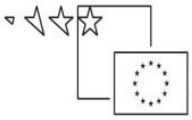
REPUBLIKA SLOVENIJA
MINISTRSTVO ZA ŠOLSTVO IN ŠPORT



Naložba v vašo prihodnost
OPERACIJO DELNO FINANCIRA EVROPSKA UNIJA
Evropski socialni sklad

IZDELAVA SPLETNIH STRANI

SIMON HORVAT
ANITA URAN

Višješolski strokovni program: Informatika
Učbenik: Izdelava spletnih strani
Gradivo za 2. letnik

Avtorja:
Simon Horvat, univ. dipl. inž.
ŠOLSKI CENTER VELENJE
Višja strokovna šola

Anita Uran, univ. dipl. inž.
ŠOLSKI CENTER VELENJE
Višja strokovna šola

Strokovni recenzent:
Islam Mušić, prof.

Lektorica:
Lidija Šuster, prof.

CIP - Kataložni zapis o publikaciji
Narodna in univerzitetna knjižnica, Ljubljana

004.774(075.8)(0.034.2)

HORVAT, Simon, 1978-
Izdelava spletnih strani [Elektronski vir] : gradivo za 2. letnik / Simon Horvat, Anita Uran. - El. knjiga. - Ljubljana : Zavod IRC, 2010. - (Višješolski strokovni program Informatika / Zavod IRC)

Način dostopa (URL): http://www.zavod-irc.si/docs/Skriti_dokumenti/Izdelava_spletnih_strani-Uran_Horvat.pdf. - Projekt Impletum

ISBN 978-961-6824-97-2
1. Uran, Anita

255546880

Izdajatelj: Konzorcij višjih strokovnih šol za izvedbo projekta IMPLETUM
Založnik: Zavod IRC, Ljubljana.
Ljubljana, 2011

Strokovni svet RS za poklicno in strokovno izobraževanje je na svoji 130. seji dne 6. 5. 2011 na podlagi 26. člena Zakona o organizaciji in financiranju vzgoje in izobraževanja (Ur. l. RS, št. 16/07-ZOFVI-UPB5, 36/08 in 58/09) sprejel sklep št. 01301-3/2011/9-2 o potrditvi tega učbenika za uporabo v višješolskem izobraževanju.

Gradivo je sofinancirano iz sredstev projekta Impletum 'Uvajanje novih izobraževalnih programov na področju višjega strokovnega izobraževanja v obdobju 2008–11'.
Projekt oz. operacijo delno financira Evropska unija iz Evropskega socialnega sklada ter Ministrstvo RS za šolstvo in šport. Operacija se izvaja v okviru Operativnega programa razvoja človeških virov za obdobje 2007–2013, razvojne prioritete 'Razvoj človeških virov in vseživljenjskega učenja' in prednostne usmeritve 'Izboljšanje kakovosti in učinkovitosti sistemov izobraževanja in usposabljanja'.

KAZALO VSEBINE

V gradivu je uporabljena slikovna podpora, ki ima naslednji pomen:


uvod poglavja


primer naloge


povzetek


osrednji del


rešitev


vprašanja za ponavljanje


vmesna vprašanja za razmišljanje


rešitev s programsko kodo

Če želite poglobiti znanje s področja izdelave spletnih strani, potem lahko dodatno posežete tudi po bogati tuji strokovni literaturi.

Avtorja

PREDGOVOR

Spoštovani,

v današnjem svetu je prisotnost interneta (medmrežja) vedno večja. Dostop do interneta ima že več kot polovica zemeljske oble in informacije, dobljene prek takšnega medija postajajo čedalje bolj iskane. Zaradi tega postajajo spletna mesta, portali oz. strani čedalje pomembnejši in predstavitev informacije na tak način je tako rekoč nuja.

Obstaja veliko takšnih ali drugačnih strani, ki pa se vendarle razlikujejo po profesionalnosti. Pojem profesionalnost je čedalje pomembnejši, saj so všečne in vsebinsko bogate spletne strani slika podjetja, ustanove ali osebe, ki se na tak način predstavlja. Pomemben je prvi stik obiskovalca, ki dobi s tem približno podjetja. Spletne strani so namenjene tako naključnim obiskovalcem kot tudi internim in zunanjim uporabnikom. Za uspešno zasnovo je zato treba natančno opredeliti ciljne skupine uporabnikov in določiti cilje, ki jih je treba doseči s postavitvijo tovrstne spletne strani na svetovni splet. Vprašanje, komu bodo strani namenjene, je osnova za grafično obliko in strukturo spletne strani.

Ne smemo pozabiti, da na spletnih mestih objavljamo najrazličnejše podatke in informacije, zato je tudi pravilno izbrana vsebina, ki zajema zahtevane tekstovne, slikovne, dinamične in tehnične elemente, zelo pomembna za uspešno predstavitev na svetovnem spletu.
Pomemben člen izdelave je načrtovanje. Vpliv informacije na obiskovalca je odvisen od načina podajanja le-te. Velikokrat že vsebina informacije narekuje način, grafično obliko in zgradbo spletne strani.

V svetu kapitala, kjer ima glavno vlogo denar, je najpomembnejše vprašanje s tega področja, kako čim hitreje, uspešneje in ceneje izdelati spletno predstavitev. Vedno večjo vlogo na tem področju pridobivajo orodja z odprto kodo, ki poleg tega, da so zastonj, vsebujejo ogromno dokumentacije. V to kategorijo spadajo Apache, PHP, PostgreSQL in Symfony. Namen učbenika je prikaz tehnologij za uspešno, cenovno ugodno, fleksibilno in uporabniško prijazno izdelavo spletnih strani.

Učbenik je razdeljen na šest sklopov. V prvem sklopu je prikazana namestitev vseh potrebnih orodij za izdelavo spletnih strani, v drugem pa osnove jezika HTML in XHTML. V tretjem poglavju so prikazane slogovne predloge, v četrtem pa programski jezik PHP in obrazci. Peto poglavje je namenjeno programskemu ogrodju Symfony, ki je namenjeno hitri izdelavi spletnih strani. V zadnjem, šestem, poglavju je opisana še varnost in najpogostejše napake programerjev, kakor tudi napotki za izdelavo spletnih strani.

Spoštovane študentke, spoštovani študenti. Želiva vam uspešen študij ter koristno uporabo pridobljenega znanja.

Avtorja

1 NAMESTITEV STREŽNIKA IN PROGRAMOV

V uvodnem poglavju učbenika bomo spoznali potrebna programska orodja in strežnike, ki jih potrebujemo za izdelavo dinamičnih spletnih strani. Ob spoznavanju le-teh se bomo srečali z brezplačnimi orodji, ki jih je na trgu kar nekaj. Ali veste, da so brezplačna orodja tudi ena izmed najbolj uporabljanih in da je spletni strežnik Apache, ki ga bomo uporabljali, najbolj uporabljan strežnik v tem trenutku? S programskim jezikom PHP pa je nerejena več kot polovica vseh spletnih strani.

Uvodna naloga:

Namestiti je potrebno strežniška in programska orodja za izdelavo dinamičnih spletnih strani. Strežnik je potrebno ustrezno nastaviti ter namestiti IDE[1] orodje. Nalogo bomo najlažje rešili, če bomo podali nekaj smernic in predloge programskih orodij, ki bi zadostovale potrebam naloge. Rešili jo bomo z razlago posameznih podpoglavij v tem poglavju.

Če želimo spletne strani deliti z drugimi, da bodo vidne na svetovnem spletu, potrebujemo strežnik. Strežnikov je več vrst, za spletne strani pa je najpomembnejši spletni strežnik v kombinaciji s strežnikom zbirke podatkov. Če želimo izdelovati in preizkušati spletne strani, potem potrebujemo bodisi lokalni ali oddaljeni strežnik do katerega dostopamo prek FTP-odjemalca. Ogledali si bomo, kako si spletni strežnik namestimo na dva najbolj razširjena operacijska sistema Windows in Linux, ter katera programska orodja potrebujemo za izdelavo spletnih strani.

1.1 ZAHTEVANA ORODJA

Za izdelavo dinamičnih spletnih strani potrebujemo spletni jezik, spletni strežnik ter strežnik in orodje za dostop do zbirke podatkov.

Apache:
Eden izmed najbolj popularnih HTTP-strežnikov je Apache. Zagotavlja varno in učinkovito platformo za spletni strežnik. Je enostaven za uporabo in konfiguracijo. Izdeluje in vzdržuje ga podjetje Apache Software Foudation, več informacij pa je dostopnih na spletni strani www.apache.org (1. 7. 2010).

PostgreSQL:
Je zmogljivo odprtokodno orodje za zagotavljanje strežnika relacijske zbirke podatkov. Več informacij je dostopnih na spletni strani www.postgresql.com (1. 7. 2010).

PHP:
PHP je skriptni jezik za izdelavo spletnih strani. Njegova integracija v XHTML je zelo enostavna in je eden najbolj popularnih Apache modulov. Ogromna zbirka knjižnic in ukazov je dostopna na spletni strani www.php.net (1. 7. 2010).

phpPgAdmin:
phpPgAdmin je orodje, ki služi za administracijo zbirk podatkov PostgreSQL. Napisan je z jezikom PHP, uporaba in dokumentacija orodja pa je opisana na spletni strani http://phppgadmin.sourceforge.net (1. 7. 2010).

[1] Integrated Development Environment

Minimalne zahteve za namestitev orodij so:

- 256 MB pomnilnika,
- 150 MB prostora na trdem disku,
- Windows, Linux ali MAC operacijski sistem.

1.2 NAMESTITEV NA OPERACIJSKI SISTEM WINDOWS XP

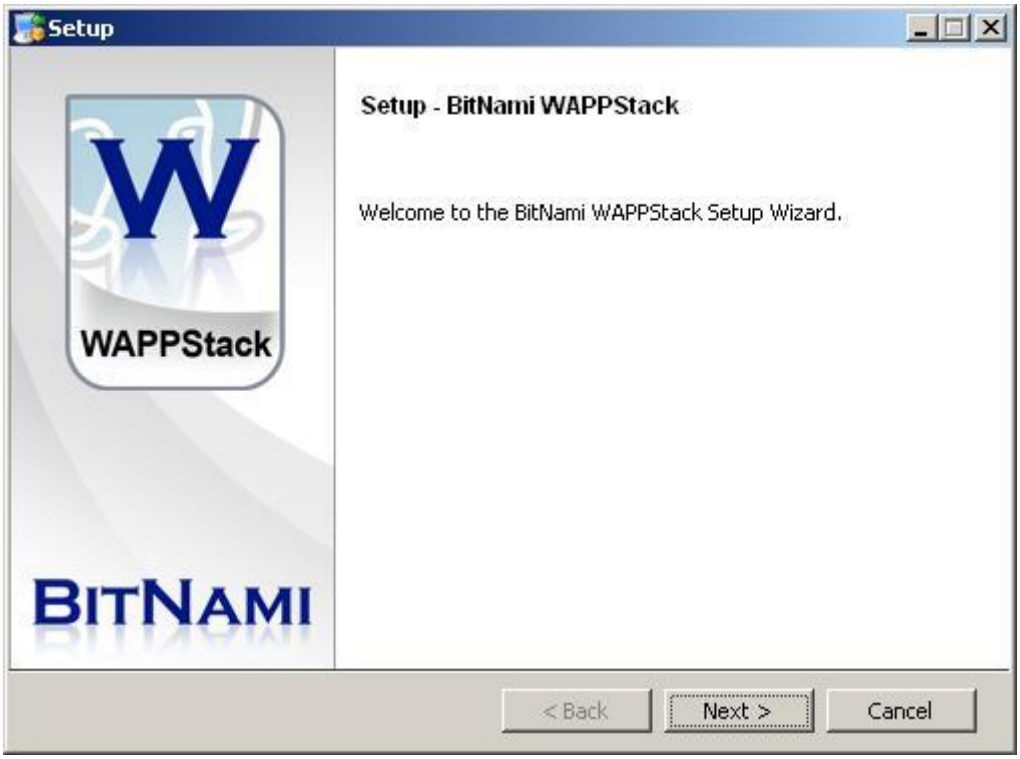
Operacijski sistem Windows XP je komercialni operacijski sistem, zato le-ta v osnovi ne vsebuje ustreznih orodij za programiranje spletnih strani. Vse programe moramo sneti z medmrežja in jih naložiti na lokalni računalnik. Namestitev, ki jo bomo prikazali, ne vsebuje podrobnejših nastavitev. Za profesionalno uporabo je potrebno strežnike dodatno zaščititi in poskrbeti za ustrezno varnost le-teh.

Za gostovanje spletnih strani potrebujemo strežnik s podporo za programski jezik, v našem primeru PHP. Danes so spletne strani mediji, ki omogočajo interakcijo med uporabnikom in spletno stranjo. V tem procesu lahko uporabnik aktivno sodeluje (nevede ali vede) v vsebini le-te, prek različnih forumov, pisanja novic, nalaganja multimedijskih vsebin in podobno. Podatki se zato shranjujejo, običajno v zbirko podatkov.

Da ne bi na naš računalnik nameščali vseh komponent posebej, obstajajo programska orodja, ki vse to že vsebujejo in se namestijo na računalnik z eno namestitvijo. Primer takega orodja je Bitnami WAPPStack (www.bitnami.org (1. 7. 2010)), ki vsebuje spletni strežnik Apache, programski jezik PHP in PostgreSQL sistem za upravljanje zbirk podatkov. Program je brezplačen.

Oglejte si spletno stran http://bitnami.org/ in ugotovite, za katere operacijske sisteme je namenjeno orodje BitNami. Kakšne pakete imamo na voljo?

Po zagonu namestitvenega programa se odpre pozdravno okno.



Slika 1: Namestitveni program BitNami WAPPStack

Namestitev ni zahtevna, priporočamo pa, da v izogib morebitnih nepravilnih nastavitev, program namestimo v mapo C:\BitNami. Potrebno je izbrati še vrata spletnega strežnika Apache, ki so privzeto 80. V enem izmed korakov namestitve moramo vpisati še geslo za PostreSQL. Le-to si je potrebno zapomniti, saj nam bo služilo za dostop do zbirk podatkov.

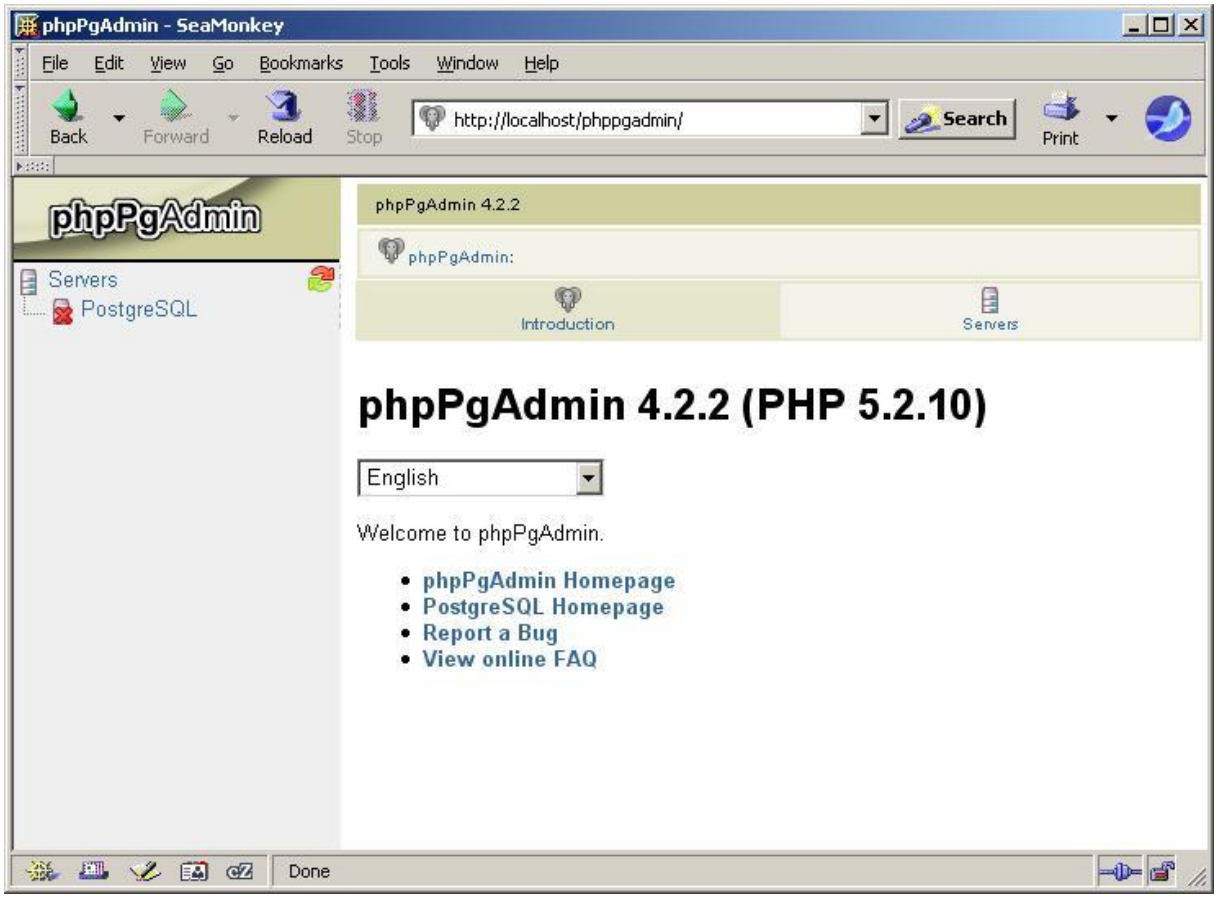
Po končani namestitvi se odpre privzeti brskalnik s prikazom osnovne strani.

Slika 2: BitNami

Do spletne strani na strežniku dostopamo prek IP-naslova 127.0.0.1 ali naslova http://localhost.

Nameščen je tudi vmesnik phpPgAdmin, s katerim dostopamo do podatkovne zbirke PostrgreSQL. Najdemo ga na naslovu http://localhost/phppgadmin (1. 7. 2010). Podatki za dostop so:

Uporabniško ime: administrator
Geslo: geslo, ki smo ga izbrali med namestitvijo



Slika 3: Vstop v phpPgAdmin

Za dostop do tabel zbirke podatkov se je potrebno prijaviti v PostgreSQL. V levem delu vmesnika kliknemo na rdeči križec zraven napisa PostgreSQL in se prijavimo s podatki:

Uporabniško ime: postgres
Geslo: geslo, ki smo ga izbrali med namestitvijo

Z uspešno prijavo se prikažejo orodja za administracijo zbirk podatkov, ki jih bomo potrebovali v nadaljevanju.

Privzeta mapa, v kateri so shranjene datoteke, je: `C:\BitNami\apache2\htdocs`.

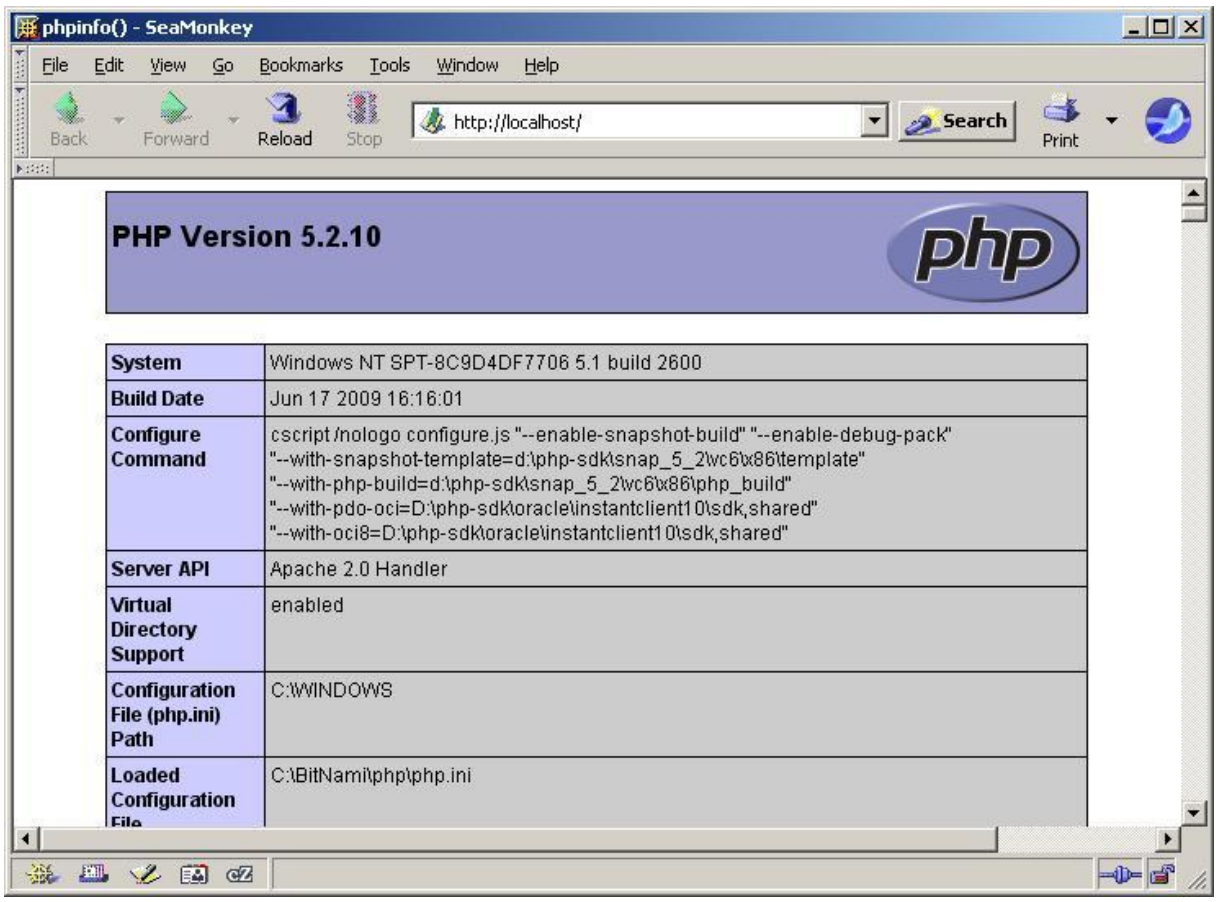
Privzeto mapo spletnih dokumentov spremenimo v datoteki `C:\BitNami\apache2\conf\httpd.conf`, pri čemer spremenimo vrstico:

```
DocumentRoot "C:/Bitnami/apache2/htdocs"
```

Pravilnost delovanja PHP-ja preverimo z naslednjo kodo:

```
<?php
phpinfo();
?>
```

Datoteko shranimo v mapo htdocs z imenom index.php. Datoteke, ki so privzeto v tej mapi, lahko izbrišemo (mapa img in datoteke index.html, bitnami.css in wappstack.png), saj jih ne potrebujemo. V brskalniku osvežimo spletni naslov http://localhost in v njem dobimo izpis informacij o nameščeni verziji PHP-ja.



Slika 4: Phpinfo()

phpinfo() je funkcija, ki prikaže vse nastavitve sistema PHP, vključno z zbirko podatkov.

1.3 NAMESTITEV NA OPERACIJSKI SISTEM UBUNTU 9.10

BitNami je na voljo za različne operacijske sisteme, tudi za Linux. Za namestitev na Ubuntu 9.10 je potrebno sneti verzijo LAPP in datoteko s končnico bin:

```
bitnami-lappstack-1.2-0-linux-installer.bin
```

Za zagon datoteke ji je potrebno nastaviti uporabniške pravice, kar storimo z ukazom:

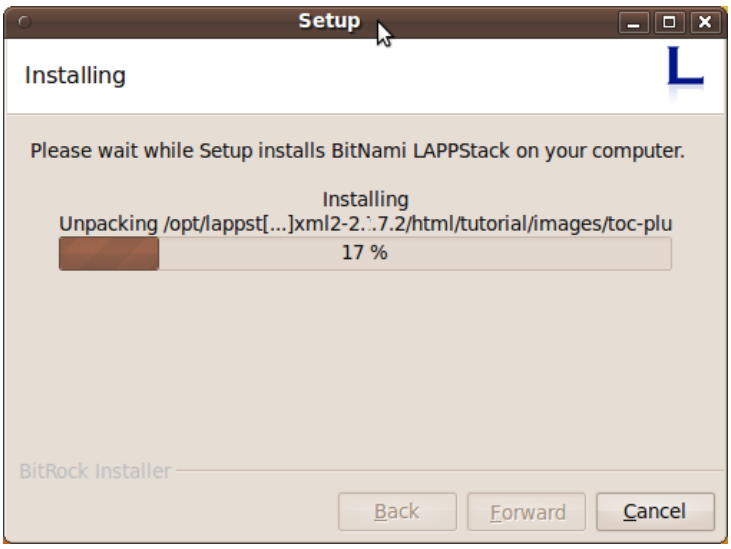
```
$ chmod 755 bitnami-lappstack-1.2-0-linux-installer.bin
```

Namestitveni proces zaženemo z ukazom:

```
$ ./bitnami-lappstack-1.2-0-linux-installer.bin
```

V nadaljevanju namestitve je potrebno izbrati namestitveno mapo. Če smo namestitveni proces zagnali kot navaden uporabnik, se bo BitNami namestil v domačo mapo, sicer pa v mapo:

```
/opt/lappstack-1.2-0
```



Slika 5: Namestitveno okno LAPP sklada

Ob koncu namestitve določimo še root geslo za PostgreSQL. Do strežnika dostopamo na enak način kot pri operacijskem sistemu Windows XP.

Privzeta vrata za Apache so 8080, za PostgreSQL pa 5432. Če je katero od vrat že v uporabi, nas bo namestitveni proces o tem pravočasno obvestil in nam ponudil alternativna vrata.

1.4 KONFIGURACIJA SPLETNEGA STREŽNIKA

Z osnovno namestitvijo spletnega strežnika je mapa domače spletne strani nastavljena v korenski imenik spletne strani, kar iz varnostnih in tudi praktičnih vidikov ni najbolj priporočljivo. V nadaljevanju bomo opisali postopek konfiguracije spletnega strežnika za Linux. Za Windows je namestitev zelo podobna.

V nadaljevanju bomo prikazali postopek, kako spletno stran premaknemo v novo mapo ter kako spremenimo URL[2] naslov, preko katerega dostopamo do spletne strani v http://blog.localhost.

Za začetek najprej ustvarimo mapo, kjer bo shranjena naša spletna stran. V svojem domačem imeniku ustvarimo mapo dev in nato v njej mapo blog.

```
$ mkdir dev
$ cd dev
$ mkdir blog
```

Nato odpremo glavno konfiguracijsko datoteko spletnega strežnika:

```
$ nano /opt/lappstack-1.2-0/apache2/conf/httpd.conf
```

Odkomentiramo vrstico:

```
Include conf/extra/httpd-vhosts.conf
```

[2] Uniform Resource Locators

Za konfiguracijo navideznega gostitelja[3] je potrebno spremeniti še datoteko:

```
$ nano /opt/lappstack-1.2-0/apache2/conf/extra/httpd-vhosts.conf
```

Podani primer spremenimo v:

```
NameVirtualHost 127.0.0.1:80

<VirtualHost 127.0.0.1:80>
    DocumentRoot "/home/simon/dev/blog"
    ServerName blog.localost
    ServerAlias blog.localhost
    ErrorLog "logs/blog.localhost-error_log"
    CustomLog "logs/blog.localhost-access_log" common
    <Directory "/home/simon/dev/blog">
      AllowOverride All
      Allow from All
    </Directory>
</VirtualHost>
```

Domensko ime blog.localhost mora biti deklarirano lokalno. Na Linux sistemu jo dodamo v `/etc/hosts`, na Windows pa v `C:\WINDOWS\system32\drivers\etc\hosts`

Dodamo naslednjo vrstico:

```
127.0.0.1       blog.localhost
```

Če želimo preizkusiti delovanje naslova `blog.localhost`, lahko ustvarimo datoteko index.php s spodaj podano vsebino v mapi `dev/blog`.

```
<?php
phpinfo();
?>
```

Za potrditev vseh sprememb ponovno zaženemo strežnik z ukazom:

```
./ctlscript.sh restart
```

Slika 6: Blog.localhost

[3] Angl. Virtual Host

1.5 NAMESTITEV IDE-ORODJA

Za programiranje spletnih strani je dovolj navaden urejevalnik besedil, kot je npr. Beležnica v operacijskem sistemu Windows ali Mousepad v operacijskem sistemu Linux.

Urejevalniki takšne vrste imajo kar nekaj pomanjkljivosti, ki jih programerji spletnih strani pri svojem delu najbolj potrebujejo. Prva pomanjkljivost je označevalnik kode, ki ustrezno barva značke, rezervirane besede in ureja tabulatorje. Pri takih urejevalnikih manjka tudi številčenje vrstic, ki v primeru napak ali iskanja v marsičem olajša delo.

1.5.1 NetBeans

NetBeans je IDE-orodje za razvoj aplikacij različnih programskih jezikov, kot npr. Java, Javascript, PHP, Python ipd. Eden izmed razlogov, zakaj smo se odločili za uporabo tega orodja, je tudi podpora za razvojno okolje Symfony, s katerim se bomo v nadaljevanju lotili izdelave spletnih strani.

Slika 7: NetBeans logo

Namestitev je sila preprosta tako v operacijskem sistemu Linux kot Windows. Obiščemo spletno stran www.netbeans.org (1. 7. 2010), izberemo področje Download in snamemo datoteko, ki vsebuje podporo za PHP. Sneli in namestili bi lahko celotni paket, vendar je za naše potrebe popolnoma dovolj podpora samo za PHP. Velikost datoteke je približno 25 MB, namestimo pa jo z običajnim namestitvenim postopkom kot pri večini Windows programih.

Nekoliko drugačna je namestitev za Linux. V tem primeru snamemo datoteko s končnico sh (npr. netbeans-6.8-ml-php-linux.sh) in program namestimo z ukazom:

```
./netbeans-6.8-ml-php-linux.sh
```

Namestitveni čarovnik nas vodi skozi proces nameščanja.

V operacijskem sistemu Linux lahko Netbeans namestimo tudi s programskih skladišč - repozitorijev z ukazom:

```
apt-get install netbeans
```

Nekateri repozitoriji ne vsebujejo različice večje ali enake 6.8, ki ima podporo za Symfony, zato je priporočljivo, da si ta paket namestimo ročno.

Povzetek:

V poglavju smo spoznali tri tehnologije, ki jih potrebujemo za delo s spletnimi stranmi:
- spletni strežnik Apache,
- programski jezik PHP,
- sistem za upravljanje zbirk podatkov PostgreSQL.

Spoznali smo postopek namestitve teh treh orodij, ki jih dobimo s paketom BitNami. BitNami vsebuje še orodje za dostop do zbirke podatkov, PhpPgAdmin. Programski paket je brezplačen in deluje na različnih operacijskih sistemih.

V poglavju je bila zajeta tudi enostavna konfiguracija spletnega strežnika z nastavitvijo navideznih gostiteljev. Nastavili smo http://blog.localhost. Za izdelavo spletnih strani potrebujemo razvojno okolje oz. ustrezen urejevalnik besedil. Navadna beležnica pri tem ni dovolj, predvsem zaradi svojih pomanjkljivosti, saj nima označevalnika kode in številčenja vrstic. Namestili smo Netbeans, ki je prav tako odprtokodno, večplatformsko orodje.

Naloge:

1. Vstopite na spletno strani www.bitnami.org in si oglejte posamezne produkte. Kakšna je razlika med njimi? Kakšna je razlika med LAMP in LAPP paketom?

2. Na spletu poiščite, kaj pomeni izraz localhost? Kateri IP lahko identificiramo s tem imenom?

3. Poiščite nekaj sorodnih komercialnih in nekomercionalnih IDE-orodij za programiranje v jeziku PHP? Poskušajte ugotoviti razlike med njimi.

