyTemplate/core/settings/settings.php
[2] Unde LOGO se inlocuieste cu valoarea sa

numele imaginii selectate[1], iar aceasta imagine este salvata in directorul uploaded aflat in root-ul template-ului.

Pentru a putea scrie calea catre noua imagine-logo ai la dispozitie variabila *logo*[2] care iti returneaza exact valoarea indicata de constanta LOGO, adica numele si extensia imaginii in cauza. Variabila *logo* se foloseste de obicei numai impreuna cu functia *checkLogo().*

Iata un exemplu de folosire a functiei:

```
{if:checkLogo()}
    <a href='{siteLink}'>
        <img src='{siteLink}/uploaded/{logo}?{time}' alt='' id='logo' />
    </a>
{else:}
    <a href='{siteLink}'>
        <img src='{siteLink}/templates/{bluo_template}/img/logo.jpg?{time}' alt='' />
    </a>
{end:}
```

In exemplu de mai sus se observa ca daca functia *checkLogo* returneaza valoarea true, adica s-a uploadat o imagine noua pe post de logo, se afiseaza imaginea respectiva, in caz contrar se afiseaza o imagine care se gaseste in directorul *img*[3] special creat pentru stocarea de imagini.

De asemenea se observa aparitia unei variabile *{time}* in numele imaginii. Acesta a fost introdusa pentru a evita eventuala memorare in cache-ul browserului a numelui imaginii logo, fapt ce ar duce la afisarea in front end a imaginii anterioare, desi din partea de administrare aceasta a fost inlocuita. Aparitia acestei variabile, care depinde de timp, face ca browserul sa interpreteze ca fiind distincte doua imagini care practic au acelasi nume. Iata cum numele unei imagini careia i s-a adaugat dupa *extensie secventa ?{time}.*

Exemple

Inserare Header

Inserarea headerului este un exemplu de folosire evidenta a modului RenderHTML.
Hai sa consideram ca pagina-template pe care o modifici contine urmatoarea secventa de cod :

[1] Daca imaginea se numeste logo.png atunci LOGO = 'logo.png'. Este important de retinut faptul ca numele imaginii se salveaza impreuna cu extensia ei, acest lucru fiind de ajutor in momentul in care se doreste scrierea sursei imagini pentru logo-ul uploadat.
[2] Modul de apelare al acesteia este {logo}
[3] Directorul img se gaseste in directorul root al templateului

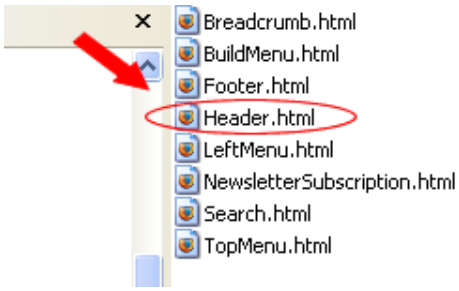
```
<div id="header">
    {if:checkLogo()}
        <a href='{siteLink}'>
            <img src='{siteLink}/uploaded/{logo}?{time}' alt='' id='logo' />
        </a>
    {else:}
        <a href='{siteLink}'>
            <img src='{siteLink}/templates/{bluo_template}/img/logo.jpg?{time}' alt='' />
        </a>
    {end:}

    {runModule(#RenderHTML#,#Search.html#)}
</div>
```

Practic, cu acest cod se insereaza logoul in pagina web.
Daca alegi sa folosesti modulul *RenderHTML* pentru introducerea codului din header atunci aceasta pagina va arata astfel:

```
<div id="header">
    {runModule(#RenderHTML#,#Header.html#)}
    {runModule(#RenderHTML#,#Search.html#)}
</div>
```

Tot ce am facut pana aici a fost sa apelam modulul la care ne referim. Acum trebuie sa cream fisierul html al carui nume a fost trimis ca parametru in linia de apelare. Atentie, insa, la adresa de creare! Structura directorului modules trebuie sa arate astfel:



modules

Continutul fisierului html este urmatorul:

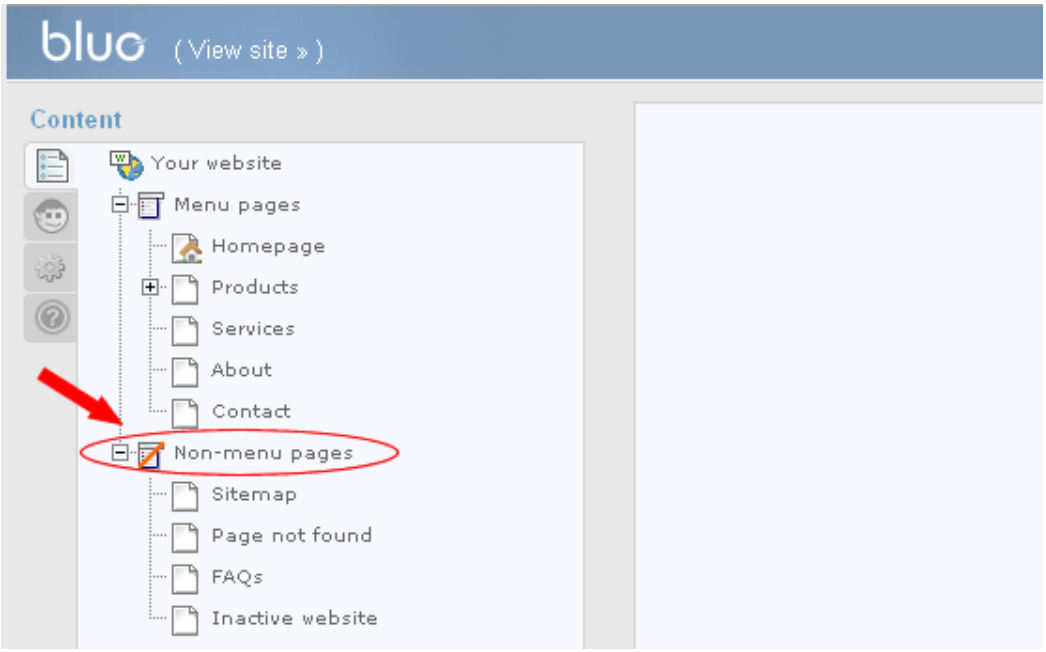
```
{if:checkLogo()}
    <a href='{siteLink}'>
        <img src='{siteLink}/uploaded/{logo}?{time}' alt='' id='logo' />
    </a>
{else:}
    <a href='{siteLink}'>
        <img src='{siteLink}/templates/{bluo_template}/img/logo.jpg?{time}' alt='' />
    </a>
{end:}
```

Header.html

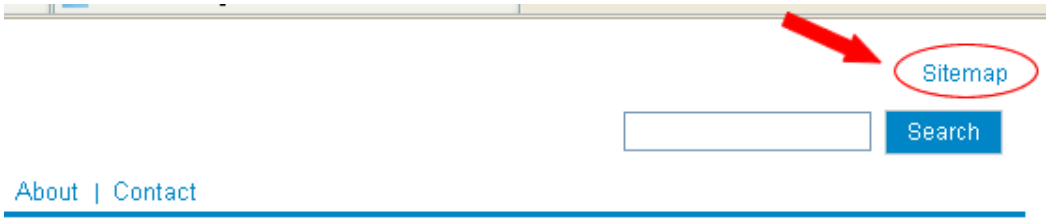
RenderInfo

Rol

RenderInfo se foloseste atunci cand vrei sa introduci numele sau link-ul unei singure pagini din site-ul tau intr-una din sectiunile template-ului, diferita de aceea unde se adauga continutul paginii[1].
Utilitatea modulului se observa atunci cand ai de introdus un link catre o pagina de genul SiteMap, FAQs, Terms of use, care de obicei nu sunt integrate in nici unul dintre meniurile de pe site. Faptul ca nu apar in nici unul dintre meniuri implica pozitionarea lor in categoria *Non-meniu pages* si se gasesc in tree-ul din partea de administrare in zona indicata de imaginea urmatoare:



Sa presupunem ca vrei sa introduci link-ul din imaginea de mai jos.



Pentru asta trebuie sa cunosti doua elemente:

- link-ul catre pagina SiteMap
- numele paginii

Probabil o sa te intrebi de ce trebuie sa pui numele paginii cu modulul *RenderInfo* cand sti aceasta se numeste SiteMap. Insa gandeste-te la

[1] In continutul paginii se poate pune link catre o pagina cu ajutorul editorului.

urmatorul aspect: in momentul in care vei vrea sa schimbi numele acestei pagini va trebui sa intervii in cod si sa faci modificarea, pe cand daca o sa folosesti RenderInfo, in momentul in care vei schimba din partea de administrare numele acestei pagini se va face automat acest lucru si in front end-ul siteului tau.

Atentie! Pentru a putea folosi acest modul trebuie sa cunosti id-ul paginii careia vrei sa-l afli link-ul/numele. Cum afli acest id? Foarte simplu. Te loghezi in partea de administrare, faci click dreapta pe pagina in cauza si apesi pe Previzualizare. In address bar o sa scrie ceva de genul:

http://www.yoursite.com/index.php?**id=*idValue***&admin=...

Id-ul de care ai nevoie este chiar *idValue*.

Mod de apelare

Apelarea modului *RenderInfo* intr-o pagina-template se face folosind urmatorul cod:

```
{runModule(#RenderInfo#,#idValue#,#infoName#)}
```

unde:

- *idValue* este id-ul paginii pentru care se solicita informatia
- *infoName* este numele informatiei. Acest parametru poate sa ia doua valori distincte:
    1. **name** – in acest caz, apelarea modului *RenderInfo* va avea ca rezultat numele paginii cu id-ul *idValue.*
    2. **link** – in acest caz, apelarea modului *RenderInfo* va avea ca rezultat linkul catre pagina cu id-ul *idValue.*

Atentie! Aceasta linie de cod se introduce exact in locul unde se doreste inserarea numelui/linkului paginii in cauza.

