UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Primož Marn

Primerjava klasičnega in hitrega razvoja spletnih aplikacij

DIPLOMSKO DELO UNIVERZITETNEGA ŠTUDIJA

Mentor:

doc. dr. Mojca Ciglarič

Ljubljana, 2008

Namesto te strani vstavite original izdane teme diplomskega dela s podpisom mentorja in dekana ter žigom fakultete, ki ga diplomant dvigne v študentskem referatu, preden odda izdelek v vezavo!

Zahvala

Zahvaljujem se mentorici doc. dr. Mojci Ciglarič za mentorstvo in nasvete pri izdelavi diplomskega dela.
Zahvaljujem se članom razvojne skupine, ki so me sprejeli medse in v meni vzbudili zanimanje za spletne tehnologije.
Zahvaljujem se družini za podporo in potrpežljivost v času študija.

Kazalo

Seznam kratic in simbolov

PHP	*(angl.) PHP: Hypertext Preprocessor;* Skriptni programski jezik.
HTML	*(angl.) HyperText Markup Language;* Označevalni jezik za oblikovanje večpredstavnostnih dokumentov.
API	*(angl.) Application Programming Interface;* Programski vmesnik.
BASIC	*(angl.) Beginner's All-purpose Symbolic Instruction Code;* Začetniška vsenamenska simbolična ukazna koda.
SQL	*(angl.) Structured Query Language;* Strukturiran povpraševalni jezik.
ORM	*(angl.) Object-Relational Mapping;* Objektno-relacijska preslikava.
AJAX	*(angl.) Asynchronous JavaScript and XML*; Asinhroni JavaScript in XML.
RAD	*(angl) Rappid Application Development*; Hiter razvoj aplikacij.
TDD	*(angl.) Test-Driven Development*; Testno voden razvoj.
MVC	*(angl.) Model-view-controller*; Model-pogled-krmilnik.
CSS	*(angl.) Cascading Style Sheets*; Prekrivne predloge.
YAML	*(angl.) YAML Ain't a Markup Language;* Lahko berljiv format zapisa podatkov.
I18N	*(angl.) Internationalization;* Internacionalizacija.
XML	*(angl.) Extensible Markup Language;* Razširljivi označevalni jezik.
FTP	*(angl.) File transfer protocol;* Protokol za prenos datotek.

Povzetek

Spletne strani niso več namenjene samo predstavitvi podjetij in njihovih izdelkov, temveč obsegajo vedno več dinamike, kot je na primer spletna prodaja in interakcija z uporabnikom. Na splet se selijo celotni informacijski sistemi, zato so spletne aplikacije vedno večje in zapletenejše. Za realizacijo tako obsežnih spletnih aplikacij pa je potrebno izbrati tehnologijo, ki bo omogočala hiter razvoj, kakovosten izdelek in učinkovito in cenovno sprejemljivo vzdrževanje. V tem diplomskem delu je predstavljen razvoj spletnih aplikacij s skriptnim jezikom PHP.

V uvodnem poglavju je predstavljena problematika izdelave spletnih aplikacij in tehnologije, ki jih je ob tem potrebno obravnavati. Predstavljen je jezik PHP, njegov razvoj skozi čas ter prednosti in slabosti.

Osrednja poglavja diplomskega dela obravnavajo različne načine razvoja z jezikom PHP in arhitekturne značilnosti aplikacij. Najpomembnejši del je predstavitev ogrodja Symfony in primerjava njegove uporabe s klasičnim razvojem. Za potrebe primerjave so bili razviti trije programski moduli.

Primerjava klasičnega razvoja in uporabe ogrodja je pokazala očitne prednosti pri uporabi slednjega, saj je občutno skrajšan čas razvoja. Tako narejena aplikacija odraža najnovejše pristope in tehnologije, ki zagotavljajo večjo kakovost.

Ključne besede: spletne aplikacije, razvoj, ogrodje Symfony, PHP

Abstract

Web sites are no longer used only for presentations of companies and their products. More and more modern web sites are dynamic, for example online shopping and other user interactions. Migration of information systems to the web results in large and complex applications. Implementation of comprehensive applications requires technology, that provides rapid development, quality product and efficient and cost effective maintenance. This work presents web application development using PHP scripting language.

In introduction the concepts of web application development are presented and technologies that need to be investigated. PHP language is introduced, it's development over time and advantages and disadvantages of using it.

Main chapters present different development strategies when using PHP and architectural characteristics of the applications. The most important part is presentation of Symfony framework and comparison of its use with conventional development. Three application modules were developed for purposes of the comparison.

The comparison of conventional development and use of the framework showed obvious advantages of using the framework. Time needed for development is significantly reduced. Applications developed with framework reflect the latest approaches and technologies, which provide higher quality.

Keywords: web applications, development, Symfony framework, PHP

1. Uvod

Predstavitev podjetij in njihove ponudbe na spletu ni več izjema, temveč pravilo. Povpraševanje po izdelavi spletnih strani je vedno večje, vedno več pa je tudi dinamičnih spletnih strani, ki omogočajo interakcijo z uporabnikom. Za izdelavo dinamičnih funkcionalnosti in administracijskih okolij, uporaba statične HTML kode ni več zadostna. Potrebna je uporaba spletnih skriptnih jezikov s katerimi podpremo poslovno in funkcionalno logiko.

Spletni jezik PHP je zaradi dostopnosti in zmogljivosti zelo razširjena izbira, zato sem ga izbral za temelj raziskave razvoja spletnih aplikacij. Za izdelavo vedno večjih in zahtevnejših aplikacij pa tudi uporaba golega skriptnega jezika lahko postane preveč okorna in zamudna. Zato se vedno več uporablja ogrodja, s katerimi se skrajša čas razvoja, hkrati pa nudijo tudi druge ugodnosti. Cilj diplomskega dela je raziskati možnosti in prednosti razvoja s pomočjo izbranega razvojnega ogrodja ter ugotoviti kako se razlikuje od klasičnega razvoja.

Za primerjavo različnih načinov razvoja pa je potrebno spoznati lastnosti jezika PHP, možne arhitekture spletnih aplikacij in sestavljanja končne kode HTML, ki jo prikazujejo spletni brskalniki.

Pri razvoju kode spletnih aplikacij si pomagamo z razvojnimi okolji. Razvoj v enostavnem urejevalniku teksta je ob velikem številu možnosti jezikov PHP in HTML lahko zelo zamuden in nepraktičen, zato je cilj spoznati kakšne prednosti prinašajo zmogljivejša razvojna orodja.

1.1. Predstavitev jezika PHP

1.1.1. Zgodovina

PHP je naslednik projekta PHP/FI, ki ga je leta 1995 zasnoval Rasmus Lerdorf. Začelo se je z nekaj enostavnimi skriptami, napisanimi v jeziku Perl, kasneje pa je sledila mnogo večja implementacija v jeziku C, ki je podpirala komunikacijo s podatkovnimi bazami in razvoj enostavnih dinamičnih spletnih aplikacij. PHP/FI je bil izdan kot projekt odprte kode. Leta 1997 je sledil PHP/FI 2.0, druga različica C implementacije, ki je imela že okrog 1000 privržencev, vendar pa je še vedno večino kode prispeval prvotni avtor.

Velik korak je bil storjen z izidom PHP 3, ki je prva različica, ki je podobna PHP-ju današnjega časa. Zanj sta zaslužna Andi Gutmans in Zeev Suranski, ki sta leta 1997 povsem na novo napisala celotno kodo. Ena največjih novosti je bila močna podpora razširljivosti. Poleg podpore za različne podatkovne baze, protokole in API-je, je možnost razširljivosti privabila veliko razvijalcev, ki so prispevali svoje module. Pomembna je bila tudi precej

močnejša in doslednejša jezikovna sintaksa, ki je podpirala tudi objektno programiranje. Do konca leta 1998 je PHP uporabljalo že na deset tisoče uporabnikov in na vrhu popularnosti je PHP 3 jezik uporabljalo že 10% spletnih strežnikov.

Razvoj PHP 4 se je začel že kmalu po izidu PHP 3, konec leta 1998. Cilj je bil na novo napisati jedro, kar bi izboljšalo hitrost izvajanja kompleksnih aplikacij in povečalo modularnost. Rezultat poimenovan »Zend Engine« je dosegel zastavljene cilje in na njem temelječ PHP 4.0 je izšel maja leta 2000. PHP 4 je uporabljalo že na sto tisoče uporabnikov in je poganjal nekaj milijonov spletnih strani, kar je pomenilo več kot 20% spletnih domen. Razvojna ekipa se je močno povečala in sedaj obsega že na ducate ljudi.

Julija 2004 je izšel PHP 5, ki temelji na Zend Engine 2.0, z novim objektnim modelom in množico novih funkcionalnosti.

1.1.2. Prednosti

Dostopnost: PHP je na voljo zastonj in vsebuje podrobno dokumentacijo v mnogih jezikih. Obstaja množica podpornih skupin, forumov in razvojnih ekip.

Enostavnost uporabe: Osnovne PHP skripte lahko napišejo tudi uporabniki brez podrobnega razumevanja programskih osnov, prevajanja izvorne kode v strojno kodo in drugih pomembnih programskih konceptov.

Programerji vajeni Jave, Perl-a, BASIC-a in drugih popularnih jezikov lahko opazijo veliko podobnosti, ki jim olajšajo prehod na PHP.

PHP je fleksibilen, možna je uporaba čistega postopkovnega ali čistega objektnega programiranja, prav tako je možno uporabljati oba pristopa hkrati.

PHP uporablja ohlapno definiranje podatkovnih tipov, kar pomeni da ni potrebno določiti tipa spremenljivke in ga sistem določi glede na vsebino. V mnogih primerih ko sestavljamo različne spremenljivke pretvarjanje med tipi ni potrebno, kar pomeni občuten prihranek časa.

Teče na veliko različnih operacijskih sistemih. V časovno kritičnih sistemih se PHP lahko optimizira za performanse, podobne drugim uveljavljenim jezikom.

1.1.3. Slabosti

Osnovni PHP se izvaja počasneje kot koda napisana v zbirnem jeziku, C-ju in drugih prevajalnih jezikih.

Lastnost jezika PHP, da uporablja ohlapno definiranje podatkovnih tipov, je hkrati tudi slabost. Manj pazljivim razvijalcem lahko povzroča pojav mnogih nepričakovanih obnašanj kode zaradi razvijalčevih napak, ki jih drugi programski jeziki ne bi dovolili.

Obstaja mnogo poti za realizacijo enake funkcionalnosti in mnogo primerov funkcij, ki imajo možnost različne uporabe, zaradi podpore starejših različic ali zgodovine razvoja.

PHP nima notranje podpore za razhroščevanje, zato je potrebna uporaba zunanjih orodij.

2. Primerjava postopkovnega in objektnega programiranja s PHP

Pri nespletnih programskih jezikih je objektno programiranje že dolgo razširjeno in prevladuje v večini primerov. Razen za posebne namene, se večinoma uporabljajo samo še objektni jeziki kot so Java, C#, C++ in nekateri drugi. Pri spletnih skriptnih jezikih oziroma pri PHP-ju je stanje drugačno. PHP je že od različice 3.0 podpiral objektno programiranje, vendar je bilo le to precej manj zmogljivo od pravih objektnih jezikov. PHP pravzaprav ni strogo objekten, temveč je hibrid. Lahko uporabljamo tradicionalno postopkovno programiranje, strogo objektno programiranje, možna pa je tudi mešanica obojega. Zaradi možnosti mešanja obeh stilov so razvijalci veliko uporabljali predvsem postopkovno programiranje, v kodo pa so vnašali objekte za podatkovne elemente, za katere je bilo to smiselno. Tak način programiranja se za manjše in srednje velike aplikacije uporablja še danes. Vendar pa je prednosti pri objektnem programiranju veliko in PHP je moral slediti trendom, zato je z različico 5.0 dobil povsem nov objektni del, ki je mnogo zmogljivejši od starega. PHP 5 je povsem primerljiv s sodobnimi objektno orientiranimi jeziki. Če je bil PHP v preteklosti zaradi svoje enostavnosti in ohlapnosti najbolj primeren za manjše in srednje velike projekte, je z izboljšano podporo objektnemu programiranju postal tudi smiselna izbira pri velikih in kompleksnih projektih.

2.1. Izbira tehnike

Za katero tehniko se odločimo, je odvisno predvsem od velikosti projekta, usposobljenosti razvijalcev in potrebe po ponovni uporabljivosti kode. Za majhne nezahtevne aplikacije včasih ni smiselno razvijati razredov in je realizacija na postopkoven način povsem ustrezna. Za večje in zahtevnejše aplikacije, ki imajo precej kompleksnejši podatkovni model, je uporaba objektnega programiranja priporočljiva. Objektno programiranje ima torej veliko prednosti. Delo se lažje razdeli na več programerjev in ni potrebno podvajati kode, zaradi tega je tudi kasnejše vzdrževanje enostavnejše. Velika prednost je tudi lažja ponovna uporabljivost razredov, saj se s pomočjo dedovanja in podobnih mehanizmov da razviti bolj standardne in modularnejše aplikacije, ki jih je lažje preurediti za nove potrebe.

2.2. Problem povezave objektov s podatki iz baze

Pri postopkovnem programiranju navadno za pridobivanje zapisov iz podatkovne baze uporabljamo poizvedbe SQL, katerih rezultati so zapisani v podatkovnih poljih. Ta polja po potrebi preuredimo in jih direktno uporabimo za potrebe obdelave in izpisa.