4. Kakšne so prednosti IDE-orodij za izdelavo spletnih strani v primerjavi z urejevalnikom besedilnih dokumentov?

5. Kaj omogoča navidezni gostitelj?

6. Kaj predstavlja kratica IDE?
   a) Integral Develop Equation
   b) Integrated Development Environment
   c) Beležnica
   d) International Development Environment

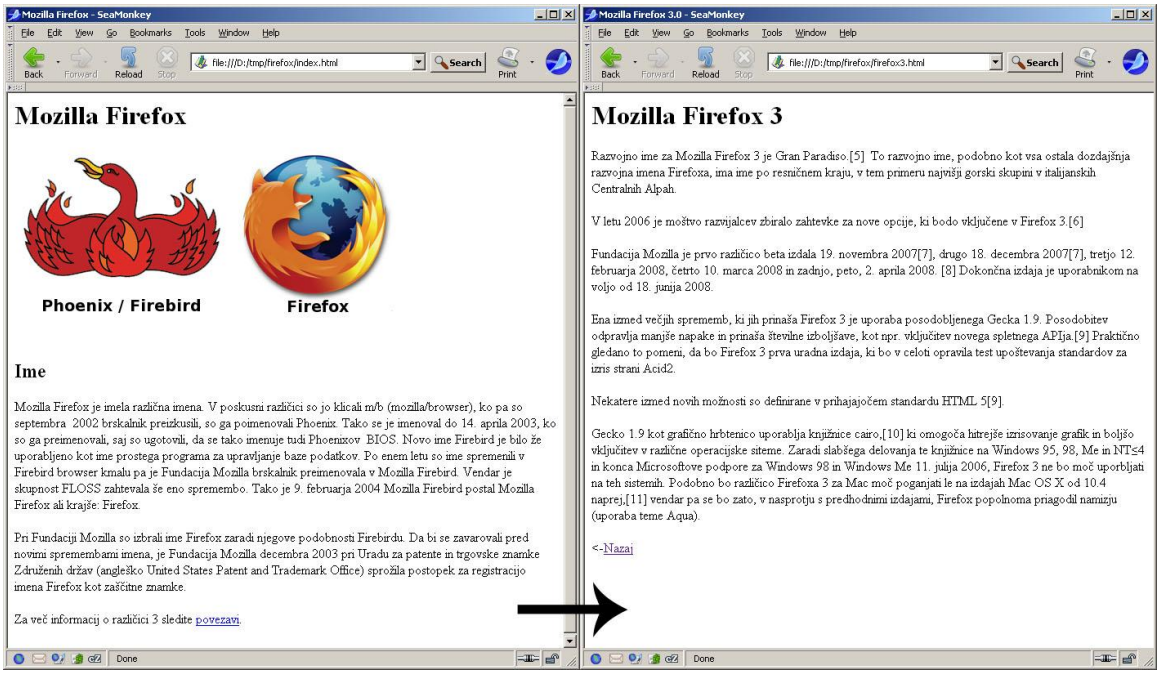
7. Na računalniku imamo vklopljen požarni zid, ki nam onemogoča delovanje spletnega strežnika in strežnika zbirke podatkov. Za spletni strežnik moramo odpreti vrata _____, za strežnik zbirke podatkov pa vrata _____.

2 JEZIKA ZA OZNAČEVANJE HTML IN XHMTL

Ali veste, da je bila prva spletna stran narejena s programskim jezikom HTML?Le ta je danes osnova vseh spletnih strani. Je standardiziran, njegova vsebina in zmožnosti pa se z vsako novo verzijo nadgrajujejo. Kako s HTML-jem izdelamo spletno stran in kako jo brskalnik prepozna? HTML je označevalni jezik za izdelavo spletnih strani in predstavlja osnovo spletnega dokumenta. Z njim lahko spreminjamo tip pisave, velikost in stil, vključujemo lahko slike, povezave ipd. Prelomno leto v izdelavi spletnih strani je bilo 1989, ko je Tim Barners - Lee, raziskovalec v evropskem centru za fiziko delcev CERN v Ženevi, razvil protokol HTTP in jezik HTML. V praksi sta zaživela dve leti pozneje. Poleg tega, da je Barners - Lee izumil protokol http, je postavil tudi prvi spletni strežnik. Leta 1990 je vzpostavil prvo HTTP povezavo med odjemalcem in strežnikom preko interneta, zaradi česar je prepoznan kot izumitelj svetovnega spleta.

Naloga:

S pomočjo navadnega urejevalnika besedil oz. IDE okolja je potrebno izdelati XHTML spletno stran. Spletna stran naj vsebuje eno podstran. Ustrezno uporabite naslove, vključite slike in povezave. Spletne strani je potrebno preveriti z W3-validatorjem in mora ustrezati standardu.



Slika 8: Naloga – Mozilla Firefox

Začetno stran imenujemo indeksna stran in se imenuje index.html. Le-to je določeno v nastavitvah spletnega strežnika.

Osnova vseh spletnih strani je HTML. Tudi dinamične spletne strani, ki omogočajo interakcijo med uporabnikom in spletno stranjo, so kombinacija programskega jezika in jezika XHTML, ki je razširjeni HTML jezik. Da bi razumeli delovanje spletnih strani, moramo najprej spoznati omenjeni jezik.

2.1 XHTML

XHTML je sestavljen iz značk[4], ki se vedno nahajajo med znakoma < in >, kot npr. `<tag>`. Značkam rečemo tudi oznake. Spletni pregledovalniki (browsers) so programi, ki berejo ukaze v jeziku XHTML in glede na to prikazujejo vsebino. Vsako ročno oblikovanje strani (npr. večkratni presledki, prazne vrstice ...) se ignorira. Če npr. vtipkamo:

```
Danes
       je

                lepo
                              vreme.
```

bo brskalnik prikazal naslednjo vsebino:

Danes je lepo vreme.

Datoteke v XHTML imajo končnico .htm ali .html, pišemo pa jih lahko v besedilnem urejevalniku, kot je beležnica (notepad), lahko pa uporabimo katerega od zmogljivejših orodij, ki npr. barvajo izvorno kodo. Primer takega urejevalnika je npr. PSPad (http://www.pspad.com/ (1. 7. 2010)). Uporabimo lahko tudi že nameščeni NetBeans.

2.1.1 Značke v XHTML

Za opis dokumenta v jeziku XHTML zadošča, da običajno besedilo opremimo z nekaj posebnimi značkami. Značke ločimo od običajnega besedila po tem, da so zapisane med posebnima simboloma, znakoma manjše (<) in večje (>).

Pri XHTML ločimo dve osnovni vrsti značk:

1. Samostojne značke, imenovane tudi prazne (blank tags).

   Zgled tovrstne značke je oznaka za prehod v novo vrsto, `<br />`.

2. Značke, ki nastopajo v parih, imenovane tudi vsebniki (containers). Pri teh oznakah velja pravilo, da se končna oznaka od začetne vedno razlikuje po dodatni poševni črti (angl. forward slash). Zgled para značk, ki omeji krepko izpisano besedilo, je:

   ```
   Stavek s <b> krepko </b> izpisano besedo.
   ```

Večina značk pozna dodatna pojasnila ali lastnosti. Zapišemo jih znotraj lomljenega oklepaja v obliki: lastnost = vrednost_lastnosti. Pri tem številčne vrednosti zapišemo neposredno, druge pa med dvojnima narekovajema (dopustna sta tudi enojna narekovaja).

Zgled samostojne oznake s parametri:

```
<img src="moja_slika.gif" width="100" height="100" />
```

Če ima ukaz več parametrov, njihov vrstni red ni pomemben.

[4] Ang. tags

2.1.2 Splošna oblika

Predpisano splošno obliko dokumenta XHTML določajo v predpisanem zaporedju zapisane naslednje oznake:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
        <title></title>
</head>
<body>
        (vsebina)
</body>
</html>
```

Vsak pravilno zapisan dokument je omejen s parom značk `<html>` in `</html>`. Bolj pregledno je, če ju zapišemo v prvi in zadnji vrstici dokumenta, čeprav to ni nujno. Navsezadnje je lahko ves dokument zapisan v eni sami vrstici. Del, ki je zapisan med paroma značk `<head>` in `</head>`, je pravzaprav glava, ki se posreduje v odgovor strežnika (angl. server response). Znotraj glave lahko zapišemo več metainformacij, ki se posredujejo v glavi odgovora.

Vsak dokument mora biti poimenovan. Njegovo ime zapišemo med par značk `<title>` in `</title>`. Sama vsebina dokumenta je zaprta med oznaki `<body>` in `</body>`. Ta vsebina se prikaže znotraj pregledovalnika.

Pri protokolu HTTP podaja obliko zapisa posebna oznaka v glavi: Content-Type. Praktično vsi brskalniki prikazujejo besedilo v kateri od standardnih kodnih strani, ki ne vsebuje slovenskih znakov. S preprosto značko, ki jo dodamo glavi, lahko vsilimo interpretacijo po standardu UTF-8, ki pozna šumnike. Ustrezna oznaka se potem glasi:

```
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
```

Parameter `http-equiv` označuje element metainformacije v glavi odgovora, parameter `CONTENT` pa njegovo vrednost. Za lažje delo pri ustvarjanju XHTML dokumentov si lahko pripravimo šablono, ki jo shranimo v datoteko. To datoteko vedno uporabimo za izhodišče nadaljnjega kodiranja. Tovrstna datoteka bi lahko imela naslednjo vsebino:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  <title></title>
</head>
<body>

</body>
</html>
```

Še bolj splošna rešitev za pisanje šumnikov pa je uporaba kode znakov v obliki:

`&#n;`

kjer je n ustrezna številka iz prvega stolpca tabele 1. Mali č zapišemo kot: `&#269;`.

Tabela 1: Tabela šumnikov

Številka	**Znak**
268	Č
269	č
352	Š
353	š
381	Ž
382	ž

2.1.3 Naslovi

Na voljo imamo šest nivojev naslovov (headings) h1 do h6. Primer za prvi nivo je:

```
<h1>Naslov</h1>
```

Naslov poravnamo na sredino z dodatkom:

```
<h1 align="center">Naslov</h1>
```

2.1.4 Odstavki in nove vrstice

Običajne odstavke (angl. paragraphs), katerih širina se prilagaja trenutni velikosti okna, dosežemo s:

```
<p>
```

Znački `<P>` lahko z lastnostjo `align` določimo način poravnavanja - left, right, center. Primer:

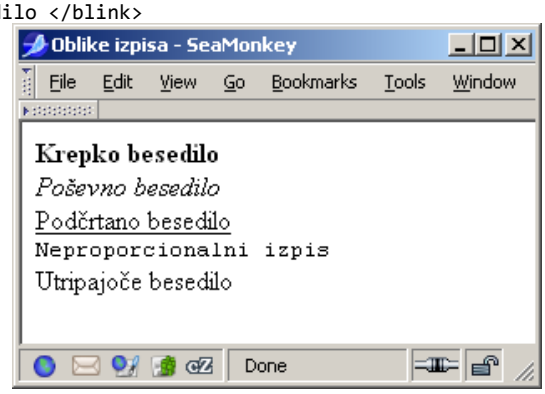
```
<p align='right'>
Ta del besedila bo <br/>
desno poravnan.
</p>
```

Prehod v novo vrstico smo določili z značko `<br/>`.

2.1.5 Oblike izpisa

Na voljo je več različnih oblik izpisa besedila.

- Krepko besedilo (angl. bold):
  `<b> Krepko besedilo </b>`
- Poševni izpis besedila (angl. italic):
  `<i> Poševno besedilo </i>`
- Podčrtan izpis (angl. underline):
  `<u> Podčrtano besedilo </u>`
- Neproporcionalni izpis (typewriter):
  `<tt> Neproporcionalni izpis </tt>`
- Utripajoč izpis (blink):
  `<blink> Utripajoče besedilo </blink>`



Slika 9: Različne oblike izpisa

Oblike lahko kombiniramo z gnezdenjem, npr. krepko in poševno hkrati.

2.1.6 Seznami -oblikovanje in prikaz

Označevalni seznami (angl. unordered lists):

```
<ul>
        <li> Prva točka</li>
        <li> Druga točka</li>
        <li> Tretja točka</li>
</ul>
```

Rezultat:

- Prva točka
- Druga točka
- Tretja točka

Če želimo uporabljati oštevilčene sezname (angl. Ordered Lists) namesto ul, uporabimo značko ol. Rezultat:

1. Prva točka
2. Druga točka
3. Tretja točka

Sezname lahko gnezdimo, npr.

```
<ol>
<li> Prva točka
   <ul>
         <li> Prva pika prve točke</li>
         <li> Druga pika prve točke</li>
         <li> Tretja pika prve točke</li>
   </ul>
<li> Druga točka
<li> Tretja točka
</ol>
```

Rezultat:

1. Prva točka
   - Prva pika prve točke
   - Druga pika prve točke
   - Tretja pika prve točke
2. Druga točka
3. Tretja točka

Razmislite in poskusite ugotoviti, kako bi orisno oštevilčevali s številkami, npr.

1 Pomlad

1.1 Poletje

1.2 Jesen

2 Zima

2.1.7 Ročno oblikovano besedilo

Če želimo, da je besedilo prikazano tako, kot smo ga natipkali (prelomi vrstic, večkratni presledki ...), moramo uporabiti značko `<pre>` (angl. preformated text):

```
<pre>
   Danes
          je
                lepo
                       vreme
</pre>
```

Tak izpis je npr. uporaben pri izpisih programov. Zaradi poravnavanja po stolpcih se v tem primeru uporabi neproporcionalni font.

Poskušajte s pomočjo značke <pre> izpisati okvir z vašim imenom in priimkom. Npr.

```
+-------------+
|    Janez    |
|             |
| NOVAK       |
+-------------+
```

2.1.8 Vključevanje slik in povezav

V dokument vključimo slike z:

```
<img src="Naslov in ime datoteke">
```

Pravila za opis, kje se slika nahaja, so enaka kot pri sidrih (relativni ali absolutni naslovi). Vsi prikazovalniki prepoznajo in prikazujejo vsaj formata GIF in JPG.
Npr.:

```
<img src="logo.jpg">
```

Sliki lahko določimo širino in višino z:

```
<img src="Naslov in ime datoteke" width="W" height="H" />
```

W in H pomenita širino in višino slike v pikah (pixels). Lahko pa navedemo tudi relativno velikost v odstotkih glede na velikost okna, tako da številki dodamo znak %.
Sliko lahko uporabimo tudi za povezavo. Primer uporabe:

```
Pritisni na logo:
<a href="http://vss.scv.si"><img src="logo.jpg" width="40" /></a>
```

Za ustvarjanje povezav se uporablja značka <a>. Njega splošna sintaksa je

```
<a href="url">Besedilo povezave</a>
```

V parameter href zapišemo url[5] naslov povezave, ki jo želimo odpreti. Le-ta je lahko spletna povezava ali povezava do lokalnega dokumenta.

Oglejte si spletno stran http://www.w3schools.com (10.7.2010) in preglejte možnosti parametrov pri uporabi značke `<a>`. Ugotovite kako bi ppvezavo odprli v novem oknu brskalnika.

[5] Ang. Uniform resource locator – internetni naslov na katerem se nahaja vsebina

2.1.9 Tabele – oblikovanje in prikaz

Tabele so zelo močno orodje v xhtml. Kot smo omenili, jezik sam ne omogoča natančnega oblikovanja strani, zato ponavadi za ta namen uporabimo tabele (in nekaj zvijač). V besedilo vključimo tabelo z uporabo značke `<table>`.

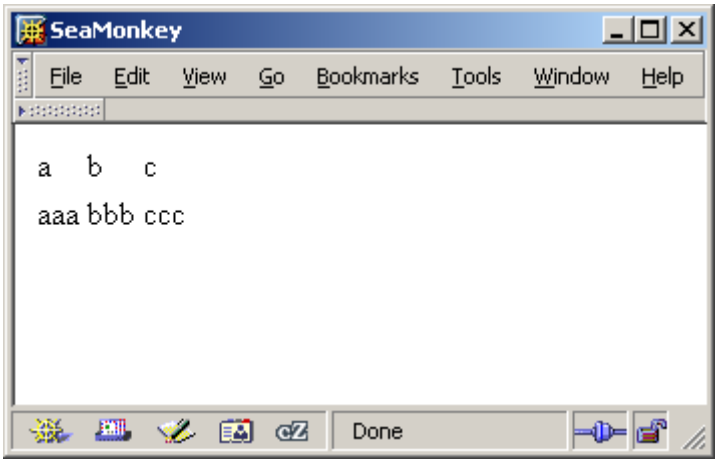
Značke, ki se uporabljajo v tabelah:

- Znački `<table>` in `</table>` označujeta začetek in konec tabele.
- Znački `<th>` in `</th>` se uporabljata za naslovne vrstice tabele – vsebina je prikazana s krepko pisavo.
- Znački `<tr>` in `</tr>` naznanjata novo vrstico. V večini primerov lahko končno značko `</TR>` izpustimo.
- Znački `<td>` in `</td>` naznanjata novo podatkovno celico v trenutni vrstici.

Pomembno: Tabela je zgrajena iz ene ali več vrstic, vsaka vrstica pa iz ene ali več podatkovnih celic in na ta način moramo tabelo tudi opisati.

Primer opisa tabele:

```
<table>
<tr> <td> a   </td> <td> b   </td> <td> c   </td>
<tr> <td> aaa </td> <td> bbb </td> <td> ccc </td>
</table>
```



Slika 10: Tabela brez obrob

Značka `<table>` ima veliko atributov oz. lastnosti. Omenimo le najpomembnejše:

- `border` se nanaša na debelino robov tabele v pikah (debelina 0 pomeni, da tabela nima roba).
- `cellspacing` se nanaša na prostor med celicami (v pikah).
- `cellpadding` se nanaša na prostor znotraj celic v pikah (prostor med mejami celic in vsebino znotraj celic).

Primer:

```
<table border=2 cellpadding=6>
<tr> <td> a   </td> <td> b   </td> <td> c   </td>
<tr> <td> aaa </td> <td> bbb </td> <td> ccc </td>
</table>
```

2.1.10 Verzije XHTML

Originalen XHTML W3C-standard, XHTML 1.0, je bil preprost prehod iz HTML 4.01 v XML. XHTML 1.0 se pojavlja v treh "različicah", pri katerih se vsaka zgleduje po njeni vzporednici v HTML 4.01.

- **XHTML 1.0 Strict** je enak HTML 4.01 Strict, vendar sledi sintaksnim pravilom XML.
- **XHTML 1.0 Transitional** dovoljuje uporabo nekaterih pogosto uporabljenih značk (tag) in elementov (attribute), ki se v XHTML 1.0 Strict ne pojavljajo, npr. , <u>, <strike> in <applet>. Podpira tudi vse drugo, kar podpira XHTML 1.0 Strict, vendar vključuje elemente in atribute za slog in je priporočen za brskalnike, ki so nastali v času okoli leta 1990, saj imajo ti brskalniki težave s prikazovanjem CSS-predlog.
- **XHTML 1.0 Frameset**: Dovoljuje uporabo HTML-okvirjev (frames).

2.1.11 Glavne razlike med XHTML in HTML

1. Pri XHTML morajo biti vse značke napisane z malimi črkami. Tako namesto:

```
<IMG SRC="logo.jpg" WIDTH="389" HEIGHT="227" BORDER="0" ALT="logotip podjetja">
```

napišemo:

```
<img src="logo.jpg" width="389" height="227" border="0" alt="logotip podjetja" />
```

2. Vse značke morajo biti zaprte. Tako kot npr. v primeru pod točko 1, kjer je značka img na koncu zaprta s poševnico. Kot primer lahko navedemo tudi značko za novo vrstico, ki se mora glasiti `<br />`.

3. Vse značke morajo biti vstavljene in zaključene v pravilnem vrstnem redu. Če npr. začnemo z značko `<b>` in nato `<u>`, mora na koncu slediti `</u>` in šele nato `</b>`.

4. Tako kot značke morajo biti vsi atributi zapisani z malimi črkami.

5. Vse vrednosti atributov morajo biti zapisane v enojnih ali dvojnih narekovajih. Primer:

```
<table width="100%">
```

6. Vsi atributi morajo biti zapisani v dolgi in ne okrajšani obliki.

Nepravilno

```
<input checked>
<input readonly>
<input disabled>
<option selected>
<frame noresize>
```

Pravilno

```
<input checked="checked" />
<input readonly="readonly" />
<input disabled="disabled" />
<option selected="selected" />
<frame noresize="noresize" />
```

7. Zaradi kompatibilnosti z modernejšimi brskalniki dodamo pred zaključno poševnico "/" presledek.

2.1.12 Deklaracija tipa dokumenta DOCTYPE

Če pogledamo izvorno kodo XHTML-dokumenta in HTML-dokumenta lahko takoj na začetku opazimo bistveno razliko v tem, da XHTML-dokument na vrhu vsebuje deklaracijo DTD oz. DOCTYPE[6]. Uporabljajo se tri možnosti, in sicer:

Strict:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "DTD/xhtml1-strict.dtd">
```

Uporabljamo jo kadar v izvorni kodi ni nobene značke, ki so bile zamenjane z drugimi. Primer take značke je recimo `<center>`, ki je zamenjana z značko `<DIV>` ali pa namesto nje uporabimo boljšo rešitev s slogovnimi predlogami CSS. Takih značk se poskušamo čimbolj izogibati in uporabljati modernejše pristope pri oblikovanju.

Transitional:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

Uporabljamo jo, kadar v izvorni kodi še vedno uporabljamo zastarele in zamenjane značke.

Frameset:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN" "DTD/xhtml1-frameset.dtd">
```

Uporabljamo, kadar dokument vsebuje okvirje.

2.1.13 Struktura XHTML dokumenta

Vsi XHTML dokumenti morajo vsebovati vsaj naslednje:

```
<!DOCTYPE Tu vstavimo doctype>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
        <title>Naslov</title>
</head>

        <body>
        </body>

</html>
```

Pravilnost XHTML-dokumenta lahko preverimo z W3-validatorjem na naslovu http://validator.w3.org/ (10. 7. 2010).

Rešitev naloge:

Zastavljena naloga zahteva, da izdelamo dva XHTML-dokumenta, ki vsebujeta naslove, slike in povezave ter da sta v skladu z XHTM-standardom.

Spoznali smo, da se XHTML-dokument prične z značko <!DOCTYPE>. HTML-dokument je zgrajen iz glave (<head>) in telesa (<body>). V glavi določimo informacije o naslovu spletne strani in metapodatke. Med metapodatke zapisujemo ključne besede, po katerih iščejo spletni iskalniki, kodiranje znakov ipd. Telo pa vsebuje jedro spletne strani s slikami, naslovi, besedilom …

[6] Ang. Document Type Declaration

Glavno spletno datoteko poimenujmo `index.html`. V njej uporabimo povezavo na datoteko firefox3.html.

index.html:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta content="text/html; charset=ISO-8859-2" http-equiv="Content-Type">
<title>Mozilla Firefox</title>
</head>
<body>
<h1>Mozilla Firefox</h1>
<img style="width: 463px; height: 220px;" alt="Firebird in Firefox"
src="Phoenix_-_Firefox.png"><br>
<br>
<h2>Ime</h2>
Mozilla Firefox je imela različna imena. V poskusni različici so jo klicali m/b (mozilla/browser), ko
pa so septembra  2002 brskalnik preizkusili, so ga poimenovali Phoenix. Tako se je imenoval do 14.
aprila 2003, ko so ga preimenovali, saj so ugotovili, da se tako imenuje tudi Phoenixov  BIOS.
Novo ime Firebird je bilo že uporabljeno kot ime prostega programa za upravljanje baze podatkov. Po
enem letu so ime spremenili v Firebird browser, kmalu pa je Fundacija Mozilla brskalnik preimenovala v
Mozilla Firebird. Vendar je skupnost FLOSS zahtevala še eno spremembo. Tako je 9. februarja 2004
Mozilla Firebird postal Mozilla Firefox ali krajše: Firefox.<br><br>
Pri Fundaciji Mozilla so izbrali ime Firefox zaradi njegove podobnosti Firebirdu. Da bi se zavarovali
pred novimi spremembami imena, je Fundacija Mozilla decembra 2003 pri Uradu za patente in trgovske
znamke Združenih držav (angleško United States Patent and Trademark Office) sprožila postopek za
registracijo imena Firefox kot zaščitne znamke.<br> <br>
Za več informacij o različici 3 sledite <a href="firefox3.html">povezavi</a>.<br>
<br>
</body>
</html>
```

firefox3.html

```
<html>
<head>
<meta content="text/html; charset=ISO-8859-2" http-equiv="Content-Type">
<title>Mozilla Firefox 3.0</title>
</head>
<body>
<h1>Mozilla Firefox 3</h1>
Razvojno ime za Mozilla Firefox 3 je Gran Paradiso.[5]  To razvojno ime, podobno kot vsa ostala
dozdajšnja razvojna imena Firefoxa, ima ime po resničnem kraju, v tem primeru najvišji gorski skupini v
italijanskih Centralnih Alpah.<br><br>
V letu 2006 je moštvo razvijalcev zbiralo zahtevke za nove opcije, ki bodo vključene v Firefox
3.[6]<br><br>
Fundacija Mozilla je prvo različico beta izdala 19. novembra 2007 [7], drugo 18. decembra 2007 [7],
tretjo 12. februarja 2008, četrto 10. marca 2008 in zadnjo, peto, 2. aprila 2008. [8] Dokončna izdaja
je uporabnikom na voljo od 18. junija 2008.<br><br>
Ena izmed večjih sprememb, ki jih prinaša Firefox 3, je uporaba posodobljenega Gecka 1.9. Posodobitev
odpravlja manjše napake in prinaša številne izboljšave, kot npr. vključitev novega spletnega API-ja.[9]
Praktično gledano to pomeni, da bo Firefox 3 prva uradna
izdaja, ki bo v celoti opravila test upoštevanja standardov za izris strani Acid2.<br> <br>
Nekatere izmed novih možnosti so definirane v prihajajočem standardu HTML 5 [9].<br><br>Gecko 1.9 kot
grafično hrbtenico uporablja knjižnice cairo[10], ki omogoča hitrejše izrisovanje grafik in boljšo
vključitev v različne operacijske sisteme. Zaradi slabšega delovanja te knjižnice na Windows 95, 98, Me
in NT≤4 in konca Microsoftove podpore za Windows 98 in Windows Me 11. julija 2006, Firefox 3 ne bo moč
uporbljati na teh sistemih. Podobno bo različico Firefoxa 3 za Mac moč poganjati le na izdajah Mac OS X
od 10.4 naprej,[11] vendar pa se bo zato, v nasprotju s predhodnimi izdajami, Firefox popolnoma
priagodil namizju (uporaba teme Aqua).<br>
<br>
&lt;-<a href="index.html">Nazaj</a><br>
<br>
</body>
</html>
```

Povzetek:

V poglavju smo spoznali osnovne značilnosti jezika XHTML. Spletno besedilo se oblikuje nekoliko drugače kot ostala besedila, saj se uporabljajo značke. Značk je veliko, tiste najosnovnejše pa so `<b>`, `<i>`, `<u>`, `<img>`, `<p>`, `<ul>`, `<ol>` in `<pre>`.

XHTML-dokument je sestavljen iz glave in telesa. V glavo zapišemo vse pomembne podatke o dokumentu, kot so metapodatki, naslov ipd. V telesu pa se nahaja vsebina spletne strani. Za pravilno uporabo in opis značk lahko uporabimo spletno stran w3schools.com http://www.w3schools.com/html/ (10. 7. 2010).
XHTML je razširjen HTML-jezik z nekaj posebnostmi. Vse značke morajo biti napisane z malimi črkami in vse vrednosti morajo biti zapisane v enojnih ali dvojnih narekovajih. XHTML je, tako kot HTML, podrobneje opisan na spletni strani http://www.w3schools.com/xhtml/ (10. 7. 2010).
Pred objavo moramo spletno stran validirati, s čimer preverimo morebitne napake. Najbolj pogoste napake so nepravilna uporaba značk, manjkajoči elementi ipd. Spletno stran validiramo na naslovu http://validator.w3.org/ (10. 7. 2010).

Naloge:

1. S pomočjo interneta ugotovite, kaj je alternativno besedilo pri slikah in kako ga pravilno uporabimo?

2. Razmislite, kako bi izdelali spletno stran, ki bi imela na vrhu sliko, ki bi bil logotip in na levi strani meni. Meni in logotip bi bila vidna na vsaki podstrani, spreminjala pa bi se samo vsebina desno od menija oz. pod logotipom.

3. Razmislite in poskušajte na spletno stran vstaviti video s spletne strani YouTube. Kaj je EMBED-koda in čemu služi?

4. Izdelajte dokument z imenom index.html in v njega vstavite poljubno besedilo ter sliko (oboje poiščite na spletu). Nato naredite sledeče:
   - V naslov spletne strani (ne URL) vstavite svoje ime in priimek.
   - Na koncu spletne strani dodajte povezavo na vaš elektronski naslov, pri čemer se mora s klikom na ta naslov odpreti odjemalec za elektronsko pošto (uporabite spletno stran http://www.w3schools.com/HTML/html_links.asp).
   - Nekje na sredini vstavite vodoravno črto.
   - Na koncu spletne strani dodajte povezavo, ki je povezava na začetek slike (http://www.w3schools.com/tags/tag_a.asp).

5. Kaj se izpiše na zaslon, če bi besedo AVTO zapisali, kot je navedeno v nadaljevanju?

   ```
   <u><i><b>AVTO</b></i></u>
   ```

   a) AVTO bi se izpisal krepko.
   b) Beseda AVTO bi bila podčrtana.
   c) Beseda AVTO bi se izpisala krepko, poševno in podčrtano.
   d) To je nepravilna oblika.

6. XHTML se pojavlja v treh različicah, in sicer _____, _____ in _____.

3 SLOGOVNE PREDLOGE CSS

Sam HTML oz. XHTML ni dovolj za oblikovanje spletnih strani. Vedno večje zahteve uporabnikov in hitrejši dostopi do interneta so provedli do tega, da smo začeli bolj estetsko oblikovati spletne strani. Zato je bil razvit CSS, ki nam pri tem še kako pomaga. Da bi bila zadeva še bolj zanimiva je na voljo množica programske opreme za oblikovanje slogovnih predlog.

Slogovne predloge (CSS) določajo slog (obliko) spletnega dokumenta. Podobno kot v urejevalnikih besedil določimo lastnosti slogov za oblikovanje dokumenta, lahko z uporabo CSS standarda določimo obliko slogov, ki jih uporabimo pri ustvarjanju spletnih portalov. Z uporabo slogovne predloge lahko s spremembo enega dokumenta (CSS) dosežemo nov izgled spletnega portala. Z uporabo slogov lahko določamo, kako naj pregledovalnik oblikuje posamezne elemente našega spletnega dokumenta. Določiti je mogoče celo vrsto oblikovnih lastnosti, med katere spadajo ozadje, robovi, razmiki, odmiki, pisava, poravnava, barva ...

Naloga:

Nalogo iz poglavja 2 bomo dopolnili. Narediti je potrebno štiri slogovne predloge in jih uporabiti v index.html. Naredili bomo slogovni predlogi za Heading 1 in 2, ozadje in povezavo na dokument Firefox3.html. Za razumevanje naloge in rešitev le-te bomo najprej podali nekaj smernic in napotkov za izdelavo slogovnih predlog.

Podrobne informacije in razlaga slogovnih predlog je dostopna na spletni strani http://www.w3schools.com/css/ (10. 7. 2010).

Jezik XHTML je premalo za estetsko oblikovane spletne strani, zato ga lahko oblikujemo s slogovnimi predlogami. Slogovne predloge so ponavadi shranjene v posebnih datotekah, omogočajo pa, da z nekaj spremembami dosežemo popolnoma drugačen videz spletnih strani. Da bi razumeli estetiko strani, bomo najprej spoznali zgradbo in uporabo CSS-dokumentov.

3.1 PROGRAMSKA OPREMA ZA IZDELAVO SLOGOVNIH PREDLOG

Za pisanje slogovnih predlog lahko uporabljamo navaden tekstovni urejevalnik ali naprednejši urejevalnik. Uporabimo lahko že nameščeno orodje NetBeans ali pa uporabimo katero od komercialnih orodij. V nadaljevanju bomo na kratko pregledali tri orodja.

NetBeans

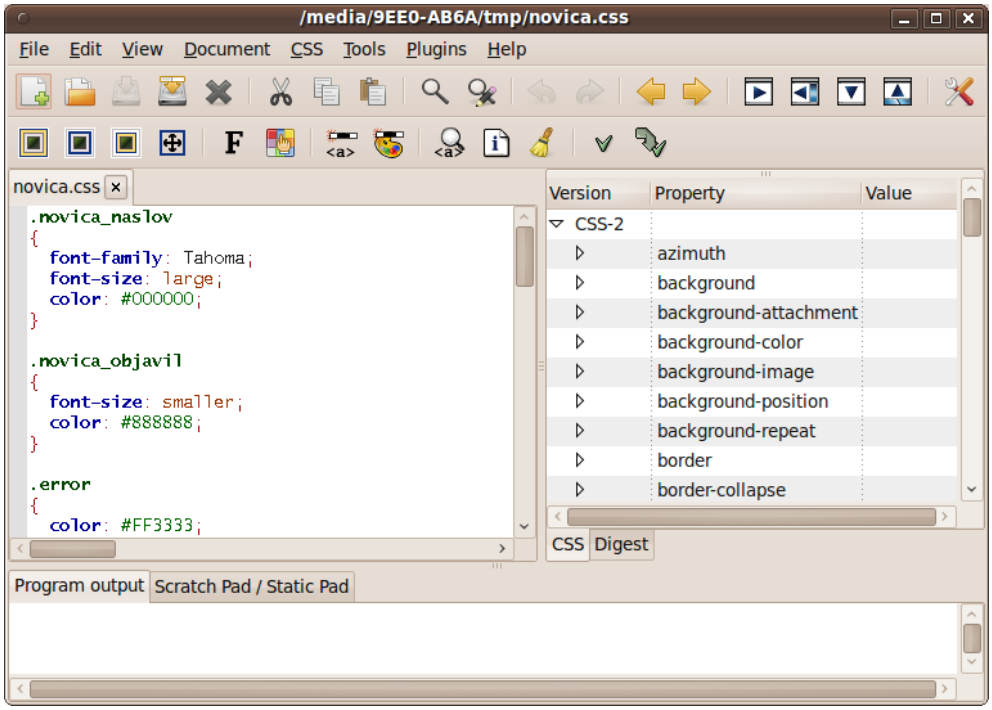
V sklopu novega projekta, ki ga lahko ustvarimo z NetBeansom ta vsebuje orodje za pregled in pisanje slogovnih predlog. Njegova dobra lastnost je, da ustrezno označi (obarva) izvorno kodo predloge, vsebuje pa tudi področje predogleda (angl. Preview), v katerem lahko vidimo vizualni izgled slogovne predloge. Njegov minus je, da nima vmesnika, v katerem bi bile vidne vse lastnosti CSS-standarda, zato jih je več ali manj potrebno vedeti. Dobra lastnost je tudi, da deluje na več operacijskih sistemih (Windows, Linux, Mac OS X, Solaris …). NetBeans je dosegljiv na naslovu www.netbeans.org (10. 7. 2010).

Rapid CSS

Gre za komercialno orodje, ki deluje v operacijskem sistemu Windows. Omogoča enostavno izdelavo slogovnih predlog, saj vsebuje različne funkcije, ki olajšajo izdelavo predlog. Vsebuje funkcijo za avtomatsko dopolnitev (angl. Auto-Complete) lastnosti predlog. Če želimo npr. nastaviti ozadje, napišemo črko b in program ponudi vse lastnosti na b, ki jih potem iz spustnega seznama tudi izberemo. Vsebuje tudi nadzornik kode (angl. code-inspector) in ostale funkcije, ki precej olajšajo izdelavo slogovnih predlog. Njegov minus je cena in omejenost na operacijski sistem Windows. Dostopen je na naslovu www.blumentals.net/rapidcss/ (10. 7. 2010).

Cssed

Cssed je odprtokodno orodje, izključno namenjeno izdelavi slogovnih predlog. Prenesemo ga lahko s spletne strani http://cssed.sourceforge.net/ (10. 7. 2010). Ponuja pa praktično vse, kar rabi povprečen uporabnik. Na voljo je za različne operacijske sisteme.



Slika 11: Cssed

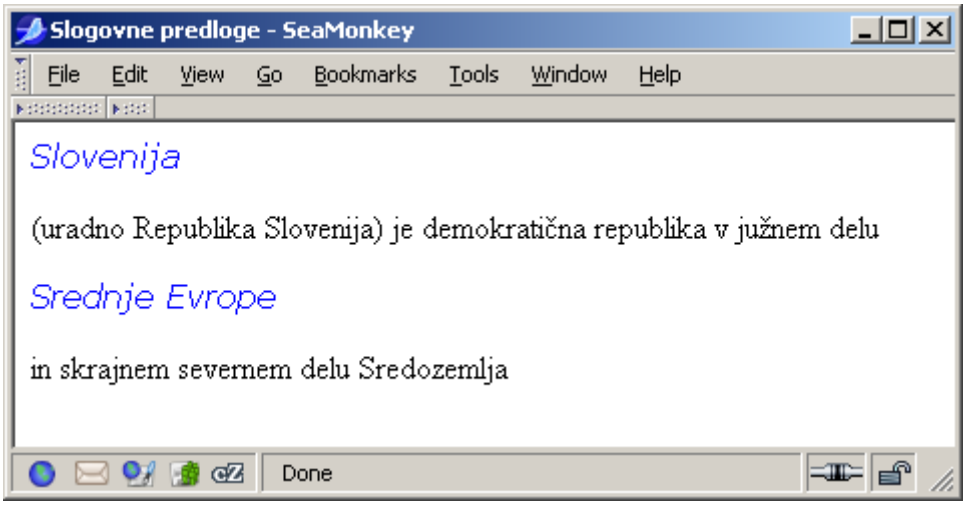
Ugotovite, kako bi lahko v operacijskem sistemu Linux namestili Cssed preko repozitorijev.

3.2 SLOGOVNE PREDLOGE IN LASTNOSTI

Slogovno predlogo izdelamo po pravilih oz. standardu za določanje slogovnih lastnosti. Poglejmo si za začetek določitev lastnosti za element p:

```
p
{
        font-family: Verdana, Arial;
        font-style: italic;
        color: blue;
}
```

Slogovna predloga, ki je podana v primeru, se sklicuje na vse značke <p> v xhtml dokumentu in za njih nastavi modro pisavo, ki je ležeča in je tipa Verdana oz. Arial.



Slika 12: Oblikovani odstavki s slogovnimi predlogami

Glavni del xhtml dokumenta, ki se nahaja v <body> je v tem primeru zapisan kot:

```
<p>Slovenija</p> (uradno Republika Slovenija) je demokratična republika v južnem delu <p>Srednje Evrope</p> in skrajnem severnem delu Sredozemlja
```

Sintaksa:

```
ime_slogovne spremenljivke
{
        lastnost_1: vrednost_lastnosti_1 [, vrednost_lastnosti_1 ...];
        [lastnost_2: vrednost_lastnosti_2 [, vrednost_lastnosti_2 ...] ...]
}
```

Vsako pravilo se začne s slogovno spremenljivko oz. selektorjem (v zgornjem primeru p). Slogovna spremenljivka je običajno eden ali več elementov (značk) XHTML-dokumenta.

Nato sledi določitev lastnosti znotraj zavitih oklepajev. Lastnost določimo s parom:

lastnost: vrednost [, vrednost ...];

Najprej zapišemo lastnost (npr. color), nato dvopičje, za njim pa vrednost (npr. blue). V primeru, da je napisanih več pravil, jih med seboj ločimo s podpičji.

**Definiranje področja z in **

Kot pomoč pri oblikovanju lahko uporabljamo tudi xhtml oznaki in . Prva je namenjena oblikovanju bloka, druga pa oblikovanju odstavka. Če jima ne določimo slogovne oblike, nimata nobenega oblikovnega učinka.

3.3 VKLJUČITEV SLOGOVNE PREDLOGE V XHTML-DOKUMENT

Slogovno predlogo lahko v spletni dokument vključimo na tri načine:

- kot zunanjo slogovno predlogo,
- kot notranjo slogovno predlogo,
- kot slog znotraj oznake oz. značke.

Zunanja predloga
Slogovno predlogo povežemo z XHTML-dokumentom tako, da v glavi (<head>) XHTML-dokumenta uporabimo element link.
Npr.

```
<link rel="stylesheet" type="text/css" href="MojSlog.css" />
```

Uporabimo lastnosti:
rel="stylesheet" (določa odvisnost na slogovno predlogo),
type="text/css" (vrsta dokumenta je besedilna datoteka, ki vsebuje CSS-pravila),
href="MojSlog.css" (referenca na dokument, ki vsebuje slogovno predlogo).

Če je potrebno spremeniti obliko, potem spremenimo le vsebino slogovne datoteke (v zgornjem primeru MojSlog.css). Tako enostavno dobimo nov izgled vseh dokumentov, ki so povezani na to datoteko (npr. v podjetjih, ki zahtevajo enoten izgled spletnih strani v svojih enotah). V tem primeru se lahko iz dokumenta XHTML sklicujemo na dokument CSS. Brskalnik dobi lastnosti iz datoteke s slogom in jih priredi pripadajočim oznakam oz. značkam v dokumentu.

Zunanjo slogovno predlogo lahko napišemo v kateremkoli urejevalniku besedil, obstajajo pa tudi programi za pisanje in preverjanje slogovnih predlog (tekstovni in grafični), ki nam le-te pomagajo napisati v krajšem času. Slogovna predloga naj ne vsebuje ukazov XHTML, saj lahko pride do napak. Slogovno datoteko shranimo v datoteko s končnico "*.css".

Notranja slogovna predloga
Notranja slogovna predloga pripada le tistemu dokumentu, v katerem je zapisana. Določimo jo v glavi dokumenta z uporabo elementa style.
Primer:

```
...
<head>
	<style type="text/css">
		hr {color: yellow}
		p {margin-left: 20px}
		body {background-image: url("slike/ozadje.gif")}
	</style>
</head>
...
```

Ker brskalnik običajno ignorira neznane ukaze, se lahko zgodi, da se slog izpiše v dokumentu kot del vsebine. Problem rešimo tako, da napišemo slog med komentar: <!-- slog -->.

```
...
<head>
	<style type="text/css">
	<!--
		hr {color: red}
		p {margin-left: 20px}
		body {background-image: url("slike/ozadje.gif")}
	-->
	</style>
</head>
...
```

`Background-image` se uporablja za določitev slike ozadja. Kje morajo biti shranjene te slike v primerjavi s CSS-dokumentom?

Ta način pa povzroča težave v XML-prevajalnikih, ki vse, kar je v komentarjih, prezrejo, torej tudi sloge. Edini strogo združljiv način je uporaba zunanjih slogovnih predlog.

Slog znotraj značke (za določen element)

Slog lahko določimo tudi vsakemu elementu znotraj oznake. V praksi ta način ne uporabljamo, saj ni ekonomičen in praktičen. Izjemoma ga uporabimo le takrat, kadar moramo določiti neko specifično lastnost nekemu elementu.

Če hočemo spremeniti pisavo odstavka, potem lahko zapišemo:

```
<p style="color: red">Besedilo.</p>
```

3.4 SINTAKSA

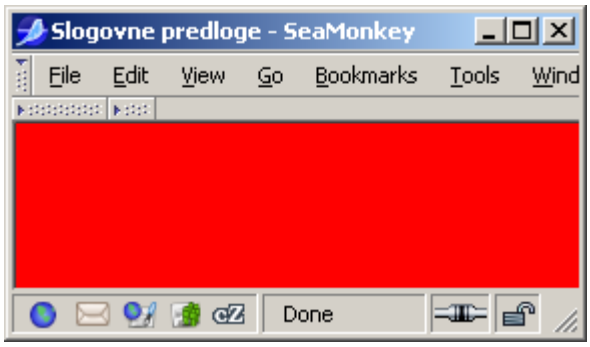
Slogovna predloga je zgrajena iz niza pravil. Pravilo je sestavljeno iz dveh delov: iz **selektorja** in iz **deklaracije**. Deklaracija je sestavljena iz lastnosti in vrednosti.

```
selektor
{lastnost: vrednost}
```

Selektor je lahko katerikoli element (značka), uporabljen v XHTML-kodi.

Primer spremembe barve ozadja na rdeče za celoten dokument.

```
body {background-color: red}
```



Slika 13: Rdeče ozadje s pomočjo značke background-color

Vrednosti se ne pišejo z narekovaji. Izjemoma so v narekovajih nizi, če je vrednost sestavljena iz dveh ali več besed:

```
p {font-family: "Arial CE"}
```

Če želimo selektorju prirediti več lastnosti, potem jih ločimo s podpičjem.

```
p
{
        text-align: center;
        color: red;
        font-size: 12pt;
}
```

Če se selektorju priredi več možnih vrednosti lastnosti, potem jih ločimo z vejico. Brskalnik v seznamu izbere prvo, ki jo prepozna.

```
p
{
        cursor: text, pointer, wait;
}

blockquote
{
        font-family : Verdana, Geneva, Arial, Helvetica, sans-serif;
}
```

Zaradi boljše preglednosti, je priporočljivo, pisanje lastnosti eno pod drugo.

```
p
{
        text-align: center;
        color: red;
        font-size: 12pt
}
```

3.5 KOMENTARJI V CSS

Začetek komentarja označimo z "/*" , zapišemo komentar in ga zaključimo z "*/":

```
/* to je komentar */
p
{
        text-align: center;
        /* drug komentar */
        color: black;
        font-family: Arial;
}
```

3.6 KASKADE

Uporabimo lahko več slogov. Posredovanje informacij o slogu brskalniku je možno na več načinov. Slog elementa je lahko vsebovan neposredno v XHTML-elementu. Lahko ga uporabimo v glavi <head> XHTML-dokumenta ali pa v zunanji datoteki. Seveda lahko te tri možnosti opisov slogov tudi kombiniramo. Nekaj jih opišemo v zunanjih datotekah, nekaj opisov vključimo kar v glavo dokumenta, preostanek pa definiramo sproti. Glavni namen slogov je ločiti oblikovanje od vsebine, zato je priporočljivo uporabiti zunanjo slogovno datoteko.

Če uporabimo več načinov, potem brskalnik upošteva slog v naslednjem vrstnem redu:

1. slog znotraj elementa;
2. slogovna predloga v glavi XHTML-dokumenta;
3. zunanja slogovna predloga;
4. privzeta vrednost v brskalniku.

Slog znotraj elementa ima največjo prioriteto, sledi mu slogovna predloga v glavi dokumenta in na koncu, če ni nikjer določen slog tega elementa, potem brskalnik vzame svojo privzeto vrednost.

V primeru, da je neko pravilo napisano dvakrat z isto prioriteto, velja, da se uporabi zadnje zapisano.

3.7 PRAVILO !IMPORTANT

Pravilo !important uporabimo, kadar želimo, da neka deklaracija velja vedno. Če npr. v zunanji slogovni predlogi ob deklaraciji uporabimo pravilo !important, potem ta deklaracija velja vedno, tudi če jo v kakšni drugi slogovni predlogi spremenimo. Če uporabimo več pravil !important, so si vsa pravila enakovredna.

Primer:

```
p
{
font-size: 12pt !important;
}
```

3.8 ZDRUŽEVANJE

Združevanje pomeni določanje lastnosti več elementom hkrati.

Primer določitve modre barve pisave naslovom od h3 do h5.

```
h3
{
        color: blue;
}

h4
{
        color: blue;
}

h5
{
        color: blue;
}
```

je enakovredno:

```
h3, h4, h5
{
        color: blue;
}
```

3.9 DEDOVANJE

Primer:

```
table
{
color: red;
font-family: Verdana;
}

td
{
color: green
}
```

V tem primeru je pisava v tabeli zelene barve vrste pisave Verdana.

Primer:

```
p
{
color:red;

font-size:12pt;
}

p #prvi

{
color:orange;
}
```

p #prvi podeduje velikost pisave font-size:12pt; barva pa je oranžna.
To velja za vse podobne primere. Povsod, kjer je določen glavni element (p, h, td ...), njegov podelement (p #prvi, p.prvi ...) podeduje vse lastnosti, ki niso predpisane.

3.10 ATRIBUT CLASS

Z atributom class v XHTML-elementu lahko določimo oblikovno lastnost več elementom v dokumentu.

Primer določitve različne poravnave pri odstavkih

```
.desno {text-align: right}
```

XHTML:

```
<p class="desno">Desna poravnava odstavka</p>
<h2 class="desno">Desna poravnava naslova</h2>
```

Z uporabo lastnosti class lahko naredimo podmnožico oblikovanja samo za določene elemente:

CSS-dokument:

```
p.desno {text-align: right;}
p.levo {text-align: left;}
```

XHTML-dokument:

```
<p class="desno">Ta odstavek ima desno poravnavo.</p>
<p class="levo">Ta odstavek ima levo poravnavo.</p>
```

Če v definiciji sloga uporabimo le atribut class z vrednostjo .levo, potem lahko lastnosti tega atributa uporabimo za več elementov.
V primeru je prikazano, kako lahko .levo uporabimo za odstavek (`<p>`) in naslov (`<h1>`):

CSS:

```
.levo {text-aligl: left}
```

XHTML:

```
<h1 class="levo">
       Naslov je poravnan levo
</h1>
<p class="levo">
       Besedilo je poravnano na levo
</p>
```

3.11 PSEVDOPODRAZREDI

Psevdorazredi in psevdoelementi so že v naprej določeni in jih ne moremo sami definirati.

Pri elementu <a> ločimo več stanj, ki jih opišemo s posebnimi podrazredi: link (neobiskana povezava), visited (obiskana povezava), active (kliknili smo na povezavo, gumb še držimo), hover (z miško smo nad povezavo, gumba nismo pritisnili).

Sintaksa psevdorazreda:

```
selektor:psevdo_razred
{lastnost: vrednost}
```

Primer:

```
a
{
font-family:Verdana
} /* velja za vse  */

a:link
{
color: yellow;
font-style: normal;
text-decoration: none;
} /* običajna oblika povezave  */

a:visited
{
color: green;
font-style: normal;
text-decoration: line-through;
}/* obiskana povezava  */

a:active
{
color: blue;
font-style: italic;
text-decoration: none;
}  /* aktivna povezava */

a:hover
{
color: red;
font-style: normal;
text-decoration: underline
} /* z miško nad povezavo  */
```

Na enak način so narejeni posevdorazredi in psevdoelementi.
Psevdorazredi: active, hover, link, visited, first, left, right, first-child, lang

Psevdoelementi: first-letter, first-line, before, after

3.12 ATRIBUT ID

Atribut id se uporablja na podoben način kot class, le da ga lahko v dokumentu XHTML uporabimo samo enkrat.

XHTML:

```
<p id="naslov">
       To je naš naslov !
</p>
```

CSS:

```
p#naslov
{
        font-size:110%;
        font-weight:bold;
        color:#0000ff;
        background-color:transparent;
}
```

Ker se lahko uporabi id v dokumentu samo enkrat, je mogoče izpustiti navedbo XHTML-elementa. Enakovredni zapis:

XHTML:

```
<p id="naslov">
       To je naš naslov !
</p>
```

CSS:

```
#naslov
{
        font-size:110%;
        font-weight:bold;
        color:#0000ff;
        background-color:transparent;
}
```

3.13 PRAVILA AD

Medijski tipi so namenjeni za prikaz vsebine na različnih medijih. Niso zastopane pri Netscape Navigatorju, Microsoft Internet Explorer pa jih podpira le tri.

3.14 PRAVILO @MEDIA

S pravilom @media v CSS oznanimo, da bomo našteli eno izmed lastnosti, ki pod to sodijo. Tu določimo, da če stran gledamo preko brskalnika, vidimo pisavo Verdana, velikosti 14 px, če pa dokument natisnemo, se tiska s pisavo Times New Roman, velikosti 10 px.

```
<style>
        @media screen
        {
                p.test {font-family: Verdana,sans-serif; font-size:14px}
        }

        @media print
        {
                p.test {font-family:"Times New Roman",serif; font-size:10px}
        }


        @media screen,print
        {
                p.test {font-weight:bold}
        }
</style>
...
```

3.15 ENOTE

Mere se pišejo brez presledka med vrednostjo in enoto. Tako je zapis 23 pt napačen, pravilno je 23pt. Za 0 ni potrebno pisati enot. Vrednosti so cela števila, brez decimalnih znakov in vrednosti.

Enote so relativne ali absolutne. Relativne so tiste, katerih določitev je odvisna od nekih drugih lastnosti (npr. od monitorja).

Absolutne enote

in: inches, palec - 1 palec je enak 2.54 cm.
cm: centimeters
mm: millimeters
pt: points - points v CSS2 je enaka 1/72 palca (moderna definicija).
pc: picas - pica je enak 12 point.

Relativne enote

em: višina trenutne pisave
ex: višina črke x v trenutni pisavi
px: points - pika na zaslonu

Odstotki

Predstavljajo velikost glede na razpoložljiv prostor ali privzeto vrednost. Pogosto se uporablja pri pisavah, slikah.

Barve

Barvne vrednosti so določene s številkami za rdečo, zeleno in modro ali z imeni. Barva je na monitorju pravilno prikazana, če je nastavljen na naravne barve (faktor gama ima vrednost 2,2). To pravilo velja samo za CSS.

Poimenovane barve:

aqua (#00ffff), black (#000000), blue (#0000ff), fuchsia (#ff00ff), gray (#808080), green (#008000), lime (#00ff00), maroon (#800000), navy (#000080), olive (#808000), orange (#ffA500), purple (#800080), red (#ff0000), silver (#c0c0c0), teal (#008080), white (#ffffff), yellow (#ffff00) itd.

Blue	#0000FF	
BlueViolet	#8A2BE2	
Brown	#A52A2A	
BurlyWood	#DEB887	
CadetBlue	#5F9EA0	
Chartreuse	#7FFF00	
Chocolate	#D2691E	
Coral	#FF7F50	
CornflowerBlue	#6495ED	
Cornsilk	#FFF8DC	
Crimson	#DC143C	
Cyan	#00FFFF	

Slika 14: Nekaj poimenovanih barv
Vir: www.w3schools.com (10. 7. 2010)

S številkami podane barve

Številčne vrednosti za barve je mogoče podati na naslednje načine:

#rrggbb (po dve šestnajstiški vrednosti za rdečo, zeleno in modro);
#rgb (po eno šestnajstiško vrednost za rdečo, zeleno in modro);
rgb(x,x,x) kjer je x je celo število med 0 in 255;
rgb(x%,x%,x%) kjer je x je število 0.0 in 100.0.

3.16 URI

URL se uporablja za naslavljanje virov v spletu. Nov izraz je URN[7]. Skupaj z URL ju imenujemo URI[8]. Torej URL + URN = URI.

Vrednost URI je zapisana po pravilih, ki jih določajo standardi. Narekovaji, vejice, presledki in druge ubežne sekvence morajo biti kodirane.

URI je lahko predstavljen kot relativen ali absoluten.

Primeri:

```
body { background: url(image_name.gif) }
body { background: url("image_name.gif") }
body { background: url(http://www.somesite.com/image_name.gif) }
```

Rešitev naloge:

Dana naloga v uvodu poglavja zahteva, da nalogo iz predhodnega poglavja 2 dopolnimo. Dodati je potrebno barvo ozadja, spremeniti Heading 1 in Heading 2 ter določiti povezavo na podstran firefox3.html.

[7] Ang. Uniform Resource Name
[8] Ang. Uniform Resource Identifiers

Slika 15: Spletna stran po dodani slogovni predlogi

V XHTML-dokument vključujemo slogovne predloge v glavi, tj. med znački <head></head>. Če se naša slogovna predloga imenuje `predloga.css`, jo vključimo z ukazom:

```
<link rel="stylesheet" type="text/css" href="predloga.css" />
```

Slogovna predloga:

```
h1
{
        color: #0066CC;
        font-family: Tahoma;
        font-size: 20px;
}

h2
{
        color: #FF0033;
        font-family: Tahoma;
        font-size: 16px;
}

body
{
        background-color: #FFFF99;
}

.povezava A:link {
        color: #002571;
        text-decoration: none;
        font-family: Verdana,Arial,Helvetica,sans-serif;
        font-size: 12px;
}

.povezava A:visited {
        color: #002571;
        text-decoration: none;
```

```
        font-family: Verdana,Arial,Helvetica,sans-serif;
        font-size: 12px;
}

.povezava A:active {
        color: #002571;
        text-decoration: none;
        font-family: Verdana,Arial,Helvetica,sans-serif;
        font-size: 12px;
}

.povezava A:hover {
        color: #002571;
        text-decoration: underline;
        font-family: Verdana,Arial,Helvetica,sans-serif;
        font-size: 12px;
}
```

Predloge h1, h2 in body se nanašajo na XHTML-značke. Če želimo privzetim značkam določiti slog uporabimo enako ime, kot je XHTML-značka. Pomembno je poudariti, da teh slogovnih predlog ne začnemo s piko. Pri h1 in h2 smo določili barvo, tip in velikost pisave. Predloga body vsebuje barvo ozadja, ki obenem predstavlja barvo ozadja celotnega dokumenta.

Nekoliko več pozornosti je potrebno posvetiti slogu povezave. V našem primeru je povezava razdeljena na štiri stanja:

- link: določa, kakšna bo privzeta neobiskana povezava.
- visited: deklarira slog obiskane povezave. Tipično bi se taka povezava lahko npr. obarvala z drugo barvo ipd. V našem primeru je enaka izvorni povezavi.
- active: deklarira stil obiskane povezave. Povezava postane obiskana, ko vsaj enkrat kliknemo na njo.
- hover: povezava je hover takrat, ko se z miško enkrat pomaknemo prek nje. V našem primeru smo za hover določili, da se povezava podčrta.

Slogovno predlogo povezava vključimo v datoteko:

index.html

```
Za več informacij o različici 3 sledite <span class="povezava"><a
href="firefox3.html">povezavi</a></span>.
```

Pri tem uporabimo značko `<span class="povezava"></span>`.

Povzetek:

Za pisanje slogovnih predlog lahko uporabljamo različna programska orodja. Nekomercionalni orodji sta NetBeans in Cssed, komercialno pa Rapid CSS. Seveda obstajaj še tudi druga orodja. Za pisanje slogovnih predlog obstajajo pravila, ki se jih je potrebno držati.

```
ime_slogovne spremenljivke
{
        lastnost_1: vrednost_lastnosti_1 [, vrednost_lastnosti_1...];
        [lastnost_2: vrednost_lastnosti_2 [, vrednost_lastnosti_2...]...]
}
```

Vsako pravilo se začne s slogovno spremenljivko oz. selektorjem (v zgornjem primeru p). Slogovna spremenljivka je običajno eden ali več elementov (značk) v XHTML-dokumentu.

Nato sledi določitev lastnosti znotraj zavitih oklepajev. Lastnost določimo s parom:

lastnost: vrednost [, vrednost ...];

Najprej je napisana lastnost (npr. color), nato dvopičje, za njim pa vrednost (npr. blue). V primeru, da je napisanih več pravil, jih med seboj ločimo s podpičji.

Slogovne predloge vključimo v glavi XHTML dokumenta, in sicer z ukazom:

```
<link rel="stylesheet" type="text/css" href="predloga.css" />
```

Pri čemer je predloga.css ime predloge in je lahko poljubno veljavno ime.

Naloge:

1. Uporabite poljuben članek iz Wikipedije in ga oblikujte v novo spletno stran po naslednjih kriterijih (uporabite slogovne predloge):
    a) Vse glavne naslove (h1) pobarvajte z zeleno barvo in jim določite črno ozadje.
    b) Za ozadje celotne spletne strani določite svetlo modro barvo.
    c) Sliko oblikujte, da bo dolžine točno 200 in višine 100.
    d) Vstavite horizontalno črto in jo obarvajte črno ter določite 40 % širino.
    e) Pisavo nastavite na Tahoma, velikosti 12.

2. Kaj je selektor?

3. Kako bi zapisali slog z imenom gumb, pri čemer bi določili velikost pisave 20, barvo pisave pa modro?

4. Številčna vrednost bele barve je _____, črne pa _____.

4 PHP IN XHTML OBRAZCI

Več kot polovica vseh spletnih strani je narejenih s programskih jezikom PHP. PHP predstavlja dodano vrednost obstoječim stranem, saj dodaja interaktivnost in dinamičnost. Spreminjanje vsebine, registracija, prijava, forumi ipd., so teme, ki zahtevajo programski jezik s krmilnimi stavki. PHP jih vsebuje. Za urejanje spletne vsebine pa potrebujemo obrazce, preko katerih uporabniki vnašajo podatke. Obrazci so elementi, vnosna polja in v tem poglavju jih bomo spoznali.

PHP[9] je odprtokodni programski jezik, ki se uporablja za razvoj dinamičnih spletnih vsebin. Podoben je nekaterim programskim jezikom, kot npr. C ali C++, zato izkušenim programerjem ni potrebno veliko dodatnega učenja za razvoj dinamičnih spletnih vsebin. PHP je strežniški programski jezik, kar pomeni, da teče na strežniku, za njegovo delovanje pa potrebujemo spletni strežnik. PHP spada v skupino interpreterskih programskih jezikov, kar pomeni, da se na začetku ne prevede celotna izvorna koda, ampak se sproti interpretira. Spletni strežnik ima to funkcijo, da interpretirano izvorno kodo pošlje brskalniku v obliki XHTML-kode. Tako uporabnik ne more videti izvorne kode.

Naloga:

V poglavju 3 smo se lotili slogovnih predlog in dokument index.html opremili s slogi. Besedilo o Firefoxu, ki je zapisano, je bilo vnaprej določeno. Našo spletno stran želimo dopolniti, da se na mestu tega besedila prikaže tekst, ki ga uporabnik sam vpiše v spletnem obrazcu. Za rešitev te naloge moramo spoznati PHP jezik in XHTML-obrazce, katerih vsebina je podana v nadaljevanju tega poglavja.

Interakcija med uporabnikom in spletno stranjo je mogoča le z uporabo programskega jezika. Predstavljajmo si, da imamo statično spletno stran, v kateri moramo za vsako spremembo posodobiti izvorni XHTML-dokument in ga ponovno naložiti (npr. prek FTP-odjemalca) na strežnik. Tako delo je zelo zamudno, zato se kot na dlani ponuja rešitev, da uporabimo metode, s katerimi bomo lažje posodabljali vsebino spletnih strani. Tu pride v poštev jezik PHP in XHTML-obrazci.

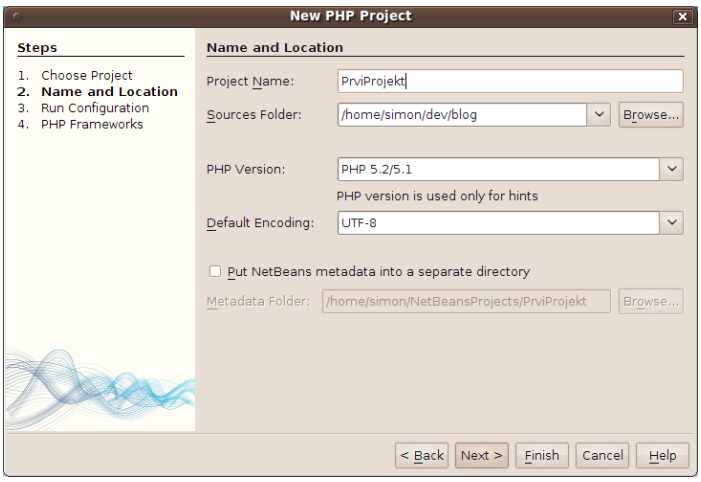
4.1 SKRIPTNI PROGRAMSKI JEZIK PHP

4.1.1 Prva skripta

V prvem poglavju smo namestili NetBeans, zato bomo prvo PHP skripto napisali s pomočjo tega orodja.

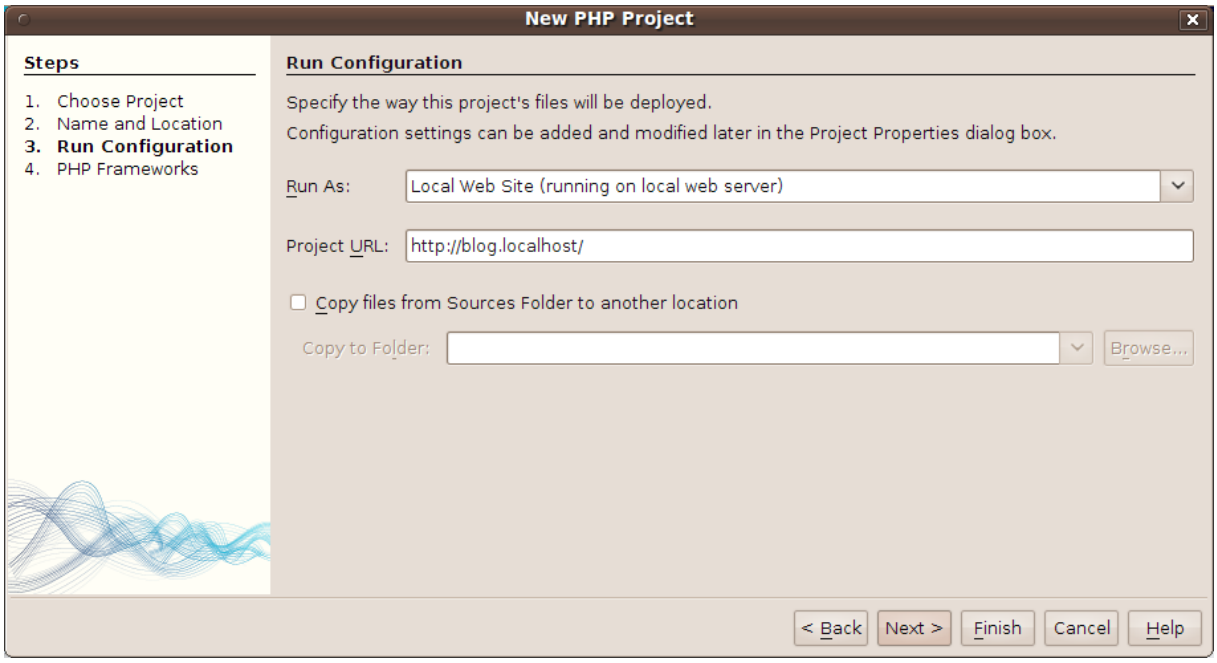
Ko zaženemo NetBeans, izberemo `File->New Project`, s čimer začnemo nov projekt. Izberemo `PHP Application` in kliknemo Next. Določimo Ime projekta, npr. PrviProjekt, ter v področju Sources Folder izberemo mapo, kjer bo shranjena izvorna koda projekta. V poglavju 1.4 smo pri konfiguraciji spletnega strežnika ustvarili mapo `dev/blog` ter nastavili navideznega gostitelja http://blog.localhost. V Sources Folder izberemo mapo `dev/blog`, kot je prikazano na spodnji sliki.

[9] Hypertext Preprocessor



Slika 16: Nov PHP Projekt – korak 2

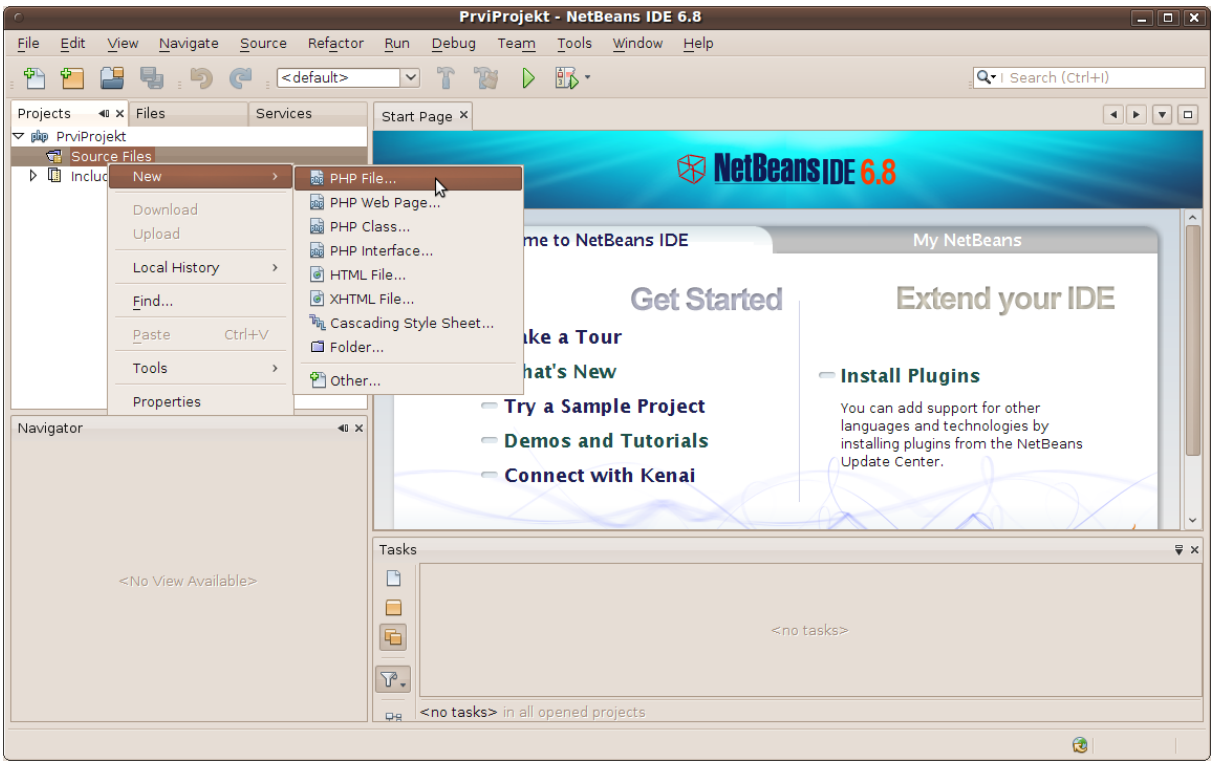
S klikom na Next pridemo do tretjega koraka, kjer najprej določimo, da bomo spletno stran preizkušali lokalno (Local Web Site (running on local web server)) in da se URL glasi http://blog.localhost/



Slika 17: Nov PHP Projekt – korak 3

V četrtem koraku imamo možnost izbire Symfony projekta. Zaenkrat to v našem primeru še ne pride v poštev, zato pustimo odkljukano.

Z zadnjim korakom smo končali z osnovnimi nastavitvami projekta in odpre se razvojno okolje NetBeans. Ustvariti moramo novo datoteko izvorne kode, zato v področju Projects izberemo `Source Files`, kliknemo z desno miškino tipko in izberemo `New->PHP File`.



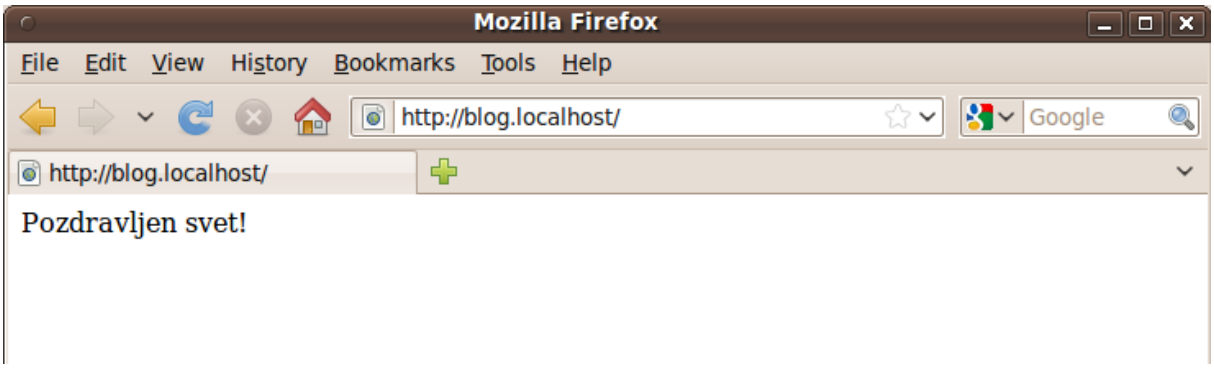
Slika 18: Nova PHP-datoteka

Datoteko poimenujemo index.php in v njo vpišemo:

```
<? php
        echo "Pozdravljen svet!";
?>
```

> *Datoteko moramo shraniti s končnico php, kar pove strežniku, da prebrano kodo interpretira kot kodo PHP.*

Datoteko shranimo, odpremo brskalnik in kot naslov vpišemo http://blog.localhost. Izpiše se Pozdravljen svet.



Slika 19: Pozdravljen svet

4.1.2 Označevanje stavkov PHP

Za pravilno delovanje kode PHP je le-to potrebno označiti s posebnimi znaki. Če tega ne naredimo, potem se koda zamenja z navadno kodo XHTML, kar pomeni izpis celotne kode v brskalniku. Spodnja tabela prikazuje štiri načine označevanja kode.

Tabela 2: Označevanje stavkov

Stil značke	*Začetna oznaka*	*Končna oznaka*
standardne značke	<?php	?>
kratke značke	<?	?>
značke ASP	<%	%>
skriptne značke	<SCRIPT LANGUAGE="php">	</SCRIPT>

Vir: http://www.php.net/

Od štirih različnih načinov sta le navaden in skriptni način tista, ki bosta zanesljivo delovala na vseh sistemih. Podporo za kratke značke in značke ASP moramo namreč ločeno vključiti v datoteki php.ini. Poiščemo vrstici short_open_tag in asp_tags ter obe omogočimo:

```
Short_open_tag = On
Asp_tags = On
```

Če omogočimo zgoraj napisani podpori, lahko kodo pišemo takole:

```
<?
echo "Pozdravljen svet!";
?>

<?php
echo "Pozdravljen svet!";
?>

<%
echo "Pozdravljen svet!";
%>

<SCRIPT LANGUAGE='php'>
echo "Pozdravljen svet!";
</SCRIPT>
```

Zapisano kodo zapišemo tudi v eni sami vrstici:

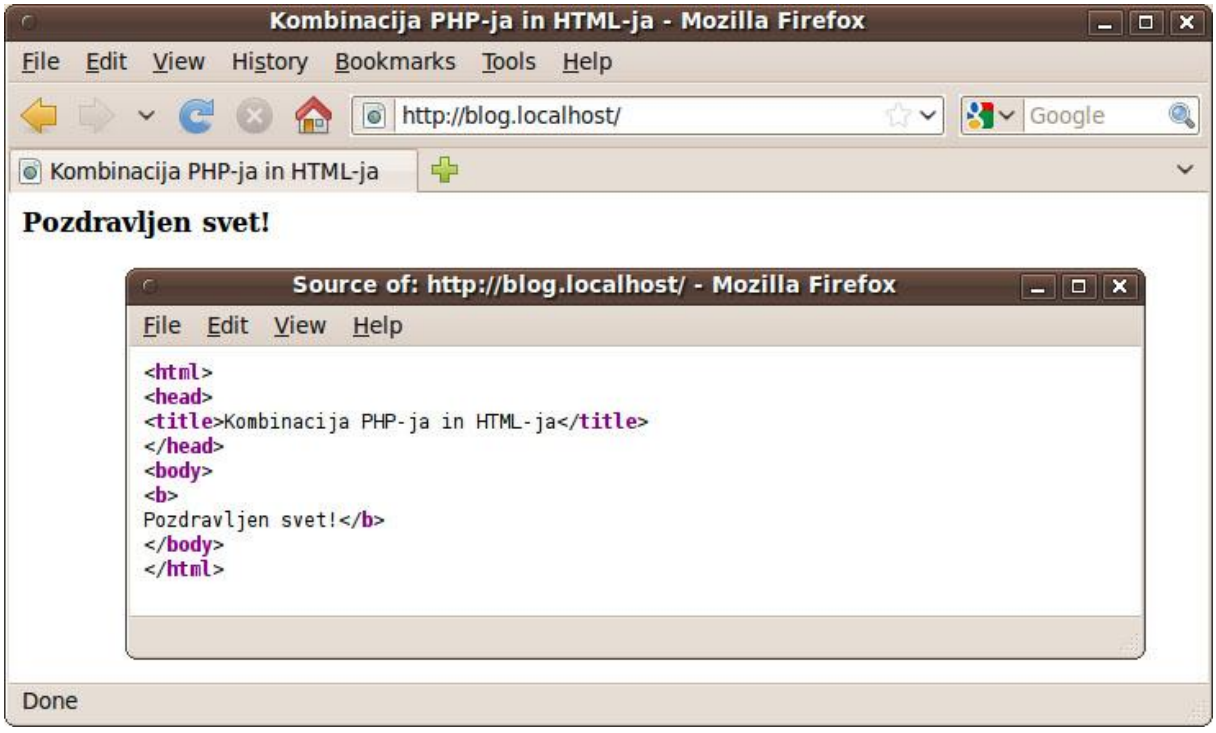
```
<? echo "Pozdravljen svet!"; ?>
```

4.1.3 Kombinacija PHP-ja in XHTML-ja

Koda, ki smo jo pisali v prejšnjem poglavju, je bila čisti php. Le-to lahko kombiniramo s kodo XHTML tako, da jo vstavimo zunaj značk, ki označujejo začetek in konec kode php.

```
<html>
<head>
<title>Kombinacija PHP-ja in HTML-ja</title>
</head>
        <body>
                <b>
                        <?php
                                echo "Pozdravljen svet!";
                        ?>
                </b>
        </body>
</html>
```

Kot lahko vidimo na primeru, je vstavljanje kode PHP v XHTML odvisno od nas samih. Vstavimo jo lahko kjerkoli, treba je le označiti začetek in konec. Izpisano kodo lahko pogledamo v brskalniku in le-ta nam s krepkimi črkami izpiše: Pozdravljen svet! Če pogledamo izvorno kodo v spletnem pregledovalniku, bo ta izpisana kot normalna koda XHTML. PHP se namreč prevede in ostane skrit samo za programerja.



Slika 20: Izpis v brskalniku in prikaz prevedene izvorne kode

V dokument XHTML lahko vključimo poljubno število blokov kode PHP. Vse skupaj je obravnavano kot en dokument in spremenljivke (funkcije, razredi …), ki jih definiramo v enem bloku, so ponavadi dosegljive tudi v blokih, ki sledijo.

4.1.4 Komentiranje kode

Koda, ki se nam zdi na prvi pogled smiselna in enostavna za razumevanje, lahko postane čez nekaj časa prav zapletena. Vstavljanje komentarjev tako postane zelo pomembno in v marsičem olajša kasnejše popravljanje ali razumevanje kode.
Enovrstični komentarji se začnejo z dvojno desno poševnico (//) ali z lojtro (#). Ves tekst, ki je za tema znakoma, se obravnava kot komentar:

```
// to je primer komentarja
# to je primer komentarja
```

Večvrstični komentar začnemo z enojno desno poravnano poševnico in zvezdico (/*), končamo pa ravno obratno (*/).

```
/*
Ves tekst,
ki je vpisan
med tema znakoma,
je komentar.
*/
```

4.1.5 Gradnja blokov

Spremenljivke

Spremenljivke uporabljamo za shranjevanje različnih vrednosti. V spremenljivko lahko shranimo številke, nize, znake, predmete, polja in logične znake. Ime spremenljivke je sestavljeno iz znaka dolar ($) in imena spremenljivke. Ime lahko vsebuje črke, številke in podčrtaj.

```
$a;
$daljsa_spremenljivka;
$2345;
$maleVELIKEcrke;
```

Podpičje na koncu vrstice označuje konec stavka PHP in ni del spremenljivke. Poznamo več različnih načinov, kako določiti vrednost spremenljivke. Definiramo jih tako, da jih vključimo v skripto in jim določimo vrednost:

```
$stevilo1 = 12;
$stevilo2 = 100;
```

Ko jim priredimo vrednost, jih lahko uporabljamo za dostop do te vrednosti. Tako lahko rečemo, da:

```
echo $stevilo1;
Pomeni isto kot:
echo 12;
```

Ta trditev velja, dokler spremenljivki $stevilo1 ne priredimo druge vrednosti, torej vse dokler je $stevilo1 enako 12.

Kaj bi se izpisalo na zaslon v primeru, če napišemo: `echo "12+3";`?
Kaj pa, če napišemo `echo 12+3;`?
Sta oba zapisa pravilna?

Dinamične spremenljivke

Kot smo spoznali, definiramo spremenljivke tako, da pred njih dodamo znak dolar. Obstaja tudi način, da spremenljivko definiramo kot kazalec:

```
$uporabnik = "Simon";
```

je enako kot:

```
$vmesnik = "uporabnik";
$$vmesnik = "Simon";
```

V obeh primerih z ukazom echo $vmesnik izpišemo Simon. $$vmesnik lahko štejemo kot dolar, kateremu sledi vrednost spremenljivke $vmesnik.

Sklicevanje na spremenljivke

Če želimo spremenljivko $Spremenljivka prirediti drugi spremenljivki $NovaSpremenljivka, to naredimo tako, da kopiramo vrednost iz prve v drugo spremenljivko. V primeru, da prvi spremenljivki spremenimo vrednost, bo vrednost v drugi spremenljivki ostala ista.

```
<html>
<head>
<title>Ni sklicevanja ene spremenljivke na drugo</title>
</head>
<body>
<?php
$Spremenljivka = 42;
$NovaSpremenljivka = $Spremenljivka; //vrednost prve sprem. priredimo drugi
$Spremenljivka = 325;
echo $NovaSpremenljivka; //izpiše vrednost 42
?>
</body>
</html>
```

V verziji PHP-ja, ki je višja od 4, lahko določimo tudi sklicevanje ene spremenljivke na drugo. To prikazuje naslednja koda:

```
<html>
<head>
<title>Sklicevanje ene spremenljivke na drugo</title>
</head>
<body>
<?php
$Spremenljivka = 42;
$NovaSpremenljivka = &$Spremenljivka; //pomnilniški naslov prve sprem. priredimo drugi
$Spremenljivka = 325;
echo $NovaSpremenljivka; //izpiše vrednost 325
?>
</body>
</html>
```

Razlika med kodama je v tem, da smo dodali znak & pred $Spremenljivka. S tem določimo, da sta spremenljivki $Spremenljivka in $NovaSpremenljivka med sabo povezani, kar pomeni, da imata enako vrednost. S tem lahko prihranimo vrstico kopiranja, kar pri večjih projektih poveča hitrost procesiranja kode.

Tipi podatkov

Različni tipi podatkov zavzemajo različno količino pomnilnika in tudi skripte jih obravnavajo drugače. V primerjavi s PHP-jem nekateri programski jeziki zahtevajo, da vnaprej določimo, kakšne vrste podatek bo katera spremenljivka vsebovala. Po eni strani to pomeni, da je PHP zelo fleksibilen, po drugi pa lahko pride pri obsežnejših skriptah do zmešnjave. Določena spremenljivka lahko vsebuje popolnoma drug podatek, kot smo pričakovali.

Tabela 3: Tipi podatkov

Tip	*Primer vrednosti*	*Opis*
Integer	5	celo število
Double	3.234	realno število
String	"Lep pozdrav"	niz znakov
Boolean	true	logična vrenost true ali false
Object		predmeti
Array		polje

Vir: http://www.php.net/

Za testiranje tipov podatkov lahko uporabimo vgrajeno funkcijo **gettype()**, ki vrne tip podatka glede na shranjeno vrednost.

```
$test = 5;
echo gettype($test); //vrne ineteger
```

Podoben primer lahko naredimo za druge podatkovne tipe. Vgrajena je tudi funkcija **settype()**, ki spremeni tip podatka.

```
$nedolocena_vrednost = 3.14
settype($nedolocena_vrednost, integer);
echo $nedolocena_vrednost; //izpise 3
```

Pri spreminjanju v podatek tipa Integer se vse številke za decimalno vejico odstranijo. Če spreminjamo podatek v tip Boolean, pa se vse vrednosti, različne od nič, smatrajo kot "true". Le-ta je pri izpisu predstavljen z 1, false pa z 0.

Operatorji in izrazi

Programski jeziki pokažejo svojo pravo moč pri operiranju s podatki. Operatorji so simboli oz. serija simbolov, ki se uporabljajo za kombinacijo vrednosti, izvedejo akcijo in določijo novo vrednost. Operand je podatek, ki se uporablja pri združitvi z operatorjem. Običajno imamo dva operanda na en operator. Primer:

```
4+5
```

4 in 5 sta operanda, operator pa je znak plus (+), ki določi novo vrednost 9. Z besedo izraz poimenujemo kombinacijo funkcij, vrednosti in operatorjev, ki določijo vrednost. Poznamo več vrst operatorjev:

- prireditveni operatorji,
- aritmetični operatorji,
- povezavni operatorji,
- kombinirani dodeljeni operatorji,
- primerjalni operatorji,
- logični operatorji.

Prireditveni operatorji

Ti operatorji so sestavljeni iz enačaja (=). Prireditveni operator vzame vrednost, ki je na njegovi desni, in jo priredi spremenljivki, ki je na njegovi levi strani.

```
$ime = "Janez";
```

Aritmetični operatorji

Aritmetičnih operatorjev je pet in vsak ima svojo funkcijo:

Tabela 4: Aritmetični operatorji

Operator	*Ime*	*Primer*	*Rezultat primera*
+	seštevanje	10+3	13
-	odštevanje	10-3	7
/	deljenje	10/3	3.3333333333333
*	množenje	10*3	30
%	ostanek pri deljenju	10%3	1

Vir: http://www.php.net/

Povezavni operatorji

Povezavni operator predstavlja pika (.). Ta operator združi vrednosti na njegovi levi in desni strani.

```
"Pozdravljen" . " svet!";
```

vrne:

```
"Pozdravljen svet!"
```

Ne glede na vrsto podatka je rezultat te operacije vedno niz (string).

Kombinirani prireditveni operatorji

PHP dovoljuje veliko kombinacij prireditvenih operatorjev, ki spremenijo levi operand. V tem primeru prireditveni operatorji spremenijo vrednost operanda, saj imajo poleg standardnega operatorja tudi enačaj.

```
$x = 4;
$x += 4; //$x dobi vrednost 8

//Ekvivalenca kombiniranemu prireditvenemu operatorju.
$x = 4;
$x = $x + 4; //$x = 8
```

Zgornji kodi sta v tem primeru ekvivalentni.

Nekaj kombiniranih prireditvenih operatorjev.

Tabela 5: Prireditveni operatorji

Operator	*Primer*	*Ekvivalent*
+=	$x += 5	$x = $x + 5
-=	$x -= 5	$x = $x – 5
/=	$x /= 5	$x = $x / 5
*=	$x *= 5	$x = $x * 5
%=	$x %= 5	$x = $x % 5
.=	$x .= "test"	$x = $x" test"

Vir: http://www.php.net/

Vsak od primerov v zgornji tabeli spremeni vrednost $x z uporabo desnega operanda.

Primerjalni operatorji

Primerjalni operatorji primerjajo operande. Če sta operanda enaka, vrnejo vrednost "true", sicer "false". Taka vrsta izrazov je zelo uporabna v kontrolnih strukturah, kot so npr. stavki if in while. Da bi preverili, ali je spremenljivka $x manjša od 5, napišemo:

```
$x < 5
```

Če bo vrednost $x enaka 3, je izraz enak "true", drugače "false".

Tabela 6: Primerjalni operatorji

Operator	*Ime*	*Vrne "true", če:*	*Primer*	*Rezultat*
==	enakovrednost	je leva stran enaka desni	$x == 5	false
!=	neenakovrednost	leva stran ni enaka desni	$x != 5	true
===	istovetnost	je leva enaka desni in sta obe istega tipa	$x === 5	false
>	večji	je leva stran večja od desne	$x > 4	false
>=	večji ali enak	je leva stran večja ali enaka desni	$x >= 4	false
<	manjši	je leva stran manjša	$x < 4	false
<=	manjši ali enak	je leva stran manjša ali enaka desni	$x <= 4	true

Vir: http://www.php.net/

Ti operatorji se največ uporabljajo z operandi tipa Integer ali Double, čeprav je mogoča tudi primerjava med znaki.

Logični operatorji

Tabela 7: Logični operatorji

Operator	*Ime*	*Vrne "true", če:*	*Primer*	*Rezultat*
||	ali	je vsaj en operand resničen	true || false	true
or	ali	je vsaj en operand resničen	true or false	true
xor	ekskluzivni ali	je natanko eden operand resničen (ne oba hkrati)	true xor true	false
&&	in	sta oba operanda resnična	true && false	false
and	in	sta oba operanda resnična	true and false	false
!	ne	operand ni resničen	!true	false

Vir: http://www.php.net/

Konstante

Spremenljivke omogočajo fleksibilen način shranjevanja podatkov, saj lahko njihove vrednosti kadarkoli spremenimo. PHP omogoča tudi definiranje vrednosti, za katere ne želimo, da se spreminjajo. S pomočjo vgrajene funkcije define() definiramo konstanto, ki je nato ne moremo več spreminjati. Kot konstanto lahko definiramo samo niz ali število.

```
<html>
	<head>
		<title>Definiranje konstante</title>
	</head>
<body>
	<?php
		define ("UPORABNIK", "Simon");
		echo "Pozdravljen" . UPORABNIK;
	?>
</body>
</html>
```

Kontrolne strukture

Velikokrat želimo, da se program izvaja pod določenimi pogoji. Sposobnost sprejemanja odločitev glede na vhodne podatke naredi spletno stran dinamično. Tako kot večina programskih jezikov tudi PHP omogoča različne vrste stikal in zank.

Stavek if

Stavek if omogoča pogojno izvedbo določene kode. Struktura je zelo podobna programskemu jeziku C. Če izraz vrne vrednost "true", potem se koda izvede, v nasprotnem primeru ne.

```
if (izraz){
	//izvedi kodo
}

//stavek if
if ($a == $b){
	echo "a je večji kot b";
}
```

Stavki if se lahko neomejeno gnezdijo v druge stavke if, kar omogoča popolno kontrolo nad pogojnim izvajanjem različnih delov našega programa.

Stavek else

Večkrat se mora izvesti določen stavek, če je izpolnjen pogoj in neki drug stavek, če ta pogoj ni izpolnjen. Za to uporabljamo pogojni stavek else. Else omogoča izvajanje nekega stavka, če stavek if vrne vrednost "false". Spodnja koda bi prikazala, da je "a je večji kot b", če je $a večji kot $b in "a ni večji kot b", če je $a manjši kot $b:

```
if ($a > $b){
        echo "a je večji kot b";
} else {
        echo "a ni večji kot b";
}
```

Stavek else se izvede samo, če stavek if vrne vrednost false.

Stavek elseif

Elseif je, kot pove že njegovo ime, kombinacija stavka if in else. Kot else se ta stavek izvede samo, če stavek if vrne vrednost false. V nasprotju z else se stavek izvede samo, če pogojni stavek vrne true.

```
if ($a > $b) {
    echo "a je večji kot b";
} elseif ($a == $b) {
    echo "a je enak b";
} else {
    echo "a je manjši kot b";
}
```

V enem stavku if je lahko več stavkov elseif. Prvi stavek elseif (če je kakšen), ki vrne vrednost true, bi se izvedel. V PHP-ju lahko napišemo 'else if' (dve besedi) in obnašanje bi bilo identično stavku 'elseif'.

Stavek elseif se izvede samo, če prejšnji stavek if ali prejšnji stavki elseif vrnejo vrednost false in trenutni stavek elseif vrne vrednost true.

Stavek while

Zanke while so najenostavnejše zanke v PHP-ju. Obnašajo se ravno tako kot v programskem jeziku C in njihova osnovna oblika je:

```
while (izraz) stavek
```

Stavek while napoti PHP k izvajanju gnezdenega stavka oz. stavkov, dokler ne vrne vrednosti true. Vrednost izraza se preveri vsakič pred začetkom zanke, kar pomeni, da tudi če se vrednost izraza spremeni med izvajanjem ugnezdenih stavkov, se zanka ne bo ustavila do konca ponovitve (vsakič, ko PHP izvede vse ugnezdene stavke v zanki, se šteje kot eno nadaljevanje). Če izraz while vrne vrednost false takoj na začetku, se ugnezdeni stavki sploh ne bodo izvedli. Tako kot pri stavku if, lahko tudi tu združimo več stavkov v eno zanko while.

```
$i = 1;
while ($i <= 10) {
    echo $i++;
}
```

Stavek do..while

Zanke do..while so zelo podobne zankam while, razlika je samo v tem, da se vrednost izraza preveri na koncu vsakega nadaljevanja in ne na začetku. Glavna razlika od zank while je, da se bo prvo nadaljevanje zanke do..while v vsakem primeru izvedlo (vrednost izraza se preveri samo na koncu nadaljevanja), medtem ko se to pri navadni zanki while ne bi zgodilo (vrednost izraza se preveri na začetku vsakega nadaljevanja, in če izraz vrne vrednost false takoj na začetku, se zanka takoj ustavi).

```
$i = 0;
do {
   echo $i;
} while ($i>0);
```

Zgornja zanka bi se izvedla samo enkrat, ker pri prvi ponovitvi (ko se preveri vrednosti izraza) izraz vrne false ($i ni večji od 0).

Stavek for

For so najbolj zahtevne zanke v PHP-ju, njihovo obnašanje je enako kot v programskem jeziku C. Sintaksa je naslednja:

```
for (izraz1; izraz2; izraz3) stavek
```

Prvi izraz (izraz1) se vedno izvede samo enkrat – na začetku zanke. Na začetku vsakega nadaljevanja se preveri vrednost izraza2. Če je le-ta true, se zanka nadaljuje in stavek oz. stavki se izvedejo. Če je vrednost izraza false, se izvajanje zanke konča. Na koncu vsakega nadaljevanja se izvede izraz 3.

Vsak od izrazov je lahko prazen. Če je prazen izraz2, se bo zanka izvajala neskončno dolgo (PHP označi to kot true, podobno kot programski jezik C). Včasih nam tak način uporabe pride še kako prav, saj velikokrat želimo, da se zanka izvaja do pogojnega stavka break in ne s preverjanjem zanke for.

```
//zanka for
for ($i = 1; $i <= 10; $i++) {
    echo $i;
}

//zanka for z uporabo break
for ($i = 1;;$i++) {
    if ($i > 10) {
        break;
    }
    echo $i;
}
```

Obiščite spletno stran php.net in si oglejte sintakso zanke foreach. V čem se ta zanka razlikuje od zanke for?

Stavek break

Break konča izvajanje zanke for, zanke do..while ali stavka switch.

Stavek switch

Stavek switch je podoben seriji stavkov if. V mnogih primerih bi si želeli primerjati vrednost iste spremenljivke in izvajati različne dele kode glede na vrednost te spremenljivke. V ta namen uporabljamo stavek switch. Naslednja primera prikazujeta različna načina za dosego istega rezultata. Prvi z uporabo serije stavkov if, drugi z uporabo stavka switch.

```
if ($i == 0) {
    echo "i je enak 0";
}
if ($i == 1) {
    echo "i je enak 1";
}
if ($i == 2) {
    echo "i je enak 2";
}
```

```
switch ($i) {
    case 0:
        echo "i je enak 0";
        break;
    case 1:
        echo "i je enak 1";
        break;
    case 2:
        echo "i je enak 2";
        break;
}
```

Da se izognemo morebitnim napakam, je treba razumeti delovanje stavka switch. Switch izvede vrstico za vrstico (stavek za stavkom). Na začetku se ne izvede nobena koda, ampak šele, ko se ujame izraz case z vrednostjo, ki je enaka vrednosti switch. PHP nadaljuje izvajanje stavkov do konca bloka switch ali do prvega stavka break. Če ne podamo stavka break na koncu stavkov izraza case, bo PHP nadaljeval izvajanje stavkov naslednjega casea.

```
switch ($i) {
    case 0:
        echo "i je enak 0";
    case 1:
        echo "i je enak 1";
    case 2:
        echo "i je enak 2";
}
```

V tem primeru bi PHP, če je $i enak 0, izvedel vse stavke echo in samo v primeru, da je $i enak 2, bi dobili pričakovan rezultat, ki bi izpisal 'i je enak 2'. Pomembno je, da ne pozabimo stavkov break.
V stavku switch je vrednost izraza preverjena samo enkrat in njegova vrednost se primerja z vsakim izrazom case. V stavku elseif se vrednost pogoja ponovno preveri. Če je pogoj zahtevnejši kot npr. enostavno primerjanje dveh vrednosti, bo stavek switch verjetno hitrejši.

Poseben primer izraza case je izraz default case. Ta izraz prebere vse vrednosti, ki ne veljajo za ostale case izraze in ga moramo praviloma podati kot zadnjega.

```
switch ($i) {
    case 0:
        echo "i je enak 0";
        break;
    case 1:
        echo "i je enak 1";
        break;
    case 2:
        echo "i je enak 2";
        break;
    default:
        echo "i ni enak 0, 1 ali 2";
}
```

Polja

Za spremenljivke smo navedli, da hranijo podatke, ki jih lahko kasneje obdelamo. Njihova slabost je, da lahko naenkrat shranijo samo eno vrednost. Ta problem rešijo polja, ki omogočajo shranjevanje poljubnega števila podatkov. Polje je spremenljivka, ki vsebuje več elementov, ki so indeksirani s številkami ali nizi. Omogoča shranjevanje in obdelavo podatkov, ki so shranjeni pod istim imenom.

Polja imajo več prednosti pred spremenljivkami. Vseeno je, ali shranijo dve ali dva tisoč vrednosti. Do podatkov dostopamo enostavno prek zanke in jih lahko urejamo po imenu ali številki. Vsaka vrednost v polju se nanaša na en element. Do elementa pa imamo lahko dostop prek indeksa. V osnovi je indeksiranje elementov številčno, kar pomeni, da je prvi element na vrednosti indeka 0, drugi na 1 itd.

Definiranje polj

PHP vsebuje veliko različnih funkcij za delo s polji. Osnova je funkcija array(), s katero ustvarimo polje:

```
$sadezi = array("banana", "ananas", "limona");
```

Do podatkov dostopamo:

```
echo $sadezi[2]; //izpise limona
```

PHP v tem primeru avtomatsko povečuje indeks za 1 in ga ni treba vedno znova nastavljati. Če želimo, lahko indeks določimo tudi sami:

```
$sadezi[0] = "banana";
$sadezi[200] = "jabolko";
```

Kako bi izpisali prvo črko spremenljivke `$sadezi[0]`?
Kako bi ugotovili dolžino tega niza?

Asociativna polja

V prejšnjem primeru smo definirali indeksirana polja številčno. Polja, v katerih številke zamenjamo z nizi, imenujemo asociativna polja.

```
$karakter = array (ime=>"Janez", poklic=>"elektro inženir", starost=>30);
```

Do podatkov dostopamo:

```
echo $karakter[starost]; //izpise 30
```

Večdimenzionalna polja

Za večdimenzionalno polje bi lahko rekli, da je polje polj in si ga lahko predstavljamo kot tabelo. Za dostop do večdimenzionalnega polja potrebujemo dva indeksa.

```
$polje[1][2];
```

Definiramo jih na naslednji način:

```
$x = array(
```

```
  array( 'a' => 'a1',
         'b' => 'b1',
         'c' => 'c1'  ),

  array( 'a' => 'a2',
         'b' => 'b2',
         'c' => 'c2'  ),

  array( 'a' => 'a3',
         'b' => 'b3',
         'c' => 'c3'  )
);
echo $x[1][b]; //izpise b2
```

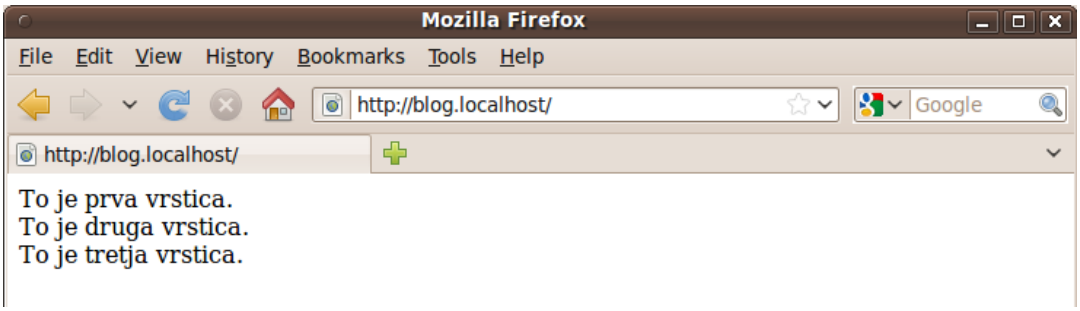
Funkcije

Naloga funkcije je, da prebere vrednosti, jih procesira in izvrši akcijo (npr. izpis v pregledovalnik) ali vrne novo vrednost ali oboje. Pravo vrednost funkcije spoznamo pri obdelavi večjega števila podatkov, ki jih moramo obdelati na enak način. Funkcija je del kode, katere naloga ni nujno takojšnja izvršitev akcije, ampak se lahko aktivira, ko to od nje zahteva skripta. Funkcije so lahko vgrajene ali jih uporabnik napiše sam in za svoje delo zahtevajo podatke, ki jih obdelajo. PHP vsebuje veliko vgrajenih funkcij in preden se lotimo pisanja svoje, je vedno dobro pogledati, ali je morebiti takšna funkcija že napisana.

Definiranje funkcij

Funkcijo definiramo z uporabo izraza (funkcije) function. Če funkcija zahteva argument, na podlagi katerega se izvrši, ga vpišemo v oklepaj in argumente ločimo z vejicami.