Mod de lucru

Ca si in cazul modulelor de pana acum iti voi explica modul de lucru plecand de la interpretarea liniei de cod care apeleaza RenderInfo:

```
{runModule(#RenderInfo#,#idValue#,#infoName#)}
```

Aceasta linie de cod s-ar “traduce” astfel: se ruleaza modulul *RenderInfo* si se trimit ca parametrii *idValue* si *infoName*. Acesti parametrii sunt folositi pentru a returna informatia dorita.

Functii si variabile disponibile

Pentru acest modul nu este justificata punerea la dispozitie a vreunei functii sau variabile.

Exemple

Crearea unui meniu

Desi ai la dispozitie si modulul *RenderMenu*, cu ajutorul caruia poti crea meniuri si pe care ti-l voi descrie mai jos, sunt cazuri in care poti realiza cu ajutorul modulului *RenderInfo* un meniu pe care nu l-ai putea crea cu nici un alt modul, asa cum este si cazul celui din imaginea urmatoare:

Iata cum arata, inainte de folosirea vreunui modul Bluo, codul html pentru sectiunea din imaginea de mai sus:

```
<div id='topRightMenu'>
    <a href="{siteLink}/faqs.html" class='topRightLink'>FAQs</a>
    <span class='topRightSpacer'>::</span>
    <a href='{siteLink}/sitemap.php' class='topRightLink'>SiteMap</a>
</div>
```

In pagina-template, codul pe care trebuie sa il scrii este insa urmatorul:

```
<div id='topRightMenu'>
    <a href="{runModule(#RenderInfo#,#306#,#link#)}" class='topRightLink'>
        {runModule(#RenderInfo#,#306#,#name#)}
    </a>
    <span class='topRightSpacer'>::</span>
    <a href='{siteLink}/sitemap.php' class='topRightLink'>
        {runModule(#RenderInfo#,#103#,#name#)}
    </a>
</div>
```

Inserarea unui link catre o pagina din sectiunea Menu Pages

Ca sa nu intelegi ca acest modul se poate aplica doar in cazul paginilor din sectiunea *Non-Menu Pages*, iata un exemplu de link catre o pagina din categoria *Menu Pages*.

Sa consideram ca nu vrei ca pagina home sa fie inclusa in meniu principal al site-ului tau[1] si ca vrei sa inserezi un link catre aceasta pagina intr-o sectiune distincta.

[1] Acest lucru este posibil prin folosirea adecvata a modului RenderMenu. Pentru mai multe detalii vezi descrierea acestui modul.

Consideram imaginea de mai jos:

t1 description

Codul html initial, fara folosirea vreunui modul Bluo, pentru sectiunea din imaginea de mai sus este:

```
<div id='topRightMenu'>
    <a href='{siteLink}' class='topRightLink'>
        Home
    </a>
</div>

{runModule(#RenderHTML#,#Header.html#)}
```

In pagina-template, codul pe care trebuie sa il scrii este urmatorul:

```
<div id='topRightMenu'>

    <a href="{runModule(#RenderInfo#,#1#,#link#)}" class='topRightLink'>
        {runModule(#RenderInfo#,#1#,#name#)}
    </a>

</div>
```

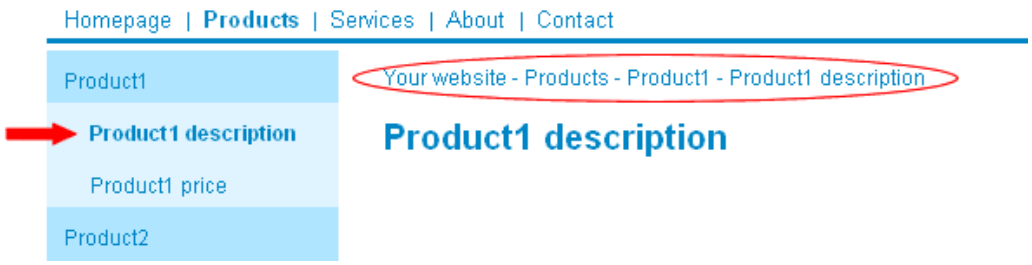
RenderBreadcrumb

Rol

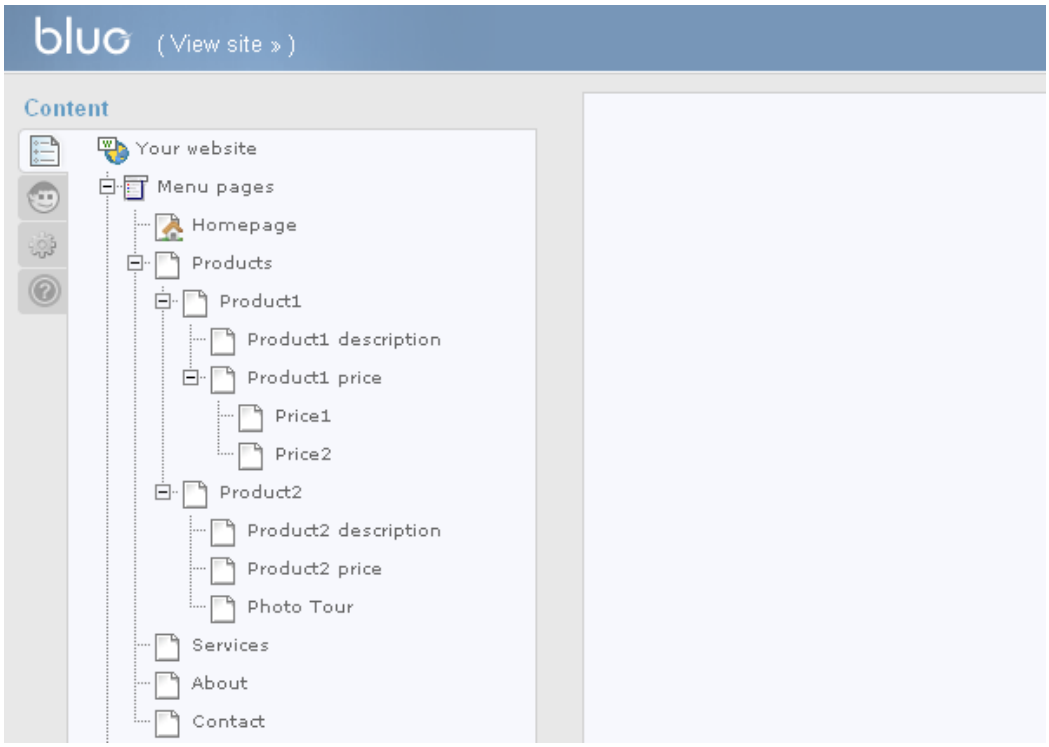
Rolul acestui modul este de a te ajuta sa generezi o sectiune de tip breadcrumb. Ce este acela un **breadcrumb**? Iata cum s-ar putea defini aceasta notiune:

Breadcrumb este o secventa de linkuri catre paginile superioare locatiei curente, aratand ruta pe care a parcurs-o vizitatorul pentru a ajunge pe acea pagina.

Daca inca nu ai inteles despre ce e vorba, iata o imagine sugestiva cu o secventa de acest tip:



Practic, breadcrumb-ul arata drumul parcurs pe niveluri pana la pagina curenta. Iata cum arata structura tree-ului pentru cazul de mai sus:



Daca site-ul tau contine o astfel de sectiune denumita Breadcrumb atunci se impune folosirea modului RenderBreacrumb.

Mod de apelare

In cazul in care siteul tau contine o sectiune de breadcrumb atunci tot ce trebuie sa faci pentru crearea ei este sa inserezi urmatorul cod in fisierul html din pagina-template in locul unde vrei sa apara in front end-ul siteului:

```
{runModule(#RenderBreadcrumb#,#NumeFisier.html#)}
```

unde `NumeFisier` este numele fisierului creat de tine in directorul **modules**.

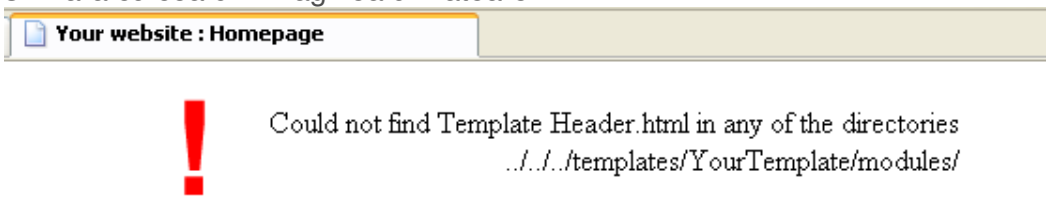
Acest mod de apelare, cat si faptul ca se lucreaza cu un fisier din directorul modules, te face poate sa crezi ca acest modul face acelasi lucru ca si *RenderHTML*, dar vei vedea ca sunt multe elemente care le deosebesc, aceasta distinctie facandu-se in special datorita functiilor si variabilelor pe care le pune fiecare la dispozitie.

Mod de lucru

Pentru a intelege mai bine felul in care acest modul opereaza haide sa plecam de la secventa de apelare scrisa mai sus. Acea linie de cod poate fi interpretata astfel: se ruleaza modulul *RenderBreadcrumb* si se trimite ca parametru sirul

de caractere *NumeFisier.html*. Acest parametru este folosit pentru a identifica fisierul html din directorul modules care contine codul necesar pentru crearea sectiunii de breadcrumb.

Foarte importanta este localizarea fisierului *NumeFisier.html* care trebuie NEAPARAT sa se gaseasca in directorul modules, deoarece in cadrul modulului *RenderBreadcrumb* se cauta fisierul html indicat de parametru NUMAI in acest director (modules) aflat in root-ul al template-ului. In cazul in care ai gresit adresa de creare a fisierului ti se va returna in front end o eroare similara cu cea din imaginea urmatoare:



In schimb, daca ai creat fisierul html in locul potrivit, modulul *RenderHTML* iti va insera in pagina-template codul html din acesta exact in locul liniei de apelare.