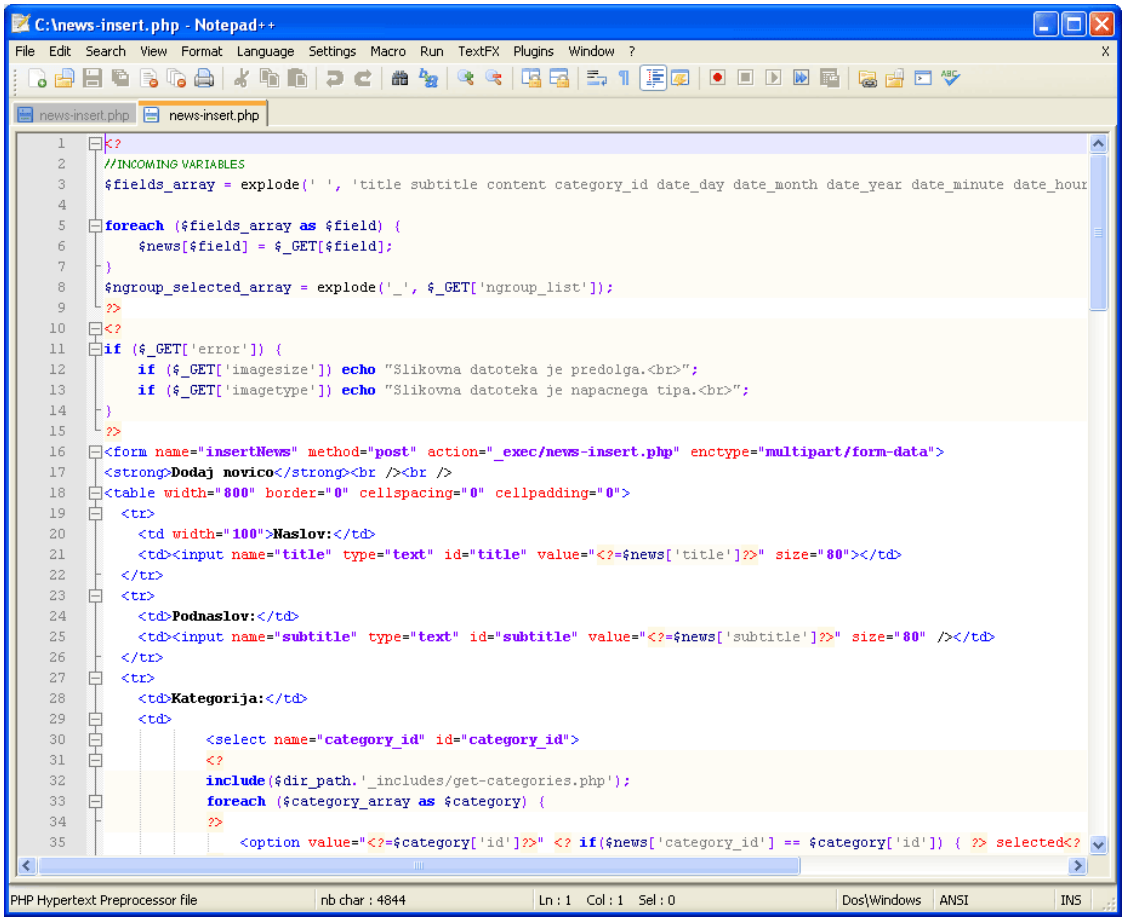
Če programiramo objektno, se rezultatov poizvedb SQL ne da zapisati direktno v objekte. V tem primeru moramo ustrezne poizvedbe vgraditi v metode razredov, prav tako pa tudi metode za dodajanje, spreminjanje in brisanje, ki vsebujejo specifične stavke SQL.

Ker je ta način lahko precej zahteven in časovno potraten, si lahko pomagamo z orodji ORM za preslikovanje med objekti in relacijami (Object-Relational Mapping). Primer takega orodja je Propel, ki je objektno-relacijsko preslikovalno ogrodje za PHP 5. Omogoča nam enostavno dostopanje do podatkov in njihovo shranjevanje, spreminjanje in brisanje preko množice generiranih razredov in metod. Na ta način nam ni potrebno skrbeti, kako se podatki berejo iz podatkovne baze, prav tako nam ni potrebno pisati stavkov SQL.

3. Razvojna okolja

3.1. Notepad in podobni urejevalniki golega besedila

Glede na to da sta PHP in HTML koda povsem tekstovni, je za njuno pisanje možno uporabiti preprost urejevalnik golega besedila. Vendar pa nam preprosti urejevalniki, kot je Notepad, ne nudijo nobenih pomagal, zato je tak razvoj težji in precej manj pregleden. Nekoliko boljši urejevalniki podpirajo ustrezno barvanje kode, kar precej prispeva k berljivosti, vendar pa boljša orodja nudijo še veliko več.

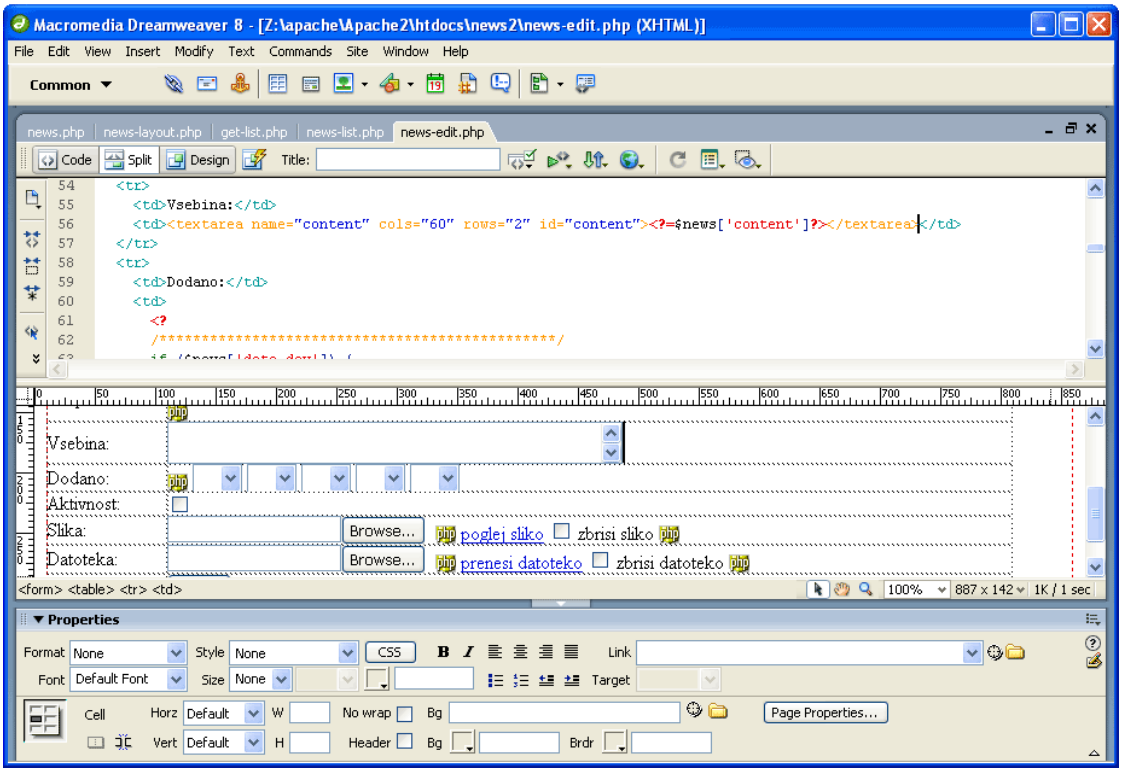


Slika 3.1 – prikaz PHP in HTML kode v urejevalniku Notepad++

3.2. Dreamweaver

Adobe Dreamweaver je orodje, namenjeno razvoju spletnih strani in podpira mnoge spletne jezike, ima pa tudi zelo dober HTML urejevalnik. Pomembna lastnost je hiter preklop med prikazom kode in predogledom, zato je razvoj HTML kode in vgrajene PHP kode precej bolj prijazen in pregleden. Datoteke ni potrebno za vsako spremembo kopirati na strežnik in si ogledati preko brskalnika, temveč lahko predogled HTML kode vidimo že v samem

programu. Za uspešen predogled moramo uporabljati ustrezno tehniko gradnje HTML kode. Za večjo preglednost je prikaz kode ustrezno obarvan. Vgrajeno PHP kodo se dobro loči od HTML kode predloge. Urejevalnik ima možnost hitrega preklopa med pogledom kode in predogledom, možno je tudi uporabljati oba pogleda hkrati.



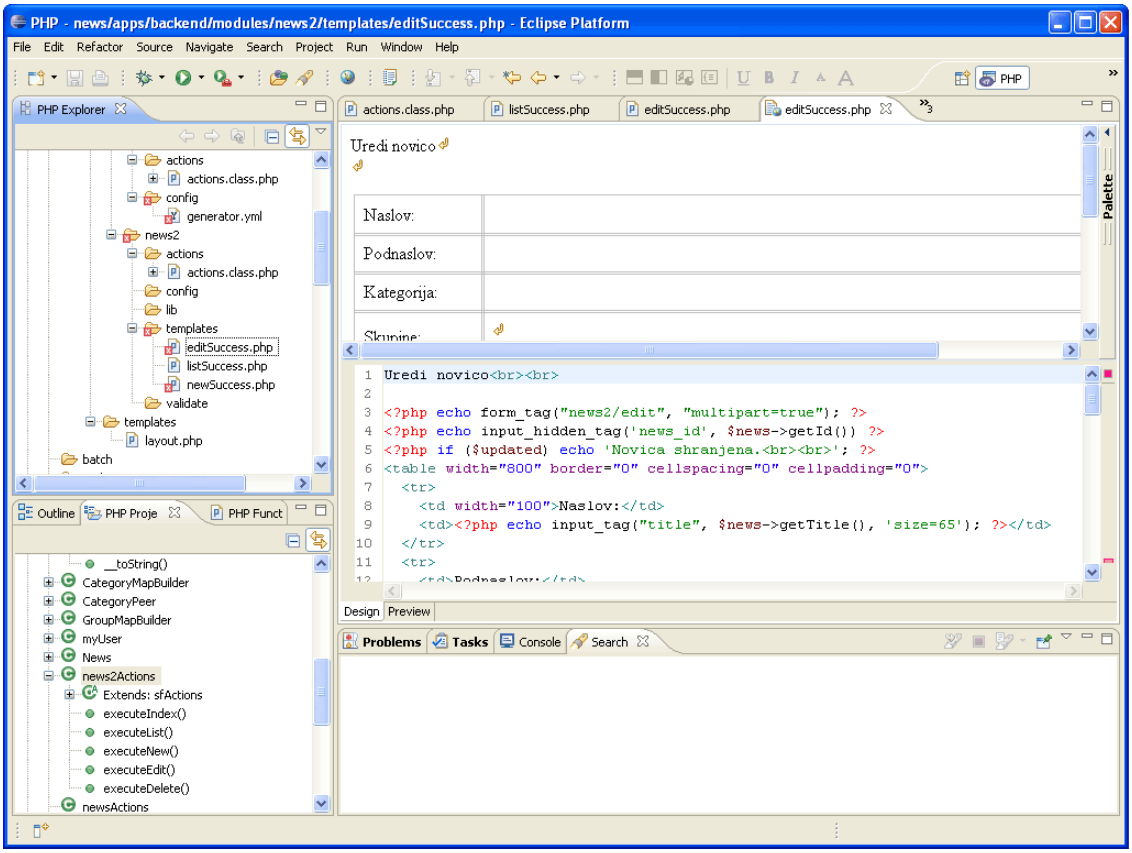
Slika 3.2 – mešan prikaz kode in predogleda v orodju Dreamweaver

Pri predogledu lahko vplivamo na parametre HTML elementov preko vmesnikov, kar je hitrejše in enostavnejše kot ročno pisanje kode. Kljub temu da urejevalnik podpira jezik PHP, pa je njegova podpora precej šibkejša kot za HTML. Podpora jeziku PHP je na nivoju barvanja kode in prikazu parametrov PHP funkcij, ne podpira pa dopolnjevanja pri uporabi objektov in le delno preverja sintaktično pravilnost kode.

Zelo uporabna možnost je vgrajen upravljalnik datotek, ki ločuje med lokalnimi datotekami za potrebe pregledovanja in pisanja, ter datotekami na strežniku, ki služijo delovanju spletne strani. Upravljalnik datotek se s strežnikom poveže preko protokola FTP in nam omogoča hitro kopiranje datotek s strežnika na razvojni računalnik. V obratni smeri popravljene datoteke prenese nazaj na strežnik, kar storimo s pritiskom na ustrezno kombinacijo tipk. S tem se delo občutno optimizira, kopiranje pa poenostavi. Urejevalnik datotek nam omogoča tudi sinhronizacijo med lokalnimi datotekami in datotekami na strežniku. V primeru razvoja več razvijalcev omogoča zaklepanje datotek. Delo na datoteki je tako dovoljeno samo enemu razvijalcu in s tem preprečeno, da bi kdo prepisal že spremenjeno datoteko.

3.3. Eclipse

Eclipse je zelo znano razvojno orodje, ki se uporablja predvsem pri razvoju Java in drugih aplikacij, vendar pa je različica Eclipse PDT razvita prav za razvoj PHP projektov. Eclipse je zelo zmogljivo orodje ravno zato, ker zanj obstaja množica dodatkov (plugin), s katerimi si olajšamo delo. Zelo uporabna dodatka sta XDebug in Zend Debugger, ki omogočata razhroščevanje znotraj Eclipse-a na nivoju PHP skripte ali PHP spletne strani. Eden od zanimivejših dodatkov je tudi podpora orodju Subversion, ki je sistem za nadzor vzporednih različic izvorne kode. Orodje nudi več perspektiv ogleda kode, uporaben pa je tudi imenik map, s pomočjo katerega hitro dostopamo do datotek. Podpora objektnemu programiranju je odlična, saj za vse objekte projekta prikaže spremenljivke in metode, za hitrejši in enostavnejši razvoj. Konstante, razredi in funkcije so zbrane v posebnem okencu, v katerem lahko hitro poiščemo željen razred in pregledamo njegovo definicijo, brez da bi bilo potrebno dostopati do implementacijskih datotek. Eclipse podpira tudi razvoj HTML kode in njen predogled, vendar je ta del pri orodju Dreamweaver občutno zmogljivejši.