```
function neka_funkcija($argument1, $argument2)
{
// koda funkcije
}
```



Slika 21: Funkcija, ki zahteva argument - izpis v novo vrstico

```
//Deklariranje funkcije, ki zahteva argument.
function echoBR($txt){
        echo ("$txt<br />");
}
echoBR("To je prva vrstica.");
echoBR("To je druga vrstica.");
echoBR("To je tretja vrstica.");
```

Najprej smo definirali funkcijo in jo nato uporabili s klicom funkcije. Ta funkcija zahteva argument (tekst), ki se vedno izpiše v novo vrstico. Funkcija se obnaša podobno kot izraz echo(), vendar vsak naslednji tekst zapiše v novo vrstico. S tem lahko prihranimo pisanje kode XHTML za novo vrstico (<br />).

Najprej smo definirali funkcijo in jo nato poklicali. Ta funkcija zahteva argument (tekst), ki se vedno izpiše v novo vrstico. Funkcija se obnaša podobno kot izraz echo(), vendar vsak naslednji tekst zapiše v novo vrstico. S tem lahko prihranimo pisanje kode XHTML za novo vrstico (<br />).

Funkcije, ki vrnejo vrednost

Vrednosti se lahko vrnejo s pomočjo neobveznega stavka return. Vrnejo lahko katerikoli tip spremenljivke. Vrnitev vrednosti konča izvajanje funkcije in skripta, iz katere smo funkcijo klicali, se normalno nadaljuje.

```
//  Funkcija, ki vrne vrednost.
function sestevanje($prvo_stevilo, $drugo_stevilo)
{
$rezultat = $prvo_stevilo + $drugo_stevilo;
return $rezultat;
}
echo sestevanje(3,5); // izpiše "8"
```

Objekti

Objektno programiranje je po svoje zahtevno in nevarno. Spremeni celotno miselnost programiranja. Objekt bi lahko poimenovali kot ograjen skupek spremenljivk in funkcij, ki ga dobimo iz razreda.
Z definiranjem razredov v bistvu določimo skupek lastnosti, ki se lahko prepoznajo kot druge vrednosti. Ustvarimo lahko npr. razred avtomobil, ki vsebuje lastnost barva. Vsi objekti avtomobila imajo to lastnost, vendar lahko zavzamejo različne vrednosti, npr. "modra", "zelena" in tako naprej.

Razred je skupek funkcij (metod) in posebnih spremenljivk (lastnosti). Razredi so predloge, iz katerih ustvarimo objekt.

Največja prednost objektnega programiranja je uporabnost. Objekti so zaokrožene celote in jih brez težav vzamemo iz enega projekta in vstavimo v drugega.

Ustvarjanje objektov

Da ustvarimo objekt, moramo najprej definirati predlogo, ki se imenuje razred:

```
class prvi_razred{
}
```

Razred prvi_razred je osnova, ki ji lahko določimo poljubno število objektov. Objekt razreda prvi_razred ustvarimo na naslednji način:

```
$obj1 = new prvi_razred();
$obj2 = new prvi_razred();
```

S tem smo ustvarili objekta $obj1 in $obj2, oba razreda prvi_razred(). S funkcijo gettype() se lahko prepričamo, da sta $obj1 in $obj2 res objekta:

```
echo $obj1 . " je " . gettype($obj1);
echo $obj2 . " je " . gettype($obj2);
```

Spremenljivke objektov

Objekti imajo dostop do vseh spremenljivk, definiranih znotraj razreda. Te spremenljivke običajno definiramo v zgornjem delu razreda in vsebujejo predpono var:

```
class prvi_razred{
        var $ime = "Simon";
}
```

Vsak objekt, ki bo ustvarjen iz tega razreda, ima dostop do spremenljivke $ime, ki jo lahko tudi spreminjamo:

```
$obj1 = new prvi_razred();
$obj2 = new prvi_razred();

$obj1->ime = "Janez";
echo $obj1->ime . "<br>"; //izpiše Janez
echo $obj2->ime . "<br>"; //izpiše Simon
```

Operator -> omogoča dostop in spreminjanje spremenljivke v objektu.

Metode

Metoda je funkcija, definirana znotraj razreda.

```
//Razred in metoda.
class prvi_razred{
        var $ime;
        function pozdrav(){
                echo "lep pozdrav";
        }
}

$obj1 = new prvi_razred();
$obj1->pozdrav(); //izpiše lep pozdrav
```

Metoda se obnaša kot normalna funkcija in je vedno definirana znotraj razreda. Do nje dostopamo z uporabo operatorja ->. Funkcije, definirane v razredu, imajo dostop do vseh spremenljivk, ki so definirane znotraj razreda, kar prikazuje spodnji primer:

```
// Dostop do spremenljivke, definirane v razredu.
class prvi_razred{
        var $ime = "Simon";
        function pozdrav(){
                echo "Pozdravljeni. Moje ime je $this->ime.<br>";
        }
}
$obj1 = new prvi_razred();
$obj1->pozdrav(); //Pozdravljeni. Moje ime je Simon.
```

S kombinacijo spremenljivke $this in operatorja -> lahko dostopamo do kateregakoli dela oz. metode v razredu, iz razreda. Pomembni deli razredov so tudi konstruktorji. Konstruktor je funkcija, ki nosi isto ime kot razred, v katerem je definirana. Konstruktorji so avtomatsko klicani, ko ustvarimo nov predmet razreda new.

Kako bi izdelali konstruktor razreda prvi_razred, ki bi privzeto ime nastavil "Janez"?

4.2 OBRAZCI

Vsaka naprednejša spletna stran vsebuje obrazce[10]. Le-ti so lahko enostavnejši od obrazcev za kontakt ali prijavo do kompleksnejših z veliko polji in tipi. Pisanje obrazcev je eno izmed zahtevnejših in zamudnejših opravil pri programiranju, saj je potrebno napisati XHTML-obrazec, implementirati validacijska pravila (npr. ali smo vnesli vse podatke, vpisali pravilen elektronski naslov …), jih »sprocesirati« (izpisati na zaslon, v podatkovno bazo …), prikazati napake, zapolniti polja obrazca v primeru napak ipd.

[10] Ang.: forms

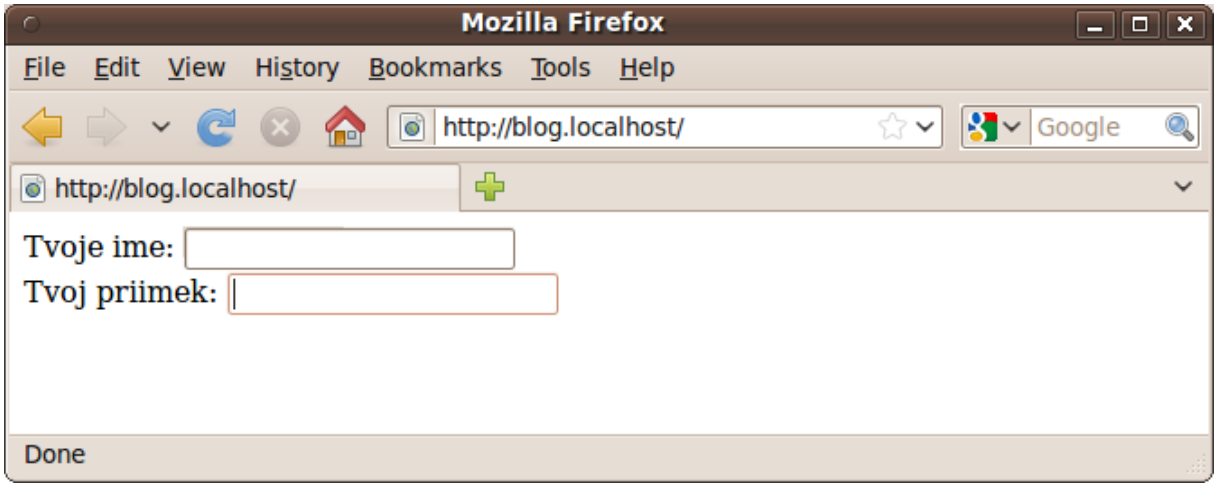
Obrazce prikažemo s standardno XHTML-obliko in značko `<form>`.

```
<form>
        ...
        elementi
        ...
</form>
```

4.2.1 Vnos niza in gesla

Eden izmed najpogosteje uporabljanih elementov obrazca je vnosno polje, ki ga označimo z značko `<input>`. Tip polja določimo s parametrom type. Za vnost teksta – niza uporabimo `type="tekst"`.

```
<form>
Tvoje ime:
<input type="text" name="ime" />
<br />
Tvoj priimek:
<input type="text" name="priimek" />
</form>
```



Slika 22: Vnosni obrazec

Večina brskalnikov prikaže dolžino vnosnega polja velikosti 20. S slogovnimi predlogami ji lahko spremenimo izgled.

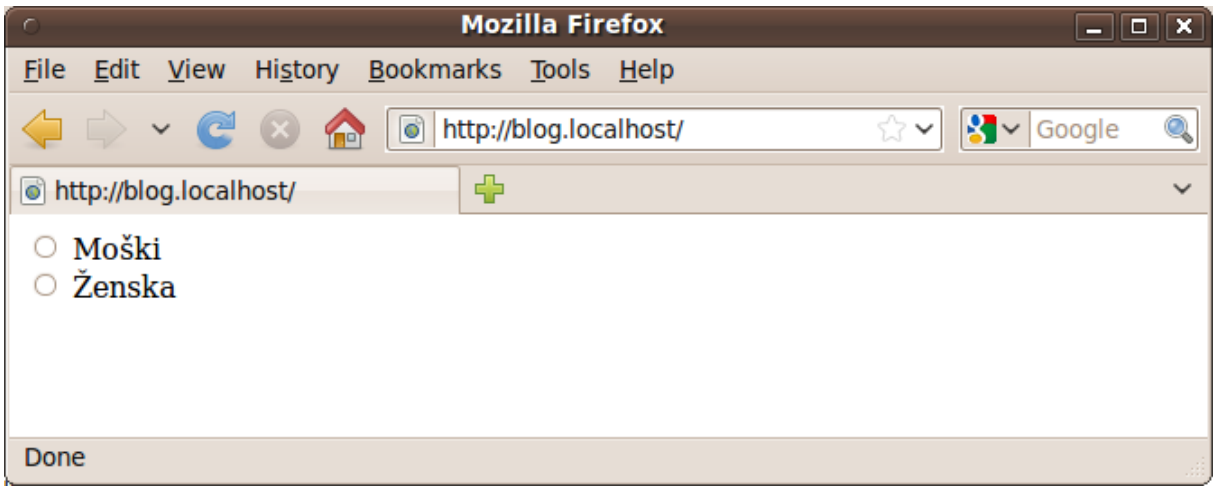
Za vnos gesla in skrivanje znakov pri vpisovanju v vnosnem polju uporabimo type="password".

4.2.2 Radijski gumb[11]

Radijski gumb je možnost, s katerim uporabnik izbere eno od možnosti.

```
<form>
<input type="radio" name="spol" value="moski" /> Moški
<br />
<input type="radio" name="sex" value="zenska" /> Ženska
</form>
```

[11] Ang: radio button

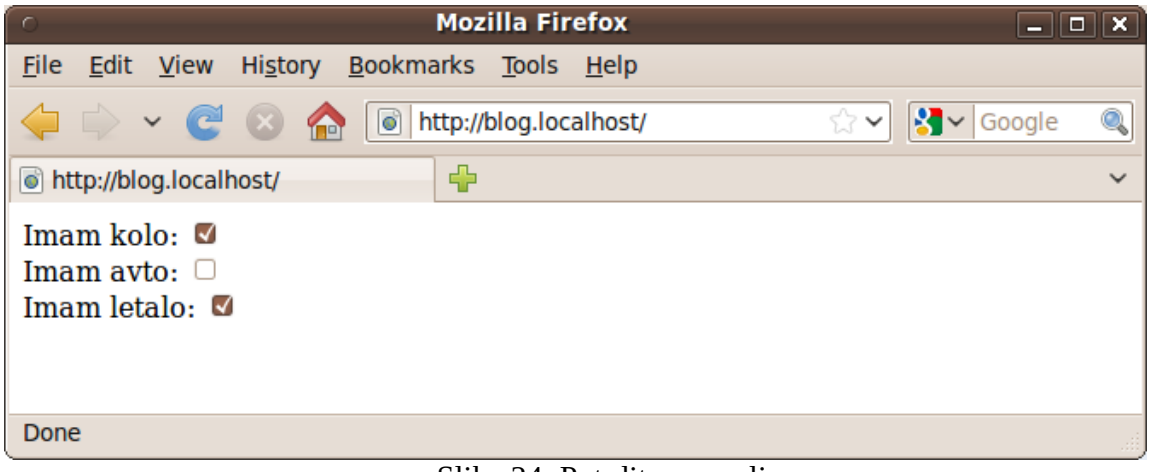


Slika 23: Radijski gumb

4.2.3 Potrditveno polje[12]

Potrditveno polje se uporablja, ko želimo izbrati več možnosti od podanih. Število vseh izbir je vnaprej določeno.

```
<form>
Imamo kolo
<input type="checkbox" name="vozilo" value="Kolo" />
<br />
Imam avto:
<input type="checkbox" name="vozilo" value="Avto" />
<br />
Imam letalo:
<input type="checkbox" name="vozilo" value="Letalo" />
</form>
```



Slika 24: Potrditveno polje

4.2.4 Gumb za predložitev[13]

Gumb za predložitev uporabljamo na koncu obrazca. S klikom na njega se podatki pošljejo do strežnika, ki jih ustrezno obdela. Obrazcu določimo ime uporabnik. Obrazec ima tudi metodo, ki je lahko `get` ali `post`. Post metoda se uporablja, kadar želimo podatke, ki jih imamo v

[12] Ang.: checkbox
[13] Angl.: submit button

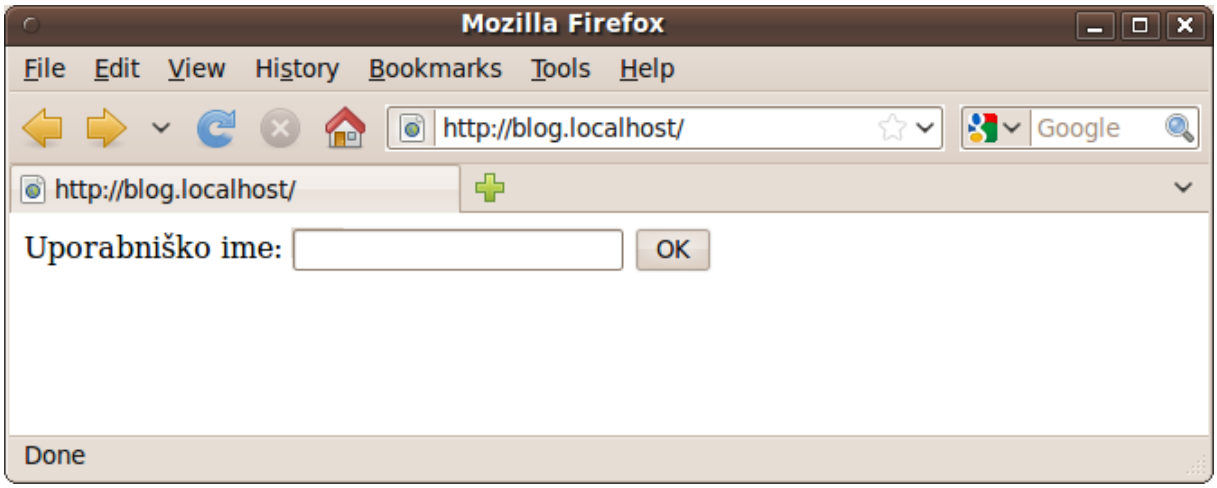
obrazcu obdelati, pri čemer obdelava pomeni, da jih želimo shraniti v podatkovno zbirko, izpisati na zaslon ipd.

Metoda get je pa veliko bolj povezana z URL-naslovom. Le-ta je lahko sestavljen iz imena datoteke kot npr. `preveri.php` in iz atributov, ki sledijo vprašaju (?). Primer: `preveri.php?id=15&title=1`. Z metodo get dobimo podatke iz URL-naslova.

Kot zadnji parameter pri obrazcu je action, ki pove, kaj se naj zgodi, ko kliknemo na gumb OK. V našem primeru se izvrši datoteka preveri.php.

Več informacij o razlikah med get in post je zbranih na spletni strani http://www.cs.tut.fi/~jkorpela/forms/methods.html.

```
<form name="uporabnik" method="post" action="preveri.php">
        Uporabniško ime:
        <input type="text" name="up_ime" />
        <input type="submit" value="OK" />
</form>
```



Slika 25: Obrazec za vnos uporabnika

Rešitev naloge:

Dana naloga iz uvoda je zahtevala, da v že izdelano nalogo index.html dodamo obrazec, preko katerega bomo sami vnesli besedilo, ki se izpiše na zaslon. To lahko storimo z uporabo spletnih obrazcev.

Ker z obrazcem izdelujemo dinamično spletno stran, ki bo kot osnovo uporabljala programski jezik PHP, moramo najprej izvorno datoteko `index.html` preimenovati v `index.php`. V nadaljevanju iz datoteke izbrišemo že napisano besedilo, ki sledi sliki. V dokumentu pa pustimo povezavo na datoteko firefox3.html. Namesto besedila vključimo XHTML-obrazec:

```
<form name="form" method="POST" action="<?php echo $PHP_SELF ?>">
Besedilo:<br>
<textarea name="besedilo" cols=40 rows=6></textarea><br>
<input type="submit" name="OK" value="OK" />
</form>
```

Obrazcu smo dali ime form, za metodo smo uporabili POST, medtem ko je action nekoliko drugačen, kot smo ga spoznali v poglavju 4.2.4. Uporabili smo `$PHP_SELF`, kar pomeni, da se s

klikom na gumb ne bomo preusmerili na drugo datoteko, ampak bomo vse nadaljnje korake izvajali v dokumentu `index.php`. To je v našem primeru tudi smiselno, saj mora besedilo biti napisano na isti strani, kot je obrazec.

Element za vnos besedila se imenuje `<textarea>`, ki mu moramo dati ime, običajno pa mu določimo tudi širino in višino. Oboje lahko oblikujemo tudi preko slogovnih predlog. Pomembno je ime obrazca, saj bomo preko imena dostopali do vsebine polja.

Gumbu smo dali ime OK. Enako ime smo uporabili tudi za napis na gumb.

Slika 26: Vneseno besedilo, prikazano na strani

V naslednjem koraku je potrebno napisati še programsko kodo, ki iz obrazca dobi besedilo in ga izpiše na zaslon. To storimo s kodo, ki je zapisana v nadaljevanju in jo vstavimo neposredno pred obrazec.

```
<?php
if (isset($_POST['OK']))
{
    if (($_POST['besedilo'])=="")
    {
        echo "Napaka. Niste vpisali besedila.";
    }
    else
    {
        echo nl2br($_POST['besedilo']);
    }
}
?>
```

Z if stavkom najprej preverimo, ali smo pritisnili gumb z imenom OK. Če gumba nismo pritisnili, se ta koda na izvede, kar je tudi logično pravilno. Pri vsakem obrazcu se soočamo tudi z obravnavo napak. Napake so lahko različne, med drugim je lahko napaka tudi prazno polje, zato z naslednjim if stavkom preverimo, ali je vnesen kakšen tekst. Zgodi se namreč lahko, da uporabnik pritisne gumb OK, ne da bi karkoli vpisal. V tem primeru se na zaslon izpiše opozorilo o manjkajočem tekstu.

Če je zadoščeno vsem pogojem, potem besedilo izpišemo na zaslon z echo `$_POST[besedilo]`. Vpisan tekst smo dali v funkcijo `nl2br()`, ki dan niz pretvori v tako obliko, da upošteva tudi nove vrstice oz. znak enter. Brez te funkcije bi se besedilo izpisalo v eni vrstici.

Oglejte si spletno stran www.php.net in poiščite razlago funkcije nl2br().

Celotna izvorna koda datoteke `index.php` je prikazana v nadaljevanju.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
  <meta content="text/html; charset=ISO-8859-2" http-equiv="Content-Type">
  <title>Mozilla Firefox</title>
  <link rel="stylesheet" type="text/css" href="predloga.css" />
</head>
<body>
<h1>Mozilla Firefox</h1>
<img style="width: 463px; height: 220px;" alt="Firebird in Firefox" src="Phoenix_-_Firefox.png"><br>
<?php
if (isset($_POST['OK']))
{
    if (($_POST['besedilo'])=="")
    {
        echo "Napaka. Niste vpisali besedila.";
    }
    else
    {
        echo nl2br($_POST['besedilo']);
    }
}
?>
<br>
 <form name="form" method="POST" action="<?php echo $PHP_SELF ?>">
Besedilo:<br>
<textarea name="besedilo" cols=40 rows=6></textarea><br>
<input type="submit" name="OK" value="OK" />
</form>
<br>
Za več informacij o različici 3 sledite <span class="povezava"><a
href="firefox3.html">povezavi</a></span>.<br>
<br>
</body>
</html>
```

Povzetek:

Programski jezik PHP v kombinaciji z obrazci omogoča interakcijo med uporabnikom in računalnikom. Pri pisanju PHP-skript lahko uporabimo razvojno okolje NetBeans, datoteke pa

shranjujemo s končnico php. PHP kodo lahko integriramo v XHTML-kodo z uporabo značk `<?php ?>` oz. krajše `<? ?>`.

PHP-jezik izhaja iz C-ja, zato je temu jeziku zelo podoben. Za programerje, ki poznajo sintakso sorodnih jezikov, prehod na PHP ne predstavlja večjih ovir. Jezik vsebuje praktično vse kontrolne strukture kot ostali jeziki, vsebuje pa tudi zanko foreach, ki je v C-ju ne poznamo. Enostavna deklaracija polj in neobvezno deklariranje podatkovnih tipov spremenljivk ga uvrščata med enostavnejše programske jezike. Celotna dokumentacija s primeri programskega jezika PHP je dostopna na spletni strani www.php.net (10. 7. 2010).

PHP- jezik omogoča enostavno delo z obrazci. Metodi GET in POST uporabljamo za posredovanje vrednosti obrazca.

Naloge:

1. Izdelajte spletno stran z imenom anketa.php in sestavite obrazec, ki naj vsebuje:
   a) vnosno polje za ime in priimek;
   b) vnosno polje za kraj šolanja (privzeta vrednost je Murska Sobota);
   c) padajoči seznam (http://www.w3schools.com/html/html_forms.asp) za izbor vrste študija (redni ali izredni);
   d) izbirno polje za razloge obiskovanja vaje (sami podajte možne odgovore – najmanj 3);
   e) vnosno polje za komentar, ki naj vsebuje že neko besedilo;
   f) gumb Počisti, ki počisti vsebino obrazca;
   g) gumb Pošlji, ki obdela vse podatke in obravnava morebitne napake ter izpiše vnesene podatke na zaslon.

2. Kakšna je razlika med gettype() in settype()?

3. Kakšna je razlika med metodo in funkcijo?

4. Funkcijam, ki se nahajajo znotraj razreda, imenujemo _____.

5. Napišite del XHTML-kode, ki ustvari gumb z napison "Potrdi", katerega ime je OK.

5 SKRIPTNI PROGRAMSKI JEZIK JAVASCRIPT

PHP je jezik, ki za svoje delovanje potrebuje spletni strežnik. To z drugimi besedami pomeni, da se za vsako spremembo na spletni strani mora le-ta osvežiti. Včasih tega ne želimo in takrat lahko uporabimo JavaScript oz. njegovo knjižnico s funkcijami, ki se imenuje jQuery. Z JavaScriptom lahko izdelamo animacijo, galerijo slik ipd. Ali veste, da JavaScript nima nič skupnega z Javo? Razen podobnega imena seveda. Kako je nastal, kje ga uporabljamo in kakšna je njegova sintaksa?

JavaScript je skriptni programski jezik, ki ga je leta 1995 razvilo podjetje Netscape. Sprva se je imenoval LiveScript, iz tržnih razlogov pa so ga kmalu preimenovali v JavaScript. Podporo temu jeziku je prvi dal brskalnik Netscape Navigator, kasneje pa so temu množično sledili tudi drugi brskalniki. Kot odgovor je Microsoft ponudil svoj programski jezik, ki ga je poimenoval VBScript, vendar je bil le-ta omejen na operacijski sistem Windows. Kasneje je tudi Microsoft začel v svoje brskalnike vključevati podporo JavaScriptu, ki je postajal vse bolj pomemben programski jezik na področju svetovnega spleta. Za razliko od programskega jezika PHP, ki za svoje izvajanje potrebuje spletni strežnik, se JavaScript izvaja na uporabnikovem računalniku, kar prinaša mnoge prednosti pri izdelavi spletnih strani. Tako npr. spletne strani ni potrebno ponovno osveževati, če želimo spremeniti barvo ozadja, spremeniti temo, velikost pisave ipd. JavaScript lahko uporabljamo brez, da bi kupili licenco, kar pomen, da je brezplačen. Pomembno je vedeti tudi to, da sta Java in JavaScript popolnoma različna jezika. Java je zmogljivejši in kompleksnejši jezik, ki ga uvrščamo v kategorijo jezikov kot sta C in C++.

Naloga:

Želimo izdelati spletno stran na kateri bi uporabnikom ponudili spreminjanje barve ozadja (background). Stran naj vsebuje dve povezavi in sicer naj ena vodi do modrega ozadja, druga pa do sivega ozadja. Ozadje se naj spremeni s klikom na povezavi.

JavaScript je jezik, ki dopolnjuje obstoječe jezike. V kombinaciji s PHP-jem se izkaže za zelo učinkovito orodje ravno zaradi osveževanja spletne strani. Dogodki se namreč izvajajo na uporabnikovem računalniku in ne na strežniku. To ima za posledico vidnost izvorne kode, ki v tem primeru, za razliko od PHP-ja, ni skrita očem uporabnikom.

5.1 ZNAČILNOSTI JAVASCRIPT-A

JavaScript je bil razvit z namenom, da doda interaktivnost statičnim xhtml spletnim stranem. Je skriptni jezik in je enostaven za razumevanje in uporabo, vsebuje pa večino krmilnih stavkov, kot ostali programski jeziki. Z značkama `<script>` in `</script>`, ga vključujemo v XHTML kodo:

```
<body>

<script type="text/javascript">
        document.write("Pozdravljen svet!");
</script>

</body>
```

Prikazana programska koda na zaslon izpiše "Pozdravljen svet" (brez narekovajev), kar je enako kot spodnja xhtml koda.

```
<body>

Pozdravljen svet!

</body>
```

Ukaz `document.write()`, služi izpisu na zaslon oz. brskalnik, znački `<script>` in `</script>` pa brskalniku narekujeta, da se med njima nahaja JavaScript koda.

Brskalniki, ki ne podpirajo JavaScript-a bodo le to prikazali kot xhtml vsebino, kar pomeni, da se ta koda izpiše na zaslon. V tem primeru je smiselno vstaviti xhtml komentar pred prvim in za zadnjim JavaScript stavkom, kot sledi:

```
<html>
<body>
<script type="text/javascript">
<!--
document.write("Pozdravljen svet!");
//-->
</script>
</body>
</html>
```

JavaScript koda se bo v tem primeru prikazala pravilno, skrita pa bo brskalnikom, ki je ne podpirajo.

5.1.1 Kdaj in kje se uporablja JavaScript?

Področje uporabe JavaScript-a je zelo široko:

- Z JavaScriptom vnesemo dinamičnot statičnemu jeziku xhtml. Preprosti ukazi in krmilne strukture dajejo xhtml razvijalcem možnost naprednejših funkcij.

- Z JavaScriptom lahko vstavimo dinačen tekst v xhtml. Z ukazom `document.write("<h1>" + ime + "</h1>")` lahko na zaslon izpišemo vsebino spremenljivke ime, ki jo oblikujemo z naslovom.

- JavaScript "pozna" dogodke. Uporabimo ga lahko pri izvajanju različnih dogodkov, kot npr. klik z miško, pomik z miško, zaznavanje pritisnjene tipke ipd.

- Z JavaScriptom lahko beremo in spreminjamo vsebino xhtml dokumentov.

- Z njim lahko validiramo podatke iz obrazcev, še predem jih pošljemo na strežnik. Validiramo lahko npr. podatke kot so pravilen elektronski naslov, enakost vpisanih gesel, prazna polja ipd. S tem zmanjšamo čas procesiranja spletne strani in le-ta postane hitrejša.

- Z JavaScriptom lahko zaznamo vrsto brskalnika, ki ga uporablja uporabnik. S tem lahko prikažemo vsebine, ki so ovisne od brskalnikov. To nam lahko pomaga pri prikazu iste vsebine, različno oblikovane na tabličnih računalnikih, telefonih prenosnikih ipd.

- Z njim lahko med drugim ustvarjamo in pridobivamo podatke iz piškotkov, kar omogoča personalizacijo spletne strani.

5.1.2 Kje vključujemo JavaScript?

JavaScript v področju <head>

V ta del spletne strani je smiselno vključevati funkcije, tj. kodo za katero ne želimo, da bi "pomešala" z ostalim delom. Funkcije lahko nato pokličemo kadarkoli, njen klic pa je enak kot pri večini ostalih programskih jezikov.

```
<html>
<head>
<script type="text/javascript">
function message()
{
alert("Pozdravljen svet!");
}
</script>
</head>

<body onload="message()">
</body>
</html>
```

Raziščite kaj omogoča funkcija alert()? Kdaj jo uporabljamo?

JavaScript v področju <body>

Ta način uporabljamo takrat kadar ne želimo vsebine dati v funkcijo oz. takrat kadar želimo, da se njena vsebina izpiše na tistem delu v dokumentu, kjer je tudi deklarirana.

```
<html>
<head>
</head>

<body>
<script type="text/javascript">
document.write("Pozdravljen svet!");
</script>
</body>

</html>
```

JavaScript v <head> in <body>

Uporabimo lahko kombinacijo prejšnjih vključevanj pri čemer funkcije pišemo v `<head>`, ostalo kodo pa v `<body>`.

JavaScript v zunanji datoteki

Zunanje datoteke uporabljamo takrat kadar želimo vsebino JavaScript-a uporabljati na različnih straneh spletne strani. Da ne podvajamo kode jo je smiselno zapisati v datoteko, ki ima končnico .js. Zunanja datoteka ne sme vsebovati značk `<script>` in `</script>`.

```
<html>
<head>
<script type="text/javascript" src="datoteka.js"></script>
</head>
<body>
</body>
</html>
```

5.2 DOGODKI

Funkciji `alert()` in `document.write()`, ki uporabljeni v primerih, opisanih na prejšnjih straneh se izvedeta oz. tolmačita takoj. JavaScript omogoča še tolmačenje programske kode ob dogodku, ki je med drugim lahko klik z miško, pritisk tipke, sprememba vsebine ipd. Dogodke običajno uporabljamo v povezavi s funkcijami, ki se ne izvedejo dokler se ne zgodi dogodek.

```
<img src="slika.jpg" alt="Slika" onclick="MojaFunkcija()" />
```

Dogodke vključujemo v elemente kot je prikazano na primeru. Običajno je to na koncu značke. V tem primeru se s klikom na sliko izvede funkcija MojaFunkcija(), ki je predhodno deklarirana v glavi dokumenta ali zunanji datoteki.

Vsi dogodki:

onload
brskalnik je do konca naložil dokument ali vse okvirje (body in frameset)
onunload
brskalnik je odstranil dokument iz okna ali okvirja (body in frameset)
onclick
uporabnik je kliknil na element
ondblclick
uporabnik je napravil dvojni klik na elementu
onmousedown
uporabnik je pritisnil miškin gumb na elementu
onmouseup
uporabnik je spustil miškin gumb na elementu
onmouseover
uporabnik je pomaknil miško na element
onmousemove
uporabnik je premaknil miško na elementu
onmouseout
uporabnik je pomaknil miško z elementa

onfocus
uporabnik je aktiviral element z miško ali tipko TAB
(a, area, label, input, select, textarea in button)
onblur
uporabnik je deaktiviral element z miško ali tipko TAB
(a, area, label, input, select, textarea in button)
onkeypress
uporabnik je pritisnil in spustil tipko
onkeydown
uporabnik je pritisnil tipko
onkeyup
uporabnik je spustil tipko
onsubmit
uporabnik je v obrazcu pritisnil gumb submit (form)
onreset
uporabnik je v obrazcu pritisnil gumb reset (form)
onselect
uporabnik je označil besedilo v polju za vnos besedila (input in textarea)
onchange
uporabnik je spremenil vrednost elementa in ga deaktiviral (input, select in textarea)

5.3 FUNKCIJE

V JavaScript je zelo pogosta uporaba funkcij. Deklarirane so lahko v `<head>`, `<body>` ali zunanji datoteki. V izogib nevšečnostim pri deklraciji in uporabi priporočamo, da funkcije deklariramo v zunanji datoteki ali `<head>`.

