Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko

Jernej Brunec

Razvoj spletnih rešitev s pomočjo ogrodja CakePHP

Diplomsko delo

Maribor, avgust 2011

Diplomsko delo univerzitetnega študijskega programa

RAZVOJ SPLETNIH REŠITEV S POMOČJO OGRODJA CAKEPHP

Študent:	Jernej Brunec
Študijski program:	UN Informatika in tehnologije komuniciranja
Smer:	Informacijski sistemi
Mentor(ica):	doc. dr. Marko Hölbl, univ.dipl.inž. rač. in inf.

Maribor, avgust 2011

Univerza v Mariboru
Fakulteta za elektrotehniko, računalništvo in informatiko

Številka: BITK-34
Datum in kraj: 17. 06. 2011, Maribor

Na osnovi 330. člena Statuta Univerze v Mariboru (Ur. l. RS, št. 1/2010)

SKLEP O DIPLOMSKEM DELU

1. **Jerneju Bruncu,** študentu univerzitetnega študijskega programa INFORMATIKA IN TEHNOLOGIJE KOMUNICIRANJA, smer Informacijski sistemi, se dovoljuje izdelati diplomsko delo pri predmetu Oblikovanje vizualne komponente spleta.

2. **MENTOR: doc. dr. Marko Hölbl**

3. **Naslov diplomskega dela:**
RAZVOJ SPLETNIH REŠITEV S POMOČJO OGRODJA CAKEPHP

4. **Naslov diplomskega dela v angleškem jeziku:**
DEVELOPMENT OF WEB SOLUTIONS USING THE CAKEPHP FRAMEWORK

5. Diplomsko delo je potrebno izdelati skladno z "Navodili za izdelavo diplomskega dela" in ga oddati v treh izvodih (dva trdo vezana izvoda in en v spiralo vezan izvod) ter en izvod elektronske verzije do 17. 06. 2012 v referatu za študentske zadeve.

Pravni pouk: Zoper ta sklep je možna pritožba na senat članice v roku 3 delovnih dni.

Dekan:

UNIVERZA V MARIBORU
FAKULTETA ZA ELEKTROTEHNIKO, RAČUNALNIŠTVO IN INFORMATIKO
1

Obvestiti:
- kandidata,
- mentorja,
- odložiti v arhiv.

ZAHVALA

Zahvaljujem se mentorju za pomoč in vodenje pri opravljanju diplomskega dela. Prav tako se zahvaljujem staršem za podporo pri študiju.

RAZVOJ SPLETNIH REŠITEV S POMOČJO OGRODJA CAKEPHP

Ključne besede: CakePHP, PHP, ogrodje, spletna aplikacija, internet

UDK: 004.777:004.43(043.2)

Povzetek

V diplomskem delu smo se lotili preučevanja ogrodja CakePHP in izdelave praktične spletne stani, tako s pomočjo ogrodja CakePHP kot tudi brez. Pregledali smo poglavite gradnike ogrodja in njihove funkcionalnosti. Izpostavili smo razlike pri procesu razvijanja spletnih aplikacij med ogrodjem CakePHP in osnovnim PHP jezikom. Obdelali smo tudi arhitekturno šablono MVC, na kateri temelji CakePHP. Poudarili smo prednosti in slabosti le-te. Vso znanje, ki smo ga pridobili iz teoretičnega dela diplome smo nato uporabili pri izdelavi spletne trgovine.

DEVELOPMENT OF WEB SOLUTIONS USING THE CAKEPHP FRAMEWORK

Keywords: CakePHP, PHP, framework, web application, internet

UDK: 004.777:004.43(043.2)

Abstract

The purpose of this diploma is to study the php framework called CakePHP and to create a demonstrative website, with the help of the CakePHP framework and without it. We described the main constituents of the framework and their functionalities. We also exposed the differences between CakePHP framework and basic php language without the aid of the framework itself. We described the basics of MVC architecture, upon which CakePHP is built. Knowledge and experience gained from studying this framework, we then applied to a practical solution – an online store made with CakePHP.

VSEBINA

Kazalo slik

Kazalo tabel

UPORABLJENE KRATICE

PHP – PERSONAL HOME PAGE HYPERTEXT PREPROCESSOR

HTML – HYPERTEXT MARKUP LANGUAGE

MVC – MODEL VIEW CONTROLLER

JSP – JAVA SERVER PAGES

ASP – ACTIVE SERVER PAGES

CGI – COMMON GATEWAY INTERFACE

HTTP – HYPERTEXT TRANSFER PROTOCOL

LAMP – LINUX APACHE, MYSQL, PHP

XAMPP – CROSS-PLATFORM APACHE, MYSQL, PHP, PERL

MAMP – MAC OS X APACHE, MYSQL, PHP

WAMP – WINDOWS APACHE, MYSQL, PHP

FAQ – FREQUENTLY ASKED QUESTIONS

CMS – CONTENT MANAGEMENT SYSTEM

SOA – SERVICE ORIENTED ARCHITECTURE

URL – UNIVERSAL RESOURCE LOCATOR

XML – EXTENSIVE MARKUP LANGUAGE

JSON – JAVASCRIPT OBJECT NOTATION

CTP – CAKEPHP TEMPLATE

PDF – PORTABLE DOCUMENT FORMAT

CRUD – CREATE, READ, UPDATE, DELETE

AJAX – ASYNCHRONOUS JAVASCRIPT AND XML

ACL – ACCESS CONTROL LIST

API – APPLICATION PROGRAMMING INTERFACE

1 UVOD

Spletne aplikacije in spletne strani v današnjem času postajajo vse bolj dinamične in zapletene. Enako velja tudi za jezike, ki definirajo in opisujejo takšne aplikacije. Spletni strežniki postajajo naprednejši, hitrost internetne povezave pa se nenehno povečuje. Te spremembe so povod za naprednejše spletne aplikacije, kar pomeni tudi spremembo v snovanju le-teh [1]. Za vsak nov programski jezik, arhitekturni stil programiranja ali uporabo nekega novega programskega orodja oziroma ogrodja je do neke mere potrebno spremeniti način razmišljanja in posledično tudi celoten proces razvoja spletnih aplikacij [3].

Pred približno 20 leti, ko spletnih aplikacij še ni bilo, smo za izdelavo spletne strani potrebovali samo označevalni jezik HTML, ki je v primerjavi s programskimi jeziki zelo enostaven, saj lahko z njim oblikujemo le vizualni del aplikacij [21]. Pisanje takšnih strani je bilo relativno lahko, predvsem pa hitro. Posledično je stran izgledala zelo preprosto, saj ni vsebovala dinamičnih elementov. Potem pa so se pojavili prvi skriptirni jeziki in z njimi tudi dinamične spletne aplikacije. To je bilo rojstvo dinamičnega spleta, po katerem brskamo še danes [2]. Ker se je zahtevnost procesa razvoja takšnih aplikacij drastično povečala, so zato nastala prva ogrodja. Eden izmed njih je tudi CakePHP, kateri bo osrednja tema naše diplomske naloge.

Osredotočili se bomo na trenutno najbolj popularen odprtokodni skriptirni jezik PHP in njegovo MVC ogrodje CakePHP. V drugem poglavju bomo na kratko opisali strežniške skriptirne jezike, ki jih danes srečujemo najpogosteje. Podrobneje bomo razčlenili jezik PHP, njegovo zgodovino in delovanje. V tretjem poglavju bomo predstavili arhitekture spletnih rešitev in zakaj jih uporabljamo. Četrto in peto poglavje bo namenjeno predstavitvi CakePHP ogrodja in primerjavi le-tega s tradicionalnim načinom razvoja spletnih aplikacij. Prikazali bomo tudi praktični primer uporabe CakePHP ogrodja, in sicer razvoj spletne trgovine.

2 RAZVOJ DINAMIČNIH SPLETNIH REŠITEV

2.1 Strežniški skriptirni jeziki

Skriptirni jezik je visokonivojski programski jezik, ki ga v času izvajanja interpretira nek določen interpretor. Ločimo strežniške (angl. server-side) in uporabniške (angl. client-side) skriptirne jezike. Za razliko od standardnih programskih jezikov (Java, C, C#), pri katerih sistem prvo prebere in prevede celotno kodo ter jo nato zapiše v človeku neberljivo strojno kodo, ki se potem izvaja na poljubnem računalniku, se skriptirni jeziki prevajajo kar v času izvajanja kode. To pomeni da se navodila, ki jih pišemo v obliki izvorne kode, sproti pošiljajo interpreteru na strežniku. Ta jih nato prevede in prikaže skladno z našimi navodili in ko pride do napake, se izvajanje programa popolnoma ustavi in ponavadi spletna aplikacija oziroma spletna stran preko strežnika to napako tudi izpiše [4]. Znano je, da so v povprečju počasnejši od programskih jezikov, zaradi omenjenega načina delovanja. V večini primerov so lažji za pisati, saj je njihova sintaksa preprostejša in v večini primerov bolj prilagodljiva. Recimo pri PHPju pri inicializaciji spremenljivk ni potrebno pripisovati tipa (angl. weak-typing). Še več, pri pretvorbi dveh različnih tipov spremenljivk (2 prištejemo »2«) PHP ne bo vrnil napake, za razliko od programskih jezikov.

Med najpopularnejše skriptirne jezike sodijo PHP, Javascript, JSP, ASP, Perl, VBScript in Ruby.

2.2 Skriptirni jezik PHP

PHP je rekurzivni akronim za PHP: Hypertext Preprocessor in je trenutno najbolj razširjen odprtokodni skriptirni jezik na svetu. Glavni razlog za takšno popularnost je cena. PHP je namreč popolnoma zastonj. Ustvarjen je pod avtorsko licenco PHP license, ki dovoljuje odprto deljenje in manipuliranje kode. Prvotno je bil namenjen za ustvarjanje dinamičnih elementov spletnih strani, danes pa je njegova funckionalnost občutno narasla. Uporablja se

ga tako za izdelovanje preprostih dinamičnih spletnih strani kot za izdelavo zapletenih celostnih rešitev kot so spletni forumi, trgovine in portali.

Še en razlog za razširjenost PHPja je prilagodljivost. Uporabljamo ga lahko na večini strežnikov [17]. Večina razvijalcev ga bo uporabljala skupaj s podatkovno bazo Mysql, združljiv pa je tudi z plačljivimi (angl. enterprise) podatkovnimi bazami kot so Oracle, Mssql in DB2 [5].