La un nivel simplificat, putem spune ca tot ceea ce face RenderBreadcrumb este de fapt sa includa fisierul al carui nume este specificat ca parametru in pagina-template.

Pentru modulul RenderBreadcrumb este foarte important continutul fisierului *NumeFisier.html*, deoarece in acesta trebuie sa se scrie codul care genereaza elementele breadcrumbului. Am spus elemente pentru ca asa cum ti-am prezentat in capitolul Rolul, breadcrumbul este compus din mai multe link-uri.

Functii si variabile

breadcrumbItems – este un vector care contine denumirile tuturor paginilor necesare construirii breadcrumbului exceptand pagina de start.

Este foarte important sa stii ca elementele sunt introduse in acest vector exact in ordinea corecta. Deci, pentru a afisa corect breadcrumbul, nu trebuie sa mai faci alte verificari, ci pur si simplu sa afisezi vectorul *breadcrumbItems*.

Cheia dupa care se face indexarea vectorului este id-ul paginii, iar valoarea este denumirea paginii:

```
breadcrumbItems [pageId] = denumirePagina;
```

Pentru a intelege modul cum trebuie sa il apelezi urmareste exemplul din capitolul Exemple.

root_name – este o constanta ce contine numele pe care l-ai ales pentru site-ul tau.

Poti sa aflii valoarea acestei constante deschizand fisierul *settings.php* care se afla la adresa *root/core/settings/settings.php*. Constanta este indicata de *SITE_ALIAS* asa cum cum se poate vedea si in imaginea de mai jos:

```
12    define('DB','bluocms_site');
13    define('DSN',"mysql://".USER.":".PASS."@".HOST."/".DB);
14
15    define('SITE_NAME','www.yoursite.com');
16    define('SITE_ALIAS','Your Site');
17    define('SITE_LINK','http://www.yoursite.com');
18    define('BUILD_PAGES_MODE','write_mode');
19
20    //users options
21    define('USERS_PER_PAGE','15');
22    define('NEWSLETTER_FROM','admin@yoursite.com');
```

settings.php

Mod de apelare

```
{root_name}
```

---

root – constanta ce contine linkul catre pagina de start a siteului dumneavoastra echivalenta cu *siteLink* din explicatia din cadrul modului *RenderHTML*

De exemplu:

```
root = http://www.exemplu.com
```

Daca vrei sa vezi exact cum arata aceasta constanta nu trebuie decat sa deschizi fisierul *settings.php* care se afla la adresa *root/core/settings/settings.php*. Valoarea acestei constante este indicata de *SITE_LINK* dupa cum se poate vedea in imaginea de mai jos:

settings.php

```
define('SITE_NAME','Your Site Name');
define('SITE_ALIAS','Your Site Alias');
define('SITE_LINK','http://www.yoursite.com');
define('BUILD_PAGES_MODE','write_mode');
```

Modul de apelare
In cadrul codului html constanta root se introduce folosind urmatorul cod:

```
{root}
```

Aceasta constanta este importanta deoarece in vectorul *breadcrumbItems*, asa cum am precizat, nu este inclusa si pagina de start, pagina pe care breadcrumbul trebuie sa o contina.

---

bluo_template – constanta ce contine numele template-ului curent folosit de Bluo CMS. Prin template curent se intelege ultimul template care a fost setat din partea de administrare ca fiind activ.
De exemplu:

```
bluo_template= YourTemplate
```

Daca vrei sa vezi exact cum arata aceasta variabila la un moment dat, nu trebuie decat sa deschizi fisierul *settings.php* care se afla la adresa *root/core/settings/settings.php*. Valoarea acestei variabile este identica cu cea indicata de *TEMPLATE* asa cum apare in imaginea de mai jos:

```
//page default options
define('DEFAULT_AUTHOR','');
define('DEFAULT_DESCRIPTION','');
define('DEFAULT_KEYWORDS','');

define('TEMPLATE','YourTemplate');

define('LOGO','');
```

settings.php

Mod de apelare pentru aceasta variabila este:

```
{bluo_template}
```

Aceasta variabila este importanta in scrierea adresei catre diverse fisiere din interiorul directorului root al templateului.

---

compare(pageId) – este o functie care primeste ca parametru un id si verifica daca acesta este egal sau nu cu id-ul paginii curente1. Functia intoarce valoarea true daca valoarea parametrului coincide cu id-ul paginii curente, in caz contrar returneaza valoarea false.

De ce este nevoie de aceasta functie? Numele paginii curente figureaza in cadrul breadcrumbului, dar este foarte important faptul ca nu apare ca link, ci doar ca text simplu. Si pentru aceasta e nevoie de o functie cu ajutorul careia sa poti sa verifici care element din vectorul breadcrumbItems nu trebuie sa apara ca link.

Apelarea acestei functii se face folosind o structura de tip IF.

```
{if:compare(id)}
```

unde *id* este id-ul paginii pentru care se face verificarea.

---

[1] Prin pagina curenta se intelege pagina vizita
[3] Prin pagina curenta se intelege pagina vizitata

getLink(pageId) – este o functie care primeste ca parametru un id al unei pagini si returneaza link-ul catre aceasta pagina.

Apelarea acestei functii se face astfel:

```
{getLink(id)}
```

unde *id* este id-ul paginii pentru care se doreste aflarea link-ului.

---

isEmpty() – este o functie cu ajutorul careia se verifica daca vectorul breadcrumbItems contine vreun element, adica daca este necesar sa se afiseze breadcrumb-ul sau nu.

Functia intoarce *true* daca *breacrumbItems* este vid si *false* daca acest vector contine cel putin un element.

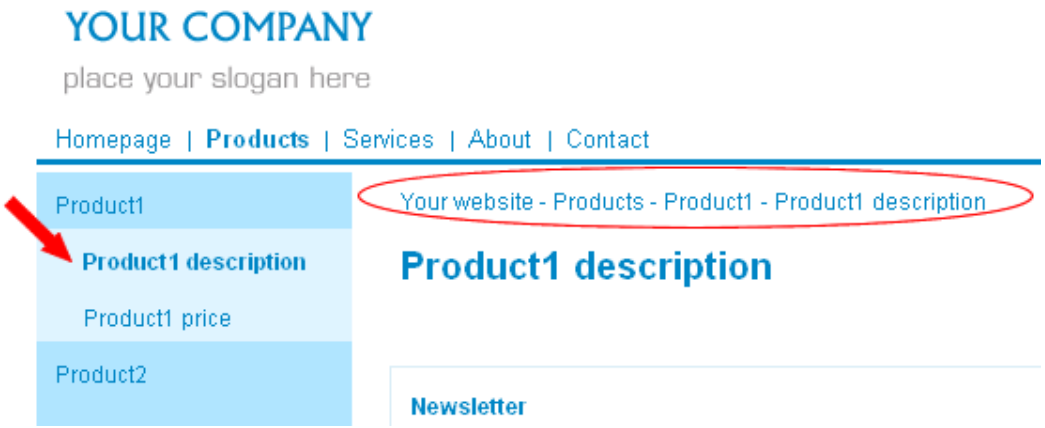
Am mentionat mai sus ca vectorul breadcrumbItems nu contine si pagina de start, ceea ce inseamna ca daca functia *isEmpty()* returneaza valoarea true pagina vizitata este chiar pagina de start, caz in care nu este necesara aparitia sectiunii de breadcrumb.

Modul de apelare pentru aceasta functie este asemanator cu cel al functiei *compare(id)*, deoarece se foloseste tot o structura de tip IF. Deoarece ne intereseaza cazul in care se poate afisa breadcrumbul, adica acela in care functia *isEmpty()* returneaza valoarea false, este preferata apelarea prin negarea rezultatului functiei, adica:

```
{if:!isEmpty()}
```

Exemple

Consideram breadcrumbul din imaginea de mai jos:



Ne propunem sa scriem codul necesar pentru generarea acestuia.

Codul care trebui scris in template-ul paginii in care va aparea acest breadcrumb este urmatorul:

```
{runModule(#RenderBreadcrumb#,#Breadcrumb.html#)}
```

Neavand decat un singur parametru, acest cod este foarte simplu de scris, singura decizie care trebuie sa o iei fiind numele fisierului in care o sa scrii codul pentru breadcrumb.

Breadcrumb.html arata astfel:

```
{if:!isEmpty()}
<a class="breadcrumbLink" href="{root}" title="{root_name}">
    {root_name}
</a> -
{foreach:breadcrumbItems,pos,name}
    {if:compare(pos)}
        <span class="breadcrumbLink">{name}</span>
    {else:}
        <a class="breadcrumbLink" href="{getLink(pos)}" title="{name}">
            {name}
        </a> -
    {end:}
{end:}
{end:}
```

Hai acum sa comentam putin codul din *Breadcrumb.html*

Primul lucru care se face este sa se verifice daca vectorul breadcrumbItems este sau nu vid. In cazul in care exista elemente in acest vector se afiseaza link-ul catre pagina de start despre care am mentionat mai sus ca nu este inclus in *breadcrumbItems*.

Pasul urmator consta din parcurgerea vectorului si afisarea tuturor elemetelor din acesta. Datorita faptului ca numele pagini curente nu trebuie sa fie link, se mai face o verificare inainte ca un element din vector sa fie afisat, si anume se foloseste functia *compare(pageId)*.

Trebuie sa acorzi o atentie sporita pentru inchiderea corecta a structurilor flexy, adica sa pozitionezi corect instructiunea *{end:}*.