Splošna sintaksa funkcije:

```
function ImeFunkcije(var1,var2,...,varX)
{
//programska koda
}
```

Glava funkcije je sestavljena iz imena function, ki pove, da gre za funkcijo, imena funkcije, ki je lahko poljubno, vendar v skladu s sintakso ter argumenti. Funkcija lahko sprejme poljubno argumentov, ki so ločeni z vejicami. Funkcija lahko vrednost tudi vrne, za kar uporabimo ukaz return.

5.4 KNJIŽNICA JQUERY

jQuery je brezplačna knjižnica z JavaScript funkcijami. Dosegljiva je uradni spletni strani jQuery-ja, www.jquery.com (10.7.2010). Omogoča izbiro xhtml elementov in manipulacijo z njimi, obravnavo CSS stilov, xhtml dogodke, JavaScript efekte in animacije, Ajax …

Knjižnica z jQuery funkcijami je zbrana v eni sami datoteki, ki jo lahko brezplačno snamemo z njihove spletne strani in vključimo v dokument na sledeč način:

```
<head>
<script type="text/javascript" src="jquery.js"></script>
</head>
```

Primer skrivanja odstavkov

Spodnji primer prikazuje način kako lahko skrijemo vsebino, ki se nahaja v odstavku (značka `<p>`).

```
<html>
<head>
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript">
$(document).ready(function(){
  $("button").click(function(){
    $("p").hide();
  });
});
</script>
</head>

<body>
<h2>To je naslov</h2>
<p>To je odstavek.</p>
<p>Še en odstavek.</p>
<button>Klikni</button>
</body>
</html>
```

jQuery vključujemo na enak način kot JavaScript. Priporočljivo je, da ga vključimo v `<head>` ali zunanjo datoteko. V danem primeru imamo dva odstavka, en naslov in en gumb. Želimo, da se s klikom na gumb skrijeta oba odstavka.

Če želimo, da se jQuery izvede, ga moramo dati v `$(document).ready(function(){});`, s čimer povemo, da bomo vse to izvajali takrat, ko se naloži celoten dokument.

```
$(document).ready(function(){

});
```

Da se določena akcija izvede ob nekem dogodku poskrbi ukaz

```
$("button").click(function(){

});
```

Z njim sporočimo, da se naj izvede koda, ko pritisnemo na značno `<button>`. Znotraj te kode pa povemo, da se naj vse značke `<p>` skrijejo,kar storimo z ukazom `$("p").hide();`.

Namesto `$("p")` lahko uporabimo katerokoli značko, razred, id in podobno. Prav tako obstaja množica efektov, selektorjev, dogodkov ipd, ki so zbrani in opisani na spletni strani jQuery-ja (http://docs.jquery.com/Main_Page) (10.7.2010).

Raziščite ukaz `show()`, ki je nasprotje ukaza `hide()`. Ustvarite še en gumb in s klikom na njega omogočite, da odstavka p znova prikažeta.

Rešitev naloge:

Naloga je zahtevala, da izdelamo spletno stran z dvema povezavama, ki naj omogočata da s klikom na njiju spremenimo barvo ozadja. Prva povezava naj spremeni barvo ozadja na modro, druga pa na sivo.

Najprej ustvarimo xhtml dokument, ki vsebuje dve povezavi, kar storimo s kodo:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  <title>Sprememba ozadja</title>
</head>
<body>
<a href="#">Modro ozadje</a>
<a href="#">Sivo ozadje</a>
</body>
</html>
```

Priporočljivo je, da nato ustvarimo funkcijo `SpremeniBarvo(barva)`, ki bo sprejela tip barve in jo nastavila za ozadje. Funkcijo deklariramo v področju <head> in sicer:

```
function SpremeniBarvo(barva)
{
        document.bgColor = color;
}
```

Za spreminjanje barve ozadja se uporablja document.bgColor.

Funkcijo je nato potrebno le še vključiti v povezavo, kot dogodek na klik. Za to bomo uporabili onclick() iz česar sledi, da povezava izgleda kot je prikazano:

```
<a href="#" onClick="SpremeniBarvo('blue')">Modro ozadje</a>
```

Celotna izvorna koda:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  <title>Sprememba ozadja</title>
  <script language="JavaScript">
        function SpremeniBarvo(barva){
                document.bgColor = barva;
        }
  </script>
</head>
<body>
<a href="#" onClick="SpremeniBarvo('blue')">Modro ozadje</a>
<a href="#" onClick="SpremeniBarvo('gray')">Sivo ozadje</a>
</body>
</html>
```

Povzetek:

JavaScript daje dinamičnost spletnim stranem in oblikovalcem omogoča, da v xhtml dokument vnesejo interaktivnost kot so dogodki ipd. Dogodki so lahko različno kot npr. klik miške, pomik miše, pritisk tipke … JavaScript lahko vključimo v `<head>`, `<body>`, oboje ali v zunanjo datoteko. V `<head>` in `<body>` pišemo programsko kodo tako, da jo damo med znački `<script>` in `</script>`. Ti dve znački pa ne uporabljamo pri zunanjih datotekah. Priporočljivo je, da pri delu uporabljamo funkcije, ki jih deklariramo z besedo function. JavaScript Vsebuje tudi krmilne stavke kot so if, while, do while, for, break ipd.

Množica že napisanih JavaScript funkcij je zbranih v datoteki jQuery. Te funkcije omogočajo različne animacije, dogodke, izbire, spremembe CSS-jev ipd.

Naloge:

1. Izdelajte spletno stran v katero vstavite dve sliki. Omogočite, da ko se spremeni barva ozadja ko se z miško pomaknemo prek ene sli druge slike. Prva slika naj spremeni barvo ozadja na rumeno, druga pa na rdečo.

2. Ka jomogoča dogodek onblur()? Kje bi ga uporabili?

3. JavaScript ukaz za zpis na zaslon se imenuje _____?

4. Znotraj katere značke vstavljamo JavaScript kodo?

5. Kako z JavaScript kodo izpišemo "Hello world!" (izpis je brez narekovajev) v novem oknu?

6. Katera jQuery metoda se uporablja za skrivanje izbranih elementov?

7. Z jQuery napišemo sledeč selektor: `$("div.intro")`. Kaj izbere?
   - vse div-e z razredom intro
   - vse div-e z id-jem intro
   - prvi div z razredom intro
   - prvi div z id-jem intro

6 PROGRAMSKO OGRODJE SYMFONY

Izdelava spletnih strani s klasičnim postopkom programiranja je zamudna. Zahteva veliko časa, saj lahko ena srednje zahtevna spletna stran razvijalcu vzame več mesecev prgramiranja in oblikovanja. Za odpravo tega obstaja mnogo programskih orodij, kot so npr. Zend, CakePHP in Symfony, ki vsebujejo že vnaprej napisane razrede. Razvijalec nato le ustvarja objekte in z metodami razredov ustvarja obliko. Na koncu je lahko prihranek časa zelo velik in ker je čas denar, se to lahko pozna tudi na debelini denarnice. Predhodno znanje jezika PHP, razredov in strukture Symfony-ja je ključnega pomena.Vsebuje tudi opisni jezik YAML, ki skrajša čas izdelave in posodabljanje podatkov v zbirki podatkov.

Symfony je spletno programsko ogrodje[14]*. Ogrodje je brezplačno in je izdano pod licenco MIT. Razvit je bil z namenom, da lahko uporabniki na hiter in enostaven način, oblikujejo in vzdržujejo spletne aplikacije brez odvečnega programiranja oz. kodiranja. Namestimo ga lahko na več platform: Unix, Linux, Mac OS in Windows. Za namestitev potrebujemo tudi spletni strežnik in programski jezik PHP.*

Trenutna verzija, 1.4.8, podpira objektno relacijsko preslikavo[15] *Propel in Doctrine. ORM je tehnika, ki omogoča dostop do podatkov podatkovne zbirke prek objektnega modela in ne preko standardnih SQL-stavkov.*

Symfony je v celoti napisan v programskem jeziku PHP 5 in ima podporo za večino podatkovnih zbirk, kot npr. MySQL, PostgreSQL, Oracle in Microsoft SQL Server.

Slika 27: Logotip Symfony
Vir: http://www.symfony-project.org/

Naloga:

S pomočjo orodja Symfony izdelajte spletno stran z imenom Blog. Blog naj vsebuje dodajanje sporočil, pri čemer želimo vnesti naslov sporočila, sliko, vsebino sporočila, ime uporabnika, ki je napisal sporočilo ter podatek, kdaj je bilo sporočilo vneseno in kdaj popravljeno. Želimo uporabiti PostgreSQL zbirko podatkov, spletna stran pa je dostopna na naslovu http://blog.localhost.

Programiranje obrazcev je zelo zamudno opravilo, sploh če se le-ti začnejo ponavljati. To ima za posledico časovno dolg razvoj spletnih strani in podvajanje kode. Da bi rešili ta problem, se za večje projekte uporabljajo programska ogrodja, ki v marsičem olajšajo omenjeni problem. Symfony s svojo arhitekturo ločuje izgled strani, podatke in logiko. Spoznajmo Symfony in njegovo logiko delovanja.

[14] angl. framework
[15] angl. Object-relational mappings (ORM)

6.1 ZNAČILNOSTI PROGRAMSKEGA OGRODJA SYMFONY

Poleg tega, da je brezplačen, podpira veliko število sistemov za upravljanje podatkovnih zbirk in je neodvisen od operacijskega sistema, ima Symfony še nekatere značilnosti:

- Enostavna namestitev in konfiguracija.
- V večini primerov je enostaven za uporabo in dovolj fleksibilen za zahtevnejše primere.
- Ogrodje je skladno s pravili programiranja in standardi.
- Koda je čitljiva in komentirana, kar omogoča lažje vzdrževanje.
- Enostaven za razširitev in nadgradnjo.

Poleg naštetega razvijalec z namestitvijo ogrodja dobi še:

- vgrajeno jezikovno podporo (lokalizacija vsebine);
- podporo za predloge, ki jih lahko spreminjamo brez posebnega znanja o delovanju ogordja in programiranju;
- podporo za Ajax in različne JavaScript efekte;
- možnost dodajanja vtičnikov;
- vgrajeno podporo za elektronsko pošto;
- podporo za usmerjanje URL-naslovov v uporabniku prijazne naslove;
- avtomatsko generiranje orodij za administracijo spletne vsebine
- itd.

6.2 JE SYMFONY PRIMEREN ZAME?

Symfony je uporabniku prijazno ogrodje in je namenjeno tako poznavalcem programiranja spletnih strani kot tudi začetnikom. Glavna odločitev za uporabo je odvisna od velikosti in obsežnosti projekta.

Če želimo razvijati enostavno spletno stran z npr. petimi do desetimi stranmi, omejenim dostopom do podatkovne zbirke in brez posebne dokumentacije, se je verjetno primerneje lotiti lastnega razvoja in programiranja.

Na drugi strani pa z razvojem kompleksih spletnih aplikacij, s težjo "logiko", PHP verjetno ne bo dovolj. Če nameravamo spletno stran vzdrževati in jo v prihodnosti razširiti, potrebujemo za to enostavno, berljivo in efektivno kodo. Če bomo npr. uporabljali napredno interakcijo spletnega vmesnika z uporabnikom (Ajax in podobno), je zelo težko in zamudno pisati nekaj sto ali tisoč vrstic kode. V vseh teh primerih se, bolj kot klasičnega programiranja, lotimo uporabe programskega ogrodja Symfony.

6.3 OSNOVNI KONCEPTI

Preden začnemo s Symfonyjem poglejmo nekaj osnovnih konceptov in lastnosti.

PHP 5

Symfony je bil razvit v PHP-ju verzije 5, zato je namenjen ustvarjanju spletnih strani v tem jeziku. Razumevanje programskega jezika PHP je zelo pomembno, če želimo iz ogrodja

izvleči čimveč. Razvijalci z znanjem jezika PHP 4 se morajo predvsem osredotočiti na objektno usmerjen pristop izdelave spletnih strani.

Objektno usmerjeno programiranje

Objektno usmerjeno programiranje (OOP) smo na kratko opisali v poglavju o PHP-ju. Ker Symfony omogoča široko uporabo objektno usmerjenih pristopov, je OOP predpogoj za učenje Symfonyja. Za referenčno knjigo o OOP-ju v PHP-ju je najbolje poseči po uradni dokumentaciji programskega jezika PHP na spletni strani:

http://www.php.net/manual/en/language.oop5.basic.php (10. 7. 2010).

Magic metode

Eno od zmogljivejših orodij PHP-ja 5 so Magic metode[16]. Te metode se uporabljajo za preglasitev privzetega obnašanja razredov brez spreminjanja kode. Z njimi postane sintaksa PHP-jezika manj kompleksna in bolj razširljiva. Spoznamo jih po tem, da se te metode pričnejo z dvojnim podčrtajem (__).

Če želimo npr. izpisati podatke nekega objekta, PHP najprej pogleda metodo __toString(). Format izpisa lahko seveda poljubno spremenimo.

```
$myObject = new myClass();
echo $myObject;
// Magic metoda
echo $myObject->__toString();
```

Symfony uporablja Magic metode, zato je potrebno razumevanje le-teh. Podrobneje je njihova raba opisana v uradni dokumentaciji PHP-ja:

http://www.php.net/manual/en/language.oop5.magic.php (10. 7. 2010).

PHP Extension and Application Repository (PEAR)

PEAR je ogrodje in distribucijski sistem za ponovno uporabljene PHP-komponente. PEAR omogoča prenos, namestitev, nadgradnjo in odstranitev PHP-skript. Pri uporabi PEARa ni potrebno skrbeti, kam se shranijo skripte, kako jih naredimo vidne in podobno.

PEAR je med drugim najbolj zanesljiva pot namestitve različnih knjižnic za PHP. PEAR ni ključen za razumevanje Symfonyja, vedeti je potrebno le, ali ga imamo nameščenega in čemu služi. Z ukazom v komandnem oknu (konzoli) lahko preverimo, ali je PEAR nameščen.

```
pear info pear
```

Ukaz vrne in izpiše različico PEARa, ki je nameščen na računalnik.

[16] angl. Magic Methods

Objektno relacijska preslikava (ORM)

Podatkovne zbirke uporabljajo relacije, vendar ker sta PHP 5 in Symfony objektno usmerjena, je potrebno do podatkovnih zbirk dostopati na objektni način. Za ta način potrebujemo vmesnik, ki bo objektno logiko "prevedel" v relacijsko logiko. Ta vmesnik se imenuje objektno relacijska preslikava (angl. object-relational mapping) oz. s kratico ORM.

Ena izmed prednosti uporabe objektnega modela dostopa do podatkovne zbirke je neodvisnost od podatkovne baze. Vmesnik avtomatično prevede objektni model v SQL poizvedbe, ki so značilne za posamezno podatkovno zbirko. To za sabo prinese enostaven prehod na drug sistem za upravljanje zbirk podatkov, eventuelno tudi sredi projekta. Predstavljajmo si, da razvijamo spletno aplikacijo in se naročnik še ni odločil, kateri sistem za upravljanje zbirk podatkov bo uporabil. Začnemo lahko npr. z SQLite, vmes uporabimo MySQL, nato PostgreSQL in tako dalje. To dosežemo s spremembo samo ene vrstice v konfiguracijski datoteki.

Uporaba objektov namesto zapisov in razredov namesto tabel ima še eno prednost. Npr. če imamo tabelo Client z dvema poljema FirstName in LastName. Če želimo iz zbirke dobiti ime in priimek, enostavno ustvarimo novo metodo, kot sledi:

```
public function getName()
{
        return $this->getFirstName().' '.$this->getLastName();
}
```

Do podatkov, ki jih želimo dobiti iz zbirke, lahko dostopamo na enostaven način z uporabo metod. Če imamo npr. spletno trgovino in razred ShoppingCart, v katerem hranimo izdelke, ki jih želi uporabnik kupiti, potem dobimo celotno vsebino košarice pred nakupom z metodo getTotal():

```
public function getTotal()
{
  $total = 0;
  foreach ($this->getItems() as $item)
  {
    $total += $item->getPrice() * $item->getQuantity();
  }
  return $total;
}
```

Odprtokodni projekt Propel (http://propel.phpdb.org/trac/) je trenutno eden izmed najbolj zmogljivih objektno relacijskih vmesnikov za PHP 5. Symfony ima vgrajen tako Doctrine kot Propel vmesnik. Izbiro le-tega lahko nastavimo v posebni konfiguracijski datoteki ogrodja.

Kakšna je razlika med objektno relacijsko preslikavo Propel in Doctrine? Oglejte si spletni strani in razmislite o razlikah.

Hiter razvoj aplikacij[17] (RAD)

Programiranje spletnih aplikacij je že dolgo časa mučno in počasno delo. Sledenje ciklom razvoja programske opreme pri izdelavi spletnih aplikacij prinese več nevšečnosti kot koristi. Razvoj se praktično ne more pričeti, dokler niso napisane vse zahteve, narisani UML[18] diagrami in napisana dokumentacija projekta. To vpliva na hitrost razvoja spletne aplikacije in

[17] angl. Rapid application development
[18] Unified Modeling Language

naročniki velikokrat niso pogosto spreminjali svojih zahtev, kot to počno danes, ko se njihove želje vsakodnevno spreminjajo in dopolnjujejo v samem razvoju programske opreme.V prvi vrsti naročniki pričakujejo, da razvojna ekipa sprejme njihove zahteve in spremeni strukturo aplikacije kar se da hitro. Na srečo uporaba skriptnega jezika, kot sta Perl in PHP, omogoča enostavno uporabo druge programske strategije, kot je »hiter razvoj aplikacij« (RAD) ali »agilen razvoj programske opreme«.
Ena od idej teh dve metodologij je, da pričnemo z razvojem programske opreme čimprej, da lahko naročnik kar se da hitro vidi prototip in ponudi nove smernice za naprej. Potem programsko opremo razvijamo naprej po iterativnem procesu v kratkih razvojnih ciklih.
Ta način ima številne posledice za razvijalca, saj mu med razvojem določene funkcionalnosti ni potrebno razmišljati, kaj bo v prihodnje in kakšne bodo zahteve. V procesu razvoja zahtev naročnika se velikokrat zgodi, da je potrebno uporabiti že napisano kodo in jo tudi ponovno napisati. Ta proces se imenuje reorganiziranje in se velikorat dogaja v procesu razvoja spletnih aplikacij. Dvojno kodo je zato potrebno premakniti na eno mesto in se držati načela, da se ne smemo ponavljati.
Na koncu se je potrebno prepričati, da spremembe in dodajanje novih zahtev naročnika ne vplivajo na obstoječo – že napisano kodo. Tu pride v poštev testiranje, ki se izkaže za zelo pomemben del razvoja programske opreme.
Symfony se izkaže kot zelo dobro orodje za hiter razvoj aplikacij.

YAML

YAML je jezik za opisovanje podatkov, podoben XML-u, vendar z enostavnejšo sintakso. Še posebej je uporaben pri opisovanju podatkov, ki jih je potrebno prevesti v polja.

Primer:

```
$house = array(
  'family' => array(
    'name'     => 'Doe',
    'parents'  => array('John', 'Jane'),
    'children' => array('Paul', 'Mark', 'Simone')
  ),
  'address' => array(
    'number'   => 34,
    'street'   => 'Main Street',
    'city'     => 'Nowheretown',
    'zipcode'  => '12345'
  )
);
```

PHP polje v zgodnji kodi lahko enostavno generiramo z uporabo YAML-niza:

```
house:
  family:
    name:     Doe
    parents:
      - John
      - Jane
    children:
      - Paul
      - Mark
      - Simone
  address:
    number: 34
    street: Main Street
    city: Nowheretown
    zipcode: "12345"
```

YAML ponuja tudi krajšo sintakso za opis zgornje strukture v samo nekaj vrsticah, in sicer z uporabo oglatih [] in zavitih {} oklepajev.

```
house:
  family: { name: Doe, parents: [John, Jane], children: [Paul, Mark, Simone] }
  address: { number: 34, street: Main Street, city: Nowheretown, zipcode: "12345" }
```

YAML je kratica za "YAML Ain't Markup Language" in se izgovarja kot "yamel". YAML lahko uporabimo za različne vrste jezikov.

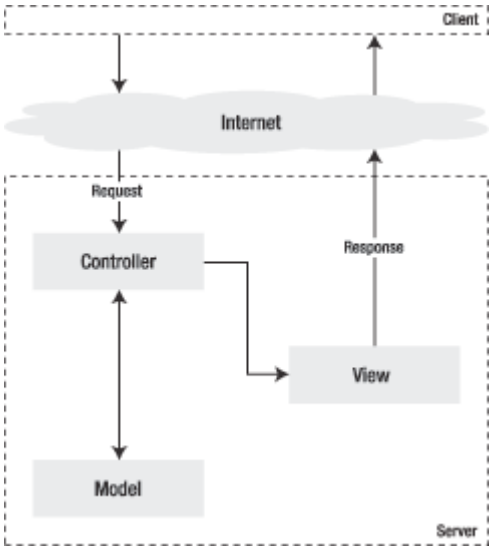
6.4 STRUKTURA OGRODJA

Po prvi namestitvi ogrodja Symfony se zdi sama struktura imenikov in datotek dokaj zastrašujoča. Sestavljena je iz številnih imenikov, skript in datotek, ki so mešanica jezika XHTML in PHP. Z razumevanjem takšne strukture bomo spoznali, da je Symfony le logično urejeno ogrodje.

MVC[19]

MVC je zgradba programske opreme, ki loči logični del od vhodnih podatkov in videza, kar omogoča neodvisen razvoj, testiranje iz vzdrževanje. Symfony temelji na tej zgradbi in je sestavljen iz treh nivojev:

- Model predstavlja informacije, na katerih aplikacija deluje – logični del.
- Pogled (view) skrbi, da postane spletna stran vizualno primerna za uporabnike.
- Kontrolni del (Controller) skrbi za pravilen odziv na uporabnikove zahteve na spletni strani.



Slika 28: MVC
Vir: http://www.symfony-project.org/gentle-introduction/1_4 (10.7.2010)

Na sliki je prikazana logika MVC-ja, ki jo uporablja Symfony, razlogov za to pa je več. Če npr. naredimo spletno aplikacijo, ki jo želimo prikazati na standardnih osebnih računalnikih in dlančnikih, bo vsa logika ostala enaka, spremeniti bo potrebno le pogled. S tem postanejo deli programske opreme neodvisni eden od drugega, kar prihrani veliko časa pri razvoju aplikacij.

[19] Model View Controller

6.5 MVC-SLOJI

Za primer si oglejmo, kako spremenimo klasično PHP-aplikacijo v MVC arhitekturo. Primer je prikazan za skripto, ki izpiše zapise iz bloga.

Nepovezano programiranje

```
<?php

// Connecting, selecting database
$link = mysql_connect('localhost', 'myuser', 'mypassword');
mysql_select_db('blog_db', $link);

// Performing SQL query
$result = mysql_query('SELECT date, title FROM post', $link);

?>

<html>
  <head>
    <title>List of Posts</title>
  </head>
  <body>
   <h1>List of Posts</h1>
   <table>
     <tr><th>Date</th><th>Title</th></tr>
<?php
// Printing results in HTML
while ($row = mysql_fetch_array($result, MYSQL_ASSOC))
{
echo "\t<tr>\n";
printf("\t\t<td> %s </td>\n", $row['date']);
printf("\t\t<td> %s </td>\n", $row['title']);
echo "\t</tr>\n";
}
?>
    </table>
  </body>
</html>

<?php
```

Skripta, ki je prikazana, ima nekaj dobrih in slabih lastnosti. Hitro jo napišemo, izvedemo, vendar jo je praktično nemogoče vzdrževati. Glavne težave s tako kodo so:

- Ni preverjanja napak. Kaj se zgodi, če "pade" povezava s podatkovno zbirko?
- PHP- in XHTML-koda sta pomešani, lahko bi rekli celo prepleteni.
- Koda nas zavezuje k uporabi MySQL podatkovne zbirke. Kaj se zgodi, če zamenjamo sistem za upravljanje zbirk podatkov?

Razmislite, kakšne so še dobre in slabe lastnosti nepovezanega programiranja? Kako bi uporabili kodo, ki ste jo že napisali v neki drugi skripti? Bi jo ponovno napisali?

Ločitev pogleda

Ukaza echo in printf iz zgornje skripte naredita kodo težko za branje. Spreminjanje XHTML-kode je povezano s trenutno sintakso in za sabo potegne veliko spreminjanja. Zato bi lahko

kodo razdelili na dva dela – ločimo logiko od pogleda. Logiko shranimo v kontrolni del (index.php), pogled pa v view.php.

Logični del, ki ga shranimo kot `index.php`.

```
<?php

 // Connecting, selecting database
 $link = mysql_connect('localhost', 'myuser', 'mypassword');
 mysql_select_db('blog_db', $link);

 // Performing SQL query
 $result = mysql_query('SELECT date, title FROM post', $link);

 // Filling up the array for the view
 $posts = array();
 while ($row = mysql_fetch_array($result, MYSQL_ASSOC))
 {
    $posts[] = $row;
 }

 // Closing connection
 mysql_close($link);

 // Requiring the view
 require('view.php');

 ?>
```

XHTML-kodo shranimo v datoteko `view.php` in predstavlja predlogo.

```
<html>
  <head>
    <title>List of Posts</title>
  </head>
  <body>
    <h1>List of Posts</h1>
    <table>
      <tr><th>Date</th><th>Title</th></tr>
    <?php foreach ($posts as $post): ?>
      <tr>
        <td><?php echo $post['date'] ?></td>
        <td><?php echo $post['title'] ?></td>
      </tr>
    <?php endforeach; ?>
    </table>
  </body>
</html>
```

Dobra praksa je, da poskušamo čimbolj ločiti XHTML- in PHP-kodo. S tem omogočimo XHTML oblikovalcem, da razumejo XHTML-kodo brez posebnega znanja PHP-jezika. Najbolj uporabljani PHP-stavki v modelu pogleda so echo, if/endif in foreach/endforeach. Vsa logika je premaknjena v index.php in ne vsebuje nobene XHTML-kode. Bitvo te ločitve je, da omogoča ponovno uporabo napisane kode v neki drugi obliki prikaza.

Ločitev podatkov

Večina kontrolne skripte (`index.php`) je namenjena manipulaciji s podatki. Tukaj se zastavlja precej vprašanj, kot npr. Kaj se zgodi, če bi želeli zapise prikazati v RSS obliki? Kaj če bi

poizvedbe podatkovne zbirke imeli na enem mestu, da se ne bi ponavljali? Kaj če se odločimo in zamenjamo ime tabele v zbirki iz post v weblog_post? Kaj se zgodi, če želimo namesto MySQL uporabljati PostreSQL?

Da zadostimo tem spremembam, ločimo iz kontrolnega dela manipulacijo s podatki. In naredimo novo skripto `model.php`.

```
<?php

function getAllPosts()
{
  // Connecting, selecting database
  $link = mysql_connect('localhost', 'myuser', 'mypassword');
  mysql_select_db('blog_db', $link);

  // Performing SQL query
  $result = mysql_query('SELECT date, title FROM post', $link);

  // Filling up the array
  $posts = array();
  while ($row = mysql_fetch_array($result, MYSQL_ASSOC))
  {
     $posts[] = $row;
  }

  // Closing connection
  mysql_close($link);

  return $posts;
}

?>
```

Kontrolna skripta `index.php` sedaj vsebuje:

```
<?php

// Requiring the model
require_once('model.php');

// Retrieving the list of posts
$posts = getAllPosts();

// Requiring the view
require('view.php');

?>
```

Kontrolna skripta je sedaj veliko bolj berljiva. Njena naloga je, da dobi podatke iz modela in jih da v model pogleda. V kompleksnejših aplikacijah kontrolni del vsebuje še seje, avtentikacijo uporabnikov in podobno.

Skripta model.php je namenjena dostopu do podatkov in kot taka mora biti ustrezno organizirana. Vse parametre, ki niso odvisni od podatkov, moramo obravnavati v kontrolni skripti. Bistveno je, da mora imeti model tako lastnost, da ga lahko ponovno uporabimo v drugem kontrolerju.

Nadaljnje ločitve

Princip MVC-arhitekture je torej ločitev kode v tri sloje: podatkovna logika je v modelu, pogled oz. videz se nahaja v view in aplikacijska logika v kontrolerju.

Seveda se lahko posamezna logika še naprej deli v manjše dele.

Zbirka podatkov

Podatkovno logiko lahko razdelimo na dva dela: logiko podatkov in logiko dostopa do podatkovne zbirke. Na ta način ustvarimo svoje funkcije za dostop do zbirke podatkov, ki pa so neodvisne od vrste podatkovne zbirke. Če bomo torej v prihodnosti zamenjali podatkovno zbirko, bo potrebno spremeniti le logiko dostopa do podatkovne zbirke.

Skripto `model.php` razdelimo na dva dela.

Dostop do zbirke podatkov:

```
<?php

function open_connection($host, $user, $password){
  return mysql_connect($host, $user, $password);
}

function close_connection($link){
  mysql_close($link);
}

function query_database($query, $database, $link){
  mysql_select_db($database, $link);

  return mysql_query($query, $link);
}

function fetch_results($result){
  return mysql_fetch_array($result, MYSQL_ASSOC);
}
```

Dostop do podatkov:

```
function getAllPosts()
{
  // Connecting to database
  $link = open_connection('localhost', 'myuser', 'mypassword');

  // Performing SQL query
  $result = query_database('SELECT date, title FROM post', 'blog_db', $link);

  // Filling up the array
  $posts = array();
  while ($row = fetch_results($result)){
     $posts[] = $row;
  }

  // Closing connection
  close_connection($link);

  return $posts;
}

?>
```

Vidimo, da v delu dostopa do podatkov ni nobene odvisnosti od podatkovne zbirke. Vse funkcije, ki smo jih ustvarili v delu dostopa do podatkovne zbirke, lahko ponovno uporabimo za druge modele.

Videz

Tako kot smo ločili podatkovno logiko, lahko ločimo tudi videz. Običajno spletne strani vsebujejo glavo, grafično razporeditev, nogo in navigacijo z menijem. Ponavadi se spreminja le notranji del posamezne strani, zato ločimo predlogo in razporeditev.
Razporeditev vsebuje podatke, ki so skupni vsem podstranem, v predlogo pa damo spremenljivke, ki jih posreduje kontrolni del.
Če sledimo tem principom, lahko kodo view.php razdelimo na tri dele

Predloga – `mytemplate.php`

```
<h1>List of Posts</h1>
<table>
<tr><th>Date</th><th>Title</th></tr>
<?php foreach ($posts as $post): ?>
  <tr>
    <td><?php echo $post['date'] ?></td>
    <td><?php echo $post['title'] ?></td>
  </tr>
<?php endforeach; ?>
</table>
```

Logični del v `view.php`

```
<?php

$title = 'List of Posts';
$posts = getAllPosts();

?>
```

Razporeditev v `view.php`

```
<html>
  <head>
    <title><?php echo $title ?></title>
  </head>
  <body>
    <?php include('mytemplate.php'); ?>
  </body>
</html>
```

Action in Front kontroler

Kot smo omenili na prejšnjih straneh, je naloga kontrolerja zelo pomembna. Njegove naloge so med drugim povezane z zahtevki, varnostjo in podobno. Kontrolni del je ponavadi razdeljen na dva dela, in sicer pa Front kontroler, ki je skupen celotni aplikaciji, in Action kontroler, ki je specifičen za posamezne podstrani.

Ena od dobrih strani Front kontrolerja je, da se preko njega vrši vstopna točka v aplikacijo. Če bi npr. kadarkoli želeli onemogočiti dostop do spletne strani, to storimo prek Front kontrolerja, namesto, da bi šli onemogočati vsak kontroler posebej.

Objekti

Vsi primeri, ki smo jih obravnavali v tem poglavju, so temeljili na proceduralnem programiranju. Objektno orientirano programiranje naredi logiko spletne strani enostavnejšo.

6.6 ZGRADBA SYMFONY OGRODJA

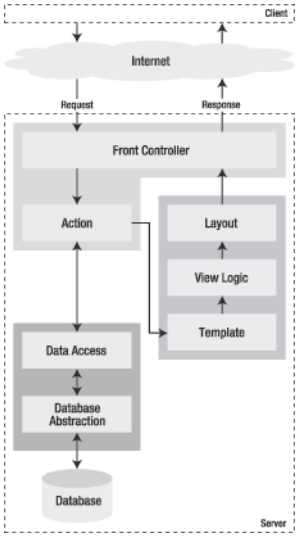
V predhodnem poglavju smo ugotovili, da za izpis zapiskov iz bloga potrebujemo naslednje dele:

Model
- podatkovna zbirke (Database abstraction)
- dostop do podatkov (Data Access)

Pogled
- pogled (View)
- predloga (Template)
- videz (Layout)

Kontroler
- Front kontroler (Front controller)
- Action (Action)

Grafični prikaz:



Slika 29: Symfoy ogrodje
Vir: http://www.symfony-project.org/gentle-introduction/1_4 (10.7.2010)

Za primer izpisa zapiskov iz bloga bi morali v Symfony ogrodju spremeniti le tri datoteke. Za vse ostalo poskrbi ogrodje, ki izdela Front kontroler, predlogo in ostale potrebne datoteke in razrede za dostop do podatkov.