Slika 3.3 – prikaz kode, predogleda, seznama datotek in razredov v orodju Eclipse

3.4. Katero razvojno okolje izbrati

Opisal sem samo najbolj razširjena in meni poznana razvojna okolja. Obstaja pa jih pravzaprav velika množica in vsako ima kako pomembno prednost. Izbira orodja je sicer subjektivna odločitev, vendar pa ravno zaradi prednosti in slabosti zelo pomembna in strokovno zahtevna. Za razvoj HTML kode in postopkovnega programiranja s PHP je Dreamweaver idealna izbira. Če pa delamo z ogrodji in objektno tehnologijo, je Eclipse uporabnejši. Osebno bom za razvoj PHP kode in osnovnih predlog pri razvoju z ogrodjem uporabljal Eclipse, za kasnejše popravke videza pa Dreamweaver, saj ima boljšo podporo za HTML in CSS kodo.

4. Arhitektura programske kode pri klasičnem razvoju

4.1. Organiziranje kode po datotekah

Ker struktura map in PHP datotek ni predpisana, je odvisno predvsem od programerja oziroma dogovorjenih standardov med razvijalci, na kakšen način naj se koda gradi. Zaradi tega so si aplikacije lahko arhitekturno zelo različne, tako za razumevanje kot tudi po kakovosti.

4.1.1. Prikaz vseh strani preko ene datoteke ali ločitev na več PHP datotek

Vsaka osnovna stran se navadno začne z datoteko index.php, ki je privzeta datoteka za prikaz v mapi. Podstrani se lahko nahajajo v ločenih PHP datotekah, lahko pa poskrbimo da se vse podstrani odpirajo preko index.php ali drugače poimenovane skupne datoteke. Izbira teh tehnik zahteva različno strukturiranje kode, ki je potrebno, da kodo ohranimo neredundantno.

4.1.2. Prikaz vseh strani preko ene datoteke

V tem primeru je index.php zunanja ovojnica vse kode, ki je potrebna za generiranje končne prikazne HTML kode. Na začetek ponavadi vključimo vse potrebne nastavitve za delovanje, vključimo potrebne knjižnice in povezave s podatkovnimi bazami, na koncu pa povezavo zaključimo. Vmes je potrebno poskrbeti za dostopanje do podatkov, ustrezno formatiranje in sestavo HTML kode. Ker različne strani prikazuje ista datoteka, je potrebno uporabiti parametre, s katerimi določamo katere vsebine se morajo prikazati.

index.php

```
<?php
//vključitev nastavitev
include('_settings/settings.php');
//vključitev knjižnic
include('_library/library.php');
//vključitev dostopa do podatkovne baze
include('_database/db_connect.php');

//dostop do spremenljivk potrebnih za krmiljenje strani
$stran = $_GET['stran'];

//glede na spremenljivko stran vključujemo PHP datoteke za dostop do
ustreznih podatkov in generiranje prikazne html kode

//vključitev zaprtja dostopa do podatkovne baze
include('_database/db_disconnect.php');
?>
```

4.1.3. Prikaz strani preko ločenih datotek

Če imamo za vsako podstran ali vsaj sklop podstrani ločeno PHP datoteko, do katere se dostopa direktno, je potrebno zaradi neredundatnosti podatkov pripraviti PHP datoteke, ki bodo vsebovale kodo skupno vsem podstranem. V PHP datoteke preko funkcije include vključimo te datoteke na začetek in konec kode ter s tem poskrbimo, da je ob spreminjanju potrebno spremeniti kodo samo na enem mestu.

_glava.php

```
<?php
//vključitev nastavitev
include('_settings/settings.php');
//vključitev knjižnic
include('_library/library.php');
//vključitev dostopa do podatkovne baze
include('_database/db_connect.php');
?>
```

_noga.php

```
<?php
//vključitev zaprtja dostopa do podatkovne baze
include('_database/db_disconnect.php');
?>
```

index.php

```
<?php
//vključitev kode glave
include('_glava.php');

//koda specifična za prvo stran, ne potrebujemo posebnih spremenljivk da bi
vedeli katero osnovno vsebino je potrebno prikazati, saj je to določeno z
ločitvijo na več datotek.

//vključitev kode noge
include('_noga.php');
?>
```

podstran.php

```
<?php
//vključitev kode glave
include('_glava.php');

//koda specifična za stran »podstran«, ne potrebujemo posebnih spremenljivk
da bi vedeli katero osnovno vsebino je potrebno prikazati, saj je to
določeno z ločitvijo na več datotek.

//vključitev kode noge
include('_noga.php');
?>
```

4.2. Organiziranje map in posebnih datotek

4.2.1. Datoteke z nastavitvami

Nastavitve je priporočljivo ločiti od preostale kode, da se jih kasneje spreminja samo na enem mestu in hkrati ne spreminjamo delov kode, ki se ne spreminjajo, temveč nastavitve le uporabljajo. Če uporabljamo modularno zasnovo se nastavitve loči na več datotek, za vsak modul svojo. S tem lahko kasneje kombiniramo obstoječe module na novih aplikacijah, brez da bi bilo potrebno spreminjati vsebino datotek, temveč se kopira samo potrebne datoteke. Datoteka, ki jo kličemo za vključitev vseh nastavitev, poskrbi da se vključijo vse datoteke v mapi z nastavitvami, kar omogoča spreminjanje imen nastavitvenih datotek in dodajanje novih, brez poseganja v kodo, ki jih uporablja.

4.2.2. Datoteke s funkcijami

Funkcije je podobno kot nastavitve smiselno ločiti od kode, ki jih uporablja, zaradi neredundantnosti, preglednosti in ponovne uporabljivosti. Funkcije združimo v datoteke po njihovem namenu in z vsakim novim projektom se knjižnica funkcij poveča. Urejenost po datotekah omogoča uporabo samo tistih družin funkcij, ki jih projekt potrebuje.

4.2.3. Datoteke za povezovanje s podatkovnimi bazami

Za vsak prikaz strani, kjer beremo podatke iz podatkovne baze, je potrebno vzpostaviti povezavo s strežnikom podatkovne baze in izbrati ustrezno podatkovno bazo. Z datotekami za povezovanje s podatkovno bazo poskrbimo, da je povezava zapisana samo na enem mestu in je v primeru sprememb potrebno popraviti samo eno datoteko.

4.3. Različne realizacije izvrševanja akcij

Spletne aplikacije ne prikazujejo samo podatkov iz podatkovne baze, temveč omogočajo tudi interakcijo uporabnika s sistemom. To so lahko enostavne forme na javnih straneh za dodajanje komentarjev ali pošiljanje vprašanj, lahko pa so tudi precej zahtevnejše forme, ki jih najdemo v administracijskih okoljih. Ne glede na mesto kjer se forme pojavijo, je potrebno akcije izvršiti. Njihovo izvrševanje in strukturiranje kode, ki jo te akcije sprožijo je lahko zelo različno. Narava akcij je taka, da se z njimi večinoma obdela vnešene podatke, se le te zapiše v bazo, se osveži že obstoječe podatke v bazi ali pa se zapise v baze briše, ter še mnoge kombinacije.

4.3.1. Koda akcij je v prikazni PHP datoteki

Ena od rešitev, ki se pogosto pojavlja, je vključevanje kode, ki jo sprožijo akcije, kar v isto datoteko, ki skrbi za prikaz podatkov in forme. Na ta način se zmanjša število datotek, koda pa je bolj lokalizirana. Vendar pa ima ta tehnika svoje slabosti. Datoteka je daljša, manj pregledna, težje razumljiva in potrebnih je več parametrov za ustrezno krmiljenje. Pri tej tehniki se vse začne z izpisom kode za prikaz forme, katere destinacija je datoteka sama. Ob oddaji podatkov forme datoteka kliče samo sebe z novimi parametri, ki določijo da se izvede ustrezni akcijski del kode. Nato pa je potrebno narediti novo preusmeritev, tokrat brez parametrov forme, vendar z morebitnimi sporočili uspešnosti akcije. Brez preusmeritve bi uporabniku omogočili, da ob osvežitvi strani ponovno izvede akcijsko kodo, kar pa v večini primerov ni pravilno.

Primer datoteke s formo in akcijo:

index.php

```
<?php
if ($_POST['akcija']) {
   //izvedi vse potrebno za akcijo
}
//preusmeritev s parametri uspešnosti
header('Location: index.php?status='.$status);
exit;
}
?>
Forma za izvrševanje akcije
<form action="index.php" method="post">
<input name="vsebina" type="text" />
<input type="submit" value="Oddaj" />
</form>
```

4.3.2. Koda akcij je v namenskih datotekah

Ideja te izvedbe je ločitev akcijske kode od prikazne kode strani. Za vsako formo se naredi nova izvršna PHP datoteka, ki jo forma kliče namesto prikazne strani. V primeru velikega števila form se lahko nabere tudi veliko teh datotek, vendar je urejenost in razumljivost precej boljša. Akcijske PHP datoteke združimo v posebno mapo, namenjeno samo njim. Oddana forma se preusmeri na nastavljeno akcijsko stran, ki vsebuje samo akcijsko kodo in ničesar ne izpisuje. Ob zaključku izvajanja se datoteka preusmeri nazaj na prikazno stran. Pri tem načinu je koda prikazne strani precej čistejša in preglednejša.

Primer datoteke s formo in akcijske datoteke:

index.php

```
Forma za izvrševanje akcije
<form action="index.php" method="post">
<input name="vsebina" type="text" />
<input name="povratnapot" type="hidden" value="index.php">
<input type="submit" value="Oddaj" />
</form>
```

_exec/akcija.php

```
<?php
if ($_POST) {
      //izvedi vse potrebno za akcijo
//preusmeritev s parametri uspešnosti
header('Location: ../'.$_POST['povratnapot'].'?status='.$status);
exit;
}
?>
```

4.4. Načini sestavljanja HTML kode

Ko strežnik dobi zahtevo po določeni spletni strani, katere datoteka je označena s končnico php, se začne izvajanje kode. Vendar pa se izvaja samo kodo med mesti, ki so označena s kombinacijo znakov za začetek in konec PHP kode. Ti znaki so <?php _koda_ ?>, lahko pa se uporabi tudi krajša različica <? _koda_ ?>, ki pa je manj prenosljiva in jo je zato možno v nastavitvah izklopiti. Njena prednost je v tem, da vsebuje manj znakov in je zato koda krajša in preglednejša. To je evidentno predvsem pri vgrajevanju PHP kode v HTML kodo, kjer samo izpisujemo vsebino spremenljivk in je veliko začetkov in koncev PHP kode. Ker PHP izvaja samo svojo označeno kodo, je več možnosti sestavljanja končne HTML kode, ki jo prikazujejo brskalniki.

Eden od osnovnih načinov je vgrajevanje PHP kode v HTML kodo. Dolgo uporabljan način je tudi sestavljanje HTML kode z izpisovalnimi funkcijami v PHP kodi, nekoliko novejša rešitev pa je uporaba predložnega stroja (template engine).

4.4.1. Vgrajevanje PHP kode v HTML kodo

Glede na to da PHP izvaja samo svojo kodo, lahko datoteka vsebuje celotno HTML kodo, ki je v večjem delu lahko statična, z vgrajevanjem PHP kode pa poskrbimo za tiste dele, kjer je potrebna dinamika. Na ta način nam je omogočen dober predogled HTML kode, če uporabljamo razvojno okolje, ki tak predogled podpira. S tem lepo vidimo obliko strani in nam je kasnejše spreminjanje HTML kode precej poenostavljeno. Vendar pa ima vgrajevanje

tudi svoje slabe lastnosti. V primeru da izvajamo v vgrajeni PHP kodi precej kontrolnih in podatkovnih operacij, dobimo skupek kode, ki združuje celotno podatkovno, krmilno in prikazno plast. To ni skladno z najnovejšimi načeli, ki priporočajo ločitev teh treh plasti. Zato se ta način poskuša nadomestiti z novejšimi, boljšimi.

Primer v preglednejšem, vendar manj standardnem načinu:

index.php (datoteka s HTML kodo, v katero je vgrajena PHP koda)

```
<Html koda>
<?
//PHP koda za branje naslova in seznama iz podatkovne baze
?>
<h1><?=$naslov?></h1>
<table>
<? foreach ($seznam as $element) { ?>
  <tr>
    <td><?=$element?></td>
  </tr>
<? } ?>
</table>
<Html koda>
```

Primer v manj preglednem, vendar bolj standardnem načinu:

index.php (datoteka s HTML kodo, v katero je vgrajena PHP koda)

```
<Html koda>
<?php
//PHP koda za branje naslova in seznama iz podatkovne baze
?>
<h1><?php echo $naslov; ?></h1>
<table>
<?php foreach ($seznam as $element) { ?>
  <tr>
    <td><?php echo $element; ?></td>
  </tr>
<?php } ?>
</table>
<Html koda>
```

4.4.2. Sestavljanje HTML kode s PHP

Poleg vgrajevanja kode, je to najstarejši način in se ga najde v mnogih predvsem starejših aplikacijah. Pri tem načinu je celotna datoteka napisana samo v PHP kodi, torej se izvaja celotna datoteka. Za izpisovanje HTML kode se uporabljajo PHP funkcije kot so echo, print in print_r, ki podpirajo izpisovanje spremenljivk, podatkovnih polj in podobnega. Ta način je mogoče najbolj primeren za skripte, ki se pravzaprav ne izvajajo preko spletnega brskalnika, ampak v ukazni lupini oziroma je izpis zelo skop in neformatiran. V primeru, da hočemo

izpisati celotno HTML kodo spletne strani, je potrebno vse dele HTML kode zapisati v tekstovne spremenljivke, jih med seboj ustrezno sestaviti in nato s funkcijami izpisati. Za obiskovalca strani pravzaprav ni pomembno kako se HTML koda sestavi, za razvijalca pa zelo. Ker je HTML koda zapisana v več spremenljivkah, predogled v razvojnih okoljih ni mogoč. Prav tako nam razvojno orodje ne more prikazati morebitnih napak v HTML kodi, ker je ne zna izluščiti in ustrezno sestaviti iz PHP spremenljivk. Tudi kasnejše vzdrževanje in spreminjanje HTML kode je lahko zelo kompleksno in časovno potratno. Eden od problemov je tudi, da kodo učinkovito in hitro lahko spreminja skoraj izključno razvijalec, ki je napisal preostalo PHP kodo, ker po navadi le on zadostno razume, kako se koda sestavi v celoto.