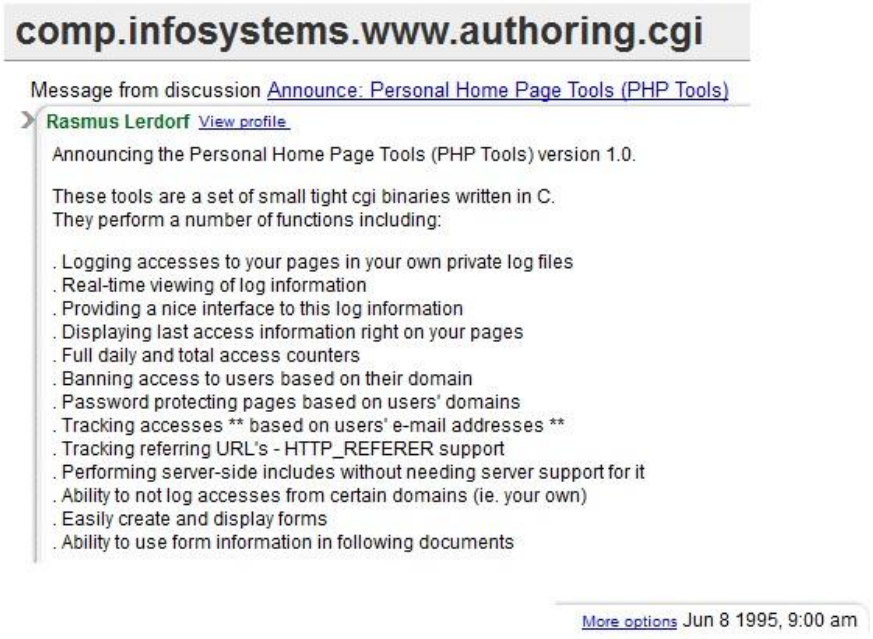
2.2.1 Zgodovina

PHP je nastal leta 1994, ko je Danski programer Rasmus Lerdorf napisal par binarnih datotek CGI napisanih v programskem jeziku C, ki jih je poimenoval »Personal home page tools«. Namen teh je bil beležiti število obiskovalcev na njegovi spletni strani. PHP, kot ga poznamo danes, je v bistvu prednik jezika PHP/FI.

```
<!--include /text/header.html-->

<!--getenv HTTP_USER_AGENT-->
<!--ifsubstr $exec_result Mozilla-->
  Hey, you are using Netscape!<p>
<!--endif-->

<!--sql database select * from table where user='$username'-->
<!--ifless $numentries 1-->
  Sorry, that record does not exist<p>
<!--endif exit-->
  Welcome <!--$user-->!<p>
  You have <!--$index:0--> credits left in your account.<p>

<!--include /text/footer.html-->
```

Slika 1: Primer kode jezika PHP/FI

Sčasoma je Rasmus skripti dodal še več funkcionalnosti. Med njimi je bila tudi zmožnost interakcije s podatkovnimi bazami, kar je odprlo novo, takrat še neznano, poglavje o snovanju dinamičnih spletnih strani. Junija leta 1995 je Rasmus tudi uradno izdal prvo javno verzijo izvorne kode PHP (PHP/FI), katero so razvijalci lahko prilagajali svojim potrebam [6].


comp.infosystems.www.authoring.cgi

Message from discussion Announce: Personal Home Page Tools (PHP Tools)

Rasmus Lerdorf View profile

Announcing the Personal Home Page Tools (PHP Tools) version 1.0.

These tools are a set of small tight cgi binaries written in C.
They perform a number of functions including:

. Logging accesses to your pages in your own private log files
. Real-time viewing of log information
. Providing a nice interface to this log information
. Displaying last access information right on your pages
. Full daily and total access counters
. Banning access to users based on their domain
. Password protecting pages based on users' domains
. Tracking accesses ** based on users' e-mail addresses **
. Tracking referring URL's - HTTP_REFERER support
. Performing server-side includes without needing server support for it
. Ability to not log accesses from certain domains (ie. your own)
. Easily create and display forms
. Ability to use form information in following documents

More options Jun 8 1995, 9:00 am

Slika 2: Originalna objava orodja PHP tools Rasmusa Lerdorfa na Google Groups

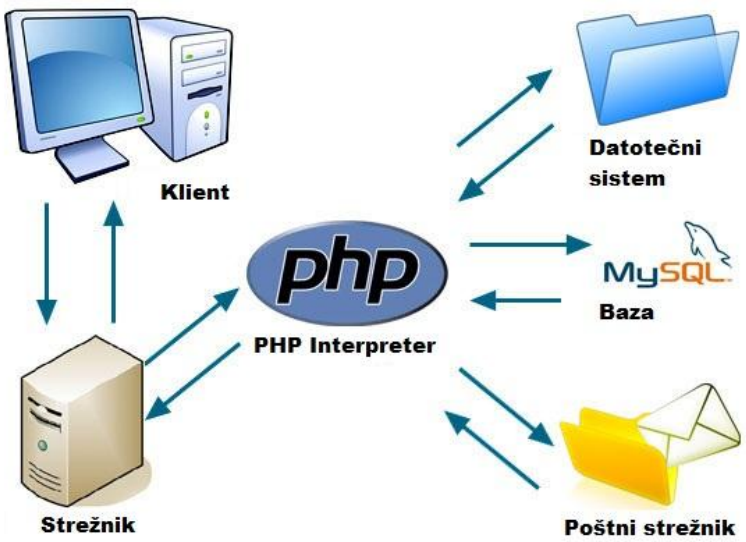
Druga generacija PHPja je izšla leta 1996 (PHP/FI 2.0). Ta je že imela začetno podobo skriptirnega jezika kot ga poznamo danes. Podpiral je podatkovne baze kot so DBM, mSQL, Postgre95, piškotke in uporabniško definirane funkcije. Leta 1997 je imel približno nekaj tisoč podpornikov [6].

Naslednji v vrsti je bil PHP 3.0, ki je nastal kot posledica projekta dveh študentov Andija Gutmansa in Zeeva Suraskija. Na univerzi v Tel Avivu sta hotela razviti spletno trgovino za svoj projekt, pri tem pa sta želela uporabiti prav PHP. Ker takratna verzija ni bila primerna za izdelavo spletne aplikacije kot je spletna trgovina, sta bila prisiljena poseči v izvorno kodo PHPja in njegovega procesorja. Po projektu sta se odločila tudi za aktivno sodelovanje pri razvoju novega programskega jezika PHP. Rezultat je bil generacija 3.0. Od takrat dalje je uradno ime PHP in ne več PHP/FI [6].

Peta in trenutna generacija PHPja je izšla leta 2004. Poganja ga sredica Zend engine 2.0 z novim objektnim modelom, kar je dodatno olajšalo delo z objekti. Podpora objektom je sicer že bila prisotna v verziji 3.0, ampak je bila preveč osiromašena. Uveden je bil nov način za definiranje funkcij. Tako kot v ostalih objektno orientiranih programskih jezikih kot sta Java in C#, je sedaj v PHPju mogoče definirati razredne spremenljivke, metode in jim določiti doseg (final, abstract, private), ter definirati konstruktorje in destruktorje [6].

2.2.2 Delovanje

PHP deluje na podoben način kot njegov predhodnik CGI. Torej razvijalcem ponuja možnost pisanja programov, ki dinamično servirajo HTML strani in procesirajo podatke iz nje. Lahko ga vključimo neposredno v statično kodo HTML in tako ustvarimo dinamične elemente ali pa to postorimo v ozadju, odvisno od arhitekturnega stila. Ker je to strežniško naravnana tehnologija, uporabniki spletnih aplikacij ne potrebujejo posebnih vtičnikov ali razširitev za brskalnike, da bi aplikacija delovala, saj je že vse naloženo in konfigurirano na samem strežniku. Kot je že bilo omenjeno je PHP »predprocesor«, kar pomeni da, ko uporabnik zahteva spletno stran s končnico PHP (zahteva), ta intervenira tik preden strežnik pošlje odgovor nazaj k uporabniku (odgovor). Ta dodaten korak med zahtevo in odgovorom PHP interpreteru omogoča, da izvede dodatne operacije kot so dostopanje do podatkovne baze, pošiljanje e-mailov ali klicanje zunanjih spletnih storitev [7]. Ta proces je ponazorjen na spodnji sliki.



Slika 3: Shema php zahteve in odgovora

Večina profesionalnih spletnih strani, vključno s spletnimi iskalniki, uporabljajo ta model, kjer ima strežnik vmesni procesni korak med dokumentom in uporabnikom.

2.3 Podatkovna baza Mysql

Mysql je med podatkovnimi bazami to, kar je PHP med skriptirnimi jeziki, najpogostejša izbira med spletnimi razvijalci. Razlog za to je seveda cena in zmogljivost. Tako kot PHP je odprtokodna, kar omogoča prosto distribucijo in jo lahko brez stroškov snamemo iz spleta. Trenutno ima največjo rast med odprtokodnimi bazami [18]. Naložimo jo lahko na večino platform, od Linuxa, Mac OS X, FreeBSD, do Windowsa.

Povezovanje do Mysql baze ne predstavlja problema. V PHPju to postorimo s preprosto funkcijo mysql_connect(), v primeru naše spletne trgovine v CakePHPju, pa se to postori v konfiguracijskem dokumentu ogrodja.

2.4 Strežniški paketi za razvijanje spletnih rešitev

Večina spletnih razvijalcev se loti razvijanja na domačem strežniku. To pomeni, da ustvarijo svoj strežnik s pomočjo vnaprej pripravljenih paketov, ki so temu namenjeni. Med te sodijo XAMPP, WAMP, LAMP in MAMP. Gre za odprtokodne pakete, prednaložene s PHPjem, Mysqlom in Apache komponento. Z njimi lahko postavimo strežnik, ki ga poganjamo na lastnem računalniku. Omogočajo nam identične razmere kot na oddaljenem spletnem strežniku. Razvijalci lahko tako razvijajo aplikacije v izoliranem okolju, kar je ne samo varno, ampak tudi bolj učinkovito in omogoča občutno hitrejše razvijanje. Ko spreminjamo kodo, nam ni potrebno čakati da se naloži, kot bi to bilo potrebno na oddaljenem strežniku. Med drugim to porablja tudi naš skupni prenos podatkov, ki pa je ponavadi na plačljivih spletnih strežnikih omejen. Za vsak prenos izvorne kode in raznih dokumentov (slik, video vsebin, itd.) se porablja prenos. Več jih imamo, veča se čas nalaganja na strežnik. Če spreminjamo po eno datoteko in jo naložimo na strežnik, še ni problemov. V kolikor pa hočemo naložiti več skript hkrati pa se lahko ta proces občutno zavleče.

Pomembno je omeniti tudi razhroščevanje aplikacije. Ko smo v procesu popravljanja ali nadgrajevanja spletne aplikacije, se nam bodo pojavljale razne napake v kodi, kar je popolnoma običajno. Če razhroščujemo na oddaljenem strežniku, bodo obiskovalci spletne aplikacije kaj kmalu pobegnili, saj bodo vse napake in spremembe vidne tudi njim. Zato uporabljamo omenjene strežniške pakete, ki proces razhroščevanja izolirajo pred zunanjostjo. Napake in nepravilnosti bodo vidne samo nam. Omembe vredna je tudi konfiguracija

strežnika. Velikokrat se dogaja, da na oddaljenem strežniku vse ne deluje kot bi moralo in to ne zaradi sintakse, ampak zaradi nastavitev strežnika. Nastavitve so pomembne, saj z njimi kontroliramo delovanje strežnika. Apache strežniki imajo module, ki jih lahko aktiviramo ali ne. Pomembnejši moduli so mod_rewrite (omogoča prepisovanje URL naslovov), mod_ssl (modul za kriptografijo) in mod_php (nudi podporo za jezik PHP). Na oddaljenih strežnikih do teh nastavitev ne moremo direktno dostopati, saj so pod nadzorom tujih upraviteljev. V tem primeru moramo kontaktirati ljudi, ki skrbijo za omenjene strežnike. Teh nevšečnosti se z lokalnim strežnikom znebimo. Lahko ga vključimo in izključimo po svojih potrebah, lahko po želji spreminjamo nastavitve. V fazi razvoja in testiranja spletne aplikacije je tako bolj smotrno uporabljati omenjene strežniške pakete na našem računalniku, kot pa oddaljen strežnik.

Problem direktive safe_mode

Pred nekaj leti so se razvijalci PHPja odločili, da bodo skušali rešiti problem varnosti dostopanja do podatkov na deljenem strežniku. Danes je znano, da posredovanje na nivoju PHPja ni bila pametna ideja, ampak ker so bile takratne alternative za ponudnike strežnikov nesprejemljive, so ti potarnali razvijalcem PHPja, naj skušajo rešiti ta problem. Na takšen način je nastala direktiva safe_mode.

Problem se je pojavljal izključno na deljenih strežnikih. Namen te nastavitve je omejiti uporabnikom dostop do datotek, ki jim ne pripadajo, tako da preverja uid/gid identifikator na klicih skript, ki zahtevajo dostop do datoteke ali datotečnega sistema. Če se identifikator skripte ujema z identifikatorjem datoteke, potem se lahko izvede operacija nad njo.