Action v `myproject/apps/myapp/modules/weblog/actions/actions.class.php`

```
<?php
class weblogActions extends sfActions
{
```

```
  public function executeList()
  {
    $this->posts = PostPeer::doSelect(new Criteria());
  }
}

?>
```

Predloga v `myproject/apps/myapp/modules/weblog/templates/listSuccess.php`

```
<h1>List of Posts</h1>
<table>
<tr><th>Date</th><th>Title</th></tr>
<?php foreach ($posts as $post): ?>
  <tr>
    <td><?php echo $post->getDate() ?></td>
    <td><?php echo $post->getTitle() ?></td>
  </tr>
<?php endforeach; ?>
</table>
```

View v `myproject/apps/myapp/modules/weblog/config/view.yml`

```
listSuccess:
  metas: { title: List of Posts }
```

Symfony avtomatsko generira privzeti videz (Layout), ki ga lahko po potrebi dopolnimo.
`myproject/apps/myapp/templates/layout.php`

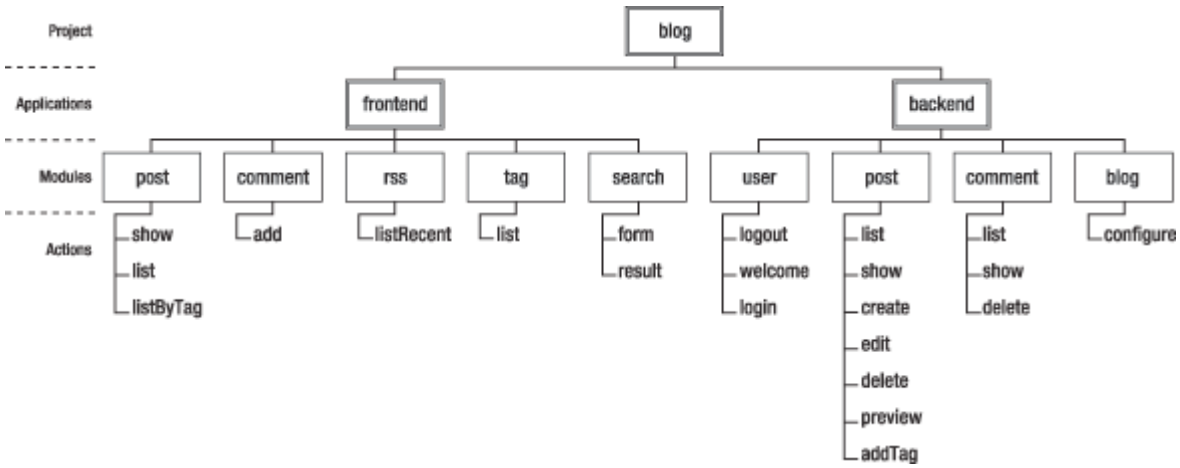
```
<html>
  <head>
    <?php echo include_title() ?>
  </head>
  <body>
    <?php echo $sf_data->getRaw('sf_content') ?>
  </body>
</html>
```

Koda, ki je prikazana, je enaka kodi v poglavju 10.5. Evidentno je, da z uporabo programskega ogrodja ni nič več programiranja in kodiranja kot z nepovezanim programiranjem. Poleg tega takšen način omogoča bolj "čisto" kodo, fleksibilnost, ponovno uporabo že napisane kode in podobno.

Organizacija projekta Symfony

Znotraj ogrodja Symfony lahko ustvarjamo projekte. Znotraj projekta so locirane aplikacije, ki jih je lahko več in tečejo neodvisno ena od druge. V večini primerov sta dve aplikaciji frontend in backend. Frontend aplikacija je spletna stran, ki jo vidijo uporabniki, Backend pa služi za administracijo spletne strani. Aplikacije uporabljajo isto podatkovno zbirko. Obstaja lahko tudi več aplikacij znotraj projekta, kar je uporabno takrat, ko imamo več majhnih spletnih strani znotraj ene spletne strani.
Vsaka aplikacija vsebuje enega ali več modulov. Modul ponavadi predstavlja stran ali skupek strani s podobno vsebino. Imamo lahko npr. module domov, članki, pomoč, košarica, račun ipd.

Moduli so sestavljeni iz akcij, ki predstavljajo različne akcije, ki jih lahko naredimo z moduli. Pri spletni trgovini je npr. košarica, ki vsebuje akcije, kot so dodaj, prikaži in posodobi.



Slika 30: Organizacija projekta Symfony
Vir: http://www.symfony-project.org/gentle-introduction/1_4 (10. 7. 2010)

Oglejte si spletno stran projekta Symfony - http://www.symfony-project.org/ (10. 7. 2010). Poskušajte najti gradivo, kjer je podrobneje opisana struktura ogrodja.

6.7 PRIMER: PRIJAVNA SKRIPTA

V tem podpoglavju bomo prikazali izvorno kodo prijavnega obrazca. Sam potem ustvarjanja projekta je enak kot je zapisano v Rešitvi naloge, zato se bomo tukaj osredotočili na izdelavo uporabniško določenega obrazca. Skripta bo omogočala prijavi, pri čemer bo upoštevala uporabniške ime in geslo, ki ju bomo sami določili in ne bo povezavna z zbirko podatkov.

Obrazce ustvarjamo v mapi `/lib/form`, kamor dodamo nov razred z imenom LoginForm, kot je prikazano v nadaljevanju.

```
class LoginForm extends sfForm
{
  public function configure()
  {
    $this->setWidgets(array(
      'username' => new sfWidgetFormInput(),
      'password' => new sfWidgetFormInputPassword()
    ));

    $this->widgetSchema->setNameFormat('login[%s]');

    $this->setValidators(array(
      'username' => new sfValidatorChoice(array('required' => true, 'choices' => array('admin'))),
      'password' => new sfValidatorChoice(array('required' => true, 'choices' => array('moje_geslo')))
    ));
  }
}
```

Metoda `setWidgets` se uporablja za dodajanje elementov v obrazec. V zgornjem primeru smo dodali vnosno polje tipa text (`sfWidgetFormInput()`) in password (`sfWidgetFormInputPassword()`). Za uspešno prijavo je potrebno ustvariti je validacijo vpisanih podatkov, kar pomeni, da določimo ali je vpisan element obvezen in kakšen parameter potrebuje. Tako uporabniško ime mot geslo sta obvezna, oba pa imata kot izbiro določeno uporabniško ime "admin" in geslo "mojegeslo". Nastaviti je potrebno še akcijo, ki se zgodi, ko se izvede obrazec. To določimo v področju actions, kot metodo executeLogin()

```
class authActions extends sfActions
{
  public function executeLogin(sfWebRequest $request)
  {
    $this->form = new LoginForm();

    if ($request->isMethod('post'))
    {
      $this->form->bind($request->getParameter('login'));
      if ($this->form->isValid())
      {
        // authenticate user and redirect them
        $this->getUser()->setAuthenticated(true);
        $this->getUser()->addCredential('user');
        $this->redirect('home/index');
      }
    }
  }

  public function executeLogout()
  {
    $this->getUser()->clearCredentials();
    $this->getUser()->setAuthenticated(false);
    $this->redirect('@homepage');
  }
}
```

Najprej ustvarimo objekt razreda LoginForm in pridobimo podatke in obrazca. Če je obrazec pravilen, potem uporabnika preusmerimo na spletno stran home/index. Še prej pa nastavimo sejo. Metoda executeLogout je namenjena odjavi, ki pobriše sejo in uporabnika preusmeri na domačo spletno stran (@homepage).
Izdelati je potrebno še predlogo na spletni strani z imenom `loginSuccess.php`:

```
<form action="<?php echo url_for('auth/login') ?>" method="POST">
  <table>
    <?php echo $form ?>
    <tr>
      <td colspan="2">
        <input type="submit" />
      </td>
    </tr>
  </table>
</form>
```

Kot action v form napišemo naslov, kjer se nahaja action skripta, za metodo pa uporabimo POST. S type="submit" smo vstavili gumb.
Na koncu je potrebno nastaviti še varnost za prijavo uporabnika, kar storimo tako, da v datoteko security.yml dodamo vsebino:

```
default:
  is_secure: on
  credentials: user
```

Rešitev naloge:

Dana naloga iz uvoda zahteva, da v že izdelano nalogo index.html dodamo obrazec, preko katerega bomo sami vnesli besedilo, ki se izpiše na zaslon. To lahko storimo z uporabo programskega jezika, ki zna prebrati vneseno besedilo z obrazcev. Symfony je brezplačno orodje, ki ga moramo najprej sneti s spletne strani www.symfony-project.org/ (10. 7. 2010). Oglejmo si primer namestitve in konfiguracije v operacijskem sistemu Ubuntu 9.10. Za operacijski sistem Windows je postopek zelo podoben.

Najprej moramo v `dev/blog` ustvariti mapo `lib/vendor`, kar storimo z ukazom:

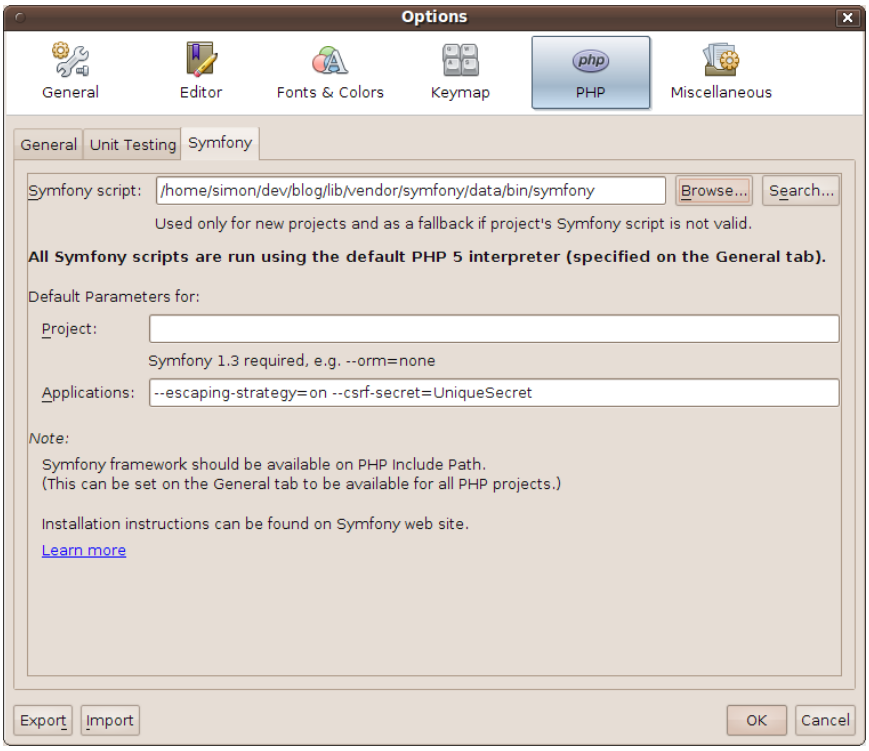
```
$ mkdir -p lib/vendor
```

Snamemo zadnjo stabilno različico, in sicer tgz datoteko. To datoteko shranimo v pravkar ustvarjeno mapo. Potrebno jo je razširiti. Uporabniki Windows operacijskega sistema snamete zip datoteko.

```
$ tar zxpf symfony-1.4.3.tgz
$ mv symfony-1.4.3.tgz symfony
$ rm symfony-1.4.3.tgz
```

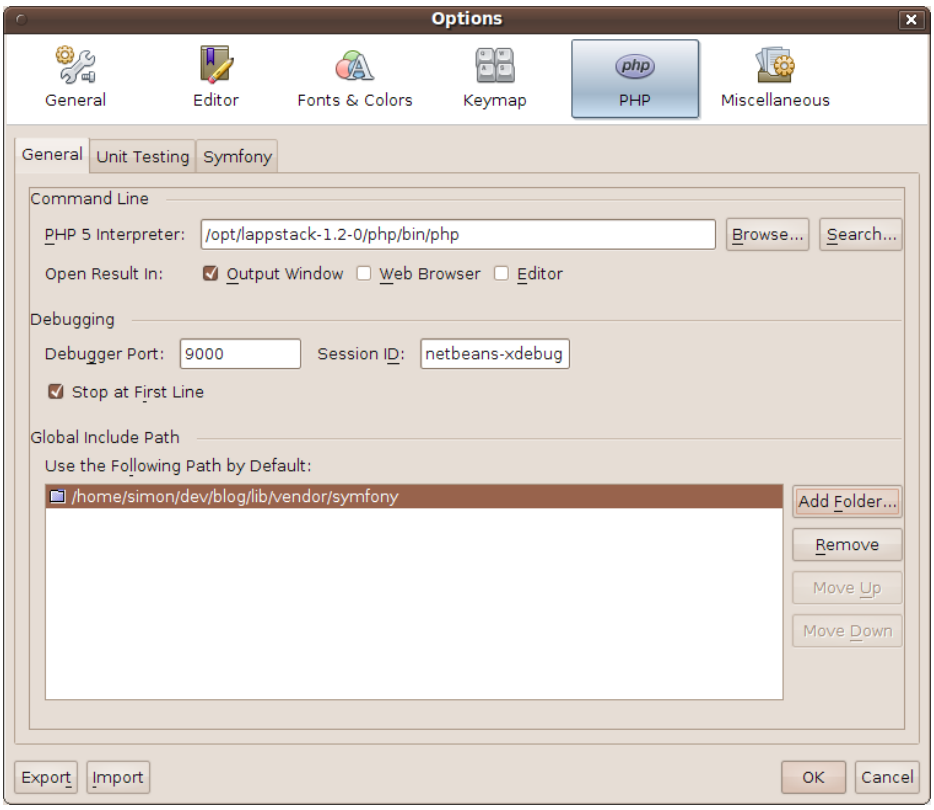
Ustvarjanje novega projekta

Nov projekt ustvarimo s programskim orodjem NetBeans. Preden začnemo z ustvarjanjem novega projekta, moramo nastaviti še nekaj parametrov. Odpremo `Tools->Options->PHP`. V zavihku Symfony določimo pot do Symfony skripte, ki jo NetBeans uporabi za generiranje novega projekta.



Slika 31: Zavihek Symfony

V zavihku General določimo še pot do PHP Interpreterja, ki se nahaja v `/opt/lappstack-1.2.0/php/bin`. V Global Include Path določimo še pot do mape, kjer je shranjen Symfony, s čimer določimo, da bodo vsi projekti lahko uporabljali Symfony. Na koncu potrdimo vpisane parametre s klikom na gumb OK.



Slika 32: Zavihek General

Sledi izdelava novega projekta, kjer izberemo PHP Application in damo projektu ime blog. V polju Sources Folder izberemo mapo `dev/blog`. V tretjem koraku izberemo lokalno spletno stran in v polje Project URL vpišemo http://blog.localhost/. V četrtem koraku ne smemo pozabiti označiti Symfony PHP Web Framework, s čimer bo NetBeans generiral vse potrebne datoteke. Zaenkrat ustvari samo uporabniško (frontend) aplikacijo. Backend se uporablja za administracijo spletne strani in jo lahko ustvarimo tudi kasneje. S klikom na gumb Finish ustvarimo nov projekt.

Posodobitev strežniških nastavitev

Symfony uporablja mapo web, v kateri so shranjene datoteke, ki so vidne na svetovnem spletu, zato je potrebno posodobiti strežniške nastavitve. Odpremo datoteko `httpd-vhosts.conf`.

```
$ nano /opt/lappstack-1.2.0/apache2/conf/extra/httpd-vhosts.conf
```

Spremenimo nastavitve:

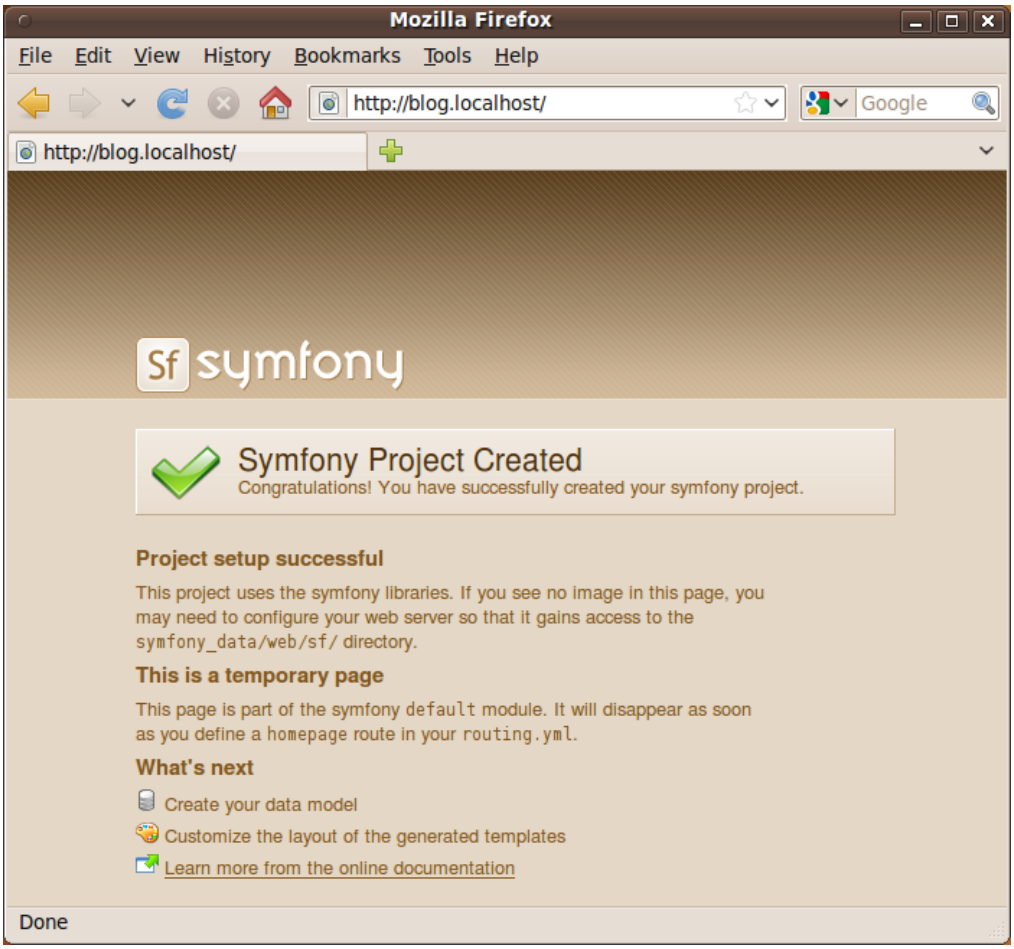
```
<VirtualHost 127.0.0.1:80>
    DocumentRoot "/home/simon/dev/blog/web"
    ServerName blog.localost
    ServerAlias blog.localhost
    ErrorLog "logs/blog.localhost-error_log"
    CustomLog "logs/blog.localhost-access_log" common
    <Directory "/home/simon/dev/blog/web">
      AllowOverride All
      Allow from All
    </Directory>

    Alias /sf /home/simon/dev/blog/lib/vendor/symfony/data/web/sf
    <Directory "/home/simon/dev/blog/lib/vendor/symfony/data/web/sf">
      AllowOverride All
      Allow from All
    </Directory>
</VirtualHost>
```

Ponovno zaženemo strežnik z ukazom:

```
$ cd /opt/lappstack-1.2.0/
$ ./ctlscript.sh restart
```

Če obiščemo spletno stran http://blog.localhost, se prikaže privzeta spletna stran Symfony projekta.



Slika 33: Symfony Project Created

Poleg privzete spletne strani lahko Symfony preizkusimo tudi v razvojnem okolju (http://blog.localhost/frontend_dev.php). Razvojno okolje je namenjeno razvoju spletne aplikacije in je nevidno vsem uporabnikom, dokler sami tega ne določimo. Od navadnega okolja se razlikuje po tem, da ima v desnem zgornjem kotu majhne ikone, s katerimi vidimo log-datoteke, SQL-stavke, ki so se izvršili, nastavitve ipd.

Podatkovni model

Da bi shranili podatke o novicah v blogu, potrebujemo relacijsko zbirko podatkov. Ker je Symfony objektno usmerjeno ogrodje, bi radi na tak način delali tudi z zbirkami podatkov. Namesto klasičnih SQL-stavkov, bomo uporabili objekte. Pri tem nam je v veliko pomoč ORM-orodje, pri čemer lahko uporabimo Propel ali Doctrine.

Propel in Doctrine sta odprtokodni knjižnici, ki omogočata dostop do zbirke podatkov prek objektov. Več informacij o Propelu (http://propel.phpdb.org (10. 7. 2010)) in Doctrineu (http://www.doctrine-project.org/ (10. 7. 2010)) je dostopnih na njihovih spletnih straneh.

V našem projektu uporabimo Doctrine. Ker ORM potrebuje opis tabel to storimo tako, da odpremo datoteko `config/doctrine/schema.yml` in dodamo:

```
BlogPosts:
  connection: doctrine
  tableName: blog_posts
  columns:
    id:
      type: integer(8)
      fixed: false
      unsigned: false
      primary: true
      autoincrement: true
    title:
      type: string(255)
      fixed: false
      unsigned: false
      primary: false
      notnull: true
      autoincrement: false
    image:
      type: string(255)
      fixed: false
      unsigned: false
      primary: false
      notnull: false
      autoincrement: false
    body:
      type: string()
      fixed: false
      unsigned: false
      primary: false
      notnull: true
      autoincrement: false
    username:
      type: string(255)
      fixed: false
      unsigned: false
      primary: false
      notnull: true
      autoincrement: false
    created_at:
      type: timestamp(25)
      fixed: false
      unsigned: false
      primary: false
      notnull: true
      autoincrement: false
    updated_at:
      type: timestamp(25)
      fixed: false
      unsigned: false
      primary: false
      notnull: true
      autoincrement: false
    slug:
      type: string(255)
      fixed: false
      unsigned: false
      primary: false
      notnull: false
      autoincrement: false
```

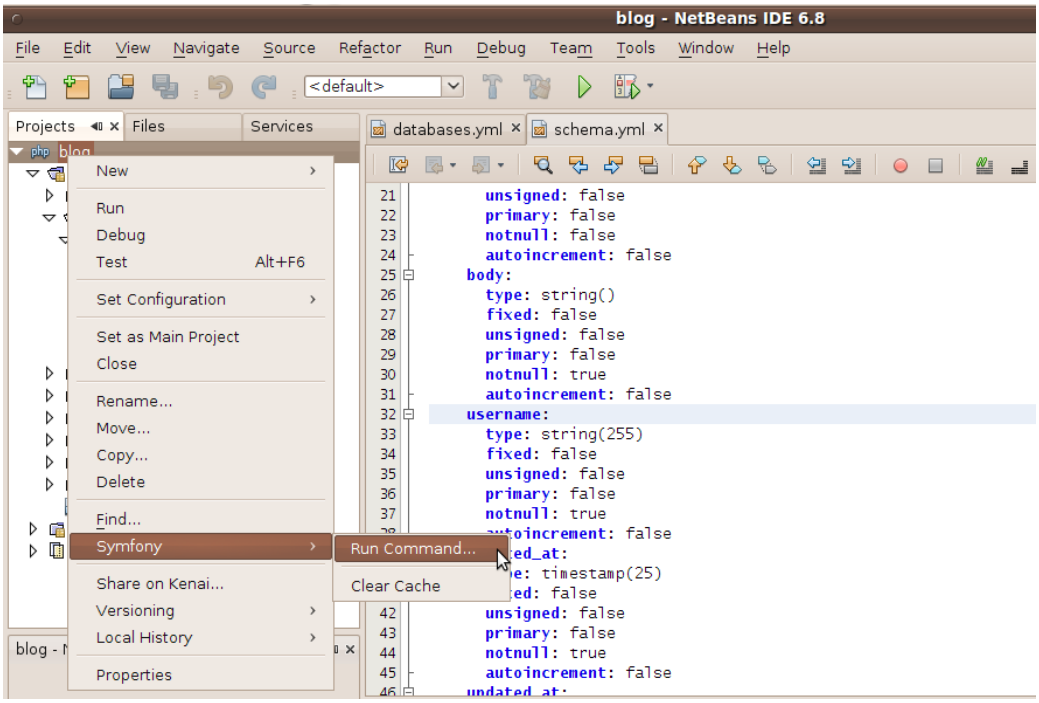
Z zgornjo kodo smo opisali tabelo blog_posts (z jezikom YAML), ki vsebuje stolpce: id, title, image, body, user, created_at, updated_at in slug. Stolpec slug bomo uporabljali za izdelavo povezav oz. prikaz URL-naslova v prijaznejši obliki. URL-naslove lahko prikažemo na dva načina:
http://www.example.com/web/controller/article.php?id=123456&format_code=6532 ali
http://www.example.com/article/Finance_in_France

Prvi primer ima kar nekaj slabosti. Poleg tega, da kaže ime php skripte, so v njem prikazane še pomembne informacije o podatkovni zbirki, kot je npr. id. V drugem primeru je povezava dosti bolj pregledna, saj je že iz nje takoj razvidno, za kakšen članek gre, poleg tega pa je njena struktura hierarhična v obliki map in podmap. V stolpec slug vpisujemo niz, s katerim identificiramo posamezen vnos oz. članek v blogu.

Če želimo z opisom zbirke vstaviti tabele v zbirko podatkov, je potrebno določiti še parametre za dostop do zbirke podatkov. V tem primeru je Symfony zelo fleksibilen, saj v samo eni datoteki (`config/databases.yml`) določimo, za kakšno zbirko podatkov gre. V našem primeru bomo uporabili PostgreSQL. Datoteko lahko konfiguriramo na dva načina:

- z ročno spremembo datoteke `databases.yml`;
- z uporabo `configure:database` ukaza.

Veliko bolj praktičen je drugi način, saj je Symfony zelo močno orodje za konfiguracijo prek ukazne vrstice. Ta ukazna vrstica je v našem primeru integrirana v NetBeans.



Slika 34: Ukaz zaženi

Do nje pridemo z desnim miškinim klikom na projekt blog. Izberemo Symfony->Run Command. Poiščemo ukaz `configure-database`, pri čemer lahko uporabimo Filter. Prikazana je uporaba tega ukaza, ki mu lahko damo tudi parametre. Tako zaženemo ukaz:

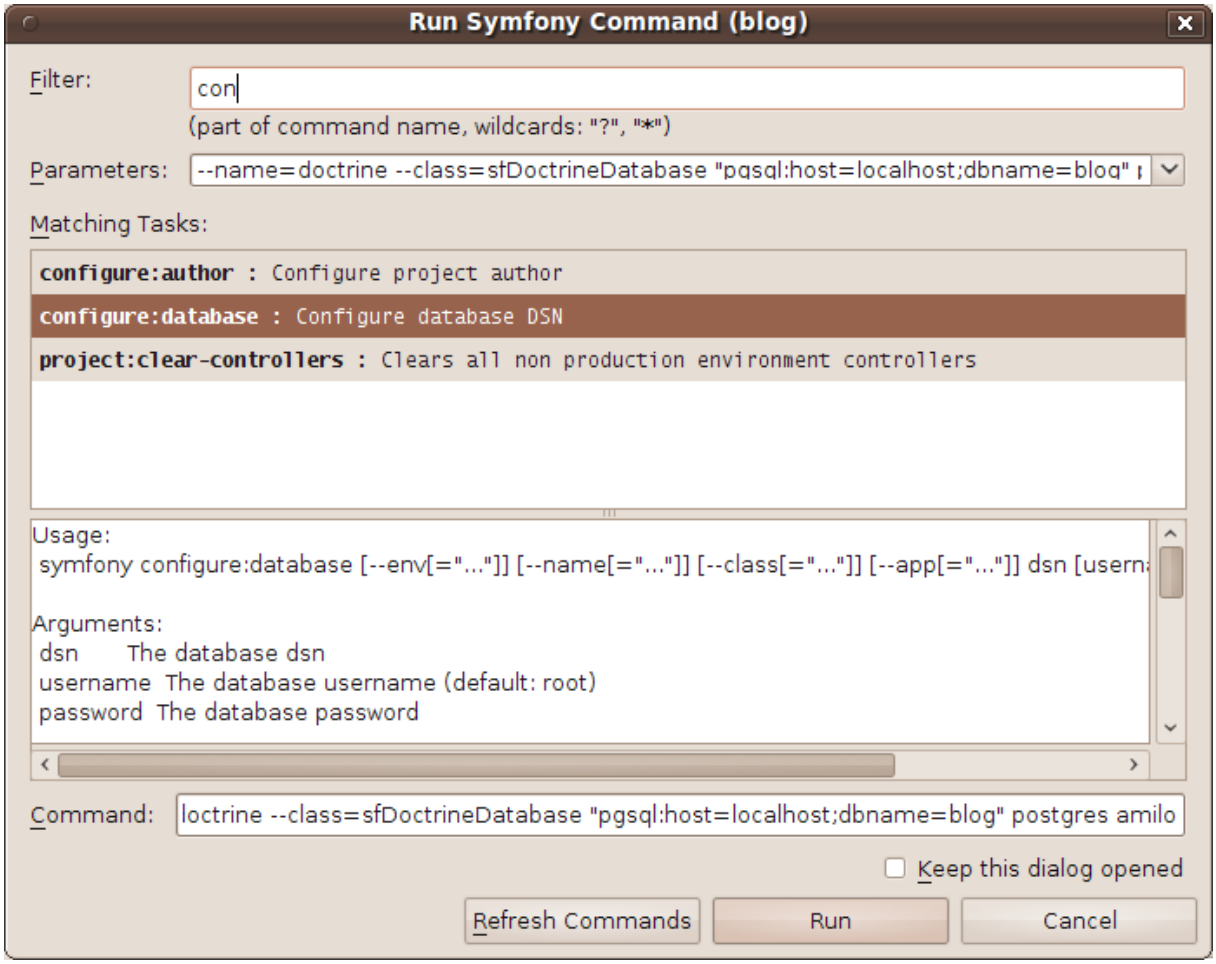
```
configure-database
```

Pri čemer uporabimo parametre:

```
--name=doctrine --class=sfDoctrineDatabase "pgsql:host=localhost;dbname=blog" postgres amilo
```

Z `--name-doctrine --class=sfDoctrineDatabase` določimo Doctrine model, `pgsql` pove tip zbirke podatkov, ki je v našem primeru PostgreSQL in ime zbirke podatkov. Zadnja dva parametra sta uporabniško ime in geslo za dostop do zbirke podatkov. Če gesla nimamo, pustimo prazno.

V primeru uporabe MySQL zbirke podatkov namesto `pgsql` napišemo `mysql`.



Slika 35: Konfiguriranje SUZP

Z zgornjim ukazom ustvarimo datoteko `config/databases.yml`, katere vsebina je:

```
all:
  doctrine:
    class: sfDoctrineDatabase
    param:
      dsn: 'pgsql:host=localhost;dbname=blog'
      username: postgres
      password: amilo
```

Z opisom zbirke podatkov v schema.yml lahko uporabimo nekatere doctrine funkcije in generiramo SQL-stavke, s katerimi bomo potem ustvarili tabele v zirki podatkov. Ponovno uporabimo vgrajeno komandno okno in zaženemo ukaza:

```
doctrine:build-model
doctrine:build-sql
```

Z doctrine:build-sql ustvarimo SQL-stavke v `data/sql` mapi:

```
CREATE TABLE blog_posts (id BIGSERIAL, title VARCHAR(255) NOT NULL, image VARCHAR(255), body TEXT NOT NULL, username VARCHAR(255) NOT NULL, created_at TIMESTAMP NOT NULL, updated_at TIMESTAMP NOT NULL, slug VARCHAR(255), PRIMARY KEY(id));
```

Da bi v resnici ustvarili tabelo v zbirki podatkov, moramo zagnati še ukaz doctrine:insert-sql.

```
doctrine:insert-sql
```

Krajši ukaz za večino ukazov, ki smo jih spoznali je, `doctrine:build --all --no-confirmation`, zato je najbolje, da ga zaženemo.

```
doctrine:build --all --no-confirmation
```

S tem ukazom ustvarimo tudi obrazce, njihovo validacijo in še veliko več.

Podatki

Na prejšnjih straneh smo s pomočjo YAML-jezika opisali strukturo tabele, ki jo bomo uporabili pri izdelavi bloga. Če želimo blog dejansko uporabiti, je potrebno tabelo napolniti s podatki. Zelo nepraktično je, če bi podatke vnašali ročno, npr. prek phppgadmina. Poleg tega se z vsakim ustvarjanjem tabel in stolpcev izgubijo vsi do tedaj vneseni podatki. Symfony uporablja mapo data/fixtures, v kateri so shranjeni podatki tabel, ki so opisani z YAML-jezikom.