Primer sestavljanja HTML kode s PHP:

index.php (datoteka ki vsebuje samo PHP kodo)

```
<?php
//PHP koda za branje naslova in seznama iz podatkovne baze
echo "<h1>".$naslov."</h1>";
echo "<table>";
foreach ($seznam as $element) {
  echo "<tr><td>".$element."</td></tr>";
}
echo "</table>";
?>
```

4.4.3. Uporaba predložnega stroja za gradnjo HTML kode

Zaradi omenjenih slabosti prejšnjih načinov gradnje HTML kode, je bilo potrebno razviti nove načine, ki bi slabosti odpravili in zadostili mnogim zahtevam in željam razvijalcev. Uporaba predložnega stroja nam prinese mnoge prednosti. Gradnja HTML kode poteka tako, da s PHP določimo podatkovne spremenljivke, ki jih zna brati predložni stroj, prav tako moramo določiti datoteko, v kateri je HTML predloga v jeziku predložnega stroja. Na ta način poskrbimo, da se v PHP kodi izvaja samo podatkovna in kontrolna plast, prikaz pa je v ločeni datoteki. Datoteka s predlogo vsebuje HTML kodo v katero se vgrajuje kodo predložnega stroja, s katero poskrbimo za dinamiko. Koda predložnega stroja se uporablja za prikazovanje podatkov iz spremenljivk in polj ter za sestavljanje različnih delov HTML kode odvisno od kontrolnih spremenljivk. Računanja in podobnih kompleksnejših operacij naj s kodo predložnega stroja ne bi izvajali, kljub temu da to omogoča. Dobra lastnost te tehnike je možnost predogleda HTML kode. Zaradi ločenosti od podatkovno/krmilnega dela je razumevanje lahko omejeno na samo predlogo. Zato lahko predlogo razvija in spreminja razvijalec, ki PHP jezika pravzaprav sploh ne pozna in se lahko osredotoči na HTML kodo in precej enostavno kodo predložnega stroja. Dobra lastnost je tudi enostavna in hitra zamenjava videza zaslonske maske v aplikacijah, kjer je to potrebno. Predložni stroj, ki je zelo popularen v kombinaciji z jezikom PHP, se imenuje Smarty.

Primer uporabe predložnega stroja Smarty:

index.php (glavna krmilna datoteka, vsebuje samo PHP kodo)

```
//PHP koda za branje naslova in seznama iz podatkovne baze
//inicializacija predložnega stroja

//določitev naslova, zapisanega v $naslov
$tpl->assign('naslov', $naslov);

//določitev seznama, zapisanega v polju $seznam
$tpl->assign('seznam', $seznam);

//določitev in prikaz predloge
$tpl->display('predloga.tpl');
```

predloga.tpl (datoteka s HTML in kodo predložnega stroja)

```
<h1>{$naslov}</h1>
<table>
{foreach from=$seznam item=element}
  <tr>
    <td>{$element}</td>
  </tr>
{/foreach}
</table>
```

4.4.4. Simuliranje predložnega stroja s PHP kodo

Uporabi predložnega stroja pa se lahko tudi izognemo in ga simuliramo z uporabo PHP kode. Pri tem načinu je potrebno ustrezno strukturirati PHP datoteke in končni rezultat je zelo podoben uporabi predložnega stroja, vendar s to razliko, da v predlogi za prikaz ne uporabljamo vgrajevanja kode predložnega stroja, temveč kar PHP kodo. S tem ohranimo dober predogled HTML kode. Prav tako zaradi ustrezne delitve datotek na podatkovno PHP datoteko in datoteko za predlogo poskrbimo, da je predložna datoteka čista in jo lahko spreminja razvijalec z osnovnim znanjem jezika PHP. Osnovna PHP datoteka vsebuje samo PHP kodo, ki jo potrebujemo za kontrolo in pridobivanje podatkov. Tehnika nam sicer ne nudi določenih dodatnih funkcionalnosti, ki jih ima pravi predložni stroj, zato moramo zanje ročno poskrbeti, če jih seveda potrebujemo.

Primer simulacije predložnega stroja:

index.php (glavna krmilna datoteka, vsebuje samo PHP kodo)

```
//PHP koda za branje naslova in seznama iz podatkovne baze

//vključitev datoteke s predlogo v index.php
include('predloga.tpl');
```

predloga.tpl (datoteka s HTML in PHP kodo)

```
<h1><?=$naslov?></h1>
<table>
<? foreach ($seznam as $element) { ?>
  <tr>
    <td><?=$element?></td>
  </tr>
<? } ?>
</table>
```

5. PHP ogrodje Symfony

Ogrodje pospeši razvoj aplikacij, saj avtomatizira in poenostavi marsikateri korak. Predpisana je tudi struktura kode in razvijalec je prisiljen, da piše boljšo in preglednejšo kodo, ki jo je tudi lažje vzdrževati. Symfony je celostno ogrodje, ki optimizira ključne značilnosti razvoja. Ena od pomembnih lastnosti je ločitev poslovnih pravil, strežniške logike in prikaza. Vsebuje veliko število razredov, ki skrajšajo čas potreben za razvoj. Avtomatizirana so nekatera veliko uporabljana opravila, zato se razvijalec lahko v celoti posveti specifikam aplikacije.

Ogrodje Symfony je v celoti napisano v PHP 5, je podrobno testirano in se uporablja tudi pri velikih in kompleksnih spletnih aplikacijah. Podpira večino podatkovnih baz in teče na Unix, Linux in Windows okoljih.

Lastnosti:

- enostavnost namestitve in nastavitve na večino platform
- neodvisnost od specifične podatkovne baze
- enostavnost uporabe, vendar vseeno dovolj velika fleksibilnost za kompleksne primere
- upoštevanje večine najboljših tehnik in načinov razvoja
- pripravljenost na velike projekte in stabilnost za dolgotrajne projekte
- preglednost kode za enostavno vzdrževanje
- AJAX interakcije so enostavne za implementacijo, saj knjižnice poskrbijo za združljivost kode JavaScript na različnih brskalnikih

Razvojno okolje in orodja:

Da bi zadostili potrebam podjetij, ki uporabljajo svoje standarde in pravila programiranja, je Symfony možno v celoti prilagoditi. Privzeto podpira različna razvojna okolja in ima vključena orodja za avtomatizacijo pogostih razvojnih opravil:

- orodja za generiranje kode so odlična za prototipiranje in izredno hitro izgradnjo osnovnih administracijskih okolij
- vgrajeno testno ogrodje omogoča testno voden razvoj
- meni za razhroščevanje pospeši iskanje napak, saj prikazuje vse informacije, ki jih razvijalec potrebuje na trenutni strani
- vmesnik ukazne vrstice avtomatizira prenos aplikacije med dvema strežnikoma
- možnost beleženja dnevnika sprememb administratorjem omogoči pregled nad celotnimi aktivnostmi, ki se izvajajo na aplikaciji

5.1. Razvoj ogrodja Symfony

Prva različica ogrodja Symfony je izšla oktobra 2005, ustanovitelj pa je Fabien Potencier, ki je CEO francoskega podjetja Sensio, znanega po razvoju inovativnih rešitev za spletne strani. Leta 2003 je avtor iskal PHP ogrodje, ki bi bilo na osnovi odprte kode, vendar ni noben projekt ustrezal njegovim zahtevam. Ob izidu PHP 5, pa je ugotovil da so vsa potrebna orodja dosegla dovolj zrelo stopnjo, da se jih integrira v celostno ogrodje. Razvoj jedra je trajal eno leto, temeljil pa je na ogrodju Mojavi Model-View-Controler, Propel-u za objektno-relacijsko preslikavo in Ruby on Rails za predložne funkcije (helper). Prvotno je bilo ogrodje razvito za avtorjevo podjetje Sensio, kajti uporaba učinkovitega ogrodja omogoča hitrejši in učinkovitejši razvoj aplikacij. Razvoj je bolj intuitiven in aplikacije so robustnejše ter enostavnejše za vzdrževanje. Ko je bilo ogrodje uspešno uporabljeno na več projektih, ga je avtor izdal pod licenco odprte kode. S tem je javnosti podaril odlično orodje, hkrati pa tudi sam pridobil z odzivom uporabnikov in podjetju naredil dobro reklamo. Za svojo uspešnost je ogrodje potrebovalo obsežno dokumentacijo v angleškem jeziku, sicer bi bilo precej manj enostavno za priučitev in uporabo.

5.2. Uporabniki ogrodja Symfony

Z izidom spletne strani projekta, je veliko razvijalcev namestilo Symfony in zanimanje je bilo vedno večje. Ogrodja so v tistem času postala zanimiva in popularna, Symfony pa je na račun kakovostne kode in dobre dokumentacije postal razumljiva izbira. Kmalu so se pojavili posamezniki, ki so predlagali popravke in izboljšave, izboljšala se je tudi dokumentacija. Uporabnikom je na voljo javni repozitorij kode, avtor Fabien pa še vedno prispeva glavni del kode in skrbi za njeno kakovost. Symfony skupnost danes obsega forum, poštne sezname in IRC kanale, zato je enostavno priti do pomoči.

5.3. Komu je namenjeno ogrodje Symfony

Symfony lahko uporabljamo ne glede na izkušnje razvijalca, seveda so le te pomembne, vendar pa je glavni dejavnik za odločitev o uporabi ogrodja velikost projekta. Za majhne nezahtevne strani, kjer se ne potrebuje zahtevnih operacij in funkcionalnosti, uporaba ogrodja ni smiselna, saj z njegovo uporabo ne bi veliko pridobili, nekatera opravila pa bi zahtevala še več časa. Na drugi strani pa je uporaba ogrodja zelo priporočljiva pri večjih projektih, kjer je veliko poslovne logike. Realizacija takih projektov z golim PHP bi bila precej težja, dolgotrajnejša, zahtevnejše pa bi bilo tudi vzdrževanje. Z uporabo ogrodja tako pridobimo na kakovosti in berljivosti kode, kode je manj, precej manjša je tudi redundantnost. Omogoča tudi precej hitrejši razvoj AJAX funkcionalnosti, ki so značilne za sodobne aplikacije.

5.4. Objektno-relacijska preslikava

Za hranjenje podatkov se večinoma uporablja relacijske podatkovne baze. PHP 5 in Symfony pa sta objektno orientirana. Za dostopanje do podatkovne baze na objektni način potrebujemo vmesnik za preslikanje objektne logike v relacijsko logiko. Tak vmesnik se imenuje ORM (object-relational mapping). ORM sestavljajo objekti, ki omogočajo dostop do podatkov v bazi, hkrati pa vsebujejo tudi poslovna pravila. Ena od prednosti objektno-relacijske abstrakcijske plasti je uporaba sintakse, ki ni specifična za posamezno podatkovno bazo. Klici objektov se avtomatsko prevedejo v stavke SQL, ki so optimizirani za trenutno uporabljeno podatkovno bazo. Zamenjava podatkovne baze med projektom je torej enostavna, zato se lahko razvoj aplikacije začne še preden se stranka odloči katero podatkovno bazo bo uporabljala. Prehod na drugo podatkovno bazo zahteva samo spremembo konfiguracijske datoteke. Abstrakcijska plast ograjuje podatkovno logiko, saj preostanek aplikacije ne potrebuje vedeti česarkoli o stavkih SQL. Uporaba objektov namesto zapisov in razredov namesto tabel ima še dodatno prednost. Omogoča dodajanje izpisa izpeljanih podatkov, ki niso zapisani v podatkovni bazi, so pa izpeljani iz zapisanih. Vsa funkcijska logika za dostopanje do podatkov in poslovna logika je zato lahko zapisana v teh objektih. Primer bi lahko bila nakupovalna košarica, v kateri hranimo postavke, ki so objekti. Za dostop do skupnega zneska košarice lahko dodamo naslednjo metodo.

```
public function skupniZnesek() {
  $skupaj = 0;
  foreach ($this->getPostavke() as $postavka)
  {
    $skupaj += $postavka->getCena() * $postavka->getKolicina();
  }
  return $skupaj;
}
```

Na primeru se vidi kako hitra in enostavna je poizvedba, prav tako pa je dostopna preko objekta na katerikoli strani. Priti do enakega podatka na klasičen način, bi zahtevalo dosti več dela. Potrebno bi bilo napisati poizvedbe SQL specifično za namen, poleg tega bi bilo omenjeno kodo potrebno ponoviti povsod kjer bi želeli podatek prebrati.

Symfony uporablja ORM z imenom Propel, ki je prav tako projekt odpre kode in je trenutno najboljše tovrstno orodje za PHP 5. Glede na to, da je Propel integriran v Symfony, se uporablja tudi Propel-ova sintaksa. Z različico 1.1 pa je vključen kot dodatek (plugin), tako da je prehod na drug ORM enostavnejši. Za uporabo konkurenčnega ORM Doctrine je tako potrebno naložiti ustrezen dodatek.