Vse lepo in prav dokler se ne pojavi naslednja situacija. Razvijalci spletnih aplikacij velikokrat uporabljajo vzorec cachinga. Uvedemo ga tako, da shranimo del aplikacije oziroma strani, kateri se ne spreminja in ga uporabniki pogosto zahtevajo (meniji, FAQ, footerji), v datoteko. Potem ta del prikličemo iz datoteke vsakič, ko nek uporabnik zahteva stran, ki vsebuje ta omenjen del in tako prihranimo čas nalaganja. Strežnik bo to datoteko seveda ustvaril, ampak težava se pojavi, ko želimo do te datoteke dostopati. Namreč vaš uid/gid se ne bo ujemal s tem od datoteke, saj ga je ustvaril strežnik in tako pride do konflikta. Največkrat se je to opazilo tudi pri instaliranju CMSov. Safe_mode je blokiral instalacijo, saj večina CMSov zahteva možnost manipulacije z direktoriji in datotekami.

Direktivo so uradno od verzije 5.3.0 naprej označili kot »deprecated« ali opuščeno in tako prisilili ponudnike strežnikov po svetu, ki temeljijo na pogonu PHP, da jo opustijo [8].

2.5 Ogrodja za hiter razvoj spletnih rešitev

Ogrodja za razvoj spletnih rešitev nam omogočajo hitrejše in lažje razvijanje novih aplikacij. Razvijalcem pomagajo pri hitrosti razvoja, zanesljivosti in skalabilnosti aplikacij. V času razvijanja naše spletne trgovine z ogrodjem CakePHP smo ugotovili, da je najpomebnejša lastnost, ki ti jo lahko nudi neko ogrodje, skrajšan čas priprave. Ogrodja v večini primerov vsebujejo najpopularnejše knjižnice, komponente, razširitve in šablonske skripte, kar nam občutno zniža čas začetne priprave. Kakor veliko stvari ki se jih lotevamo v življenju, je tudi pri razvijanju spletnih aplikacij najtežji začetek. Priprava obsega od prebiranja člankov in nasvetov na spletu do pisanja prvih delčkov kode. Od kvalitetne priprave je pozneje odvisna tudi kvaliteta same aplikacije. Več planiranja posledično pomeni manj napak, ampak tudi več zapravljenega časa. Ogrodje nam na tem področju izjemno pomaga. To vsekakor ni samo skupek funkcijskih knjižnic pripravljenih za ponovno uporabo. Če uporabljate samo knjižnice, boste morali objekte, ki se nahajajo v njih instancirati znotraj svoje aplikacije, še več, natančno boste morali vedeti kje jih klicati in katere metode uporabiti . Ogrodje pa nam omogoča uporabljanje že instanciranih objektov in metod znotraj njih, ki so vnaprej pripravljeni in prilagojeni naši aplikaciji. Splošno gledano nam ogrodje definira pretok nadzora grajenja aplikacije. V primeru da nam nastavljene metode niso v korist, jih lahko vedno predefiniramo znotraj razredov, kar večina razvijalcev sicer ne priporoča, saj posegamo v integriteto samega ogrodja, ali pa jih predefiniramo v kodi svoji aplikacije (metode znotraj objektov ogrodij ponavadi sledijo načelom polimorfizma, kar poveča uporabnost in zviša skalabilnost aplikacije). Moramo pa vedeti, da za vso to funkcionalnostjo in širino stoji kompromis. S tem ko se naše aplikacije večajo v obsegu, se hitro zgodi, da ogrodja začnejo pešati z vidika odzivnosti in performans, kar je posledica številnih abstrakcij. Bolj kot se oddaljujemo od jedra preprostih programskih konstruktov, bolj smo neučinkoviti [9]. Na spletu trenutno obstaja 6 večjih PHPjevih ogrodij. Dejstvo je, da se med seboj do neke mere razlikujejo. Vsako ogrodje ima svoje prednosti in pomanjkljivosti. Odločitev, katero ogrodje bomo izbrali za razvijanje spletne aplikacije, je odvisna od tega, čemu dajemo prednost [20]. Mi smo se odločili za CakePHP zato, ker ima odlično uporabniško bazo in podporo ter zelo urejeno sintakso.

Prednosti in slabosti ogrodij za razvoj spletnih rešitev so sledeči.

Prednosti:

- Ponovna uporabljivost kode, ki je bila optimizirana in testirana s strani razvijalcev ogrodja,
- Povečana zanesljivost spletne aplikacije, ki jo gradimo,
- Zmanjševanje potrebnega časa za konfiguracijo aplikacije,
- Včasih nam pomaga vzpostaviti stil »lepega« in praktičnega programiranja ter uporabe preverjenih praks in vzorcev,
- Po definiciji nam ogrodje poda možnost nadgrajevanja brez poseganja v izvorno kodo naše aplikacije.

Slabosti:

- Izdelava ogrodja je časovno potratno in zahtevno delo,
- Krivulja učenja je ponavadi strma,
- Čez čas lahko ogrodje postane izredno kompleksno.
- Pri večjih in zapletenejših aplikacijah večina ogrodij postane neučinkovitih.

Tabela 1: Najpopularnejša ogrodja jezika PHP [19], [20]

PHP ogrodje	**Trenutna stabilna verzija**	**Datum izida**	**Licenca**
CakePHP	1.3.10	30-05-2011	MIT
CodeIgniter	2.0.2	07-04-2011-	BSD
Lithium	0.10	18-06-2011	BSD
Symfony	2.0	28-07-2011	MIT
Yii	1.1.8	26-06-2011	BSD
Zend framework	1.11.10	03-08-2011	BSD

Kakor ostala programska ogrodja, tudi PHP ogrodja bazirajo na prednaloženih in konfiguriranih knjižnicah, objektih in metodah, ki nam pomagajo pri hitrem razvoju spletnih aplikacij. Vsa od navedenih temeljijo na arhitekturnem vzorcu MVC, ki smo ga podrobneje opisali v naslednjem poglavju [10].

3 ARHITEKTURA SPLETNIH REŠITEV

Eden od ključnih elementov konstrukcije katerekoli spletne aplikacije je arhitektura sistema. Ta definira interakcijo med posameznimi deli aplikacije in kakšno funkcijo imajo. V času statičnih spletnih strani je bila arhitektura preprosta. S prihodom dinamičnih spletnih aplikacij pa se je arhitektura občutno spremenila. Takšne aplikacije potrebujejo močno in zanesljivo ogrodje, ki ga lahko nudi samo spretno načrtovana arhitektura. Pri manjših spletnih aplikacijah arhitektura nima takšnega pomena kot pri večjih in ni tako pomembno na kakšen način jih bomo zgradili, saj bo obremenitev na njih manjša kot pri večjih spletnih aplikacijah. Pri večjih spletnih aplikacijah so obremenitve velike in izstopa vsaka manjša nepravilnost v arhitekturi. V glavnem niso samo povečane verzije manjših aplikacij, ampak imajo v ozadju tudi drugačno arhitekturo, ki jim omogoča optimalno delovanje.

Trenutno obstaja veliko različnih arhitekturnih vzorcev, ki vsak za sebe služijo točno določenemu namenu. Generalno klasificiramo aplikacije glede na to, kateri arhitekturni vzorec uporabljajo [11].

Večina spletnih aplikacij uporablja enega izmed spodaj navedenih arhitekturnih vzorcev:

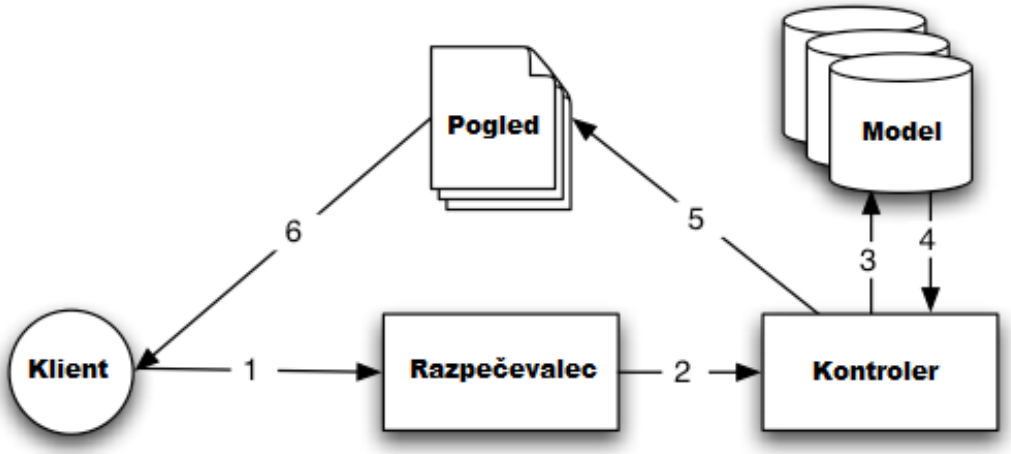
- Event-driven architecture (sem spada arhitektura SOA),
- Model-View-Controler ali Presentation-Abstraction-control,
- Multitier architecture,
- Layers,
- Pipeline and filters,
- Implicit invocation.

3.1 Model-View-Controller arhitektura

Arhitekturni vzorec MVC je temelj ogrodja CakePHP. Namen MVC arhitekture je ločiti poslovno logiko in aplikacijske podatke od prezentacije za uporabnika. Srečamo jo tako v spletnih kot namiznih aplikacijah. Vsak programski jezik ima neke vrste prikrojeno implementacijo MVCja, ampak osnovni pretok nadzora aplikacije ostaja enak:

1. Uporabnik komunicira z aplikacijskim vmesnikom preko klika miške na določen gumb,
2. Kontroler prestreže sproženo akcijo klika gumba oziroma prestreže podatke, ki so se poslali ob kliku gumba,
3. Kontroler o dogodku obvesti model (ali mu spremeni stanje) in mu ponavadi pošlje podatke, če so mu bili posredovani,
4. Model lahko predstavlja več podatkovnih virov. To so lahko LDAP vnosi, RSS viri, datoteke na sistemu ali podatkovna baza. S temi podatki torej manipulira tako, da izvaja CRUD operacije nad svojim virom,
5. Pogled te podatke preko modela direktno prikaže, ali pa jih model najprej posreduje kontrolerju za dodatno manipulacijo in jih ta pošlje pogledu, odvisno od sistema,
6. Uporabniški vmesnik počaka na možno interakcijo z uporabnikom, kar ponovi celoten cikel.

V našem izbranem ogrodju CakePHP je MVC način rokovanja z aplikacijo enak omenjenemu. Iz prakse smo ugotovili, da je možno tudi odstopati od te šablone, kar ni priporočljivo, saj negativno vplivamo na celotno arhitekturo aplikacije. Je pa tudi možno prilagoditi ta cikel našim potrebam, kar je dobrodošlo. Preprost cikel klicov na CakePHP MVC arhitekturi izgleda kot je prikazano na sliki.



Slika 4: Shema klicev v MVC arhitekturi ogrodja CakePHP

Klient

Predstavlja uporabnika, ki komunicira z našo aplikacijo preko aplikacijskega vmesnika.

Razpečevalec

Je razpečevalec zahtev, ki prejme uporabnikovo zahtevo preko spletnega brskalnika in jo posreduje kontrolerju. V primeru CakePHPja je med razpečevalcem in kontrolerjem še »router« ali usmerjevalnik, ki skrbi za mapiranje URL naslovov v akcije kontrolerja.

Kontroler

Gre za razred, ki skrbi za prestrezanje in prevajanje uporabnikovih vhodov v akcije, ki prožijo model. Predstavljamo si ga lahko kot neke vrste klicni center 112. Ta prejema klice (vhod uporabnika) in glede na vrsto klica, ga vežejo dalje do ustrezne službe (kontroler, ki nas veže do modela). Policaji, gasilci ali reševalci bi v tem primeru predstavljali model aplikacije. Akcije kontrolerja so funkcije oziroma metode, ki jih uporabnik proži, ko zahteva nek naslov v obliki URL naslova.