RenderMenu

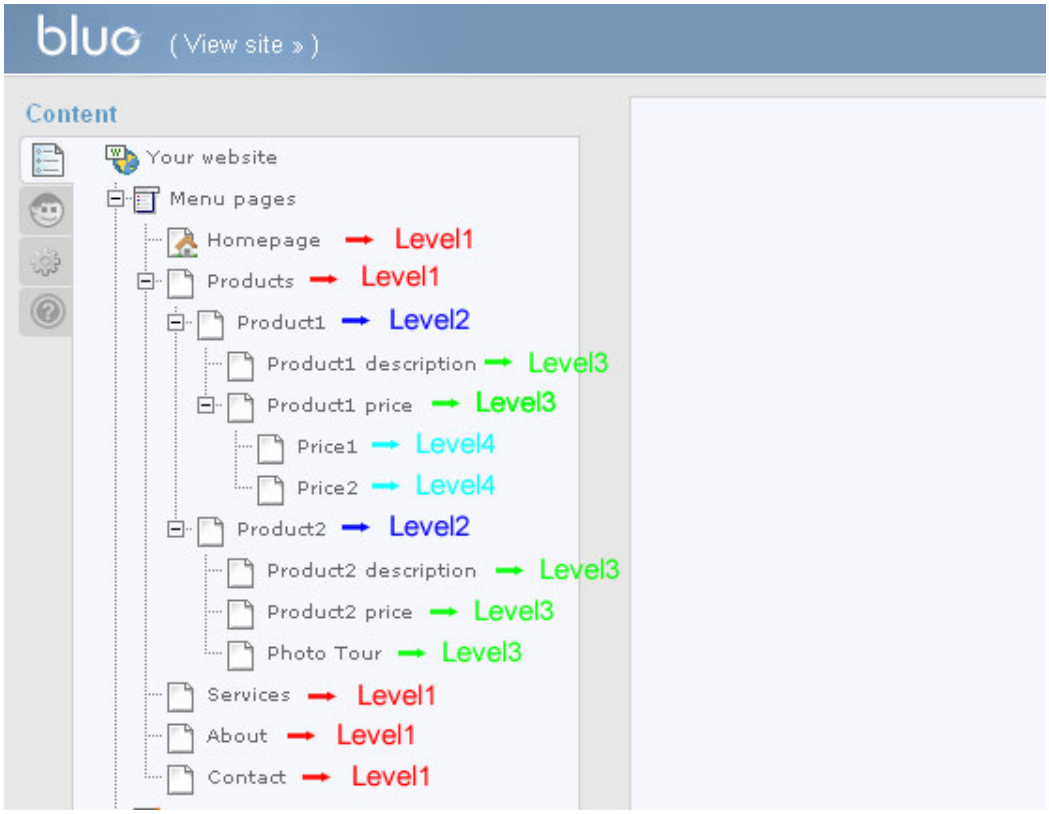
Rol

Pentru un site web meniul de navigare este un element foarte important!

Modulul *RenderMenu* este util tocmai pentru acest lucru: te ajuta sa creezi diverse meniuri pe care site-ul tau le pune la dispozitia utilizatorilor.

Asa cum bine stii, pot exista meniuri in care sunt afisate doar pagini de pe nivelul 1 sau doar pagini de pe nivelul 2 sau meniuri care sa contina toate paginile de pe nivelurile 1 si 2. Bineinteles ca aceasta afirmatie poate fi extinsa pana la n niveluri, in functie de modul de organizare a paginilor in site-ul tau.

Daca nu cunosti modul de numerotare al nivelurilor imaginea urmatoare te ajuta mult:



Practic, *RenderMenu* nu face altceva decat sa genereze un meniu in functie de mai multi parametrii pe care ti-i voi prezenta in randurile urmatoare.

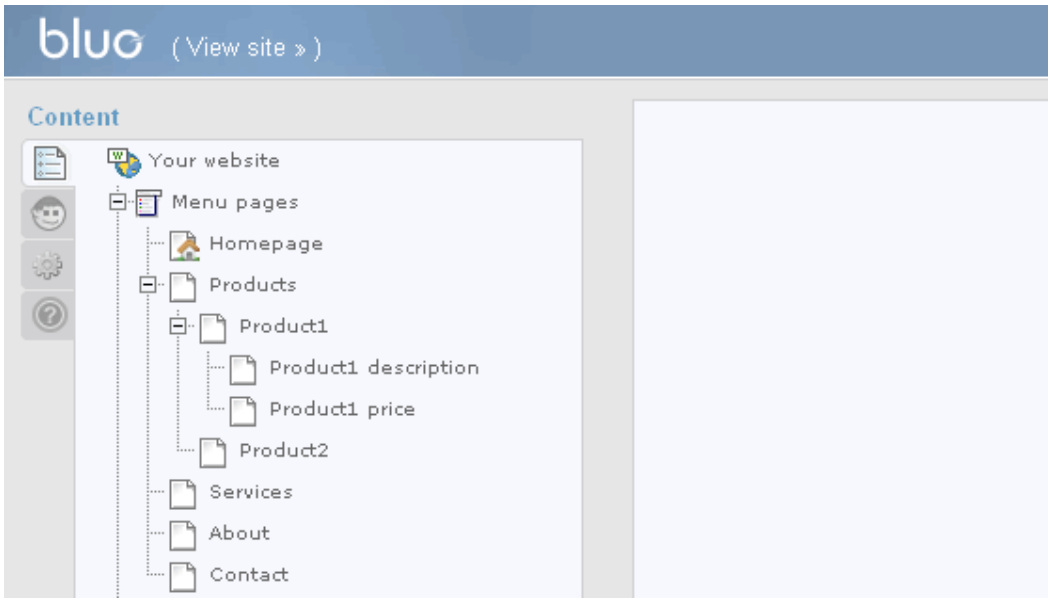
Mod de apelare

Pentru a crea un meniu cu *RenderMenu*, linia de cod care trebuie inserata in pagina-template este urmatoarea:

```
{runModule(#RenderMenu#,#includeHomepage#,#start#,#end#,#expand#,#NumeFisier.html#)}
```

Unde:

- primul parametru constituie denumirea modulului si va primi intotdeuna valoarea *RenderMenu* pentru a se rula acest modul
- **includeHomepage** este un parametru care poate lua valorile true sau false, in functie daca se doreste sau nu includerea paginii de start intre elementele meniului.
- **start** este parametrul ce indica nivelul de la care se porneste crearea meniului. Valorile pe care acesta le poate lua sunt:
  - 1,2,3 … - adica indicarea numarului nivelului dorit ca nivel de start
  - *CURRENT*, *CURRENT+1*, *CURRENT+2* … - ceea ce inseamna ca nivelul de start este ales in functie de nivelul pe care se afla pagina curenta. Astfel, daca parametrului *start* i se da valoarea *CURRENT+1*, iar pagina vizitata se afla pe nivelul 2, atunci nivelul de la care se porneste generarea modulului este 2+1. Practic, constanta *CURRENT* indica nivelul paginii curente.
- **end** este parametru similar lui *start* care indica nivelul pana la care se creaza meniul. Valorile pe care le poate lua sunt:
  - 1,2,3 … - adica indicarea numarului nivelului dorit ca nivel de sfarsit
  - *CURRENT*, *CURRENT+1*, *CURRENT+2* … - ceea ce inseamna ca nivelul pana la care se construieste meniul este ales in functie de nivelul pe care se afla pagina curenta. Astfel, daca parametrului *end* i se da valoarea *CURRENT+1*, iar pagina vizitata se afla pe nivelul 3, atunci nivelul de sfarsit al meniului este 3+1.
- **NumeFisier.html** – este numele fisierului in care se introduce codul html pentru meniu; acest fisier trebuie sa se gaseasca in directorul modules din cadrul directorului root al templateului tau.
- **expand** este un parametru care poate lua valorile:
  - **all** – adica meniul tau va contine toate paginile aflate pe nivelurile dintre *start* si *end*, inclusiv.
  - **selected** – care este un expand-all particular. Adica: se porneste de la nivelul start si se expandeaza meniul numai pe ramura selectata. Pentru a intelege mai bine haide sa consideram urmatoarea configuratie a paginilor:

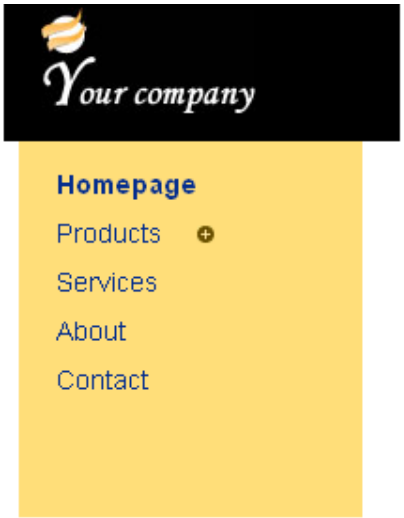


Apelam in felul urmator modulul RenderMenu:

```
{runModule(#RenderMenu#,#true#,#1#,#3#,#selected#,#NumeFisier.html#)}
```

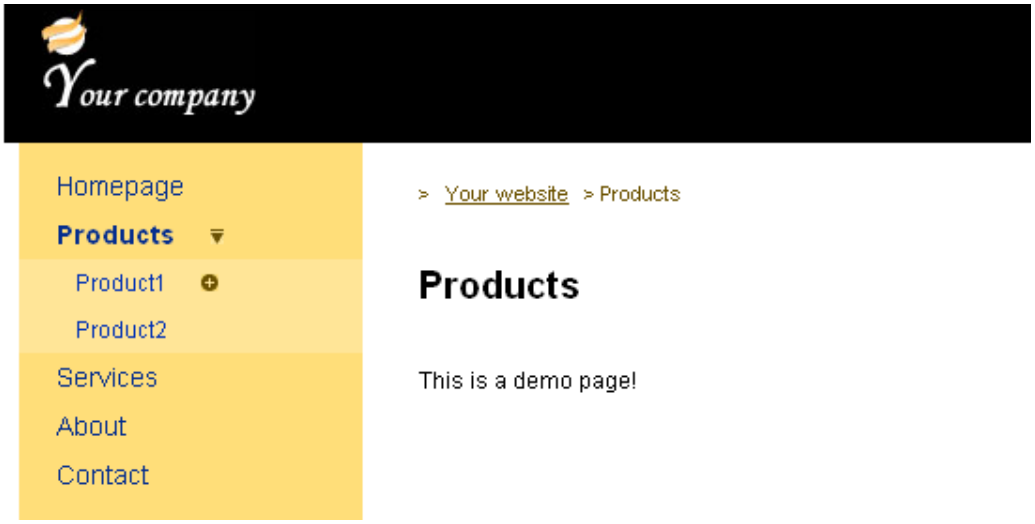
Numele fisierului html nu ne intereseaza deocamdata. Ceea ce vreau sa iti explic acum este modul de selectare a paginilor in cazul in care parametrului expand i se da valoarea selected.