5.5. Rapid Application Development (RAD)

Programiranje spletnih aplikacij je bilo v preteklosti dolgotrajno in počasno. Razvoj se je navadno začel šele po obsežni analizi in načrtovanju, prav tako pa je bilo kasnejše spreminjanje zahtev neugodno. Glede na to, da se poslovanje odvija vedno hitreje, se tudi zahteve spreminjajo precej hitreje in stranka se lahko med razvojem projekta večkrat premisli. Na srečo uporaba skriptnih jezikov kot je PHP, omogoča uporabo drugačnih programerskih strategij, kot je na primer RAD. Ena od idej te metodologije je začeti z razvojem čim prej, da lahko naročnik čim prej pregleda delujoč prototip in na ta način lažje dokonča specifikacijo zahtev. Aplikacija se nato razvija iterativno. Razvijalcem posledično ni potrebno upoštevati kaj bi se lahko spremenilo še v prihodnosti. Princip razvoja naj bi bil tak, da so rešitve čimbolj enostavne. Ko pa pride do novih zahtev, oziroma se dodaja nove funkcionalnosti, pa je potrebno obstoječo kodo delno prepisati. Temu procesu pravimo preurejanje (refactoring) in se med razvojem aplikacije veliko izvaja. Koda se premika na različna mesta glede na namen, podvojene dele kode pa združimo na eno samo mesto. Zaradi konstantnega spreminjanja kode je potrebno zagotoviti, da se aplikacija še vedno pravilno izvaja. Ta problem rešujemo z uporabo avtomatiziranih testov. Z dobro napisanimi testi zagotovimo, da ugotovimo vse napake do katerih lahko pride pri spreminjanju kode. Nekatere metodologije pa spodbujajo celo pisanje testov pred samim kodiranjem, imenujejo se testno voden razvoj (test-driven development ali TDD).

5.6. Struktura ogrodja Symfony

Symfony temelji na znani MVC arhitekturi. Črke MVC predstavljajo besede model (model), pogled (view) in krmilnik (controller). Model predstavlja tisti del aplikacije, ki skrbi za zapis podatkov in realizacijo poslovnih pravil in logike. Pogled poskrbi za predstavitev podatkov v obliki, ki bo primerna za uporabnika in je torej vmesnik med uporabnikom in sistemom. Krmilnik se odziva na akcije uporabnika in komunicira z modelom in pogledom ter ju povezuje in upravlja. Model poskrbi tudi za podatkovno abstrakcijo, da pogled in krmilna logika dostopata do podatkov samo preko modela, s čemer se izognemo vezanosti na posamezno podatkovno bazo. MVC arhitektura ima več prednosti. Omogoča enostavnejše vzdrževanje, koda je preglednejša in naloge se lažje deli na več uporabnikov. Ti so lahko specializirani za uporabniške vmesnike in se lahko posvetijo samo pogledu, krmilno logiko pa prepustijo drugim. Ena od posledic arhitekture je tudi, da v primeru uporabe aplikacije na različnih napravah ni potrebno spreminjati in podvajati krmilne logike in modela, temveč se izdela samo več pogledov.

MVC arhitekturo je možno najučinkoviteje realizirati z objektno tehnologijo, zato je Symfony v celoti napisan objektno.

Symfony-jev MVC model lahko razdelimo na sedem komponent:

Plast modela (model layer)

- abstrakcija podatkovne baze (database abstraction)
- dostop do podatkov (data access)

Plast pogleda (view layer)

- pogled (view)
- predloga (template)
- temeljna predloga (layout)

Kontrolna plast (controller layer)

- prvi krmilnik (front controller)
- akcija (action)

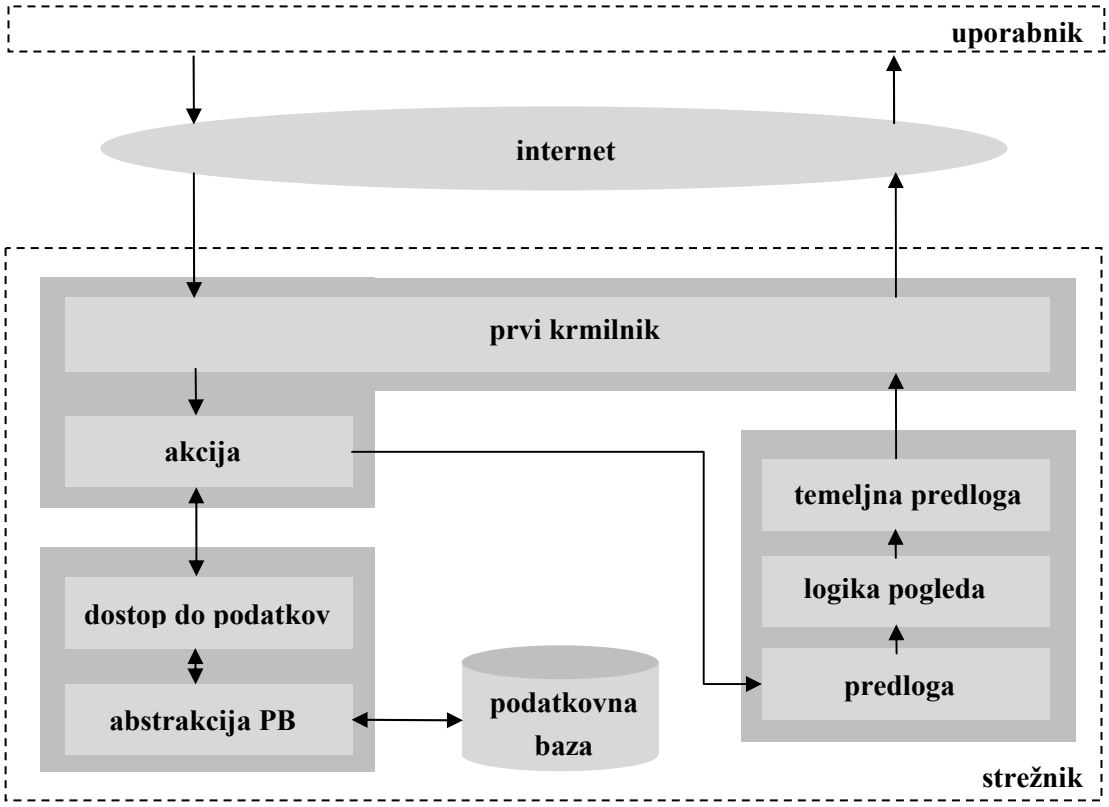


Diagram 5.1 – diagram povezav med komponentami

Iz navedenega bi lahko sklepali, da je potrebno za vsako novo stran narediti sedem datotek, vendar to ni potrebno. Prvi krmilnik in temeljna predloga sta skupni vsem stranem, tako da potrebujemo samo po eno različico. Prvi krmilnik generira Symfony sam in nam ga ni potrebno napisati. Razredi plasti modela so generirani avtomatsko iz podatkovne strukture, za kar poskrbi knjižnica Propel. Podatkovno abstrakcijo pa zagotavlja komponenta Creole. Spremembe podatkovne baze torej niso boleče, spremeniti je potrebno samo nastavitev.

Razvijalcu je potrebno napisati samo tri datoteke. Krmilne akcije, predloge in temeljno predlogo, ki je osnova za vse strani. Posledica delitve je manjše število vrstic kode, večja preglednost, možnost ponovne uporabe in večja fleksibilnost.

5.7. Temeljni razredi

Symfony uporablja nekaj temeljnih razredov, s katerimi se srečamo pri razvoju.

Razred	Opis razreda
sfController	krmilni razred, njegova naloga je interpretacija zahteve (request) in usmeritev na ustrezno akcijo
sfRequest	vsebuje vse elemente zahteve (parametre, piškotke, glavo)
sfResponse	vsebuje glavo odziva in vsebino, iz tega objekta bo nastala HTML koda, ki bo poslana uporabniku.
sfContent	je referenca na vse temeljne objekte in je dostopen povsod

Tabela 5.1 – opis temeljnih razredov

5.8. Organizacija kode

V ogrodju Symfony je projekt množica storitev in operacij na voljo na posamezni domeni, ki si delijo skupni objektni model. Operacije so združene v aplikacije, ki so neodvisne od drugih aplikacij projekta. Pri običajnih projektih sta aplikaciji dve, uporabniška aplikacija in administracijska aplikacija za skrbnike. Aplikacije se podrobneje deli na module, ki združujejo strani s podobnim namenom. Module sestavljajo akcije.

5.8.1. Struktura map korena

```
apps/
  frontend/
  backend/
cache/
config/
data/
  sql/
doc/
lib/
  model/
log/
plugins/
test/
  bootstrap/
  unit/
  functional/
web/
  css/
  images/
  js/
  uploads/
```

Mapa	Opis mape
apps/	vsebuje mapo za vsako aplikacijo projekta, tipično sta to frontend in backend
cache/	vsebuje delno prevedene nastavitve, akcije in predloge, za hitrejšo odzivnost
config/	vsebuje glavne konfiguracijske datoteke projekta
data/	mapa namenjena shranjevanju podatkovnih datotek
doc/	mapa namenjena projektni dokumentaciji
lib/	mapa namenjena tujim knjižnicam
model/	vsebuje datoteke objektnega modela projekta
log/	vsebuje datoteke dnevnikov različnih ravni
plugins/	vsebuje dodatke naložene v aplikacijo
test/	vsebuje »unit« in funkcionalne teste napisane s PHP in kompatibilne s Symfony testnim ogrodjem
web/	korenska mapa spletnega strežnika, edine datoteke, ki so dostopne s spleta, se nahajajo v tej mapi

Tabela 5.2 – opis map korena

5.8.2. Struktura map aplikacije

```
apps/
  [application name]/
    config/
    i18n/
    lib/
    modules/
    templates/
      layout.php
```

Mapa	Opis mape
config/	vsebuje YAML konfiguracijske datoteke, večina konfiguracije je v teh datotekah, razen osnovne konfiguracije ogrodja
i18n/	vsebuje datoteke za večjezično podporo aplikacije, večinoma prevodi uporabniškega vmesnika
lib/	vsebuje razrede in knjižnice, ki so specifične za aplikacijo in niso namenjene celemu projektu
modules/	vsebuje vse module, ki zagotavljajo funkcionalnost aplikacije
templates/	vse globalne predloge aplikacije, ki si jih delijo vsi moduli, privzeto vsebuje layout.php, datoteko, ki je glavna predloga v katero so vstavljene predloge modulov

Tabela 5.3 – opis map aplikacije

5.8.3. Struktura map modulov

Vsaka aplikacija vsebuje enega ali več modulov. Vsak modul ima svojo mapo znotraj mape modules, katere ime je določeno ob inicializaciji modula.

```
apps/
  [application name]/
    modules/
      [module name]/
          actions/
            actions.class.php
          config/
          lib/
          templates/
            indexSuccess.php
```

Mapa	Opis mape
actions/	privzeto vsebuje samo datoteko actions.class.php, v kateri so shranjene vse akcije modula
config/	lahko vsebuje konfiguracijske datoteke z lokalnimi parametri za module
lib/	vsebuje razrede in knjižnice, ki so specifične za modul
templates/	vsebuje predloge, ki se navezujejo na akcije modula, privzeta predloga indexSuccess.php je generirana ob incializaciji modula

Tabela 5.4 – opis map modula

5.8.4. Struktura mape web

Pri oblikovanju mape web je zelo malo omejitev, vendar pa je vseeno priporočljivo uporabljati osnovno strukturo, saj s tem zagotovimo predvideno obnašanje in uporabo bližnjic v predlogah.

```
web/
  css/
  images/
  js/
  uploads/
```

Mapa	Opis mape
css/	vsebuje CSS datoteke za videz strani
images/	vsebuje grafične datoteke za slike in grafične elemente, ki se prikazujejo na straneh
js/	vsebuje JavaScript datoteke
uploads/	vsebuje datoteke, ki jih uporabniki naložijo na strežnik, ponavadi so to grafične datoteke, lahko pa tudi vse druge vrste

Tabela 5.5 – opis podmap mape web

6. Primerjava tehnik razvoja modula administracijskega okolja

Vse več spletnih strani uporablja dinamične vsebine, ki jih je potrebno vnesti v sistem. Za potrebe vnašanja vsebin in upravljanje spletne strani uporabljamo administracijska okolja. Okolja se lahko močno razlikujejo po obsegu, vsem pa je skupno, da so razdeljena na module, ki so specializirani za specifične podatke in informacije. Moduli so večinoma sestavljeni iz seznama zapisov, ki jih lahko spreminjamo in brišemo. Dodajanje novih zapisov je tudi sestavni del modula. Glede na funkcijo modula so dodane še dodatne akcije, ki ustrezno spreminjajo lastnosti zapisov. Na primer zapisi novic imajo lahko samo stanje aktivnosti in neaktivnosti, zapisi naročil v spletni trgovini pa lahko prehajajo med kompleksnejšimi stanji, za katere je potrebno realizirati akcije, ki te prehode omogočijo.

Za primerjavo razvoja sem izbral tri različne tehnike. Osnovna je klasični razvoj s PHP, brez dodatnih knjižnic in ogrodij. Ostali tehniki se nanašata na ogrodje Symfony, prva je uporaba generatorja administracijskega okolja, druga pa razvoj modula s pomočjo samega ogrodja. Za lažjo primerjavo sem realiziral modul, ki sem ga poimenoval »novice«, vendar pa je sestavljen iz večine elementov, s katerimi se srečamo pri izdelavi modulov.

Ti elementi so tekstovno polje, spustni meni, polje za vpis datuma, potrditveno polje za nastavitev logične spremenljivke, skupina potrditvenih polj za N:N povezavo zapisa z zapisi drugih tabel ter nalaganje datotek na strežnik.

Za podatkovno bazo sem pri vseh treh tehnikah izbral MySQL, saj je najpogosteje uporabljena podatkovna baza v kombinaciji s PHP platformo.

6.1. Razvoj modula administracijskega okolja na klasičen način

Najprej moramo razviti podatkovni model in po njem kreirati podatkovno bazo. Kljub temu da je pri razvoju na klasični način največ stvari potrebno narediti ročno, nam ni potrebno napisati stavkov SQL za kreiranje tabel. Na skoraj vsakem PHP strežniku je namreč naložena PHP aplikacija phpMyAdmin, s katero upravljamo s podatkovno bazo in kreiramo tabele na enostaven način. Tabele kreiramo tako, da vpišemo imena atributov in njihove lastnosti v formo, aplikacija pa sama zgradi ustrezen stavek SQL za vpis v bazo.