Model

Predstavlja aplikacijske podatke in vsebuje logiko za dostopanje in manipuliranje z njimi. Katerikoli podatek, ki je del aplikacije je na voljo v modelu. Metode ki jih ustvarimo v modelu, morajo biti dovolj robustne in fleksibilne, da omogočajo rokovanje z večimi različnimi tipi podatkov, saj si podvajanja metod ne smemo privoščiti. Model vsebuje funkcionalnost za validiranje podatkov, definiranje in združevanje tabel v podatkovni bazi (relacije) in asociacije z drugimi modeli v aplikaciji.

Pogled

Je zadolžen za generiranje specifičnega izhoda. To je lahko v obliki celotne spletne strani, dela spletne strani, XML ali JSON dokumenta. Zadolžen je tudi za različne datotečne tokove kot so kreiranje datotek in PDFov. Pogledi so datoteke, ki imajo v večini primerov končnico HTML, v primeru CakePHPja pa so to datoteke s končnico CTP [12].

4 OGRODJE CAKEPHP

CakePHP je tako kot njegov jezik, ki ga sestavlja, odprtokodni in zastonjski. Je ogrodje za hitro razvijanje spletnih aplikacij in predstavlja pomemben mejnik v evoluciji PHPja, saj jeziku PHP podaja nov, strukturiran in modularen način snovanja spletnih aplikacij. Ima aktivno razvijalsko ekipo in skupnost, ki skrbi za nenehno posodabljanje izvorne kode, popravljanje hroščev in pisanje dokumentacije. CakePHP nekako spominja na ogrodje Ruby on rails, ki je ogrodje za objektni programski jezik Ruby, saj je bil modeliran prav v njegovi podobi. Delita si namreč nekaj lastnosti kot so MVC arhitektura, scaffolding, podporo za Javascript, itd.

Glavne funkcionalnosti ogrodja so:

- Kompatibilnost s PHP4 in PHP5,
- Integrirana podpora CRUD operacijam,
- Scaffolding,
- Interaktivna konzola za hitro generiranje izvorne kode,
- MVC arhitektura,
- Usmerjevalnik, ki nam omogoča čiste URL naslove,
- Vgrajena validacija podatkov,
- Hiter in fleksibilen sistem šablon,
- Pomočniki pri ustvarjanju pogledov (AJAX, Javascript, HTML, podpora obrazcem),
- Pomočniki za pomembne spletne komponente (Email, Cookie, Security, Request Handler),
- Sanacija vhodov,
- Lokalizacija.

4.1 Datotečna struktura ogrodja CakePHP

Paket ogrodja CakePHP prihaja v specifični obliki. Datoteke so logično razporejene v drevesno strukturo, ki nam zagotavlja jasen pregled in ločevanje nad posameznimi elementi aplikacije.

Tabela 2: Datoteke ogrodja CakePHP

config	Vsebuje konfiguracijske datoteke, kot so povezava za bazo in jedro (core).
controllers	Kontrolerji in njihove komponente.
libs	Knjižnice, ki smo jih ustvarili sami, posebej za aplikacijo in ki niso v lasti zunanjih izvajalcev (3rd party).
locale	Tekstovne datoteke potrebne za internacionalizacijo aplikacije.
plugins	Vtičniki.
tmp	Tu se nahajajo začasni podatki, kot so logi, sejne informacije in caching podatki.
vendors	Katerikoli zunanji razredi, ki jih želimo vključiti v našo aplikacijo. S tem omogočimo dostop do njih kjerkoli v naši aplikaciji preko funkcije App::import().
views	Predstavitveni dokumenti tipa CTP, elementi (elements), error strani, pomočniki (helpers) in šablone (layouts).
models	Modeli, behaviors in datasources.
webroot	Ko aplikacijo naložimo na strežnik, je ta datoteka korenski element. Prav tako se v njej nahajajo CSS datoteke, slike in Javascript skripte.

4.2 Scaffolding

Ta funkcionalnost nam pomaga pri razvijanju uporabniškega vmesnika in osnovnih CRUD operacij za našo spletno aplikacijo. To dosežemo z inicializacijo spremenljivke scaffold v kontrolerju.

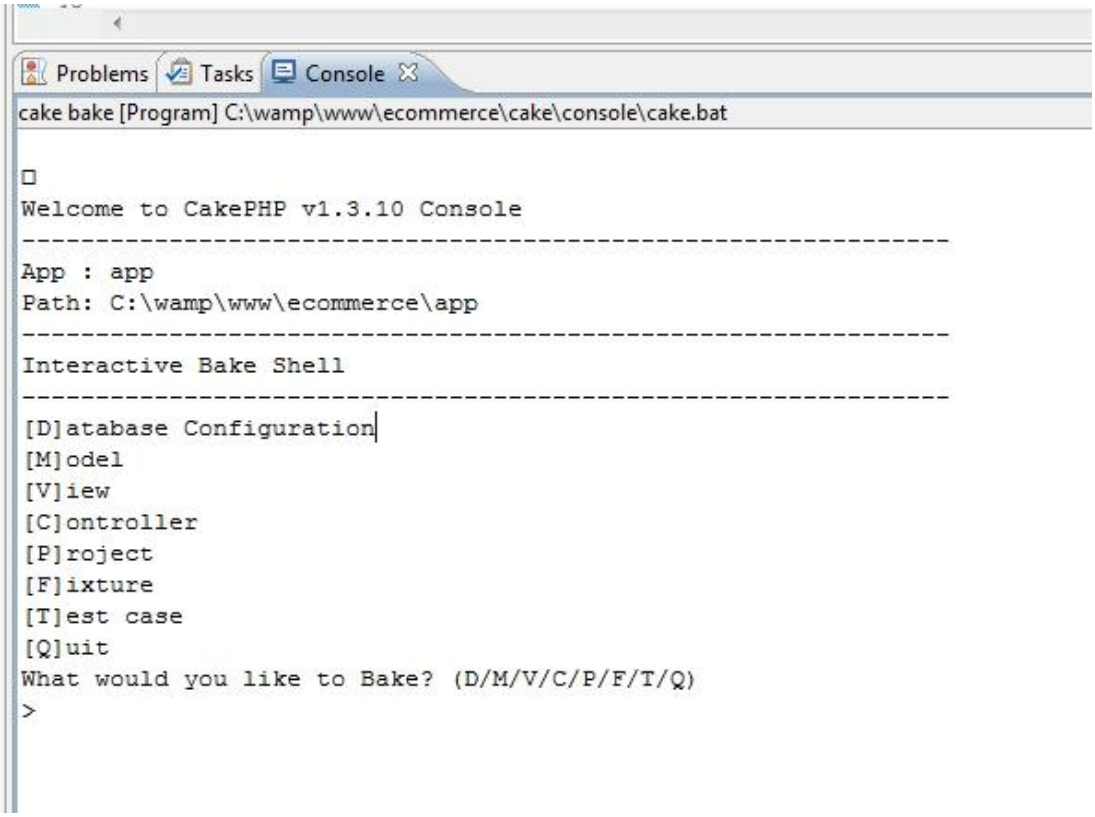
```php
<?php
class CategoriesController extends AppController {
    var $scaffold;
}
?>
```

Slika 5: Inicializacija funkcionalnosti scaffolding

Potrebujemo samo še funkcije v kontrolerju in modelu, ki se bodo klicale ob zahtevi. Ko se bo zahteva za določeno akcijo sprožila, nam bo direktiva $scaffold v ozadju generirala pogled preko katerega bomo lahko prožili metode kontrolerja. Ni pa priporočljivo scaffold uporabljati v produkcijske namene, saj je to le začasna rešitev. Uporablja se predvsem v začetnih fazah razvoja, ko še nimamo definiranega spletnega vmesnika in bi vendarle želeli preizkusiti funkcije aplikacije. To nam občutno skrajša čas testiranja, namreč za izgled ponavadi porabimo veliko časa, še posebej ko bi radi prikazali podatke v tabelah ali pa bi radi generirali obrazce.

4.3 Interaktivna CakePHP konzola

S paketom CakePHP pride tudi preprosta konzolna aplikacija cake. Precej ironično, je ime poglavite funkcije konzole »bake« ali po slovensko peči. Ta direktiva sproži proces kreacije novih razredov (pogledov, kontrolerjev in modelov), konfiguracijo podatkovne baze ali testnih primerov.



Slika 6: Izgled cake konzole po klicu direktive "bake" v okolju Eclipse PDT

V večini primerov začnemo spletno aplikacijo razvijati z načrtovanjem podatkovne baze. Tudi pri CakePHPju nas nekako napeljejo, da najprej naredimo načrt baze in pričnemo ustvarjati tabele ter njene atribute. V kolikor smo s procesom zaključili, lahko potem uporabimo konzolo in ustvarimo konfiguracijsko datoteko, ki bo vsebovala nastavitve naše podatkovne baze. V naslednjem koraku že lahko ustvarimo prvi model. Konzola nam, če smo pravilno nastavili konfiguracijsko datoteko, izpiše vse možne tabele v bazi, na katere je možno vezati model. Eden model je lahko vezan le na eno tabelo (recimo komitenti). Ko smo tabelo izbrali nam konzola ustvari datoteko s končnico PHP in jo skladno s tabelo tudi imenuje (komitenti.php). Možni so še vmesni koraki kot so ustvarjanje validacijskih pravil na posamezne atribute.

Potem je na vrsti kontroler. Konzola nam spet avtomatično prepozna vse modele, ki smo jih do sedaj kreirali. Ker je kontroler odvisen od modela, razen v redkih primerih ko ne potrebujemo vira podatkov, je nujno da smo pred tem že ustvarili model.

Nazadnje kličemo še direktivo View, ki nam ustvari poglede glede na metode oziroma funkcije izbranega kontrolerja.

4.4 Convention over configuration

»Convention over configuration« je stil programiranja, kjer razvijalec favorizira načela jezika oziroma ogrodja v katerem razvija, in ne rekonfiguracije le-tega in predstavlja temelj CakePHPja. V kontekstu to pomeni, da če se držimo priporočene sintakse (od deklaracij funkcij in spremenljivk do poimenovanja razredov in tabel), bomo na koncu imeli lažje delo kot pa če bi vsilili svoj način, za kar bi morali posledično posegati v konfiguracijo ogrodja. Prednost takšnega stila programiranja je v tem, da v času razvijanja občutno zmanjšamo čas odločanja in tudi število odločitev, pri tem pa pridobimo na preprostosti.

Praktični primer bi bil poimenovanje razredov v CakePHPju. Če poimenujemo razrede pogledov, kontrolerjev in modelov kot narekuje konvencija CakePHP, nam bo ogrodje avtomatsko prepoznalo in povezalo te razrede med seboj, brez dodatne konfiguracije. Na primer, če imamo kontroler z imenom komitenti_controller.php, bo ogrodje avtomatsko domnevalo da se bodo pripadajoči pogledi nahajali v datoteki views/komitenti/ in da bo njegov model /models/komitent.php. Vsa konfiguracija velja tudi za podatkovno bazo. Če bomo poimenovali naš model user.php, bo orodje predpostavljalo, da je tabela v podatkovni bazi, ki pripada temu modelu imenovana users (tabele se poimenujejo v množini, modeli v ednini, kotrolerji v množini s pripisom besede »controller«, pogledi pa po akcijah kontrolerjev) [13].

4.5 Komponente in pomočniki

CakePHP že omenjene funkcionalnosti razširi še z dodatnimi pomožnimi razredi, ki se delijo na jedrne (angl. core) in tiste, ki jih ustvarimo sami.

4.5.1 Komponente

Komponente nam služijo kot pomagala kontrolerjem in predstavljajo celovite rešitve, ki jih lahko implementiramo v svojo aplikacijo (tabela 3).

Tabela 3: Komponente CakePHP

Komponenta	**Opis**
Acl	Nam poda vmesnik za dostop do baze in manipulacijo s tabelami pravic dostopanja.
Auth	Razširi aplikacijo s sistemom avtentikacije.
Cookie	Podaja funkcije za manipulacijo s piškotki.
Email	Vmesnik za pošiljanje emailov.
RequestHandler	Omogoča izboljšano sledenje uporabnikovim zahtevam.
Security	Izboljša varnost aplikacije preko manipulacije s HTTP avtentikacijo.
Session	Ovojnica PHPjevim sejam.