Ustvarimo datoteko `data/fixtures/posts.yml` in v njo shranimo:

```
BlogPosts:
  prvi:
    title:              Lepo pozdravljeni
    image:              asus.jpeg
    body:               To je moj prvi vnos v katerem vas prav lepo pozdravljam.
    username:           Simon
    created_at: '2010-01-05'
    updated_at: '2010-01-05'
    slug:   lepo-pozdravljeni

  drugi:
    title:              Drugi post
    image:              memory.jpg
    body:               Lorem ipsum
    username:           Simon Horvat
    created_at: '2010-01-06'
    updated_at: '2010-01-06'
    slug:   drugi-post
```

Pri tem moramo biti pozorni, da obe sliki shranimo v mapo `web/uploads/novica`. Slike so lahko poljubne, zato lahko po želji spremenimo imena. Prav tako lahko dodamo več vnosov. Symfony zna iz yml-datoteke prebrati tudi kontrolne strukture, kot npr. zanke. S tem lahko na enostaven način vnesemo več vnosov naenkrat.

Z ukazom doctrine:data-load potrdimo vnos v tabelo.

```
doctrine:data-load
```

Uporabimo lahko tudi ukaz build, s čimer potrdimo vse spremembe.

```
doctrine:build --all --and-load
```

Dodajanje modula

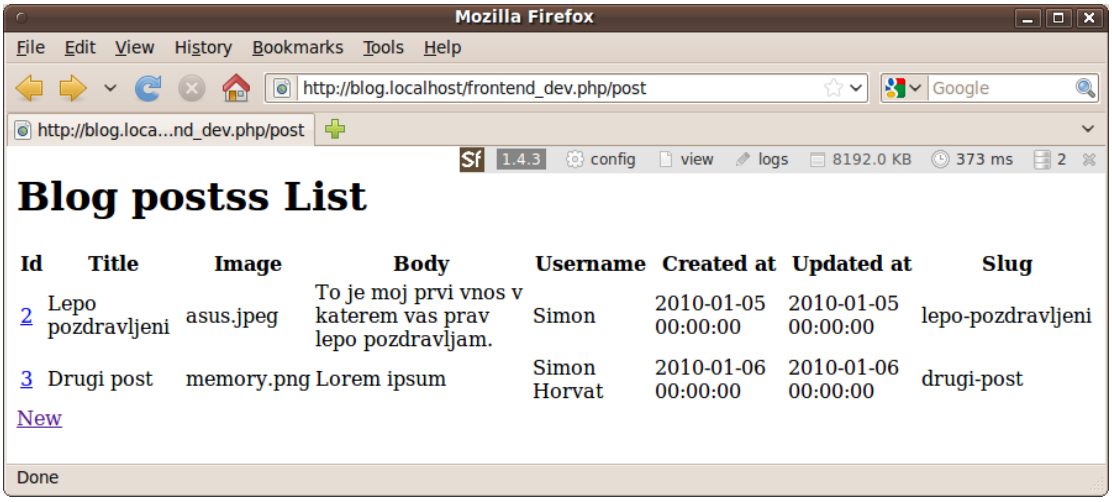
V prejšnjih korakih smo naredili vse potrebno za izdelavo spletne strani, ostane le še izdelava modula. Le-ta omogoča, da lahko dodajamo, spreminjamo ali brišemo vnose. Uporabimo ukaz `doctrine:generate-module`.

```
doctrine:generate-module –with-show –non-verbose-templates frontend post BlogPosts
```

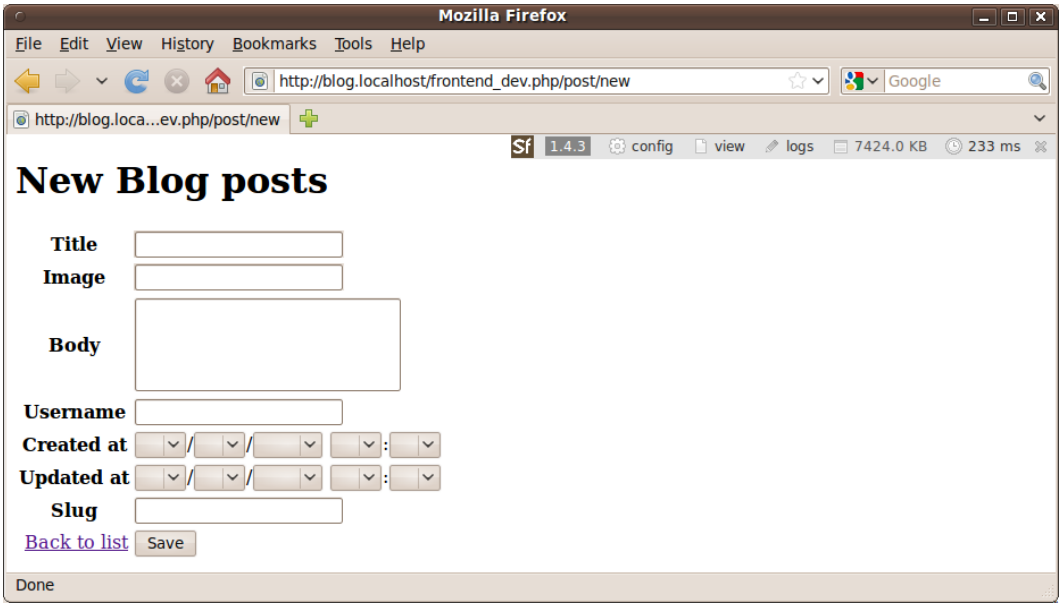
S tem ustvarimo modul z imenom post v uporabniški (frontend) aplikaciji za BlogPosts model. S tem so se ustvarile datoteke in mape v `apps/frontend/modules/post`, in sicer:

`actions/` - v tej mapi so vsebovane vse akcije oz. logika modula post. Akcije lahko po želji spreminjamo.

`templates/` - mapa vsebuje predloge (dodajanje, brisanje, ustvarjanje …) modula. To kategorijo lahko po želji spreminjamo, dodajamo slogovne predloge in nove predloge.
Sedaj lahko preizkusimo pravkar ustvarjeni modul z obiskom naslova http://blog.localhost/frontend_dev.php/post.



Slika 36: Seznam vnosov dnevnika



Slika 37: Obrazec za vnos

Osnovne nastavitve in konfiguracija je s tem končana. Do te stopnje smo prišli brez pisanja ene same vrstice izvorne kode. Če bi se izvorne kode lotili sami, bi za to porabili precej več časa in programiranja.

Veliko gradiva in primerov kako spletno stran ustvariti po svojih željah se nahaja na domači spletni strani Symfony projekta v področju dokumentacija.

Povzetek:

Symfony je brezplačno programsko ogrodje z enostavno integracijo v NetBeans. Je popolnoma objektno usmerjeno orodje in v celoti uporablja razrede in objekte. Z uporabo objektno-relacijskega modela dostopamo z objekti tudi do zbirke podatkov. Pri tem za opis uporabljamo jezik YAML.

Sicer je Symfony logično urejeno ogrodje s tremi neodvisnimi deli:
- Model,
- Pogled,
- Kontrolni del.

Na ta način ločimo posamezne dele, npr. zbirko podatkov od videza ipd.

Symfony je hierarhično urejen, sestavljen iz aplikacij kot npr. uporabniški (angl. Frontend) oz. upraviteljski (angl. Backend). Uporabniški del je aplikacija, ki jo vidi uporabnik na spletni strani, upraviteljski del pa je namenjen administraciji. V vsaki aplikaciji so moduli, ki predstavljajo neodvisno logiko posameznega dela. Tako lahko na spletni strani ločimo npr. novice, forum, ankete ipd.

Naloge:

1. Nalogo iz tega poglavja dopolnite, in sicer:
   - Vključite in uporabite slogovne predloge.
     - Oglejte si: (http://www.symfony-project.org/jobeet/1_4/Doctrine/en/04 (10. 7. 2010)).
   - Uporabite preusmerjanje URL naslovov (routing). URL-naslov http://blog.localhost/frontend_dev.php/post/show/id/2 spremenite v bolj razumljivega z uporabo slug stolpca. Primer: http://blog.localhost/frontend_dev.php/2/lepo-pozdravljeni.
     - Oglejte si: http://www.symfony-project.org/jobeet/1_4/Doctrine/en/05 (10. 7. 2010).
   - Uporabite in posodobite obrazce. Ne želimo, da bi uporabnik lahko pri novem vnosu vnašal polja Created At, Updated At, Slug in Id.
     - Oglejte si: http://www.symfony-project.org/jobeet/1_4/Doctrine/en/10 (10. 7. 2010).
   - Spremenite izpis v bolj praktičnega, pri čemer bo prikazana tudi slika in ne ime slike.

2. Izdelajte spletno stran z imenom Galerija slik. Spletna stran naj omogoča:
   - dodajanje slik, pri čemer pri dodajanju vnesemo še ime albuma, ime uporabnika, ki je sliko naložil in datum nalaganja slike,
   - urejanje podatkov o albumu in slikah,
   - brisanje podatkov o albumu in slikah,
   - prikaz albumov v vizualno privlačni obliki, pri čemer uporabite slogovne predloge.

7 VARNOST IN NAPOTKI ZA IZDELAVO SPLETNIH PORTALOV

Področje varnosti je eno najpomembnejših pri izdelavi spletnih portalov in morebitnem konfiguriranju strežnika za odprto področje – medmrežje. Če se odločimo, da bomo portal izdelovali lokalno na svojem računalniku in ga prek protokola FTP-ja prenesli na zakupljen strežnik, obsega naše področje le pravilno programiranje vseh skript. V nasprotnem primeru smo odprti v svet, kjer je nekaj milijard ljudi – potencialnih uporabnikov naših strani. Nekateri se bodo na naših straneh ustavili bežno, drugi bodo uporabljali portal brez slabih namenov, spet tretji pa komaj čakajo, da lahko zlorabijo slabo varovan strežnik. V našem primeru se bomo omejili le na pravilno izdelavo spletnih strani, spoznali bomo prednosti in slabosti sej ter piškotkov in pogledali konfiguracijo obrazcev za varen prenos podatkov.

Na začetku je treba poudariti, da je popolnoma varen sistem praktično nemogoče narediti. Ljudje, ki rušijo in vlamljajo v sisteme, so vedno korak pred izdelovalci programskih jezikov in spletnih portalov oz. mest. Obstajajo načini, kako se zavarovati pred takšnimi vlomi, in treba se je zavedati, da je veriga tako močna, kot je močan njen najšibkejši člen. Lahko npr. opravljamo spletno transakcijo in uporabnika sprašujemo po imenu, priimku, naslovu, tipu transakcije, geslu in podobno. Te podatke moramo shranjevati v več piškotkov ali sej, ker s tem zmanjšamo možnost zlorabe in povečamo varnost podatkov.

Zavedati se moramo tudi, da pri testiranju ne bomo mogli preizkusiti vseh možnosti, pa čeprav gre za nezahtevno in enostavno stran. Vsak uporabnik bo na stran gledal drugače, zato je najbolje, da programer gleda na stran z logičnega vidika. Treba je najti skrajne in nenavadne možnosti, ki jih pričakujemo. To je eden od najtežjih korakov. Vendar pa krivda za slabo napisano kodo ni vedno na strani pisca, saj lahko napake obstajajo tudi znotraj PHP-ja, kar pomeni, da moramo sistem vedno dopolnjevati. Vsaka nova verzija programa odpravi določene hrošče in pomanjkljivosti.

Veliko programerjev, predvsem začetniki, v želji po čimprejšnji izdelavi spletne strani pozabijo na verjetno najpomembnejši kriterij – varnost. Vse več aplikacij se seli na svetovni splet, zato si to področje zasluži posebno poglavje. Različne zlorabe prek piškotkov so že nekaj časa stalnica svetovnega spleta, prav tako obrazci. Preko obrazcev in neprimerne zaščite lahko hekerji dodajajo vnose in s tem nekontrolirano polnijo zirko podatkov, kar ima lahko za posledico "sesutje" strežnika. Zlorabe se dogajajo tudi prek URL-naslovov, ki jih je prav tako potrebno zaščititi, tako da morebitni napadalci čim manj zvedo o arhitekturi in zgradbi naše spletne strani. S področjem varnosti se lahko kosa tudi sama estetika spletne strani. Še tako funkcionalna in uporabna spletna stran lahko postane spletna stran brez obiskovalcev. V srednjem delu poglavja se bomo lotili teh dveh vidikov, varnosti in estetike.

7.1 OSNOVNE METODE ZAŠČITE

Vstavljena koda

Vstavljena koda je niz podatkov, ki jih uporabnik napiše s slabim namenom, to je da bi vplival na delovanje skript, zbirke itn. Ob slabo napisani kodi lahko dostopa do zaščitenih podatkov, jih briše, popravlja ter vnaša, in to brez kakšnega posebnega napora. Recimo, da si je avtor zamislil prikaz uporabnikov v stilu:

```
/uporabniki.php?id=45
```

Avtor želi prikazati uporabnika s številko 45. Izvede se skripta "uporabniki.php" in prebere podatek, da je identifikacijska številka enaka 45. Ta skripta pošlje zahtevek zbirki po podatkih za uporabnika s številko 45 (njegov ID oz. userID). Če programer pozabi na pretvarjanje spremenljivk in postavitev narekovajev v stavku SQL, je zloraba zelo preprosta. Če številko 45 zamenjamo s "46%20or%userID%3d45", imamo popoln dostop do podatkov za uporabnika s številko 46, saj pomeni vpisani niz "46 or userID = 45". Pred seboj imamo torej neke vrste terminal z neposrednim dostopom do strežnika MySQL.

Pri taki vrsti programiranja je najboljša zaščita pretvorba vrednosti identifikacijske kode uporabika (userID) v celoštevilčno vrednost – integer. Na razpolago so različne vrste pretvorb:

```
$userID = intval($userID);
$userID = (int) $userID;
settype($userID, 'int');
```

Prvi dve pretvorbi vrneta novo vrednost tipa integer, zadnja pa pretvori spremenljivko v tip integer. Vrednosti Boolean (true/false) se pretvorijo v 1 oz. 0 (false), vrednosti nizov (string) pa vrnejo 0, razen če je v nizu (ali na začetku le-tega) številka – v tem primeru vrne ustrezno številko.

Pri vrednostih, ki niso številke, je najbolje uporabljati vgrajene funkcije **mysql_escape_string()**, **pg_escape_string()** in **add_slashes()**, ki poskrbijo, da strežnik SQL ne upošteva posebnih znakov, ki bi lahko kakorkoli vplivali na prikaz podatkov.

Obrazci

Zelo pogost primer na spletnih portalih so obrazci, prek katerih uporabnik vpiše določene podatke, ki se potem obdelajo in vrnejo rezultat. Recimo, da imamo obrazec za prijavo uporabnikov:

```
//Nezanesljiva prijava preko obrazca.
<form name="form1" method="POST" action="test.php">
  <p>uporabnik:
    <input type="text" name="uporabnik">
    <br>
    geslo:
    <input type="password" name="geslo">
    <br>
    <input type="submit" name="Submit" value="V redu">
  </p>
</form>

<?php
if (($Submit) && ($uporabnik == "Janez") && ($geslo == "Janez123")){
        $prijava = TRUE;
}

if ($prijava){
//naredi nekaj kar lahko naredijo samo registrirani uporabniki
echo "Prijava uspešna";
}
?>
```

Datoteko shranimo kot test.php in jo zaženemo (http://localhost/test.php). V polje uporabniško ime vpišemo "Janez", v geslo pa "Janez123" in kliknemo "V redu". Če je vse v redu, se izpiše "Prijava uspešna". Kodo lahko poskusimo zagnati na tak način:

http://localhost/test.php?prijava=1

Vidimo, da se tudi tokrat izpiše "Prijava uspešna", kar pove, da smo prišli do podatkov, ne da bi vpisali uporabniško ime. Rešitev se skriva v datoteki php.ini in vrstici "register_globals". Ta vrstica vpliva na način obravnave celotnega spektra uporabniško poslanih podatkov. Vrednost "On" povzroči, da se vsem spremenljivkam, poslanim z uporabo GET in POST, avtomatično dodeli spremenljivka php z enakim imenom in vrednostjo. V primeru vrednosti "Off" pa so spremenljivke, na katere ima uporabnik vpliv, ločene od naših. V tem primeru je torej spremenljivka $prijava dobila vrednost 1, ne da bi uporabnik vedel za uporabniško ime, in geslo in enako se zgodi s piškotki.

Zaščita je v postopku spremembe vrednosti nastavitve "register_globals" na Off in uporabe **$_GET**, **$_POST**, **$_COOKIE**, **$_SESSION**, **$_ENV**, **$_SERVER** …

Varna prijava bi bila takšna:

```
// Varna prijava preko obrazca.
<?php
$uporabnik = $_POST['uporabnik'];
$geslo = $_POST['geslo'];
$potrditev = $_POST['Submit'];

if (($potrditev) && ($uporabnik == "Janez") && ($geslo == "Janez123")){
$prijava = TRUE;
}

if ($prijava){
//naredi nekaj kar lahko naredijo samo registrirani uporabniki
echo "Prijava uspešna";
}
?>

<form name="form1" method="POST" action="<? echo $PHP_SELF ?>">
  <p>uporabnik:
    <input type="text" name="uporabnik">
    <br>
    geslo:
    <input type="password" name="geslo">
    <br>
    <input type="submit" name="Submit" value="V redu">
  </p>
</form>
```

Možnost "register_globals" je od verzije PHP 4.2.0 privzeto izklopljena in najbolje jo je tako tudi pustiti.

7.2 MAJHNE DATOTEKE - PIŠKOTKI

Piškotki so majhne količine podatkov, shranjene na uporabnikovem računalniku. Shranijo jih spletne strani z namenom, da jih bodo kasneje uporabile pri prikazovanju strani in podobno. Vsak piškotek je sestavljen iz imena, vrednosti in datuma zapadlosti ter iz informacij o gostitelju. Posamezen piškotek je omejen na 4 KB. Potem ko je piškotek shranjen, ga lahko

prebere samo gostitelj oz. tista stran, ki ga je shranila. Uporabnik si lahko sam določi, ali želi sprejemati piškotke ali ne, in piškotek je tako praktičen način shranjevanja določenih informacij.

Uporabimo ga lahko v več namenov. Lahko ga uporabimo npr. v prijavnem sistemu za določeno stran. Da uporabnik ne bi vedno vpisoval uporabniškega imena in gesla, spletni portal lahko shrani piškotek na njegov računalnik. Ko se uporabnik naslednjič vrne, se piškotek prebere in ga avtomatsko prijavi. Drugi zanimiv primer je npr. pri anketah za preprečitev večkratnega glasovanja enega uporabnika. Uporabniku shranimo piškotek, in ko se uporabnik vrne, ga anketa prebere in mu prepreči ponovno glasovanje. Možnosti uporab je torej veliko, največja pomanjkljivost pa je, da se shranijo na uporabnikov računalnik, kjer jih lahko le-ta spreminja, zlorabi sistem in pride do neželenih podatkov.

Preden lahko preberemo piškotek, ga moramo najprej shraniti. PHP ima v ta namen vgrajeno funkcijo setcookie():

```
setcookie (ime, vrednost, čas zapadlosti, pot, domena, varnost);
```

```
setcookie ("mojpiskot", "uporabnik", time() + 86400, "/", "domena.com", 0);
```

Nastavili smo naslednje:
- piškotek smo shranili pod imenom mojpiskot,
- vsebuje vrednost uporabnik (lahko shranimo tudi spremenljivko),
- piškotek se bo izbrisal oz. bo potekel po enem dnevu (86400 sekund),
- dostopen bo na vsaki strani,
- dostopen bo na domeni domena.com,
- ni posebej definiranih varnostnih nastavitev.

Ko smo piškotek enkrat shranili, nam postane dostopen kot običajna globalna spremenljivka, do katere dostopamo prek funkcije `$_COOKIE['mojpiskot']`. Če želimo vrednost piškotka izpisati na ekranu, to storimo z ukazom:

```
echo $_COOKIE['mojpiskot'];
```

Piškotek izbrišemo na ta način, da ga enostavno prepišemo z vrednostjo nič.

Razmislite, kaj bi lahko uporabili namesto piškotkov. Ali bi lahko kako uporabili IP-naslove?

7.3 ALTERNATIVE PIŠKOTKOM - SEJE

Seje imajo podobno funkcijo kot piškotki, vendar je bistvena in zelo pomembna razlika, da se seje shranjujejo na strežnik in ne na uporabnikov računalnik. S tem se zelo poveča varnost, saj je v tem primeru možnost zlorabe minimalna. Med njima je še ena razlika, in sicer ta, da lahko piškotku nastavimo čas zapadlosti, česar pri seji ni mogoče, zato se seja večinoma uporablja za shranjevanje trenutnih podatkov, ki se po končanem delu oz. zapiranju odjemalca odstranijo.

Sejo začnemo s funkcijo start_session() in tedaj PHP registrira 32-bitno edinstveno kodo, ki označuje trenutnega uporabnika. Ta koda je veljavna, dokler uporabnik ne zapre odjemalca. Z naslednjim obiskom se uporabniku dodeli nova koda. Ko je seja enkrat zagnana. lahko vanjo shranimo poljubno število spremenljivk in s tem prenašamo podatke med posameznimi

stranmi iste domene. Ta način je veliko prikladnejši kot prenašanje spremenljivk prek skritih obrazcev ali piškotkov. Tako lahko npr. prenašamo informacije o uporabniku, ko se enkrat registrira in brska po portalu. Njegovo uporabniško ime in podatki bodo vedno na voljo programu, da jih kadarkoli uporabi.

Na vrhu vsake strani, v kateri želimo uporabiti ali registrirati sejo, mora biti funkcija:

```
session_start();
```

Znotraj skript lahko sejo kadarkoli registiramo z ukazom:

```
session_register("ime");
$ime = "Janez";
```

Do registrirane seje in spremenljivke lahko dostopamo prek ukaza:

```
echo $_SESSION['ime'];
```

Za izbris spremenljivk in končanje seje uporabljamo funkciji **session_unset()** in **session_destroy()**. Primer uporabe bi bil lahko odjava uporabnika s spletnega portala.

Razmislite in poskušajte ugotoviti, kam se shranjujejo seje?

7.4 NAPOTKI ZA IZDELAVO SPLETNIH PORTALOV

Dobro izdelane strani so vizualno prijetne strani in dobri ustvarjalci vedo, kako spraviti v harmonijo tekst, grafiko, naslove, povezave in ostale vsebine spletne strani. Večina obiskovalcev oz. uporabnikov strani ne ve skoraj nič o pisavah, poravnavah, tabelah in podobno, vedo pa zagotovo, katera stran je všečna in privlačna na pogled. Le-taka pritegne obiskovalce dan za dnem, seveda z ustrezno vsebino.

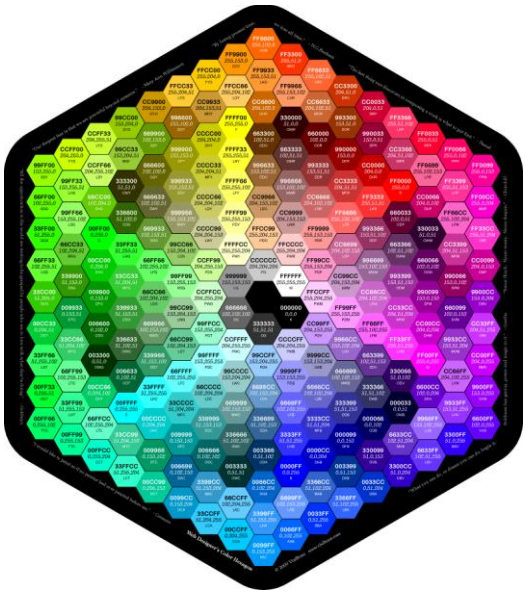
Prvo pravilo pri spletnih portalih je, da portal ni preveč raznolik in da najdemo pravo ravnovesje med oblikovno zasnovo in vsebino. Vse skupaj mora biti zaokrožena celota in ne sme preveč odstopati od osnovnega koncepta. Obiskovalec mora v vsakem trenutku vedeti, kje se posamezne strani pod isto domeno ne smejo razlikovati in morajo biti razpoznavne. Centralni del tega je izbira logotipa, ki mora biti skrbno izbran in viden na vsaki strani portala. Naredimo lahko en večji logotip za osnovno stran in manjše za druge strani, vsekakor pa pazimo, da je zelo kakovosten.

Poleg logotipa mora vsebovati portal tudi pravo harmonijo barv, pisav in kompozicije. V ta namen lahko uporabimo sloge CSS, ki omogočajo spreminjanje pisav, barv in postavitve skozi cel portal, torej globalno. Namesto da bi na vsaki strani na novo nastavljali pisavo, določimo za vse enako oz. podobno. Če bomo kasneje želeli, kaj spremeniti, bomo spreminjali samo eno datoteko in ne vseh strani. Stili nas varujejo tudi pred neželeno postavitvijo besedila.

Tretji najpomembnejši del je besedilo oz. tema strani. Velja pravilo, da grafična zasnova ne sme izstopati od grafične podobe besedila na portalu. Obiskovalci se morajo pogovarjati o vsebini in ne o videzu portala. Preveč barv, nenavadne pisave in preveliko poudarjanje besedila so stvari, ki obiskovalce odvračajo od ogleda in jim pozornost usmerjajo drugam. Ves čas se moramo zavedati poslanstva strani in ne smemo pretiravati z razno glasbo, videom

in domišljijskimi skriptami Java. Seveda pa je vse odvisno od tega kaj želimo prikazati in kakšen je namen postavitve portala. Grafična zasnova mora odražati sporočilo strani.

Delo z grafiko oz. barvami za spletne strani se povsem razlikuje od tistega, kar je namenjeno za tiskanje. Zavedati se moramo, da spletni pregledovalniki prikazujejo le delček od 16 milijonov možnih RGB-barv, zato poznamo t. i. zanesljive in nezanesljive barve. Zanesljivih spletnih barv je 216 in bodo vidne enako v različnih pregledovalnikih, operacijskih sistemih in pri ločljivostih monitorjev. Če vseeno želimo uporabljati nezanesljive barve, se moramo vedno prepričati, kakšen je izgled takšnih barv pri nižjih ločljivostih (barve monitorja nastavimo na 256). Pri snovanju spletnih dokumentov ponavadi uporabljamo skupek treh dvomestnih šestnajstiških števil. Pri zanesljivi barvni paleti uporabljamo kombinacijo parov 00, 33, 66, 99, cc in ff. Vse možne kombinacije so prikazane na sliki 8-1.



Slika 38: Zanesljive spletne barve
Vir: http://www.visibone.com/

Svetle primarne in sekundarne barve (modra, rdeča, rumena, oranžna, zelena) so kričeče in v osnovi izražajo veselje. Pri izdelavi spletnih portalov se takih barv izogibamo, uporabljamo jih le za pritegnitev pozornosti. Snovalci veliko bolj uporabljajo hladne barve, ki se v kombinaciji s črno izkažejo za zelo privlačne. Le-te vsebujejo veliko bele barve in delujejo pomirjajoče. Take barve se imenujejo tudi pastelne barve.

Pomemben del so tudi ozadja, ob katerih se moramo zavedati, da bodo nepravilno izbrana povzročila težje branje in razpoznavanje besedila. Najbolje je, da izberemo svetlejšo in nevtralno barvo in nikoli preveč izstopajoče (kontrastne). Če smo v dvomih, je najbolje uporabiti belo barvo. Za ozadje lahko uporabimo tudi sliko, in sicer se lahko odločimo, ali bomo uporabili večjo, ki se bo razprostirala čez celo stran, ali manjšo, ki se bo ponavljala. Takih primerov se je najbolje izogibati, saj lahko povzročijo efekt roba, kjer se ena slika dotika druge, poleg tega se bo stran nalagala veliko počasneje.

Pri izbiri pisave se moramo držati pravila: manj je boljše. Najbolje je za celo stran uporabiti eno ali največ dve različni pisavi, ki ju po potrebi odebelimo ali povečamo. Za pisave je najbolje uporabljati standardne, pa čeprav se komu zdijo eksotične pisave lepše. Razlog tiči v tem, da lahko posamezen pregledovalnik prikaže samo tiste pisave, ki jih ima uporabnik nameščene znotraj operacijskega sistema. V nasprotnem primeru bo prikazal nekaj drugega, kar je lahko v nasprotju s tistim, kar si želimo. Najbolj uporabljane pisave so tiste, ki jih ima večina uporabnikov nameščenih na svojem računalniku: Arial, Times New Roman in Verdana. Poudarjen tekst se praviloma uporablja za naslove, pri čemer moramo paziti, da ne pišemo vse z velikim črkami. Poševni tekst ponavadi uporabljamo za poudarke in tuje besede. Zadnji stil so podčrtana besedila, ki se ponavadi ne uporabljajo, ker v osnovi tak tekst označuje povezave.

Na različne načine lahko ustvarimo tudi povezave, kjer imamo več možnosti in se lahko oddaljimo od standardnega modro-vijoličnega podčrtavanja povezav. Slogi CSS omogočajo različne načine podčrtavanj. Povezave so lahko vedno podčrtane, lahko so druge barve,

podčrta lahko ob prehodu na povezavo izgine in podobno. Tudi tu velja pravilo, da je enostavno boljše.

Pozorni moramo biti tudi na ločljivost strani. Danes večina uporabnikov koristi srednjo ločljivost zaslonov 1024 x 768 in višjo, zato je tudi največ strani prilagojenih oz. načrtovanih v teh velikostih. Problem se pojavi pri tistih, ki imajo ločljivosti, ki so manjše, npr. 640 x 480 ali 800 x 600. Take strani morajo vsebine ročno premikati. Najboljši način za kompenzacijo je, da naredimo stran, ki se avtomatsko povečuje ali pa ima standardno velikost 640x480 in se pri večjih ločljivostih samodejno namesti na sredino zaslona.

V zadnji najpomembnejši del spletnih portalov spada ažurnost podatkov. Portal oz. strani smo naredili z namenom, obveščati ljudi oz. uporabnike z našega področja. Stran uspešno živi, če so na njej redno objavljene novosti, če vsebuje ankete, forum, poštne novice, nagradne igre in podobno (kar je odvisno od upravitelja strani oz. zasnove informacijske nadgradnje portala).

Povzetek:

Vstavljena koda je ena izmed najpogostejših napak programerjev spletnih strani. Z nepravilno uporabo URL-naslovov hote ali nehote prikažemo na nepravilen način zaščitene podatke.

```
/uporabniki.php?id=45
```

S pomočjo id-številke lahko zlorabimo zbirko podatkov, zato je veliko bolje uporabiti usmerjanje URL-naslovov brez prikaza id-številk. Velik poudarek je potrebno dati obrazcem, ki jih je potrebno zaščititi z validacijo in prikazom napak. Pomembna je tudi funkcija register_globals, ki mora biti na Off. Zlorabe so dogajajo tudi preko piškotkov, nekoliko manj pa tudi preko sej.

Pri izdelavi spletnih portalov se moramo držati standardnih smernic pri vizualnem izgledu, ločljivosti, barvah, pisavi ipd. Uporabljamo predvsem zanesljive spletne barve in slogovne predloge.

Naloge:

1. Poiščite spletno stran z anketo in glasujte v anketi. Ko se naslednjič vrnete na spletno stran, običajno ne morete več glasovati. Nato v brskalniku preverite, kam na vaš trdi disk se shranjujejo piškotki. Najdite piškotek, kjer so shranjene informacije o anketi. Kaj vse se nahaja v tej datoteki?

2. Kakšna je razlika med $_GET in $_POST?

3. Od česa je odvisno, ali bo spletna stran znala prikazati pisavo, ki jo določimo v nastavitvah? Ali bo pisava na vseh računalnikih enaka? Utemeljite odgovor.

4. Piškotki se shranjujejo na _____, seje pa na _____.

8 LITERATURA

1. Mrvar, A. *OSNOVE HTML* (online). 2009. (citirano 26. 6. 2009). Dostopno na naslovu http://mrvar.fdv.uni-lj.si/sola/VIS2/html/htmlslo.htm.

2. Zorman, S. *HTML HELP*. Velenje: Višja strokovna šola, 2008.

3. *PHP* (online). 2009. (citirano 15. 7. 2010). Dostopno na naslovu www.php.net.

4. *W3C* (online). 2009. (citirano 15. 7. 2010). Dostopno na naslovu http://www.w3.org/.

5. *Symfony* (online). (citirano 7. 9. 2010). Dostopno na naslovu http://www.symfony-project.org/.

6. *jQuery* (online). (citirano 7.9.2010). Dostopno na naslovu http://jquery.com/

7. *w3schools* (online) (citirano 8.9.2010). Dostopno na naslovu http://www.w3schools.com/

Projekt **Impletum**
Uvajanje novih izobraževalnih programov na področju višjega strokovnega izobraževanja v obdobju 2008–11

Konzorcijski partnerji:

Operacijo delno financira Evropska unija iz Evropskega socialnega sklada ter Ministrstvo RS za šolstvo in šport. Operacija se izvaja v okviru Operativnega programa razvoja človeških virov za obdobje 2007–2013, razvojne prioritete Razvoj človeških virov in vseživljenjskega učenja in prednostne usmeritve Izboljšanje kakovosti in učinkovitosti sistemov izobraževanja in usposabljanja.