Ko imamo kreirane tabele v podatkovni bazi, se lahko lotimo izdelave datotek modula. Za razvoj modula sem uporabil najbolj tradicionalen način, torej postopkovno programiranje brez uporabe predložnega stroja. Uporabil sem metodo vgrajevanja PHP kode v HTML kodo in delno upošteval arhitekturo MVC. Arhitektura MVC ni upoštevana na plasti modela, saj ni uporabljena abstrakcija podatkovne baze in dostopa do podatkov, PHP sam tega ne zagotavlja. Upoštevana pa je ločitev prikaza in krmilne logike ter logike za izvrševanje akcij.

Osnovno mapo sestavljajo naslednje mape in datoteke:

Mapa/datoteka	Opis mape ali datoteke
_database/	mapa vsebuje datoteki za vzpostavitev in zaprtje povezave s podatkovno bazo
_exec/	mapa vsebuje akcijske datoteke
_includes/	mapa vsebuje datoteke za pridobivanje podatkov za potrebe predlog
uploads/	mapa kamor se nalagajo datoteke naložene preko aplikacije
_desc.php	datoteka z opisom poti, uporabljena v vseh datotekah za večjo fleksibilnost
news-edit.php	predloga s formo za spreminjanje zapisov
news-insert.php	predloga s formo za dodajanje zapisov
news-list.php	predloga s seznamom zapisov
news-layout.php	temeljna predloga v katero se vključijo preostale predloge
news.php	prvi krmilnik, datoteka preko katere se izvajajo vsi dostopi preko interneta

Tabela 6.1 – opis map in datotek modula pri klasičnem razvoju

Izvajanje spletne strani se začne z datoteko news.php, ki vključi temeljno predlogo in glede na parametre vključi eno od možnih preostalih predlog, odvisno od tega ali gre za dodajanje, spreminjanje ali pa ogled seznama. Prva stran se začne s seznamom, ki je sestavljen iz predloge s HTML kodo in kode (_includes/get-list.php), ki dostopa do podatkovne baze in pripravi spremenljivke za prikaz podatkov v predlogi. Za dostop do podatkov je potrebno napisati ustrezno poizvedbo SQL in rezultat primerno zapisati.

Primer poizvedbe:

```
$SQL_select = "SELECT...";
$result = mysql_query ($SQL_select);
while ($row = mysql_fetch_array ($result)) {
  $news_list[] = $row;
}
```

Spremenljivka $news_list je polje in hrani vse zapise, ki jih je dala poizvedba. Spremenljivko uporabi razvijalec predloge, ki se mu ni potrebno obremenjevati, kako se do podatkov dostopa. Nad vsakim elementom seznama je možno izvršiti dve akciji, to sta brisanje zapisa in odprtje zapisa v načinu za spreminjanje. Akcija je realizirana z uporabo HTML elementov kot so forma, skrito polje in gumb za oddajo. Brisanje preusmeri na akcijsko datoteko (_exec/news-delete.php), ki vsebuje samo PHP kodo. Ta koda mora poskrbeti za brisanje zapisa iz podatkovne baze in brisanje morebitnih datotek. Po koncu izvajanja akcijske datoteke se izvrši preusmeritev nazaj na seznam in prikaže sporočilo o uspešni akciji. Akcija se izvede samo na strežniku, zato je ta korak za uporabnika neviden. Druga akcija, ki odpre zapis v načinu za spreminjanje, pa naloži predlogo news-edit.php, ki uporabi datoteko _includes/get-news-data.php za dostop do podatkov zapisa. Podatki se zapišejo v ustrezna

polja v HTML formi, za pravilno nastavitev teh polj pa moramo poskrbeti z vgrajeno PHP kodo.

Primer sestavljanja kode HTML polj:

Tekstovno polje, ki hrani naslov:

```
<input name="title" type="text" id="title" value="<?=$news['title']?>"
size="80">
```

Spustni meni za nastavitev kategorije:

```
<select name="category_id" id="category_id">
<? include($dir_path.'_includes/get-categories.php');
foreach ($category_array as $category) { ?>
  <option value="<?=$category['id']?>" <? if($news['category_id'] ==
  $category['id']) { ?> selected<? } ?>><?=$category['category']?></option>
<? } ?>
</select>
```

S pomočjo datoteke za dostop do polja kategorij izvedemo zanko, ki sestavi izbirne možnosti (option tag) spustnega menija. Na podoben način se sestavi tudi potrditveno polje za logično spremenljivko in za N:N povezave s skupinami.

Predloga news-insert.php služi dodajanju novih zapisov v podatkovno bazo, sama forma pa je zelo podobna tisti za spreminjanje podatkov. Razlika je v tem, da forma za vnašanje ne naloži predhodnih podatkov zapisa, saj le ta še ne obstaja. Pomembna razlika je tudi v tem, da formi kažeta na različni akciji, saj je za dodajanje in spreminjanje zapisa potrebno uporabiti različne stavke SQL.

Pri razvoju na klasičen način torej uporabljamo PHP programiranje, stavke SQL in HTML kodo sestavljamo z vgrajevanjem PHP kode. Določitev strukture map, datotek in kode je prepuščena razvijalcu. Sam se odloči kako bo ločil posamezne dele kode, ki sicer sodelujejo, vendar so različnega tipa. Kakovost strukture in kode je posledično v veliki meri odvisna od izkušenj in usposobljenosti razvijalca in je lahko drugim težko razumljiva.

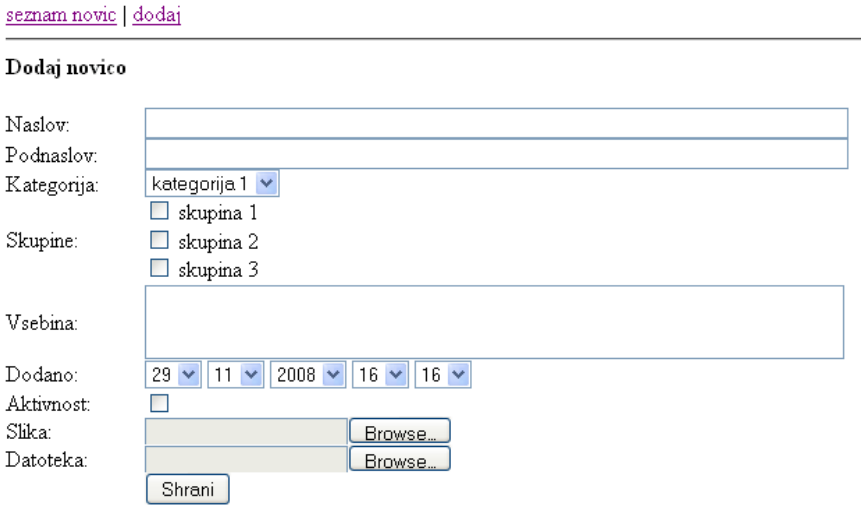
6.1.1. Primer uporabniškega vmesnika, razvitega na klasičen način

Ker predmet raziskave ni bila zmožnost oblikovanja HTML kode zaslonske maske, temveč hitrost razvoja vmesnika in logike, je HTML koda zapisana v minimalistični obliki, zato je tak tudi videz.

Seznam zapisov nam omogoča sortiranje po stolpcih, gumba Popravi in Zbriši pa izvedeta akciji.

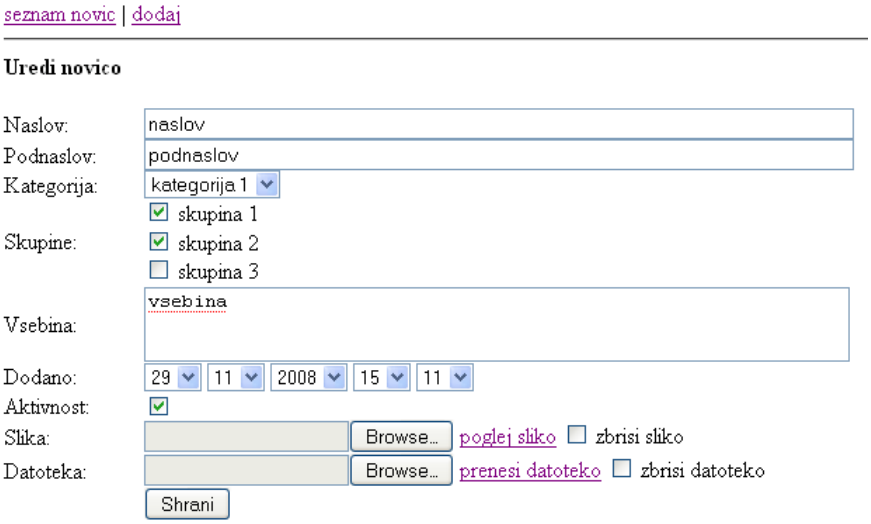
Slika 6.1 – seznam zapisov pri klasičnem razvoju

Prazna forma za dodajanje ne vsebuje nobenih podatkov, razen datuma, ki je privzeto trenutni čas. V primeru napačnega vnosa, se forma napolni s podatki vpisanimi pred oddajo, ter izpiše razlog napake.



Slika 6.2 – prazna forma za dodajanje pri klasičnem razvoju

Do forme za pregledovanje in spreminjanje zapisov pridemo s klikom na gumb Popravi pri posameznem zapisu v seznamu. Datoteke zapisa je možno tudi prenesti ter zbrisati.



Slika 6.3 – forma za spreminjanje obstoječih zapisov pri klasičnem razvoju

6.2. Razvoj modula administracijskega okolja z ogrodjem

Razvoj modula v ogrodju Symfony se začne z inicializacijo modula. Izvršimo ga z ukazom »symfony init-module ime_aplikacije ime_modula«. Symfony sam kreira mapo za modul in potrebne datoteke. Vse akcije modula so zapisane v datoteki actions.class.php, ki na začetku ne vsebuje nobenih metod. Vse potrebne datoteke, ki sledijo, moramo napisati sami. Modul lahko vsebuje več strani in vsaka stran ima svojo predlogo. Vsaka predloga ima svojo akcijo v actions.class.php, seveda pa so možne tudi akcije, ki predloge nimajo, saj ničesar ne izpišejo, temveč po izvajanju preusmerijo na drugo stran. Ime modula v primeru je »news«, njegova prva stran pa je stran s seznamom, ki se imenuje »list«. Naslov preko katerega je stran dostopna je sestavljen iz obeh, torej »news/list«. Za vsako stran obstaja predloga, ki je poimenovana po strani, ime_straniSuccess.php, oziroma pri seznamu listSuccess.php. Datoteka je predloga s HTML kodo in vgrajeno PHP kodo, vendar se PHP koda uporablja samo za dinamično gradnjo HTML kode. Funkcionalna logika in pridobivanje podatkov iz modela se izvede v akciji, predloga pa prikaže podatke na podlagi spremenljivk, ki jih akcija nastavi. Zaradi večje kompatibilnosti v ogrodju Symfony za izpis spremenljivk v HTML kodo uporabljamo daljši način vgrajevanja kode.

Primer vgrajevanja kode:

```
<?php echo $news->getTitle(); ?>
```

Kljub temu da lahko pišemo poljubno PHP in HTML kodo, pa je priporočljivo uporabljati posebne predložne funkcije (helper), ki poskrbijo za pravilen izpis HTML elementov. Enostaven primer sta funkciji link_to in image_tag.

Primer predložnih funkcij:

```
<?php echo link_to(image_tag("icons/edit_icon.png"), "news2/edit?news_id="
. $news->getPrimaryKey()); ?>
```

Enakovredna izvedba s HTML kodo:

```
<a href='/pot_do_stani/news2/edit?news_id=<?php echo $news->getPrimaryKey()
?>'><img src='/pot_do_slike/slika.png'></a>
```

Prednost uporabe predložnih funkcij je v tem, da ni potrebno pisati poti do strani, ali poti do slike, saj funkcija sama poskrbi za to. Prednost je tudi to, da je potrebno manj pisanja kode in da je verjetnost za napako manjša. Je pa taka predloga slabše vidna v razvojnem okolju, ki podpira predogled HTML kode.

S strani za prikaz seznama je možno sprožiti tri akcije, spreminjanje in brisanje posameznega zapisa ter dodajanje novih.

Na spreminjanje zapisa vodi naslov strani »news/edit/news_id/X«, ki pove, da gre za predlogo editSuccess.php, akcijo executeEdit, podan pa mora biti tudi id zapisa. Akcija executeEdit preveri ali je bila zahteva za stran sprožena s strani forme za spreminjanje ali gre za prvi obisk strani. Ob prvem obisku se namreč izvede dostop do podatkov zapisa preko objekta NewsPeer, če pa gre za odziv na formo, se spremembe zapišejo v objekt, ki se ga na koncu shrani.
Pridobivanje podatkov iz baze je precej drugačno od realizacije pri klasičnem načinu. Za razliko od klasičnega načina, tu ni potrebno pisati poizvedb SQL, temveč kličemo metode nad Peer razredom.

Primer:

```
$categories = CategoryPeer::doSelect(new Criteria());
```

Zgornji stavek zapiše v spremenljivko $categories vse zapise iz tabele kategorij, v obliki objektov. Za podobno operacijo bi na klasičen način potrebovali mnogo več kode. V večini primerov želimo rezultate tudi filtrirati, zato se uporablja objekt Criteria(). Z nastavljanjem objekta podamo različne možne omejitve in povezave, ki bi jih sicer morali zapisati s pomočjo kode SQL. Kadar hočemo dostopati do točno določenega zapisa za potrebe spreminjanja, uporabimo kodo:

```
$news = NewsPeer::retrieveByPk($news_id));
```

V spremenljivko $news se zapiše objekt, ki ustreza zapisu v tabeli novic z primarnim id-jem, ki je zapisan v spremenljivki $news_id.

Pri klasičnem načinu smo morali za spremembe zapisov v bazi izvesti stavek SQL, pri ogrodju Symfony pa vse potrebne spremembe naredimo na nivoju objekta. Za spremembo naslova na primer kličemo »$news->setTitle('naslov');«. Spremembe se v bazi izvedejo šele, ko nad objektom kličemo metodo »save«. Tudi shranjevanje naloženih datotek je z uporabo ogrodja enostavnejše.