4.5.2 Pomočniki

Pomočniki so podobni komponentam, vendar z njim razširjamo funkcionalnosti predstavitvenega nivoja in ne logičnega. Ponujajo nam številne funkcije, ki jih lahko uporabimo v elementih, šablonah in pogledih.

Tabela 4: Pomočniki CakePHP

Pomočnik	**Opis**
AJAX	V sklopu razreda JsHelper ponuja bližnjice do integracije z AJAX metodami.
Cache	Nudi metode za caching elementov in pogledov.
Form	FormHelper razred nam omogoči uporabo bližnjic pri pisanju obrazcev.
HTML	Pomočnik pri kreaciji HTML elementov.

Javascript	Enako kot pri AJAX imata od verzije 1.3 naprej skupni razred JsHelper, ki nam pomaga pri implementaciji Javascript funkcij.
Number	NumberHelper razred nam nudi funkcije za upravljanje s števili.
Paginator	Vključimo ga, ko želimo izpise podatkov na pogled oviti v oštevilčene strani.
RSS	Pomagač pri generiranju RSS virov.
Session	Tako kot komponenta, je Session hkrati tudi pomagač. Uporablja se pri izpisovanju podatkov iz seje.
Text	Razred, ki nam ponuja funkcije za manipuliranje s tekstom.
Time	Podaljšek PHPjevim funkcijam za čas.
XML	Uporablja se ga pri branju in pisanju xmlov.

5 RAZVOJ SPLETNE APLIKACIJE S POMOČJO OGRODJA CAKEPHP

V okviru diplomske naloge smo izdelali spletno trgovino Ecommerce, ki odraža jedrne funkcionalnosti ogrodja CakePHP.

5.1 Predstavitev razvoja spletne rešitve Ecommerce

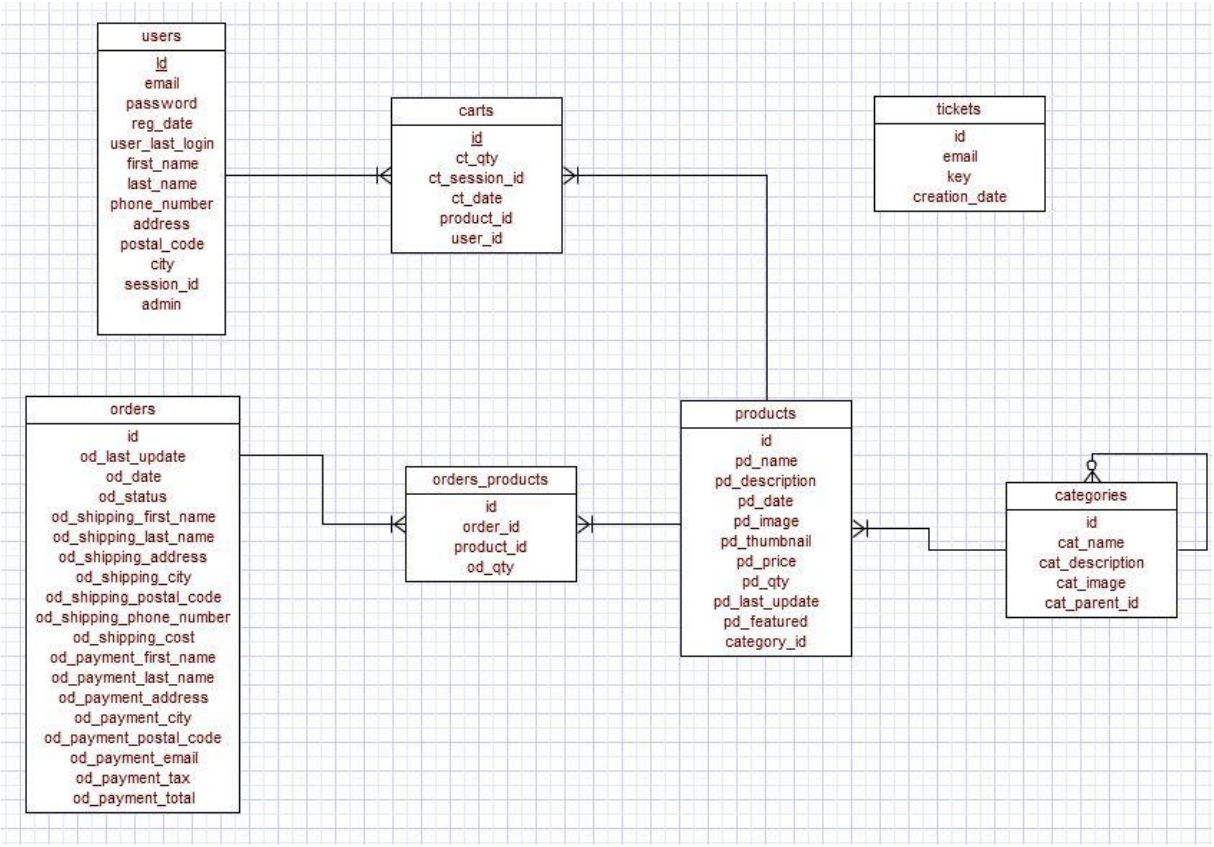
Proces razvoja smo pričeli z raziskovanjem potrebnih informacij in gradiva za izdelavo. Trajal je približno 45 delovnih dni in je zajemal naslednje faze:

- Študija ogrodja CakePHP,
- Raziskovanje funkcionalnosti znanih spletnih trgovin (mimovrste.com, enaa.com),
- Analiza arhitekture spletne trgovine,
- Raziskovanje MVC arhitekture,
- Študija modelov podatkovnih baz za spletne trgovine,
- Priprava okolja za razvoj,
- Implementacija baze,
- Implementacija osnovnih funkcionalnosti,
- Implementacija spletnega vmesnika,
- Ločitev uporabniških in administratorskih funkcionalnosti in pogledov,
- Polnjenje baze s testnimi podatki,
- Testiranje aplikacije.

5.1.1 Načrtovanje

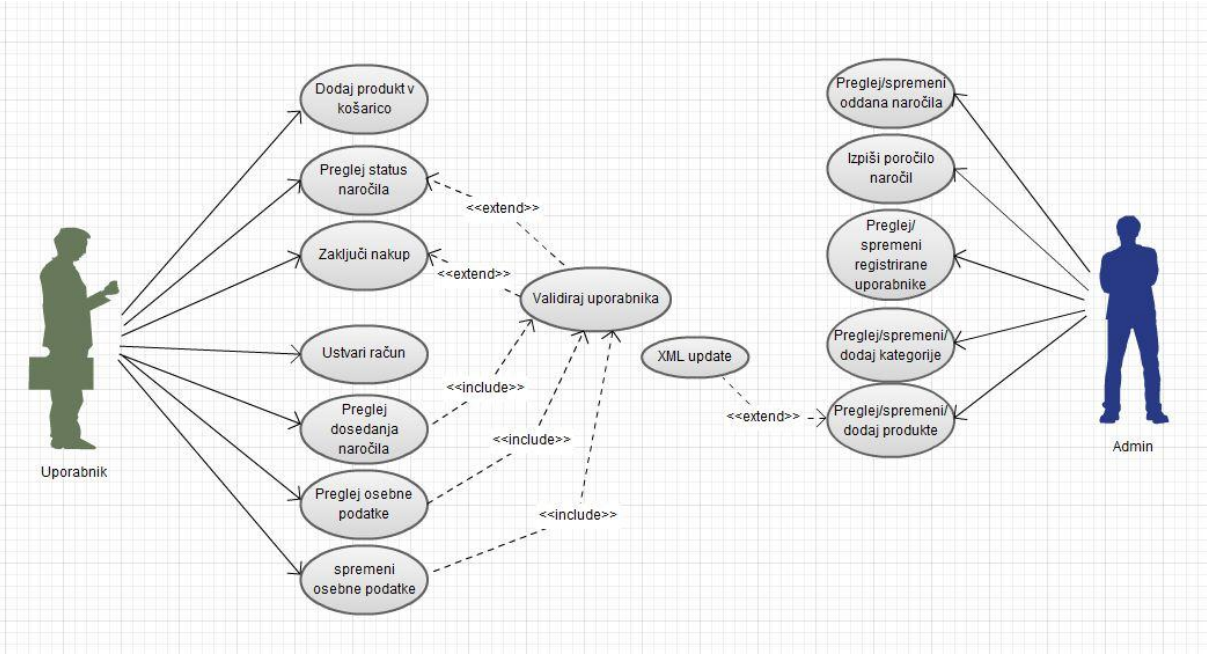
Spletna trgovina je spletna aplikacija, ki obiskovalcem omogoča nakup želenega blaga preko spleta. Kompleksnost takšnega tipa aplikacije je relativna. Pri načrtovanju smo posebej pazili, da bo aplikacija imela potrebno globino ampak omejeno širino, saj bi se v nasprotnem

primeru časovni okvir razvoja kaj hitro povečal. Proces načrtovanja smo pričeli s študijo drobovja ogrodja CakePHP. Raziskali smo njihov uradni API in e-knjigo, ki ponuja hitro in preprosto razlago na podlagi praktičnih primerov. Sledila je študija aktualnih spletnih trgovin. Prebrskali smo po njihovih straneh in artiklih za pridobitev nekega osnovnega koncepta zgradbe spletne trgovine. Zabeležili smo si funkcionalnosti, ki so se nam zdele primerne za našo aplikacijo. Ko je bil zajem potrebnih informacij končan, smo pričeli z načrtovanjem arhitekture naše spletne trgovine. Prvo smo se lotili načrtovanja podatkovne baze, saj CakePHP spodbuja takšen način razvoja. Skicirali smo si E-R diagram za osnovni pregled entitet in relacij med njimi (slika 7).



Slika 7: E-R diagram podatkovne baze naše aplikacije

Potem smo ta diagram preslikali v konkretno bazo, ki smo jo naredili s pomočjo orodja Phpmyadmin. Ko je bila baza pripravljena, smo se lotili načrtovanja logičnega dela aplikacije. Preučili smo zajete funkcionalnosti in jih vključili v diagram primerov uporabe (slika 8).



Slika 8: Diagram primera uporabe

Diagram nam prikazuje interakcijo med uporabniki in našim sistemom. V njem so zajete poglavite funkcionalnosti, ki smo jih pozneje vključili v našo spletno trgovino. Definirali smo dva uporabnika – obiskovalca strani (potencialni kupec) in administratorja. Obiskovalec bo imel možnost registracije in vzdrževanja svojega profila ter opravljanje nakupov. Administrator bo imel na voljo celoten pogled na trgovino iz ozadja aplikacije. Izvajal bo lahko jedrne funkcionalnosti vnašanja/brisanja/posodabljanja artiklov in njihovih (pod)kategorij. Pregledoval bo lahko oddana naročila in jim spreminjal status. Na voljo bo imel tudi časovno poročilo zaključenih naročil. Za oba akterja bo potrebna tudi avtentikacija in validacija, ki nam bo nudila potrebno varnost.

Ker nam CakePHP že nudi osnovno datotečno strukturo in MVC ogrodje za programiranje nam ni potrebno izgubljati časa za pripravo posameznih datotek in postavljanje začetne arhitekture.

Pri izdelavi spletnega vmesnika je zelo pomembna struktura. Posamezni gradniki in elementi vizualnega nivoja so ogledalo naše aplikacije. To je del preko katerega komunicirajo naši uporabniki. Ker razvijamo spletno trgovino, bo naš spletni vmesnik spletna stran. Upoštevali smo, da je najpomembnejša kvaliteta na področju dizajna spletnih trgovin preprostost. Večina jih uporablja enostavno tri-stolpično strukturo strani, brez kakšnih požrešnih medijskih vsebin (audio/video). S tem pridobimo na preglednosti in hitrosti, kar je nujno za trgovino. Uporabniki obiščejo spletno trgovino z namenom nakupa izdelkov in pri tem ne sme biti

motečih elementov, saj jih bodo od tega odvrnili. Enako velja za hitrost. Počasnejša je naša spletna aplikacija, večja je možnost da jo bodo zapustili.