Astfel, daca pagina curenta[3] este Homepage, atunci in momentul in care se genereaza meniul se va tine cont de toate paginile de pe acelasi nivel cu Homepage la care se vor adauga toate paginile direct legate de pagina selectata (dar in cazul nostru nu avem astfel de pagini). Practic, in front end, meniul va arata astfel:



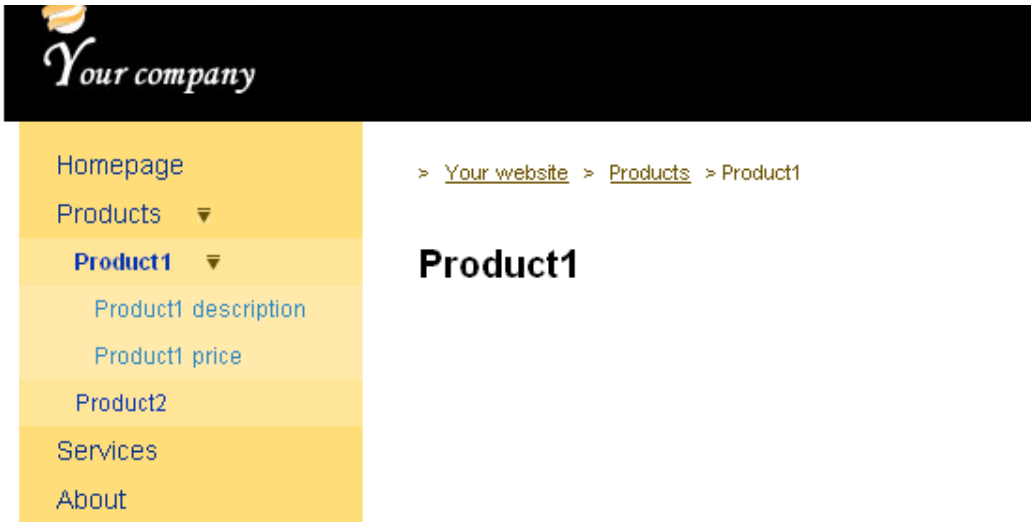
Aceeasi structura va avea meniul si in cazul in care vom avea ca pagina selectata una dintre paginile: Services, About sau Contact.

Situatia se schimba insa in momentul in care pagina selectata va fi Products. Meniul se va modifica astfel:

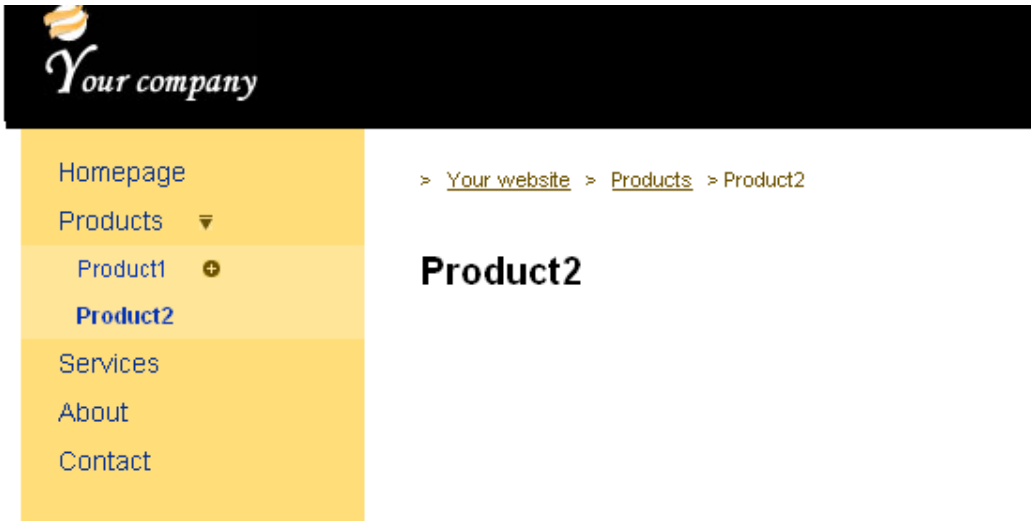


Dupa cum observi s-a expandat nivelul 2 numai in cazul paginii selectate, adica Products.

Mergand mai departe, daca selectam pagina Product1 meniul va avea urmatoarea configuratie:



Din situatia in care ne aflam acum (adica pagina Product1 este pagina curenta) selectam pagina Product2 si meniul va deveni:



Ca o concluzie pentru cazul in care parametrul expand primeste valoarea selected putem spune ca meniul este format din toate paginile de pe nivelul de start plus toate paginile de pe ramura selectata pana la nivelul n, unde n este egal cu:

- *end* in cazul in care pagina vizitata este pe nivelul *end*
- N+1 (unde N este nivelul pe care se afla pagina selectata) in cazul in care *N<end*.

Mod de lucru

Linia de apelare a modulului RenderMenu s-ar treaduce in felul urmator:

```
{runModule(#RenderMenu#,#includeHomepage#,#start#,#end#,#expand#,#NumeFisier.html#)}
```

Se ruleaza modulul *RenderMenu*, in care se trimit ca parametrii *includeHomepage*, nivelul de start (indicat de *start*), nivelul de sfarsit (indicat prin *end*), modul de expandare (*all* – sunt expandate toate paginile; *selected* – e expandata numai ramura care contine pagina selectata) si fisierul in care se face formatarea meniului (indicat de *NumeFisier.html*).

Atentie! Fisierul *NumeFisier.html* trebuie sa se gaseasca, asa cum am mai spus, in directorul modules, in caz contrar in front end va aparea urmatoarea eroare:

In cazul de mai sus parametrului *NumeFisier.html* i s-a dat valoarea *Menu.html*. Eroarea de mai sus a rezultat datorita faptului ca in directorul modules nu se afla nici un fisier html cu numele *Menu.html*.

Practic, in acest fisier se scrie sub forma de cod html modul in care sa se afiseze meniul generat in functie de parametrii trimisi in momentul apelarii (adica *includeHomepage*, *start*, *end*, *expand*, *NumeFisier.html*). Generarea meniului consta in popularea unui vector care este parcurs in fisierul NumeFisier.html si care contine toate paginile ce trebuiesc afisate. Despre acest vector iti voi spune mai multe in urmatoarea sectiune.

Functii si variabile

Setul de functii si variabile pe care le poti utiliza in cadrul fisierului *NumeFisier.html* pentru a putea afisa meniul tau sunt urmatoarele:

nodes – este vectorul care contine toate denumirile paginilor din meniu, selectate in functie de parametrii setati. Indexarea acestui vector se face dupa id-ul paginilor.

```
nodes [pageId] = denumirePagina;
```

Apelarea acestui vector se face intr-o structura de tip foreach[1], deoarece pentru afisarea meniului este necesar ca acesta sa fie parcurs. Exemple cu moduri de utilizare pentru *nodes* iti voi arata la sectiunea Exemple.

---

bluo_template – constanta ce contine numele template-ului curent folosit de Bluo CMS. Prin template curent se intelege ultimul template care a fost setat din partea de administrare ca fiind activ.
De exemplu:

[1] Vezi capitolul introductiv despre limbajul Flexy

```
bluo_template= YourTemplate
```

Daca vrei sa vezi exact cum arata aceasta variabila la un moment dat, nu trebuie decat sa deschizi fisierul *settings.php* care se afla la adresa *root/core/settings/settings.php*. Valoarea acestei variabile este identica cu cea indicata de TEMPLATE asa cum apare in imaginea de mai jos:

Mod de apelare pentru aceasta variabila este:

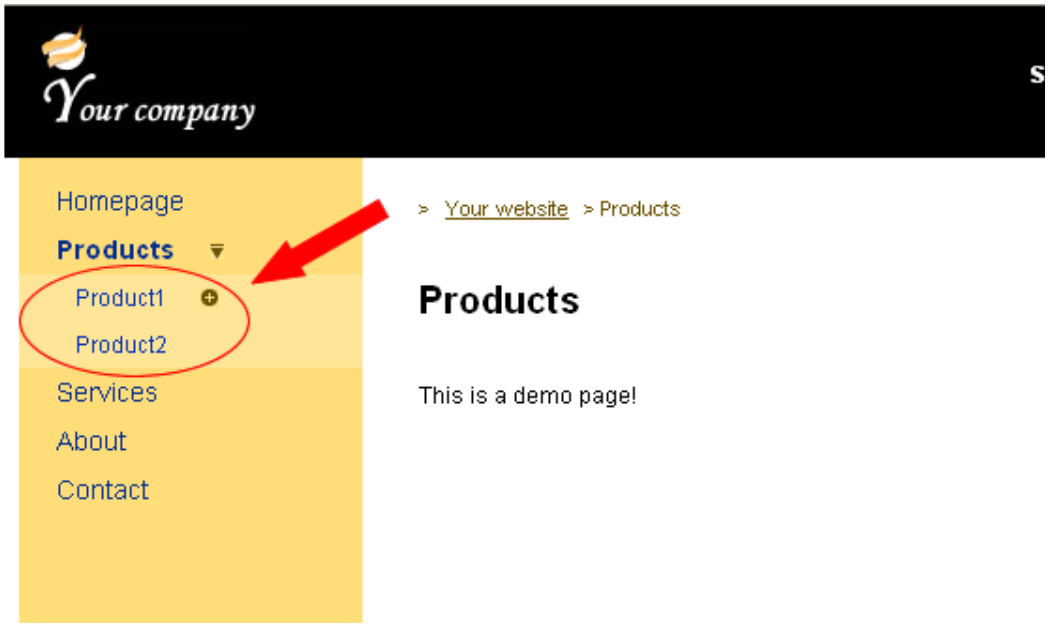
```
{bluo_template}
```

Aceasta variabila este importanta in scrierea adresei catre diverse fisiere din interiorul directorului *root* al templateului.