Primer shranjevanja naložene datoteke:

```
$this->getRequest()->moveFile('ime_html_polja',
sfConfig::get("sf_upload_dir") . $this->getRequest()-
>getFileName('ime_html_polja'));
```

Na objektu zahteve kličemo metodo »moveFile«, ki ji je potrebno podati ime HTML polja, ki je naložilo datoteko, pot in ime kamor naj se datoteka shrani. Tu si lahko pomagamo s klicem nastavitve privzete mape za nalaganje datotek in originalnega imena naložene datoteke. Možno je datoteko shraniti tudi drugam in s poljubnim imenom.

Tudi pri gradnji form je možno uporabljati običajno HTML kodo, vendar je priporočena uporaba predložnih funkcij. Za gradnjo vsakega HTML polja obstaja predložna funkcija, s katero lažje in hitreje zgradimo kodo. Za tekstovno polje uporabimo »input_tag«, za spustni meni uporabimo »select_tag«, za potrditveno polje »checkbox_tag«, za vpis datuma »input_date_tag«, za nalaganje datoteke pa uporabimo »input_file_tag«. Funkcijam s pomočjo parametrov nastavimo imena, razpoložljive vrednosti, izbrane in trenutne vrednosti ter obliko. Svojo predložno funkcijo imata tudi forma (form_tag) in gumb za oddajo forme (submit_tag). Tako kot pri drugih predložnih funkcijah je v primerjavi s pisanjem HTML elementov manj dela, vendar pa je predogled slabši, oziroma se elementov v predogledu ne vidi.

Primer razlike med HTML elementom in uporabo predložne funkcije:

Tekstovno polje
HTML zapis
```
<input name="title" type="text" id="title" value="
<?php echo $news->getTitle() ?>" size="65">
```

Zapis s predložno funkcijo
```
<?php echo input_tag("title", $news->getTitle(), 'size=65'); ?>
```

Potrditveno polje
HTML zapis
```
<input name="is_active" type="checkbox" id="is_active" value="1" <?php if
($news->getIsActive()) { ?>checked="checked"<?php } ?> />
```

Zapis s predložno funkcijo
```
<?php echo checkbox_tag('is_active', 1, $news->getIsActive()) ?>
```

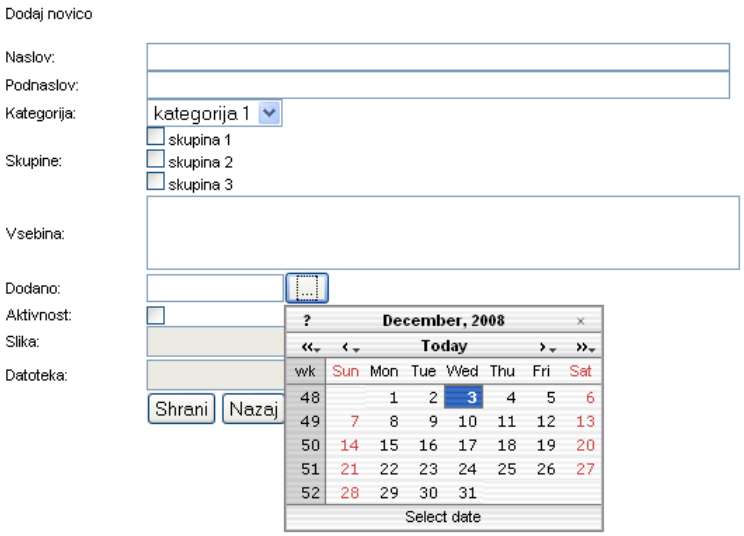
6.2.1. Primer uporabniškega vmesnika, razvitega z ogrodjem Symfony

Podobno kot pri klasičnem razvoju, je tudi tu HTML koda minimalistična, na nekoliko drugačen videz pa vplivajo privzeti stili ogrodja. Seznam zapisov omogoča povsem enake funkcionalnosti.

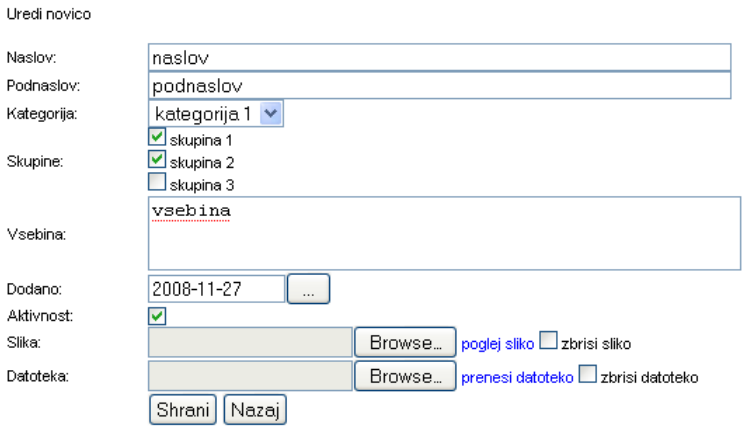
Slika 6.4 – seznam zapisov pri uporabi ogrodja

Formi za dodajanje in spreminjanje se po delovanju od različic pri klasičnem razvoju razlikujeta samo po realizaciji datumskega polja. Pri klasičnem razvoju je potrebno uporabiti spustne menije ali pa prepustiti uporabniku da pravilno obliko zapiše v tekstovno polje. Symfony ima vgrajene pomožne funkcije, ki izpišejo tekstovno polje z datumom, poleg polja

pa se nahaja gumb, ki odpre priročen koledarček, s pomočjo katerega je izbira datuma zelo enostavna. V primeru da datumsko polje pustimo prazno se nastavi trenutni datum.



Slika 6.5 – prazna forma za dodajanje pri uporabi ogrodja



Slika 6.6 – forma za spreminjanje obstoječih zapisov pri uporabi ogrodja

6.3. Razvoj modula administracijskega okolja z generatorjem

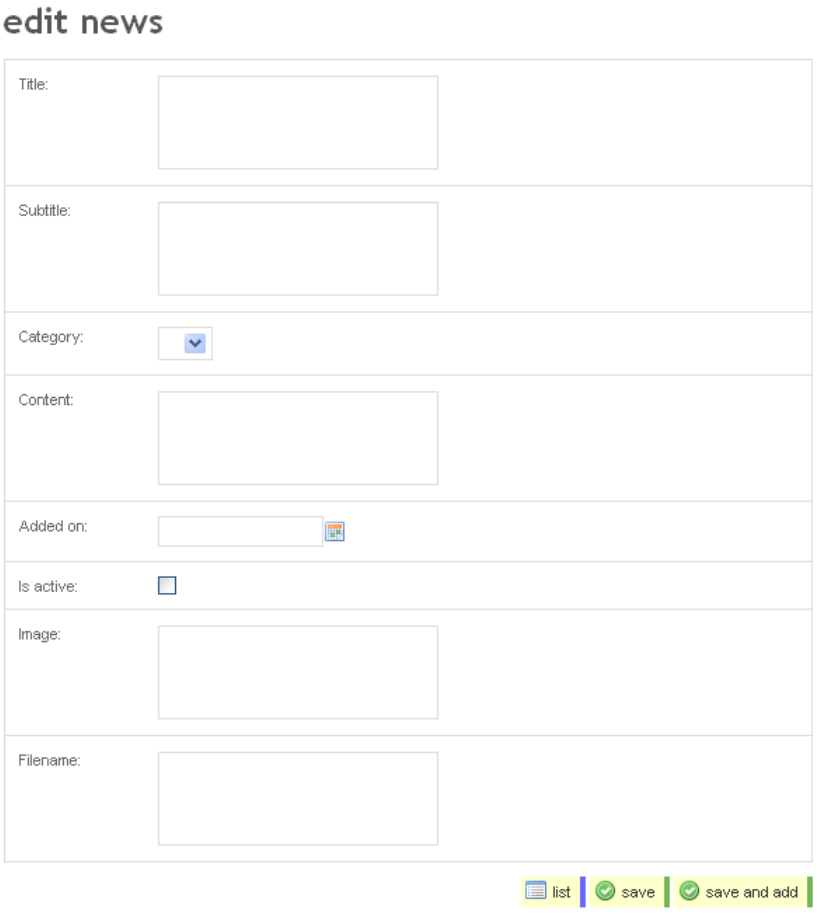
Za razliko od prejšnjih načinov izgradnje modula, je generiranje modula najhitrejša možnost, saj v večini primerov zadostuje zelo majhna količina kode. Modul vzpostavimo s klicem symfony ukaza »symfony propel-init-admin ime_aplikacije ime_modula ime_razreda«. Generator kreira potrebne mape za modul in akcijsko datoteko actions.class.php ter generator.yml. Akcijska datoteka je na začetku prazna in jo potrebujemo samo v primeru, da je potrebno predelati generirane akcije. Vso pozornost torej lahko usmerimo v datoteko generator.yml, ki je specifikacija administracijskega vmesnika v YAML zapisu. Generirana datoteka vsebuje samo pet osnovnih vrstic, kljub temu pa administracijski vmesnik že deluje v osnovni obliki.

6.3.1. Primer administracijskega vmesnika brez nastavljene specifikacije

Brez nastavitev se v seznamu prikažejo vsi atributi, ki so poimenovani po atributih v podatkovni bazi. Za ogled zapisa je potrebno klikniti na id zapisa, brisanje pa je možno znotraj ogleda. Sortiranje zapisov po atributih izvedemo s klikom na atribut v glavi stolpcev.

Slika 6.7 – neformatiran seznam zapisov

Tudi forma za dodajanje in spreminjanje podatkov brez specifikacije ne more delovati v celoti. Poleg manjkajočih prevodov ni vključena N:N povezava s skupinami. Atributa, ki hranita ime datotek nimata možnosti nalaganja datoteke, saj ju v modelu predstavlja tekstovni podatkovni tip za zapis imena datoteke, iz česar generator ne more sklepati da gre za datoteko.



Slika 6.8 – forma za dodajanje in spreminjanje zapisov

Ker v osnovni obliki niso podprte vse možnosti na način, ki bi si ga želeli, moramo napisati ustrezno YAML specifikacijo, ki opisuje vmesnik. Z njo določimo imena polj, katera polja se prikazujejo v seznamu, katera polja je možno spreminjati in podobno.

Primer specifikacije za modul novic:

```
generator:
  class:              sfPropelAdminGenerator
  param:
    model_class:      News
    theme:            default

    fields:
      title: {name: Naslov }
      subtitle: { name: Podnaslov }
      category_id: {name: Kategorija }
      category_name: {name: Kategorija }
      content: {name: Vsebina }
      added_on: {name: Dodano }
      is_active: {name: Aktivnost }
      news_ngroup: {name: Skupine }
      image: {name: Slika }
        help: Izberete sliko z diska!
        type: admin_input_file_tag
        upload_dir: photos
        params: include_link=photos include_text="poglej sliko"
                                                    include_remove=true
      filename: {name: Datoteka}
        help: Izberite datoteko z diska!
        type: admin_input_file_tag
        upload_dir: files
        params: include_link=files include_text="prenesi datoteko"
                                                    include_remove=true
    edit:
      title: Urejanje
      display: [title, subtitle, category_id, news_ngroup, content,
                                    added_on, is_active, image, filename]
      fields:
        news_ngroup: { type: admin_check_list, params:
                                                through_class=NewsNgroup }
        title: { params: size=100x1 }
        subtitle: {params: size=100x1}
      actions:
        _list:       { name: "Na seznam" }
        _save:       { name: "Shrani" }
    list:
      title: Seznam novic
      display: [title, subtitle, category_id, content, added_on, is_active]
      fields:
        added_on: {name: Vnešeno, params: date_format='dd.MM.yyyy'}
      object_actions:
        _edit:         ~
        _delete:       ~
      batch_actions:
        _delete:       ~
      actions:
        _create:       { name: "Dodaj"}
```

Za celotno specifikacijo primera je potrebno samo 50 vrstic. Pri kodi YAML se struktura določa z zamikanjem, to pomeni da je element, ki je za en zamik zamaknjen glede na prejšnji element, njegov atribut. Specifikacija vsebuje 3 glavne vrstice, »fields«, »edit« in »list«. »Fields« določa lastnosti atributov kot je na primer ime, če pa je atribut kompleksnejše narave, mu lahko določimo še druge parametre. Nastavitve veljajo za atribute v vseh možnih pogledih, razen pri tistih ki nastavitve prepišejo. V razdelku »edit« določimo vse kar se navezuje na formo za dodajanje in spreminjanje zapisov. To so naslov, atributi ki jih forma prikazuje, oblika v kateri se atribut predstavi in akcije, ki jih je možno izvesti. Pri seznamu so nastavitve podobne.

Za večino zahtev zadostuje specifikacija, na podlagi katere generator izdela predloge in akcijske funkcije, vendar pa se programiranju vedno ni možno izogniti. Osnovni primer je predstavitev kategorije v spustnem meniju. Osnovni prikaz v spustnem meniju prikaže primarne ključe zapisov v tabeli kategorija, ne pa imena kategorije, ker ima tabela lahko večje število atributov in generator ne more sam ugotoviti, na kakšen način je ime potrebno prikazati. Za ustrezen prikaz imena moramo dodati funkcijo __toString() v datoteko razreda kategorije.

Primer:

```
class Category extends BaseCategory
{
  public function __toString() {
    return $this->getCategory();
  }
}
```

V primeru sicer prikažemo ime kategorije kot je zapisano v tabeli. Lahko bi ime sestavili iz več morebitnih atributov in ga, kadar gre za daljša imena, skrajšali na določeno dolžino, da ne bi pokvarilo videza.