Za našo spletno trgovino smo uporabili enostaven tri-stolpični dizajn. Levi stolpec je namenjen meniju, sredinski prikazu glavne vsebine (artikli, vsebinska sporočila) in desni prikazu košarice ter po možnosti oglasnih vsebin (pasice). CakePHP nam nudi veliko izbire pri razvoju vizualnega nivoja. Strani lahko ustvarjamo enostavno s CTP datotekami v direktoriju views tako, da vsaka CTP datoteka predstavljala stran za sebe ali pa spletno stran razdelimo na posamezne elemente, ki jih nato lepimo skupaj in posebej osvežujemo. Slednja opcija se nam je zdela idealna, saj potrebujemo tri vsebinsko različne elemente, ki bi jih neodvisno spreminjali. Takšen tip oblikovanja tudi izboljša modularnost aplikacije. Bolj kot je razčlenjena, lažje jo spreminjamo, razhroščujemo in posodabljamo brez vpliva na ostale elemente aplikacije. To je tudi glavno vodilo celotne arhitekture MVC.

5.1.2 Implementacija podatkovne baze in modelov

Pričeli smo z implementacijo podatkovne baze. Ustvarili smo tabele in atribute kot smo prikazali na E-R diagramu.

Tabela 5: Tabele v bazi naše spletne trgovine

Ime tabele	**Opis**
users	Tabela za hrambo uporabnikov in administratorjev.
products	Tabela, ki vsebuje podatke o artiklih.
categories	Tabela za (pod)kategorije.
orders	Shranjuje podatke o naročilih.
tickets	Tabela za shranjevanje uporabnikovih zahtev po spremembi gesla.
carts	Tabela, ki predstavlja košarico. V njej se shranjujejo osebe, ki so v košarico dodale artikel in artikel.
orders_products	Vmesna tabela, ki povezuje naročila z artikli

Ko je bila podatkovna baza nared, smo zagnali CakePHP konzolo in z njeno pomočjo ustvarili konfiguracijski razred in razrede modelov. Tako smo za vsako od tabel dobili natanko en model (razen za vmesne tabele).

```php
<?php
    class DATABASE_CONFIG {

        var $default = array(
            'driver' => 'mysql',
            'persistent' => false,
            'host' => 'localhost',
            'login' => 'root',
            'password' => '',
            'database' => 'ecommerce',
        );
    }
?>
```

Slika 9: Generirana konfiguracijska datoteka za bazo

V vsakem modelu moramo določiti morebitne asociacije med tabelami. Če smo upoštevali konvencijo poimenovanja CakePHP, nam ogrodje kar samo prepozna in definira povezave med tabelami glede na tuje in primarne ključe. To nam omogoči popolno izkoriščanje funkcije branja iz baze - find(), ki ob proženju v modelu izvleče tudi podatke povezanih tabel. Torej, če zahtevamo nekega uporabnika, nam bo funkcija find() vrnila tudi vsebino njegove košarice (has many asociacija).

5.1.3 Implementacija funkcionalnosti

Podlaga za našo aplikacijo je bila nared. Naslednja faza je bila implementacija centralne logike. Preden smo se lotili razvijanja razredov kontrolerjev, smo v namen testiranja postavili neko osnovno šablono spletne strani kjer se bodo izpisovali podatki, tako da bomo ob preizkušanju videli ali funkcije delujejo pravilno.

Funkcionalnosti aplikacije:

- Uporabnik
    - Pregled artiklov,
    - Dodajanje artiklov v košarico,
    - Pregled košarice,
    - Brisanje/spreminjanje količine artiklov v košarici,
    - Oddaja naročila,
    - Registracija uporabnika,
    - Pregled računa uporabnika,
    - Spreminjanje uporabnikovih podatkov (vpisni/dostavni podatki),
    - Pregled posameznih naročil uporabnika na njegovem računu.
- Administrator
    - Pregled zalog artiklov
    - Spreminjanje zalog,
    - Dodajanje/spreminjanje/brisanje artiklov,
    - Dodajanje/spreminjanje/brisanje kategorij,
    - Manipuliranje z uporabniki,
    - Pregled registriranih uporabnikov,
    - Pregled oddanih naročil,
    - Spreminjanje stanja naročil,

- Izpis poročila glede na čas.

Pregled artiklov

Prvo smo razvili kontroler products_controller.php - razred ki bo skrbel za logiko artiklov. V njem bodo funkcije kot so pregled vseh artiklov, pregled posameznega artikla in pa funkcije administratorja za manipulacijo z njimi. Funkcija namenjena prikazu artiklov se bo prožila, ko bo uporabnik kliknil na kategorijo v meniju in bo ta funkcija vrnila seznam, ki ga bomo v pogledu formatirali v uporabniku berljivo obliko.

```
function listProducts(){

    $this->autoRender = false;
    $this->Product->recursive = 1;

    $categories = $this->Product->Category->getAllCategories();
    $categories = $this->Product->Category->buildCategories($categories,$this->passedArgs['c']);
    $catChildren = $this->Product->Category->getChildCategories($categories,$this->passedArgs['c'],true);
    $allCatIds = array_merge(array($this->passedArgs['c']),$catChildren);

    //pr($allCatIds);
    $this->paginate = array('conditions' => array('Product.category_id' => $allCatIds),'limit' => 10);
    $data = $this->paginate();

    //Saniramo html znake iz opisa izdelka
    $i=0;
    foreach($data as $product){
        $data[$i]['Product']['pd_description'] = Sanitize::html($product['Product']['pd_description'], array('remove' => true));
        $i++;
    }

    $this->set('products',$data);
    $this->helpers['Paginator'] = array('ajax' => 'Ajax');
    //pr($this->helpers);
    if ($this->RequestHandler->isAjax()) {
        $this->render('/elements/product_list');
        return;
    }

    if(isset($this->params['requested'])){
        //ClassRegistry::getObject('view')->loaded['paginator']->params = $this->params;
        //$this->set('paging', $this->params['paging']);
        return array('products' => $data, 'paging' => $this->params['paging']);
    }else{

        $this->render('/elements/product_list');
    }
}
```

Slika 10: Funkcija listProducts() za izpis artiklov glede na izbrano (pod)kategorijo

Dodajanje artiklov v košarico

To je ena izmed poglavitih funkcionalnosti spletne trgovine. Uporabniku moramo omogočiti da bo želeni artikel lahko dodal v košarico in ga pozneje tudi kupil. To funkcionalnost smo implementirali v kontrolerju carts. Naredili smo funkcijo addToCart(), ki se bo prožila ob kliku na gumb »add to cart« v pogledu. Prvo pridobimo ID od artikla, ki ga želimo dodati in se vprašamo ali je ta produkt že v košarici. Če je, potem mu povečamo kvantiteto, če ne, pa ga shranimo v bazo, v tabelo carts. Poleg tega se še bo shranil uporabnikov ID oziroma sejni ID,

odvisno od tega ali je uporabnik registriran in vpisan v sistem ali ne. Tako lahko potem povežemo uporabnika z vnešenim artiklom. Na koncu funkcije smo dodali še funkcijo cleanUp(), ki skrbi za redno čiščenje polnih košaric. Nastavili smo, da vsakič ko uporabnik vnese nek artikel v košarico, se proži omenjena funkcija, ki pregleda celotno tabelo carts za morebitne zastarele vnose in jih potem izbriše. S tem preprečimo odvečno obremenitev podatkovne baze.

Za košarico smo ustvarili dva različna pogleda. Eden je v obliki elementa, ki ga najdemo v desnem stolpcu strani in predstavlja pomanjšan ter hiter predogled vsebine košarice. Drugi pogled pa je celotna stran, ki omogoča pregled artiklov v košarici in proženje dodatnih funkcionalnosti (povečanje količine artikla, nadaljevanje z nakupom, zaključitev nakupa).

Slika 11: Dodajanje artikla v košarico

Oddaja naročila

Naslednja faza nakupa je stvaritev in zaključitev naročila po tem, ko je uporabnik v košarico vnesel želene produkte. To fazo smo razdelili na dva koraka. Prvi korak je vnos oziroma potrditev osebnih/dostavnih podatkov in tipa plačila, drugi je pregled končnega naročila in potrditev.

V prvem koraku uporabnika vprašamo ali bi se želel prijaviti v sistem ali pa izpolni vnosna polja za dostavne podatke. To smo storili zato, da lahko uporabnik kupi izdelek tudi če ni registriran. Ko to postori, ga pošljemo na naslednjo stran, kjer lahko preveri končno naročilo (dostavni podatki, artikli, poštnina, končna skupna vrednost plačila). V primeru da naročilo

potrdi, se to preko orders kontrolerja in order modela shrani v tabelo orders in vmesno tabelo orders_products.

```
//shranjevanje narocila
function saveOrder($orderData, $session_id ,$user_id = null){
    $time = new TimeHelper();
    //shrani Order v orders table
    $orderData['Order']['od_date'] = $time->format("Y-m-d H:i:s", time());
    $orderData['Order']['od_payment_tax'] = 0.00;
    $this->save($orderData);

    //pridobimo ID pravkar vnesenega narocila, za nadaljno manipulacijo
    $order_id = $this->getInsertID();

    //pridobivanje IDjev produktov iz vozicka
    if(!empty($user_id)){
        $result = $this->Product->Cart->getCartContent($session_id, $user_id);
    }else{
        $result = $this->Product->Cart->getCartContent($session_id);
    }

    //shranjevanje IDjev produktov in narocene kolicine
    $product_ids = array();
    $orderQty = array();
    $i = 0;
    foreach ($result as $item){
        $product_ids[$i] = $item['Product']['id'];
        $orderQty[$i] = $item['Cart']['ct_qty'];
        $i++;
    }
    //tu shranimo podatke v vmesno tabelo (hack)
    $this->addAssoc('Product', $product_ids, $order_id, $orderQty);


    //updatanje zaloge v tabeli products
    $products = array();
    foreach($result as $item){
        array_push($products, $item['Product']);
        $value = $item['Product']['pd_qty'] - $item['Cart']['ct_qty'];
        $this->Product->id = $item['Cart']['product_id'];
        $this->Product->saveField('pd_qty', $value);
    }

    //brisanje vsebine kosarice
    foreach($result as $item){
        $this->Product->Cart->emptyCart($item['Cart']['id']);
    }

    $order = $this->findById($order_id);

    return $order;
```

Slika 12: Orders_controller in funkcija shranjevanja naročila saveOrder

Ker je uporabnik nakup zaključil, izpraznimo vsebino košarice in zmanjšamo zalogo artiklov v bazi. Poleg teh funkcionalnosti smo posebej za našo aplikacijo razvili tudi preprosto komponento MyEmail za pošiljanje emailov, ki razširja že vgrajeno komponento EmailComponent. Dodali smo svojo šablono za pošiljanje glede na to katera akcija se proži. Tako recimo ob zaključenem nakupu preko kontrolerja kličemo komponento MyEmail in preko nje pošljemo email s podatki o naročilu. Enako storimo, ko administrator spremeni stanje naročila (recimo iz »new« na »shipped«), da obvestimo uporabnika o napredku procesiranja njegovega naročila.

```
}else{
    $this->Session->setFlash('Order placed!');
    //nastavi kupljene produkte za prikaz v e-mailu
    $this->set('order',$order);
    //poslji email za oddano narocilo uporabniku
    $this->MyEmail->sendOrderReceivedEmail($this->data['Order']['od_payment_email']);
}
```

Slika 13: Pošiljanje emaila kupcu znotraj kontrolerja

Registracija in vpis v sistem

CakePHP nam nudi izjemo pomoč pri operacijah kot so vpisovanje (login) in izpisovanje iz sistema, hkrati pa zagotavlja potrebno varnost. Na začetku razvijanja aplikacije smo imeli podatke nezavarovane in odprte, saj nam je to omogočalo transparentnost in olajšalo testiranje. Ko smo začeli razvijati sistem za registracijo in vpis je bilo nujno, da aplikacijo zapremo. Določiti smo morali taksonomijo in kakšen doseg bodo imeli določeni uporabniki. Stalna praksa je, da aplikacijo prvo zapremo in potem odpiramo posamezne funkcionalnosti glede na tip uporabnika. V CakePHP ogrodju je za zapiranje aplikacij zadolžena komponenta Auth. Nudi nam funkcije s katerimi lahko popolnoma avtomatiziramo avtentikacijo uporabnikov in zaklepamo oziroma odklepamo posamezne segmente aplikacije.