---

hasChildren(pageId) – este o functie cu ajutorul careia se verifica daca pagina cu id-ul *pageId* are copii, adica pagini direct legate de ea, sau nu. Functia intoarce valoarea true daca pagina pentru care se face verificarea are cel putin un copil si, respectiv, valoarea false daca aceasta pagina nu are nici un copil.

Pentru a intelege mai bine notiunea de copil pentru o pagina, in imaginea urmatoare sunt indicati copii de ordinul 1 pentru pagina Products.



Functia *hasChildren(pageId)* se apeleaza intr-o structura de tip if, astfel:

```
{if:hasChildren(pageId)}
```

---

isFirst(pageId) – este o functie cu ajutorul careia poti sa verifici daca pagina cu id-ul pageId este prima din vectorul *nodes*, adica daca este primul element din meniu.

Functia returneaza valoarea true daca pagina cu id-ul pageId este primul element din vector. In caz contrar returneaza valoarea false.

Un exemplu de apelare pentru aceasta functie este:

```
{if:isFirst(pageId)}
```

---

getLink(pageId) – este o functie care primeste ca parametru un id al unei pagini si returneaza link-ul catre aceasta pagina.

Apelarea acestei functii se face astfel:

```
{getLink(id)}
```

unde *id* este id-ul paginii pentru care se doreste aflarea link-ului.

---

isSelected(pageId) – este o functie care iti da posibilitatea sa verifici daca o pagina este sau nu vizitata in momentul generarii meniului.

Functia intoarce valoarea true daca pagina cu id-ul *pageId* este pagina curenta, in caz contrar valoarea returnata este false.

Modul de apelare al acestei functii este:

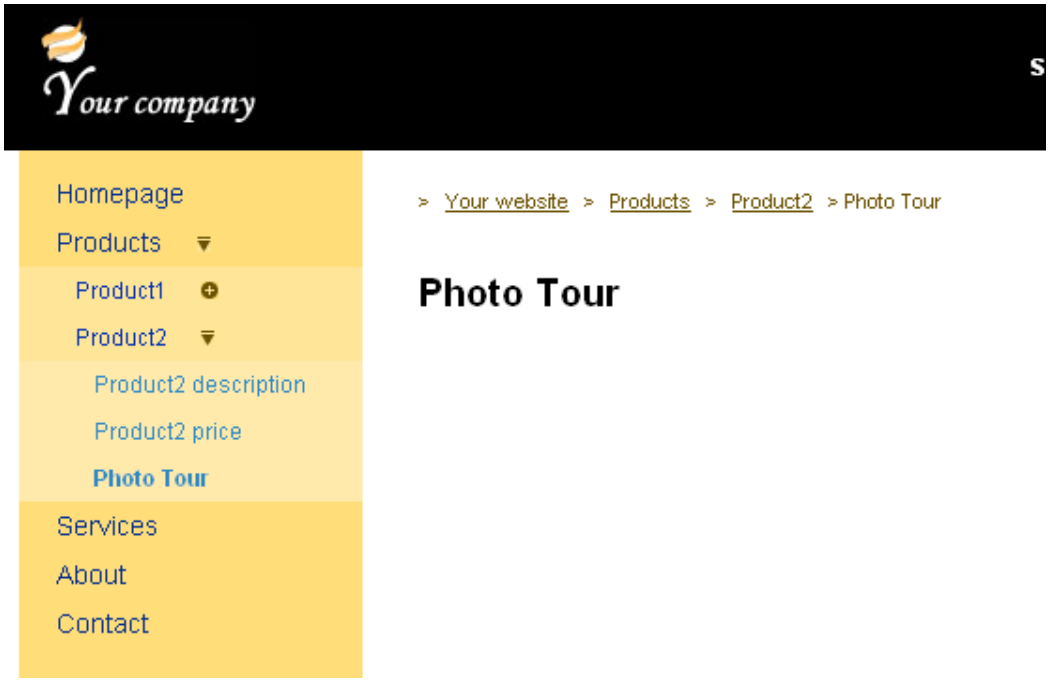
```
{if:isSelected(pageId)}
```

isSelected(pageId) este utila pentru cazul in care vrei sa aplici alte stiluri elementului din meniu care indica pagina curenta.

---

isActive(pageId) – functie care returneaza true daca pagina cu id-ul *pageId* este pagina selectata sau daca se afla pe drumul fiu-tata de la pagina selectata la pagina corespunzatoare de pe primul nivel al meniului.

Sa consideram cazul din imagine:



Dupa cum se poate vedea, pagina selectata este Photo Tour. Drumul fiu-tata de la aceasta pagina pana la nivelul de start este urmatorul:
Photo Tour -> Product2 -> Products. Deci functia *isActive(pageId)* va returna valoarea true daca pageId va indica id-ul uneia dintre paginile mentionate mai sus.

Daca privesti cu atentie imaginea, drumul fiu-tata pe care l-am mentionat mai sus se poate identifica si in breadcrumb[1] dar scris in ordine inversa.

Modul de apelare al acestei functii este:

```
{if:isActive(pageId)}
```

Ca observatie putem spune ca *isActive* se comporta la fel ca *isSelected* in cazul in care *pageId* indica valoarea id-ului paginii selectate.

---

checkLogo() – este o functie cu ajutorul careia se verifica daca a fost uploadat un nou logo din partea de administrare, sectiunea Settings-> General Options. Functia returneaza valoarea true daca sunt indeplinite simultan urmatoarele doua conditii:

1. in fisierul settings[2], constanta LOGO este setata, adica nu este nula
2.
3. daca LOGO nu e nula, atunci exista imaginea aflata la adresa root/uploaded/LOGO[3]

[1] Breadcrumb-ul apare in partea dreapta a imaginii
[2] Aflat la adresa MyTemplate/core/settings/settings.php
[3] Unde LOGO se inlocuieste cu valoarea sa

4.

Daca una dintre acestea nu este adevarata atunci functia *checkLogo()* returneaza valoarea false si in acest caz trebuie afisat logo-ul default, adica imaginea asociata ca logo pentru care se stie adresa unde se afla.

Trebuie mentionat faptul ca, in momentul in care se selecteaza un nou logo din partea de administrare, automat constanta LOGO din *settings.php* ia ca valoare numele imaginii selectate[1], iar aceasta imagine este salvata in directorul uploaded aflat in root-ul template-ului.

Pentru a putea scrie calea catre noua imagine-logo ai la dispozitie variabila *logo*[2] care iti returneaza exact valoarea indicata de constanta LOGO, adica numele si extensia imaginii in cauza. Variabila *logo* se foloseste de obicei numai impreuna cu functia *checkLogo().*

Iata un exemplu de folosire a functiei:

```
{if:checkLogo()}
    <a href='{siteLink}'>
        <img src='{siteLink}/uploaded/{logo}?{time}' alt='' id='logo' />
    </a>
{else:}
    <a href='{siteLink}'>
        <img src='{siteLink}/templates/{bluo_template}/img/logo.jpg?{time}' alt='' />
    </a>
{end:}
```

In exemplu de mai sus se observa ca daca functia *checkLogo* returneaza valoarea true, adica s-a uploadat o imagine noua pe post de logo, se afiseaza imaginea respectiva, in caz contrar se afiseaza o imagine care se gaseste in directorul *img*[3] special creat pentru stocarea de imagini.

De asemenea se observa aparitia unei variabile *{time}* in numele imaginii. Acesta a fost introdusa pentru a evita eventuala memorare in cache-ul browserului a numelui imaginii logo, fapt ce ar duce la afisarea in front end a imaginii anterioare, desi din partea de administrare aceasta a fost inlocuita.

Aparitia acestei variabile, care depinde de timp, face ca browserul sa interpreteze ca fiind distincte doua imagini care practic au acelasi nume

---

compareShift(pageId,level) – functie care returneaza true daca pagina cu id-ul pageId se afla pe nivelul level si false in caz contrar.

Ceea ce trebuie mentionat aici este faptul ca *level* ia valori incepand cu valoarea 0. Deci in cazul in care vrei sa testezi daca pagina cu id-ul *pageId* este pe primul nivel din tree trebuie sa apelezi functia astfel:

---

[1] Daca imaginea se numeste logo.png atunci LOGO = 'logo.png'. Este important de retinut faptul ca numele imaginii se salveaza impreuna cu extensia ei, acest lucru fiind de ajutor in momentul in care se doreste scrierea sursei imagini pentru logo-ul uploadat.
[2] Modul de apelare al acesteia este {logo}
[3] Directorul img se gaseste in directorul root al templateului

```
{if:compareShift(pos,0)}
```

Pentru a face acelasi lucru pentru urmatorul nivel, parametrul level trebuie sa ia valoarea 1.

Modul de apelare al acestei functii este cel indicat mai sus.