Dokler nam osnovne funkcionalnosti in videz ustrezajo, zadostuje YAML specifikacija in __toString() funkcije v razredih. Če pa želimo nad generiranim modulom izvesti večje spremembe, je potrebno spremeniti predloge in akcije. Koda za predloge in akcije pri generiranem modulu v osnovi sploh ne obstaja, temveč se generira po zahtevah iz specifikacije. Generirana koda predlog in akcij se nahaja v »cache« mapi, od koder se izvaja aplikacija. Če želimo del modula realizirati drugače, kot to stori generator, lahko v »cache« mapi poiščemo ustrezne predložne datoteke in akcijsko datoteko ter na njihovi osnovi izdelamo potrebno kodo. V primeru da želimo spremeniti akcije, poiščemo ustrezno generirano akcijo in njeno funkcijo zapišemo v že omenjeno datoteko actions.class.php. Funkcije zapisane v tej datoteki prepišejo generirane funkcije. Podobno je s predlogami, ki so v ločenih datotekah. Zapišemo jih v »templates« mapo in tudi te prepišejo generirane

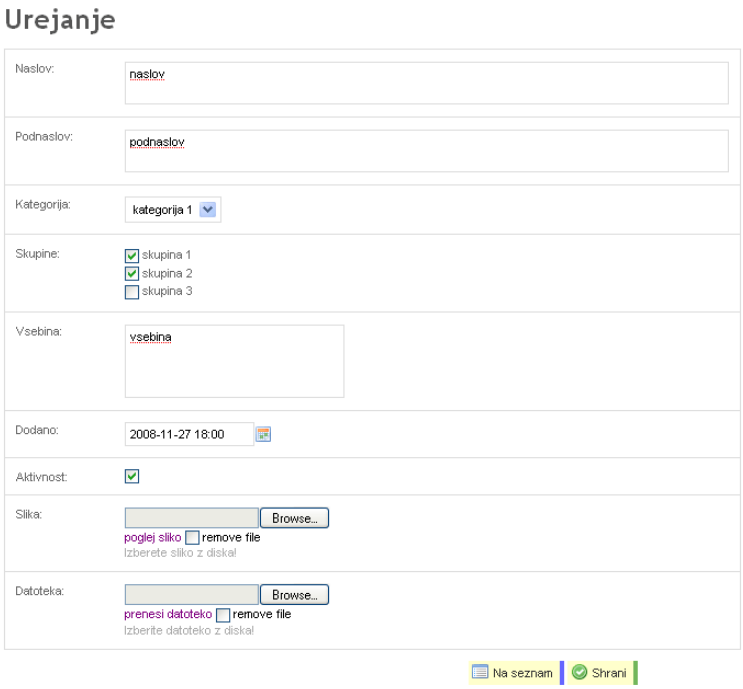
predloge. Na ta način lahko modul močno prikrojimo svojim zahtevam, upoštevaje dejstvo da spreminjanje akcij in predlog zahteva natančno poznavanje PHP in HTML kode, kot tudi strukture in razredov ogrodja. Je precej kompleksnejša naloga kot izdelava YAML specifikacije.

6.3.2. Primer administracijskega vmesnika po ustrezno nastavljeni YAML datoteki

Seznam novic je pravilno poimenovan, prikazujejo pa se samo nastavljeni atributi, katerim smo nastavili tudi prevode. Spremenjeno je tudi ime gumba za dodajanje. Nad vsakim zapisom lahko izvedemo akciji brisanja in spreminjanja s klikom na ikono. Besed, ki niso prevedene, ni možno nastaviti v YAML datoteki, temveč je potrebno izdelati jezikovne datoteke I18n.

Slika 6.9 – ustrezno oblikovan seznam zapisov

Pomanjkljiva in neustrezna specifikacija najbolj vpliva na formo za spreminjanje in dodajanje. Poleg naslovov in prevedenih atributov, se prikazujejo tudi potrditvena polja za povezovanje s skupinami ter možnosti, ki jih prinese nalaganje datotek.



Slika 6.10 – forma za dodajanje in spreminjanje po ustrezni specifikaciji

6.4. Primerjava uporabljenih tehnik

Tehnike je smiselno primerjati po času potrebnem za izdelavo, po količini napisane kode, po kakovosti kode in zahtevnosti vzdrževanja ter možnosti ponovne uporabe kode. Ko sem izdeloval primere za vse tri tehnike, sem že dobro poznaval vse tri, zato v času, ki sem ga potreboval za izdelavo, ni bilo potrebno učenje novih funkcionalnosti. Pri klasičnem načinu sem moral izdelati fizično podatkovno bazo in kreirati tabele, kjer sem si pomagal z orodjem phpMyAdmin. Za obe tehniki z ogrodjem je zadostoval skupni podatkovni model, ki je zapisan v XML datoteki, nato pa je orodje Propel poskrbelo za generiranje vseh potrebnih tabel podatkovne baze in razredov podatkovnega modela. Glede na to, da pri klasičnem načinu ni bilo potrebno pisati stavkov SQL, pri ogrodju pa je bilo potrebno napisati samo specifikacijo, sta obe operaciji vzeli približno enako časa.

Pri razvoju modulov je bila razlika v porabljenem času precejšnja, za generiranje modula sem porabil eno časovno enoto, za razvoj z ogrodjem 5 časovnih enot, za klasični razvoj pa kar 10 časovnih enot. Podobna razmerja so tudi pri količini napisane kode. V primerjavi z generatorjem je pri uporabi ogrodja približno 7 krat več kode, pri klasičnem načinu pa kar 17 krat toliko.

Kakovost kode je pri generatorju težko analizirati, saj je razen majhnih izjem praktično ne pišemo, temveč napišemo specifikacijo. Če pa se osredotočimo na generirano kodo, je le ta zagotovo optimalno napisana in dobro testirana. Pri ostalih dveh tehnikah je kakovost kode odvisna predvsem od doslednosti pri razvoju in izkušenosti razvijalca, vendar pa je razvijalec pri uporabi ogrodja prisiljen uporabljati določene arhitekturne standarde, ki zagotavljajo del kakovosti. Tudi koda pri klasičnem razvoju je lahko zelo kakovostna, vendar pa je zaradi pomanjkanja omejitev vse odvisno od razvijalca. Od njega je odvisno, kakšno arhitekturo bo uporabil in kakšno kodo bo pisal, zato je veliko večja možnost, da koda ne bo kakovostna.

Zahtevnost vzdrževanja je povezana s količino in preglednostjo kode. Vzdrževanje generiranega modula je najenostavnejše, saj je najmanj kode, ki jo je potrebno pregledati in spremeniti. Pri klasičnem razvoju je veliko odvisno od tega, kakšne stile je uporabil razvijalec, v splošnem pa velja, da je aplikacija najbolj razumljiva ravno razvijalcu, ki jo je razvil, nekdo drug pa ima pri tem lahko precej težav. Vzdrževanje modulov, napisanih z ogrodjem, je lažje zaradi manjše količine kode in večjih omejitev.

Ponovna uporabljivost kode je najlažja pri razvoju z ogrodjem, ker je abstrakcija precej večja kot pri klasičnem razvoju. Pri generiranju modula težko govorimo o ponovni uporabnosti kode, ker le te ponavadi skoraj ni, lahko pa si pomagamo s samo specifikacijo. Prihranek časa je v tem primeru zaradi hitre narave tehnike precej manjši kot pri klasičnem razvoju in razvoju z uporabo ogrodja, kjer moramo napisati veliko kode.

	Čas razvoja	Količina kode	Zahtevnost vzdrževanja	Možnost ponovne uporabe
Klasični razvoj	10	17	Visoka	Nizka
Uporaba ogrodja	5	7	Srednja	Visoka
Uporaba generatorja	1	1	Nizka	Srednja

Tabela 6.2 – elementi primerjave uporabljenih tehnik

Če primerjamo zgoraj uporabljene tehnike po opisanih parametrih, je racionalno trditi, da je najbolj smiselno uporabiti generator. Če ne gre za preveč zapletene module, katerim ustrezajo zmožnosti generatorja, je njegova uporaba res smiselna. Če pa imamo posebne zahteve in zapletenejše prikaze, pa je uporaba generatorja lahko preveč omejujoča, oziroma bi za predelavo generirane kode v potrebno kodo porabili toliko časa, da se časovna prednost povsem izgubi. V zapletenih primerih je torej smiselno uporabiti ogrodje ali pa klasični razvoj. Kljub temu da smo pri ogrodju bolj omejeni in je izvajanje kode včasih počasnejše in manj učinkovito, pa nam nudi veliko prednosti. Prava MVC arhitektura, hitrejši razvoj, lažje vzdrževanje, ob dobrem poznavanju pa je možno tudi optimizirati potratne operacije. Za razliko od generatorja lahko ogrodje in klasični razvoj uporabimo za gradnjo vseh možnih aplikacij, od administracijskih okolij do uporabniških okolij in datotek, ki samo izvajajo operacije brez izpisa.

6.5. Kdaj uporabiti posamezno tehniko

Kadar zahteve za administracijske module niso preveč zahtevne in imamo veliko šifrantov in nezahtevnih seznamov, je uporaba generatorja najhitrejša rešitev. Omogoča tudi omejevanje dostopa za uporabnike z različnimi vlogami, zato je za marsikatero manjše administracijsko okolje generator prava izbira. Zaradi zelo hitrega razvoja z generatorjem lahko hitro in v zgodnji fazi razvoja izdelamo prototipe, s katerimi lažje uskladimo zahteve z naročnikom. Prototip pa nam lahko služi tudi za hitro in enostavno vnašanje testnih zapisov. Za srednje in velike sisteme je uporaba ogrodja najbolj smiselna, saj omogoča največ z najmanj vložka časa. Uporaba klasičnega razvoja za izgradnjo administracijskih vmesnikov ob izkušenem razvijalcu in uporabi ustrezne arhitekture prav tako lahko proizvede kakovostne rešitve, vendar pa je čas izdelave občutno daljši. Za izdelavo manjših uporabniških aplikacij je včasih uporaba klasičnega razvoja celo hitrejša izbira, predvsem ko je malo dinamičnih vsebin. Uporaba ogrodja je smiselna pri srednjih in večjih projektih. Ponavadi uporaba klasičnega razvoja pri zelo obremenjenih sistemih omogoča hitrejše delovanje.

Odkar sem spoznal ogrodje in generator, bom vedno manj stvari realiziral na klasičen način, saj je pri uporabi ogrodja veliko prednosti. Predvsem za izdelavo novih, zahtevnih modulov, ki jih na klasičen način še nisem razvil, je uporaba ogrodja racionalna izbira. Modulov, ki pa sem jih že razvil na klasični način in jih bo možno še večkrat uporabiti, verjetno še ne bom

realiziral s pomočjo ogrodja, saj vložek novega časa ni upravičen, če že razvita rešitev zadostuje. Seveda je na enem projektu najbolj smiselno uporabiti samo en pristop, zato se sčasoma realizira z ogrodjem tudi že narejene module.

Sklepne ugotovitve

Skriptni jezik PHP je zrel, dostopen in zmogljiv, njegova uporaba pa se bo v prihodnosti še povečevala.

Danes obstaja zelo veliko ogrodij napisanih v jeziku PHP, prav tako obstaja veliko razvojnih okolij. Raziskovanje in testiranje vseh ogrodij in razvojnih okolij bi preseglo okvire diplomskega dela, zato sem se omejil na najbolj razširjene in priljubljene.

Zaradi vedno večjih potreb po realizaciji velikih in kompleksnih spletnih aplikacij je potrebno uporabljati nove in učinkovite arhitekture in pristope. Uporaba objektnega programiranja, MVC arhitekture in ogrodja, omogoča hitrejši razvoj, možnost ponovne uporabe, večjo modularnost in lažje vzdrževanje. Klasični razvoj aplikacij s skriptnim jezikom PHP se uporablja vedno manj. Uporaba generatorjev kode je odlična rešitev za izdelavo prototipov, včasih pa zadostuje tudi za izvedbo enostavnejših modulov. Kljub temu da obstajajo zmogljivejša in enostavnejša razvojna okolja, načeloma ne vplivajo na kakovost izdelka, temveč razvijalcu olajšujejo delo. Njihova izbira je subjektivna.

Kot sem že pred začetkom dela predpostavljal, se je izdelava aplikacij s pomočjo ogrodja izkazala za bolj učinkovito od klasičnega razvoja, nisem si pa predstavljal, da bo pohitritev razvoja tako velika. Še bolj me je presenetila razlika v količini kode. Izkazal se je tudi generator, za katerega sem pričakoval, da bo bolj omejen in manj uporaben. To še dodatno potrjuje tezo, da je uporaba ogrodja in generatorja ne samo upravičena, temveč priporočljiva.

Slike

Diagrami

Tabele

Viri

[1] Rasmus Lerdorf, Kevin Tatroe, Peter MacIntyre, *Programing PHP, Second Edition*, Sebastopol: O'Reilley Media, Inc., 2006

[2] History of PHP and related projects. Dostopno na: http://si2.php.net/history

[3] What are the advantages and disadvantages of PHP? Dostopno na: http://wiki.answers.com/Q/What_are_the_advantages_and_disadvantages_of_PHP

[4] Smarty - the compiling PHP template engine. Dostopno na: http://www.smarty.net/manual/en/

[5] The Definitive Guide to Symfony: Chapter 1 – Introducing Symfony. Dostopno na: http://www.symfony-project.org/book/1_2/01-Introducing-Symfony

[6] The Definitive Guide to Symfony: Chapter 14 – Generators. Dostopno na: http://www.symfony-project.org/book/1_0/14-Generators

[7] Classed and Objects (PHP 5). Dostopno na: http://us3.php.net/manual/en/language.oop5.php

[8] An Introduction to Propel. Dostopno na: http://propel.phpdb.org/trac/wiki/Users/Introduction

[9] Eclipse: PDT Project. Dostopno na: http://www.eclipse.org/pdt/

[10] Practical PHP Programming. Dostopno na: http://hudzilla.org/phpwiki/index.php?title=Main_Page

Izjava o samostojnosti dela

Izjavljam da sem diplomsko delo izdelal samostojno pod vodstvom mentorice doc. dr. Mojce Ciglarič.