Preden smo se lotili kodiranja vpisa, izpisa in registracije uporabnika, je bilo prvo potrebno inicializirati Auth komponento.

```
var $components = array('Session','RequestHandler','MyEmail','Auth');
```

Slika 14: Inicializacija komponent v app_controller razredu

To smo storili v razredu AppController, ki je nadrazred vsem ostalim kontrolerjem. Vse funkcije in razredne spremenljivke iz AppControllerja so na voljo vsem kontrolerjem, saj od njega to podedujejo. Najlažje je zato zapreti celotno aplikacijo iz AppControllerja in posamezne funkcionalnosti aplikacije odklepati pri posameznem kontrolerju (podrazred). Auth komponento je bilo še pred tem potrebno prekonfigurirati. To smo storili z inicializacijo njenih atributov.

```
function beforeFilter(){

    /*
     * Nastavitve Auth componente
     */
    $this->Auth->authorize = 'controller';
    $this->Auth->authError = 'You have to be logged in to access this page!';

    //omogoca klic akcij, ki jih zahtevajo elementi, drugace se aplikacija obesi na redirect loop >.<
    if (isset($this->params['requested'])) {
        $this->Auth->allow($this->action);
    }

    //nastavitev login parametrov za komponento
    $this->Auth->fields = array(
        'username' => 'email',
        'password' => 'password'
        );
    //$this->Auth->autoRedirect = false;
    $this->Auth->loginAction = array('controller' => 'users', 'action' => 'login');
```

Slika 15: Nastavitve komponente Auth v AppControllerju

Funckija beforeFilter() nam zagotovi da se bo koda znotraj nje izvedla pred proženjem akcij v naših kontrolerjih. Torej vsakič ko uporabnik zahteva neko akcijo oziroma naslov, se bo sprožil beforeFilter. To nam zagotavlja ohranitev nastavitev Auth komponente čez celotno aplikacijo. Tako jo lahko nadzorujemo izključno preko enega razreda.

V naslednji fazi naredimo funkcijo login in logout. Zaradi pomoči ogrodja nam ni potrebno pisati logike prijave v sistem, razen če želimo izvajati še dodatne operacije ob prijavi uporabnika (hramba piškotkov, spreminjanje posameznih polj v bazi ob prijavi, itd.). Primerjava vnešenih podatkov in podatkov v bazi se preverijo avtomatsko, ko se funkcija kliče. Vse kar moramo postoriti je inicializirati funkcijo. Pri izpisu pa je pametno klicati sejno funkcijo delete za izbris vseh podatkov o uporabniku.

```
function logout(){
    $this->Session->delete('Auth.User');
    $this->Auth->logout();
    $this->redirect(array('controller' => 'carts', 'action' => 'index'));
}
```

Slika 16: Logout funkcija za izpis uporabnika

Pri registraciji smo morali biti malenkost bolj pazljivi, saj pri tem ogrodje ne pomaga avtomatično. Prvo smo naredili pogled v katerem se nahaja obrazec za registracijo, nato smo ustvarili funkcijo register() v users_controllerju. Ker kreiramo novega uporabnika je nujno, da preverimo podatke, ki jih je uporabnik vnesel, zaradi možnih napak. Jedro validacije podatkov se nahaja v modelu. Tukaj definiramo validacijska pravila, ki se preglejujejo za vsaki vnos uporabnika. CakePHP nam omogoča, da vežemo posamezna pravila na vnosna polja obrazcev. Lahko celo vežemo več pravil na en vnos.

```
var $validate = array(
    'email' => array(
        'notempty' =>   array(
            'rule' => 'notEmpty',
            //'required' => true,
            'message' => 'this field cannot be left blank!',
            'last' => true,
            ),

        'email' => array(
            'rule' => array('email'),
            'message' => 'your email is not valid!',
            //'required' => true,
            ),
        'unique' => array(
            'rule' => 'isUnique',
            'message' => 'This email is already taken!'),


    ),
    'password' => array(
        'between' => array(
            'rule' => array('between',2,10),
            'message' => 'Password must be between 2 and 10 chars long'
```

Slika 17: Validacijska pravila v modelu User

Ker se v našem primeru uporabnik prijavlja v sistem z emailom in geslom, smo validirali email in password polja v obrazcu. Na voljo imamo kupico pravil, tukaj smo jih uporabili le peščico. NotEmpty preverja če je vnos prazen. Email pravilo preverja ali je email pravilno napisan. Zadnje pravilo, ki smo ga vezali na polje email je isUnique, ki preverja unikatnost vnosa. Če vnos že obstaja v naši bazi, bo proces registracije ustavljen in na pogled se bo izpisalo definirano sporočilo.

5.2 Razlike med tradicionalnim razvojem in razvojem s pomočjo ogrodja CakePHP

Dokazali smo, da je razvoj spletnih rešitev s pomočjo ogrodij nekoliko drugačen kot tisti na konvencionalen način. Če se lotevamo razvoja zahtevnih in obširnih spletnih aplikacij je zelo pomembno pretehtati naše opcije, saj je v igri veliko denarja in časa. Če vzamemo za primer skriptirni jezik PHP, ga v času pisanja te diplome veliko podjetij uporablja kot primarni jezik za razvijanje njihovih spletnih aplikacij. Večina teh podjetij se razvijanja loteva na konvencionalen način brez pomoči ogrodja, saj jim to nudi popolno kontrolo nad kodo in posledično končnim izdelkom ali pa preprosto ne potrebujejo ogrodja, ker gre za nek specifičen končni izdelek za katerega ne potrebujemo dodatne pomoči ogrodja (predstavitvene strani, spletne storitve).

Pri vseh načinih razvijanja spletnih aplikacij največ časa porabimo za načrtovanje in pripravo. Pri tradicionalnem programiranju si moramo ustvariti neko približno sliko razredov in povezave med njimi, glede na funkcionalnosti spletne aplikacije. Na takšen način določimo arhitekturo, ki nam pomaga pri nadaljnem razvijanju. Sama arhitektura je odvisna od tega kakšno spletno aplikacijo razvijamo. Če se ne zanašamo na pomoč ogrodja, je v začetni fazi razvoja težko vzpostaviti idealno arhitekturo, saj nam še ni popolnoma jasno kako bo izgledal končni produkt in kako obsežen bo zaključen projekt. Ogrodje pa nam ponavadi vzpostavi začetne temelje in od nas je pričakovano grajenje na teh temeljih. Čeprav to zmanjša čas razvoja, nas hkrati do neke mere omejuje. V kolikor smo se odločili razvoja s pomočjo ogrodja, smo omejeni izključno na njihovo strukturo in arhitekturo. Odstopanje od tega ponavadi pomeni rušenje integritete ogrodja. Še več, ogrodja prenehajo ponujati svoje bližnjice in funkcionalnosti. Če pri CakePHPju ne uporabljamo funkcije find() za črpanje podatkov iz baze ampak prilagojeno funkcijo query() s svojo SQL sintakso, nam ogrodje ne bo več nudilo pomoči pri sanaciji vnosov in preventivi SQL vsiljevanja.

Pri razvoju spletnih aplikacij je čistost in urejenost kode odvisna od posameznega razvijalca. Ogrodje tukaj do neke mere pomaga s šablonskim pristopom, medtem ko je pri tradicionalnem razvoju to odvisno izključno od razvijalca. Funkcionalno gledano, urejenost kode nima nekega neposrednega vpliva na aplikacijo. Ima pa vpliv na sam proces snovanja in organizacijo. V kolikor imamo na projektu več ljudi, je pomembno da je sintaksa aplikacije berljiva in razumljiva za vse udeležence projekta, zato da lahko nadzorujejo pretok snovanja aplikacije. CakePHP je zelo dobro strukturirano in urejeno ogrodje. Pri snovanju spletne trgovine z njegovo pomočjo nismo imeli problemov pri organizaciji datotek in sintakse v njih.

Zaradi postavljene datotečne strukture je preprosto opredeliti katera skripta bo kam spadala. Pri kodiranju je stvar podobna. Zaradi konvencije CakePHPja convention over configuration, je stil kodiranja določen in bi odstopanje od tega izničilo pomoči ogrodja. Pri tradicionalnem razvijanju si moramo datotečno strukturo postaviti sami, kakor se nam zdi smiselno. Kodiramo pa v stilu, ki je nam najbljižje.

Ko fazo načrtovanja zaključimo, je na vrsti implementacija. V obeh primerih imamo vzpostavljeno datotečno strukturo in arhitekturo aplikacije. Pri konvencionalnem razvijanju na začetku sicer nimamo pomagal in bližnjic pri implementaciji funkcionalnosti za razliko od ogrodij, ampak jih lahko dodamo preko zunanjih knjižnic. Ugotovili smo, da CakePHP v našem primeru deluje preveč napihnjeno in invazivno, saj enostavno vsebuje preveč pomagal, ki niso nujna za našo aplikacijo in tako nepotrebno zmanjšuje odzivnost in veča velikost celotne aplikacije. Teh problemov pri standardnem razvoju aplikacij nimamo. Vključujemo samo tiste knjižnice, ki jih zares potrebujemo. Res da vključevanje tujih knjižnic predstavlja dodatno delo, saj moramo pregledati njihov nabor funkcij in ali so primerne za našo aplikacijo, ampak lahko na tak način omejimo število datotek in nepotrebnih funkcij, ki zavirajo našo aplikacijo.

CakePHP ima za določene funkcije jedrnih komponent v svojem APIju nepopolne definicije in je težava razbrati njihov namen in delovanje dokler jih ne preizkusimo v praksi. Pri razvijanju naše spletne trgovine smo velikokrat obšli ponujeno pomoč CakePHPja prav zaradi tega, še posebej pri ustvarjanju HTML elementov, saj nam ogrodje avtomatsko vsili svoj stil oblikovanja vmesnika (prekomerna uporaba div elementov).

Zadnja fazo razvoja predstavlja testiranje spletne aplikacije. Glede na izkušnje, ki smo jih pridobili pri razvijanju spletne trgovine, smo ugotovili, da je proces testiranja s pomočjo ogrodja enostavno preprostejši. CakePHP nam pomaga pri izpisovanju aktivnosti podatkovne baze v povezavi s spletno aplikacijo. Opazujemo lahko vse transakcije baze preko spletnega vmesnika. Če nam je to odveč (ko smo končali s testiranjem), lahko to opcijo po volji izključimo v konfiguracijski datoteki. Če ne uporabljamo ogrodja, moramo takšno funkcionalnost ustvariti ročno ali pa uporabiti temu namenjeno knjižnico. CakePHP ima od verzije 1.2 privzeto vgrajeno tudi ogrodje za testiranje spletnih aplikacij imenovano SimpleTest, ki ga lahko med drugim uporabljamo tudi brez ogrodja CakePHP. Omogoča nam izolirano testiranje posameznih delov spletne aplikacije (poglede, kontrolerje in modele).