Functia este importanta pentru situatia in care meniul tau contine pagini de pe mai multe niveluri pentru ca te ajuta sa faci distinctia intre pagini de pe niveluri diferite.

---

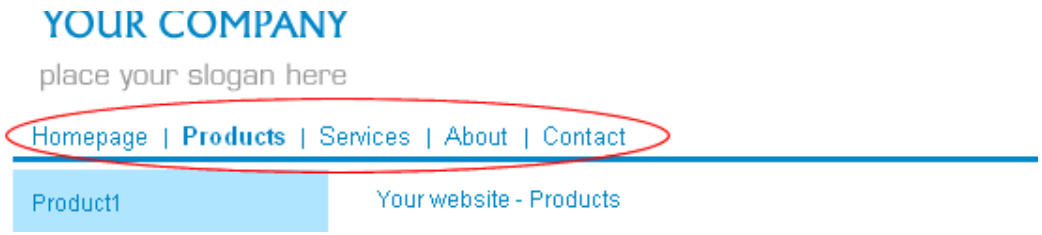
getShift(pageId) – functie care returneaza nivelul pe care se afla pagina cu id-ul pageId.

Mod de apelare:

```
getShift(pageId)}
```

Exemple

Hai sa consideram meniul de mai jos.



Ne propunem sa scriem codul necesar generarii acestui modul.

Observatii:

- toate paginile incluse se gasesc pe primul nivel al tree-ului din partea de administrare, deci parametrul start va fi egal cu parametrul end, ambele indicand nivelul comun al paginilor.
- pagina de start este inclusa in meniu, deci parametrul *includeHomepage* va avea valoarea true.

Codul care trebuie inclus in template-ul paginii unde vrei sa apara acest meniu este urmatorul:

```
{runModule(#RenderMenu#,#true#,#1#,#1#,#all#,#TopMenu.html
#)}
```

Am ales sa denumesc fisierul in care voi scrie codul pentru meniu *TopMenu.html*. In locul acestei denumiri se poate alege orice altceva cu conditia sa se respecte extensia.

TopMenu.html arata in felul urmator:

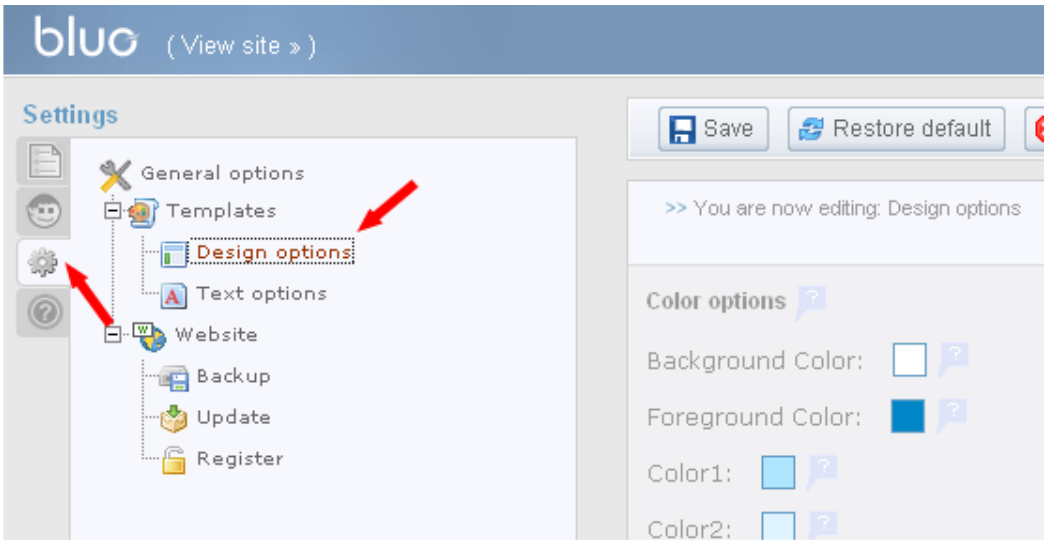
```
{if:hasPages()}
{foreach:nodes,pos,name}
    {if:isFirst(pos)}
        {if:isActive(pos)}
            <a href="{getLink(pos)}" title="{name}" class="topMenuLinkSel">
                <b>{name}</b>
            </a>
        {else:}
            {if:isSelected(pos)}
                <a href="{getLink(pos)}" title="{name}" class="topMenuLinkSel">
                    <b>{name}</b>
                </a>
            {else:}
                <a href="{getLink(pos)}" title="{name}" class="topMenuLink">
                    {name}
                </a>
            {end:}
        {end:}
    {else:}
        {if:isActive(pos)}
            |   <a href="{getLink(pos)}" title="{name}" class="topMenuLinkSel">
            <b>{name}</b>
            </a>
        {else:}
            {if:isSelected(pos)}
                |   <a href="{getLink(pos)}" title="{name}" class="topMenuLinkSel">
                        <b>{name}</b>
                    </a>
            {else:}
                |   <a href="{getLink(pos)}" title="{name}" class="topMenuLink">
                        {name}
                    </a>
            {end:}
        {end:}
    {end:}
{end:}
{end:}
```

Singurul lucru care trebuie explicat in acest cod este modul extragere a elementelor meniului din vectorul **nodes**. Astfel, cu ajutorul structurii *foreach* din limbajul flexy se face parcurgerea acestui vector. Elementele meniului se afla in vector in ordinea in care trebuie afisate.

CSS-ul

Acest capitol este dedicat in intregime modului in care trebuie sa creezi fisierul *style.css*. Probabil te intrebi de ce trebuie sa aiba o anumita structura din moment ce contine stilurile pentru template-ul tau pe care le ordonezi, creezi si folosesti dupa cum vrei.

Importanta respectarii unei anumite structuri iti permite sa ai acces la unele facilitati oferite de Bluo. Astfel, poti sa folosesti o serie de stiluri din *style.css* in editorul din partea de administrare a site-ului. Sau poti sa schimbi unele culori, stiluri pentru font, bordere din cadrul site-ului tau fara sa modifici fisierul *style.css*, ci pur si simplu prin niste click-uri in partea de administrare, mai exact in sectiunea *Settings->Design options*



Pentru ca template-ul tau sa poata fi modificabil in ceea ce priveste culorile, fontul si borderele trebuie sa respecti niste structuri de declarare a claselor care seteaza aceste elemente.

Elementele unei clase create de tine in *style.css* devin modificabile din partea de administrare numai in cazul in care clasa respectiva a fost declarata respectand urmatoarea structura:

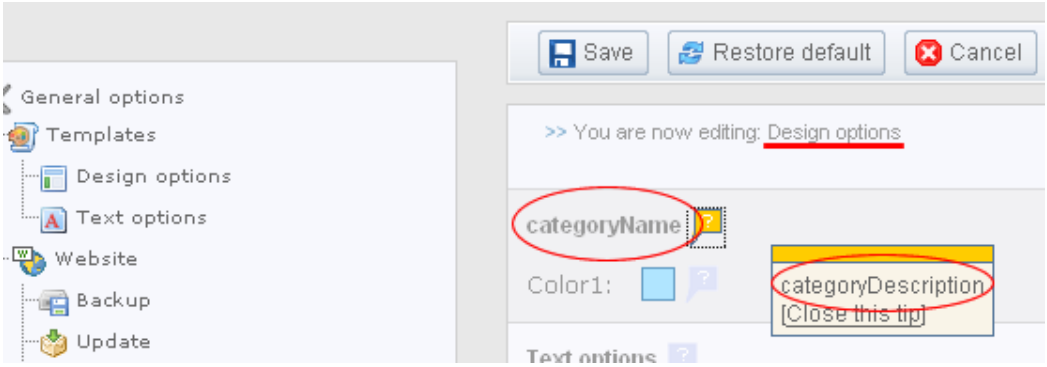
```
/* category | classTitle | classDescription */
.Change_className{
     styles
}
```

unde:

category - este numele categoriei din care face parte clasa respectiva.
O categorie se declara la inceputul fisierului style.css astfel:

```
/* categoryName: categoryDescription */
```

categoryName si *categoryDescription* sunt parametrii pe care ii alegi tu in functie de necesitate. Valorile alese pentru acestia pot fi vizualizate in partea de administrare, sectiunea *Settings->Design options*.

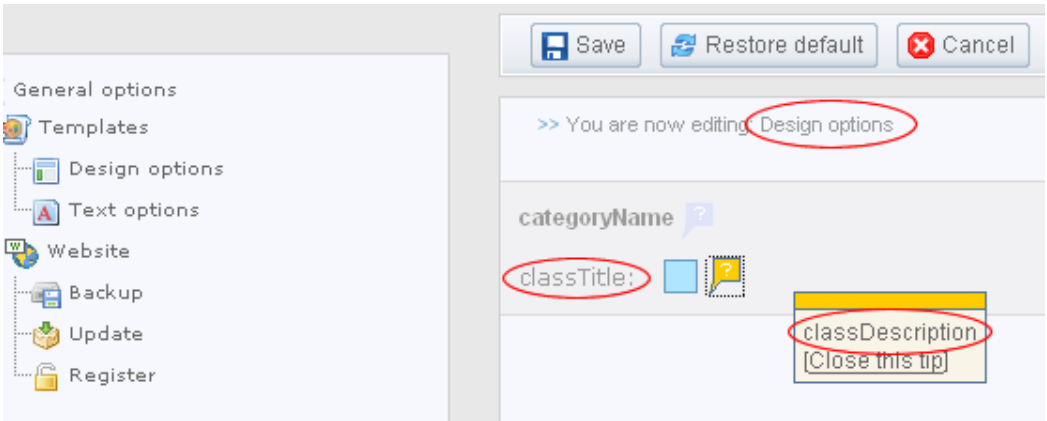


classTitle – este numele pe care il ia clasa in partea de adminstrare

classDescription – este descrierea clasei respective pentru partea de administrare

className – este un pseudo-nume pentru clasa, pseudo-nume care se concateneaza cu sirul de caractere "*Change_*" si rezulta numele propriu-zis al clasei. Atentie!!! In fisierele html se va apela aceasta clasa cu numele *Change_className.*

classTitle si *classDescription* sunt vizibile in partea de administrare astfel:

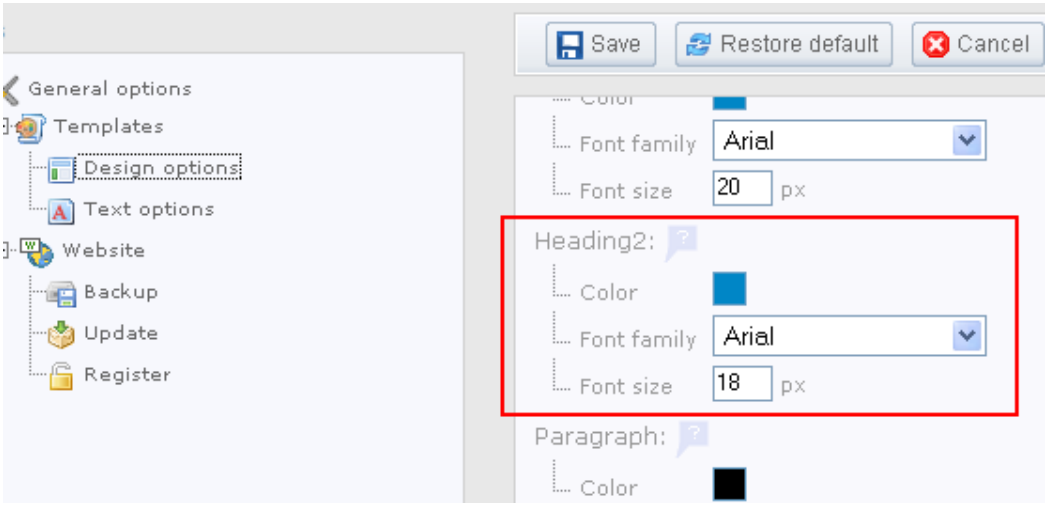


styles – reprezinta stilurile care devin modificabile. Atentie!!! Nu toate stilurile incluse in clasa sunt neaparat modificabile. Asa cum am mentionat si mai sus, in aceasta categorie se afla doar proprietatile legate de border, color, font-size, font-family si background-color.

Cu alte cuvinte, daca avem o clasa declarata astfel:

```
/* Text options | Heading2 | Sets the text style for h2 */
.Change_heading2{
      color: #0086C6;
      font-size: 18px;
      font-family: Arial, sans-serif;
      text-align: left;
      margin: 0px;
      padding-left: 23px;
      padding-top: 0px;
}
```

,dintre acestea, vor deveni modificabile doar urmatoarele: *color, font-size si font-family*, restul ramanand asa cum sunt. Acestea din urma nu vor fi incluse nici in partea de administrare, ceea ce inseamna ca in cazul clasei respective se vor putea vizualiza in *Settings->Design options* doar urmatoarele elemente:



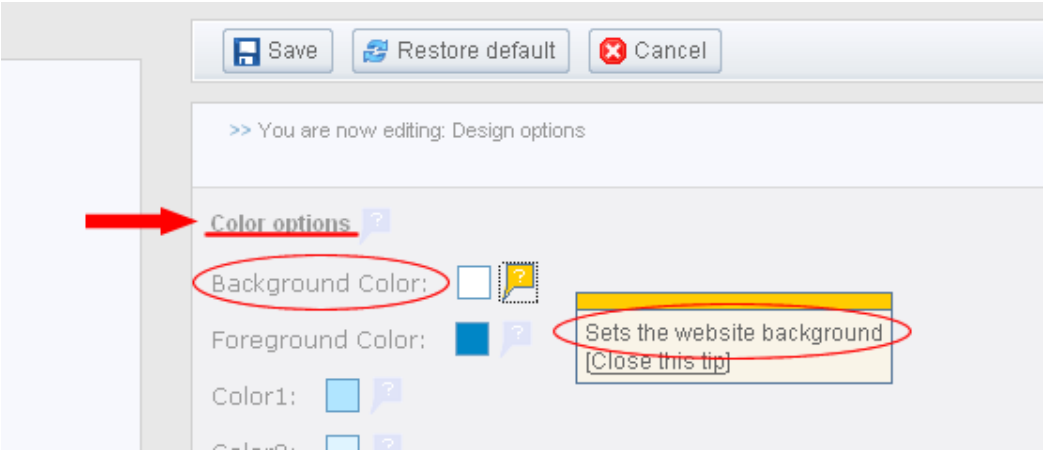
Haide sa vedem un exemplu concret de inceput de fisier *style.css*

```
/* Color options: You can use these options to modify the colors from the website */
/* Text options: You can use these options to modify the text from your website */
/* Color options | Background Color | Sets the website background */
.Change_siteBackground{
    background-color: #ffffff;
}
/* Text options | Main Font | Sets the main font of text */
.Change_mainFont{
    font-size: 12px;
    font-family: Arial, sans-serif;
}
/* Color options | Foreground Color | Sets the default color of the text */
.Change_textColor{
    color: #0086C6;
}
/* Color options | Color1 | Sets the background color for the menu items from level 1 */
.Change_color1{
    background-color: #b0e5ff;
}
/* Color options | Color2 | Sets the background color for the menu items from level 2 */
.Change_color2{
    background-color: #dff4ff;
}
```

Prima si cea de-a doua linie contin partea de declarare a categoriilor in care se vor incadra clasele care contin elemente modificabile.

Urmatorea linie, cea de-a treia, contine incadrarea clasei cu titlul *Background Color* si descrierea *Sets the website background* in categoria *Color options*.

Dupa aceasta linie urmeaza scrierea efectiva a clasei, fara insa a se omite utilizarea sintaxei corespunzatoare pentru numele clasei, adica inceperea acestuia cu sirul de caractere "*Change_*". In urma acestei declaratii, in partea de administrare se vor putea vizualiza urmatoarele elemente:



In ceea ce priveste importarea unor clase din *style.css* in editorul din partea de administrare, trebuie sa impartim clasele in:

- clase pentru elementele p, h1, h2 etc[1]
- clasele custom realizate de tine.

Integrarea claselor pentru elementele p, h1, h2 etc se face automat. Adica daca in *style.css* am clasa urmatoare:

```
p{
      font-size: 12px;
}
```

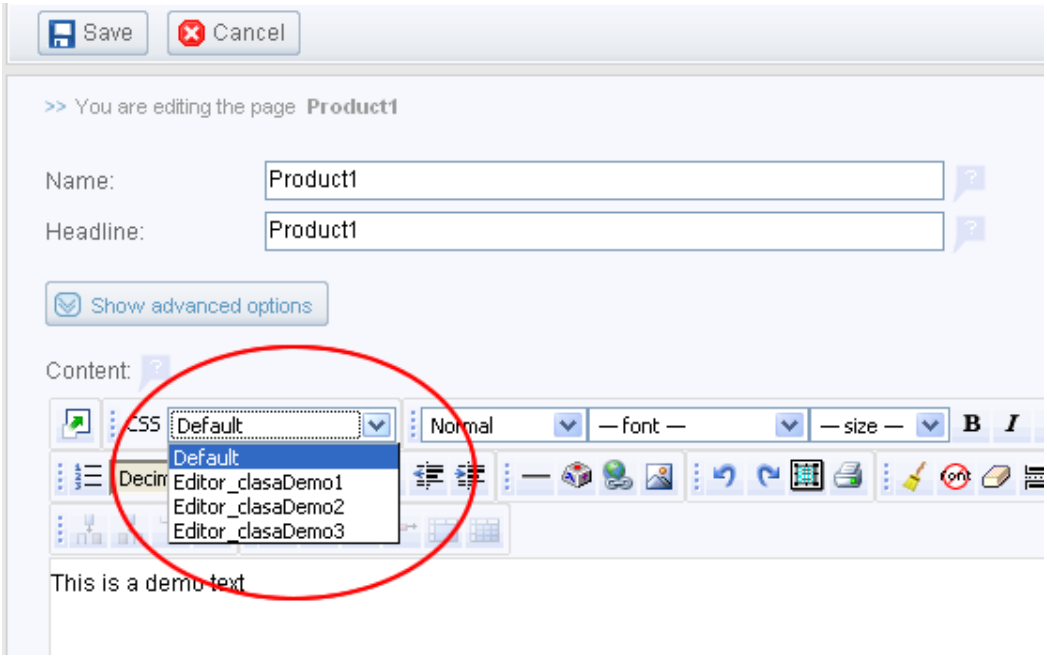
atunci orice paragraf din editor va avea aceasta clasa ca si clasa default. La fel in cazul celorlalte elemente.

Clasele custom, realizate de tine, pot fi importate in editor prin modificarea numelui lor tinand cont de urmatorul aspect:
Orice clasa care contine in denumirea sa sirul de caractere "*Editor_*" este importata in editor.
Practic, ceea ce trebuie sa faci tu este sa iei fiecare clasa de care ai nevoie in editor si sa incluzi "*Editor_*" in denumirea ei.

[1] Orice clasa care este atribuita unui container (p, div, ul, etc) si care este declarata folosind doar numele acestui container.

Atentie! Nu uita sa modifici numele claselor respective si in fisierele html unde le folosesti.

Clasele astfel importate in editor vor fi incluse intr-un dropdown care apare in meniul editorului, asa cum este indicat in imaginea urmatoare:



Clasele de mai sus au aparut datorita urmatorului cod din fisierul *style.css*:

```
.Editor_clasaDemo1{
}
.Editor_clasaDemo2{
}
.Editor_clasaDemo3{
}
```