Izpostaviti moramo tudi čas učenja ogrodja. Tisti, ki želijo uporabljajo ogrodja za hiter razvoj spletnih aplikacij, se morajo dodatno izobraziti. Učenje ogrodja je v večini primerov zahteven

proces in pogoj za obvladovanje le-tega je tudi poznavanje jezika na katerem temelji. Po drugi strani pa je čas razvoja spletnih aplikacij, kot smo ugotovili iz praktičnega dela diplome, občutno skrajšan, ko gre za večje primere. Celoten proces, od načrtovanja do praktične implementacije, je razpolovljen, zaradi narave pomoči ogrodja.

6 SKLEP

V diplomski nalogi smo podrobno predstavili skriptirni jezik PHP, ki je podlaga CakePHPju. Predstavili smo njegovo zgodovino in pojasnili delovanje. Pojasnili smo delovanje strežniških orodij in zakaj jih uporabljamo. Razjasnili smo arhitekturo spletnih aplikacij. Na posamezne elemente smo razčlenili arhitekturni vzorec MVC in vsakega posebej predstavili.

Prikazali smo obsežno ogrodje CakePHP, njegove elementarne dele, strukturo in arhitekturo. Pojasnili smo kako se lotiti razvijanja spletnih aplikacij z njegovo pomočjo. Primerjali smo proces razvoja s CakePHP in tradicionalnim razvojem spletnih aplikacij.

V praktičnem delu diplome smo razvili spletno trgovino podobno tem na svetovnem spletu (mimovrste.com, enaa.com), ki odraža poglavite funkcionalnosti ogrodja CakePHP. Spletno aplikacijo smo razvili v skladu s CakePHP pravili (poimenovanje, arhitektura). Videz aplikacije smo implementirali s pomočjo pomočnikov kot so HtmlHelper, FormHelper, TimeHelper in JsHelper.

CakePHP je izjemno zmogljivo ogrodje za hiter razvoj spletnih aplikacij. Vedno več razvijalcev se odloča razvijati spletne aplikacije z njegovo pomočjo, saj je takšen pristop hitrejši, preprostejši in produktivnejši. Ugotovili smo, da je s pomočjo pomagal (komponente, pomočniki in utility razredi), ki nam jih nudi CakePHP, čas oblikovanja spletnega vmesnika in aplikacijske logike občutno skrajšan. Moramo pa razumeti, da če želimo izkoristiti vse funkcionalnosti CakePHPja, se moramo obvezno držati njegove konvencije. MVC arhitektura, ki jo nudi ogrodje je zelo učinkovita. Pomaga nam pri ločevanju vizualne, aplikacijske in podatkovne logike ter nas usmerja v strukturirano programiranje.

Odkrili smo tudi nekaj pomanjkljivosti. Ugotovili smo, da je API ogrodja pri pojasnjevanju delovanja nekaterih funkcij komponent in pomočnikov nekoliko pomanjkljiv, kar je problem za vse, ki se želijo naučiti razvijanja spletnih aplikacij s tem ogrodjem. Podrobno delovanje razumemo šele, ko funkcije preizkusimo v praksi, oziroma če pregledamo jedrne datoteke v ogrodju, kjer so funkcije definirane. Pri snovanju zapletenih SQL poizvedb smo spoznali, da funkcije v modelu, zadolžene za interakcijo z bazo, velikokrat ne dosežejo želenega učinka (shranjevanje v vmesno tabelo) in smo tako bili prisiljeni v poseganje v jedrni model. Tudi pri

posodabljanju posameznih atributov v tabelah je prišlo do težav in smo morali uporabiti prilagojen SQL stavek in ne temu namenjene vgrajene funkcije updateField(). Interakcija med modeli in podatkovno bazo bi lahko bila bolj izpopolnjena. Recimo funkcija find(), ki je zadolžena za črpanje izbranih podatkov iz baze nam vrne tekstovno polje. Boljša rešitev bi bila če bi lahko pridobili polje objektov in bi tako dostopali do vseh podatkov preko njih. Predvsem sintaksa bi lahko bila čistejša, če bi se CakePHP manj zanašal na polja, saj ko uporabnik pošlje veliko količino podatkov iz vmesnika, so ti na voljo v kontrolerju v obliki multidimenzionalnega polja, ki je zelo težko berljivo.

Pomoč, ki jo nudi pri snovanju spletnega vmesnika je zelo dobrodošla, predvsem ko izdelujemo obrazce, saj jih lahko ustvarimo v par vrsticah kode, ampak smo prišli do spoznanja, da v določenih primerih pomočniki ogrodja ne izpolnijo naših pričakovanj (pomanjkljiva podpora HTML oznakam, podpora večim obrazcem na eni strani). To seveda ni kritična napaka, ampak definitivno prispeva k zaviranju celotnega procesa razvijanja.

Na spletu je trenutno veliko različnih PHP ogrodij. Njihovi trendi popularnosti se nekako prepletajo. Težko bi z gotovostjo trdili, katero ogrodje je najboljše oziroma najpopularnejše, saj imajo vsako za sebe edinstvene karakteristike in podpornike na različnih koncih sveta. Najpomembnejše je, da razvoj le-teh napreduje, saj nam bodo na ta način nudili možnost razvijanja kakovostnih in konkurenčnih spletnih aplikacij, brez napora obvladovanja nizkonivojskih procesov.

V primerih ko želimo razviti večje spletne aplikacije po našem okusu, kot so spletni portali, spletni forumi, spletne trgovine ali avkcije, se CakePHP izkaže kot idealno ogrodje. Če pa razvijamo preprostejše spletne aplikacije ali zelo specifične aplikacije pri katerem pomoč ogrodja ne pride do izraza, se je tega bolje lotiti brez njegove pomoči.

7 VIRI IN LITERATURA

[1] A. Valums, Web apps vs desktop apps, http://valums.com/web-apps/, nazadnje obiskano: 3.9.2011.

[2] M. Bryant, 20 years ago today, the World Wide Web opened to the public, http://thenextweb.com/insider/2011/08/06/20-years-ago-today-the-world-wide-web-opened-to-the-public/, nazadnje obiskano: 3.9.2011.

[3] History of web application, http://www.roseindia.net/servlets/HistoryOfWebApplication.shtml, nazadnje obiskano: 3.9.2011.

[4] R. Morin, Script languages, http://www.mactech.com/articles/mactech/Vol.15/15.09/ScriptingLanguages/index.html, nazadnje obiskano: 3.9.2011.

[5] The unbeatable benefits of PHP, http://www.itsabacus.com/blog/2011/01/the-unbeatable-benefits-of-php/, nazadnje obiskano: 3.9.2011.

[6] History of PHP, http://php.net/manual/en/history.php.php, nazadnje obiskano: 3.9.2011.

[7] J. Coggeshall, An introduction to PHP, http://onlamp.com/pub/a/php/2001/02/22/php_foundations.html, nazadnje obiskano: 3.9.2011.

[8] I. Alshanetsky, PHP's safe_mode or how not to implement security, http://ilia.ws/archives/18_PHPs_safe_mode_or_how_not_to_implement_security.html , nazadnje obiskano: 3.9.2011.

[9] M. Baker,What is a Software Framework And why should you like 'em, http://info.cimetrix.com/blog/bid/22339/What-is-a-Software-Framework-And-why-should-you-like-em, nazadnje obiskano: 3.9.2011.

[10] B. Moon, PHP frameworks, http://brian.moonspot.net/php-frameworks, Nazadnje obiskano: 3.9.2011.

[11] G. Booch,The architecture of Web appliacations, http://www.ibm.com/developerworks/ibm/library/it-booch_web/, nazadnje obiskano: 3.9.2011.

[12] Understanding Model-View-Controller, http://book.cakephp.org/view/890/Understanding-Model-View-Controller, nazadnje obiskano: 3.9.2011.

[13] Convention over configuration, http://en.wikipedia.org/wiki/Convention_over_Configuration, nazadnje obiskano: 3.9.2011.

[14] Q. Zervaas , Practical web 2.0 applications with PHP, Apress, 2008.

[15] D. Golding, Beginning CakePHP: From Novice to Professional, Apress, 2008.

[16] Scripting vs. programming: is there a difference?, http://www.killersites.com/blog/2005/scripting-vs-programming-is-there-a-difference/, nazadnje obiskano: 7.9.2011.

[17] Installation and configuration, http://www.php.net/manual/en/install.php, nazadnje obiskano: 7.9.2011.

[18] Market share, http://www.mysql.com/why-mysql/marketshare/, nazadnje obiskano: 7.9.2011.

[19] HOW TO: Choose a PHP Framework for Your Next Project, http://mashable.com/2011/06/23/best-php-frameworks/#17495Yii, nazadnje obiskano: 8.9.2011.

[20] PHP frameworks, Part 1: Getting started with three popular frameworks. http://www.ibm.com/developerworks/opensource/library/os-php-fwk1/, nazadnje obiskano: 8.9.2011.

[21] Programs vs. markup or why HTML authoring is not programming, http://www.cs.tut.fi/~jkorpela/prog.html, nazadnje obiskano: 9.9.2011.

Univerza v Mariboru
Fakulteta za elektrotehniko, računalništvo in informatiko

IZJAVA O ISTOVETNOSTI TISKANE IN ELEKTRONSKE VERZIJE DIPLOMSKEGA DELA IN OBJAVI OSEBNIH PODATKOV DIPLOMANTOV

Ime in priimek diplomanta-tke: Jernej Brunec

Vpisna številka: E1002757

Študijski program: Informatika in tehnologije komuniciranja

Naslov diplomskega dela: Razvoj spletnih rešitev s pomočjo ogrodja CakePHP

Mentor: Doc. dr. Marko Hölbl

Somentor: _____

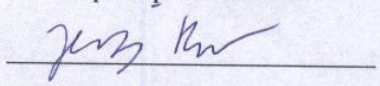
Podpisani-a Jernej Brunec izjavljam, da sem za potrebe arhiviranja oddal elektronsko verzijo zakljucnega dela v Digitalno knjižnico Univerze v Mariboru.
Diplomsko delo sem izdelal-a sam-a ob pomoci mentorja. V skladu s 1. odstavkom 21. clena Zakona o avtorskih in sorodnih pravicah (Ur. l. RS, št. 16/2007) dovoljujem, da se zgoraj navedeno zakljucno delo objavi na portalu Digitalne knjižnice Univerze v Mariboru.

Tiskana verzija diplomskega dela je istovetna elektronski verziji, ki sem jo oddal za objavo v Digitalno knjižnico Univerze v Mariboru.

Podpisani izjavljam, da dovoljujem objavo osebnih podatkov vezanih na zakljucek študija (ime, priimek, leto in kraj rojstva, datum diplomiranja, naslov diplomskega dela) na spletnih straneh in v publikacijah UM.

Datum in kraj:
Maribor, 5. 9. 2011

Podpis diplomanta-tke:



Univerza v Mariboru

Fakulteta za elektrotehniko, računalništvo in informatiko

IZJAVA O USTREZNOSTI DIPLOMSKEGA DELA

Podpisani mentor Marko Hölbl (ime in priimek mentorja) izjavljam, da je
študent Jernej Brunec (ime in priimek študenta-tke) izdelal diplomsko

delo z naslovom: Razvoj spletnih rešitev s pomočjo ogrodja CakePHP

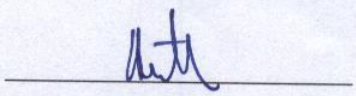
(naslov diplomskega dela)

v skladu z odobreno temo diplomskega dela, Navodili o pripravi diplomskega dela in mojimi navodili.

Datum in kraj:

5.9.2011

Podpis mentorja:



Univerza v Mariboru
Fakulteta za elektrotehniko, racunalništvo in informatiko

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani/-a Jernej Brunec,

z vpisno številko E1002757,

sem avtor/-ica diplomskega dela z naslovom:

Razvoj spletnih rešitev s pomočjo ogrodja CakePHP

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal/-a samostojno pod mentorstvom (naziv, ime in priimek) Doc. dr. Marko Hölbl

  in somentorstvom (naziv, ime in priimek)

- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela

- soglašam z javno objavo elektronske oblike diplomskega dela v DKUM.

V Mariboru, dne 5.9.2011

Podpis avtorja/-ice: