UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Gregor Beslič

Razvoj spletnih aplikacij z integracijo WordPress in Zend Framework

DIPLOMSKO DELO NA UNIVERZITETNEM ŠTUDIJU

Ljubljana, 2011

UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Gregor Beslič

Developing web applications by integrating WordPress and Zend Framework

DIPLOMSKO DELO NA UNIVERZITETNEM ŠTUDIJU

Mentor: izr. prof. dr. Marko Bajec

Ljubljana, 2011

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01745/2011
Datum: 01.04.2011

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **GREGOR BESLIČ**

Naslov: **RAZVOJ SPLETNIH APLIKACIJ Z INTEGRACIJO WORDPRESS IN ZEND FRAMEWORK**

DEVELOPING WEB APPLICATIONS BY INTEGRATING WORDPRESS AND ZEND FRAMEWORK

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Razvoj spletnih aplikacij se v marsičem razlikuje od klasičnega razvoja. Čeprav so na voljo številne razvojne platforme, programska ogrodja ter tudi postopki in metodologije, razvoj spletnih aplikacij še vedno ni dosegel takšne ravni stabilnosti, kot je bila dosežena pri klasičnem razvoju. V okviru diplomske naloge predstavite ogrodji WordPress in Zend Framework, ki lahko pripomoreta k učinkovitejšemu razvoju tovrstnih aplikacij. Na osnovi osebnih izkušenj z razvojem spletnih aplikacij predlagajte metodološke smernice za stroškovno in časovno učinkovit razvoj spletnih aplikacij.

Mentor:
prof. dr. Marko Bajec

Dekan:
prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko
1
UNIVERZA V LJUBLJANI

Zahvala

Hvala mentorju za vso pomoč,

hvala Maši za podporo

in hvala očetu za vzpodbudo.

Kazalo

Seznam uporabljenih kratic in simbolov

RAD – Rapid application development

AJAX – Asycnronus Javascript and XML

XML – Extensible markup language

BSD – Berkeley software distribution

CPL – Corporate public license

MPK – Model - Pogled Krmilnik

MVC – Model - View - Controller

URL – Uniform resource locator

UCIRA - University of California Institute for Research in Arts

WP – WordPress

ZF – Zend Framework

HTTP – Hypertext transfer protocol

EOM – Efektivna obrestna mera

SGE – System generated e-mail

LAMP – Linux Apache PHP MySql

FTP – File transfer protocol

CSS – Cascading style sheet

1 POVZETEK IN KLJUČNE BESEDE

Povzetek

V tem delu diplomskem delu bomo najprej predstavili ključne razlike javno dostopnih spletnih aplikacij napram internim (intranet) in eksternim (ekstranet) spletnim aplikacijam glede na različne parametre, kot so časovni roki za izvedbo, omejenost finančnih sredstev za razvoj, enostavnost uporabe, izgled, dostopnost s spletnimi iskalniki, hitrost delovanja, možnost zlorab in podobno.

Sledi opis metodologije razvoja, ki smo jo večkrat uspešno uporabili pri preteklih spletnih projektih in je primerna za izredno majhne (3-članske) razvojne ekipe. Opisana metodologija temelji na kombinaciji različnih metodologij RAD (angl. Rapid Application Development), pri čemer pripisujemo velik pomen prototipiranju zaslonskih mask. Razvoj spletne aplikacije predstavimo tudi kot del dolgoročne storitve, ki jo izvajalec nudi naročniku.

Na koncu bomo podali konkretne rešitve, ki smo jih uporabili v praksi ter predstavili dve razvojni ogrodji, WordPress (v kombinaciji z ogrodjem Carrington) in Zend Framework, katerih integracija nam je omogočila stroškovno in časovno učinkovitost pri izvedbi projektov. WordPress uporabimo kot sistem za urejanje vsebin spletne aplikacije (statičnih in dinamičnih vsebin strani), Zend Framework pa za izvedbo kompleksne poslovne logike. Na ta način zagotovimo, da največ časa posvetimo tistim delom aplikacije, ki so za naročnika najpomembnejši in se izognemo reševanju klasičnih problemov, za katere že obstajajo učinkovite rešitve.

Ključne besede

razvoj spletnih aplikacij, metodologija razvoja spletnih aplikacij, WordPress, Carrington, Zend Framework

Excerpt

We will first introduce key differences between the development of public web applications and the development of intranet and extranet web applications by comparing different parameters, such as time-constraints, budget limitations, ease of use (usability), design, search engine accessibility, page load times and security issues.

After the first chapter we explain the web development methodology which allows a team of only three people to successfully develop different web projects. The methodology is a combination of different RAD (Rapid application development) methodologies with a strong emphasis on GUI prototyping. We introduce the web application development as part of a long-term service the development company has to offer to the potential client.

In the last chapter we describe the technologies used, WordPress (combined with the Carrington framework) and Zend Framework. Integrating both frameworks allowed the development to be very quick and cost-effective. We use WordPress as a content management system (building static and dynamic content pages) and Zend Framework for the development of custom business logic. This allows us to spend most of the time developing crucial parts of the application rather than »reinventing the wheel« by developing solutions for common problems that have already been solved.

Keywords

web application development, web development methodology, WordPress, Carrington, Zend Framework

2 UVOD

Dandanes se vedno več gospodarskih subjektov zaveda ogromnega tržnega potenciala interneta in internetnih tehnologij ter jih s pridom izkorišča tudi v praksi. Seveda so del teh tehnologij uporabljali leta, morda celo več kot desetletje nazaj. Pri tem mislim na enostavne spletne strani, ki so služile za osnovno predstavitev dejavnosti podjetja ter uporabo elektronske pošte kot učinkovitega komunikacijskega kanala. Bolj “zavedni” oz. tehnologiji bolj naklonjeni gospodarski subjekti so svoje prihodke iz poslovanja investirali tudi v razvoj (spletnih) aplikacij in informacijskih storitev za učinkovitejšo podporo lastnih poslovnih procesov. V tem obdobju, torej dobro desetletje nazaj, je uporaba (vsaj nekaterih) internetnih tehnologij postala tako rekoč obvezen del poslovnega sveta.

Prvi val, ki ga lahko imenujemo tudi Splet 1.0 (angl. Web 1.0), je bil v mnogih pogledih revolucionaren, in ga je - tudi s stisnjenimi zobmi -, sprejela velika večina gospodarskih subjektov. Danes že težko najdemo organizacijo ali samostojnega podjetnika, ki ne uporablja elektronske pošte. Tudi tisti, ki se poslovne rabe interneta namenoma izogibajo v velikem loku, v sodobnem svetu nikakor niso imuni na njegov vpliv. Podatki o njihovem podjetju so javno dostopni v mnogih uradnih in neuradnih registrih gospodarskih subjektov (v Sloveniji npr. http://www.ajpes.si, http://www.infocity.si), spletni forumi, blogi in družbena omrežja pa polni komentarjev, kritik, mnenj in ocen njihovih izdelkov ali storitev. Velik del te vsebine zabeležijo in kategorizirajo spletni pajki (programi spletnih iskalnikov, kot je npr. GoogleBot), zato postane še lažje dostopna. Vse to seveda brez njihove privolitve. Tudi če gospodarski subjekti tehnologije ne sprejmejo, tehnologija brez težav sprejme njih. Predvsem zato, ker so tehnologijo sprejeli povprečni uporabniki, med katerimi so seveda tudi njihove (potencialne) stranke.

A ključni premiki so se začeli dogajati sredi prvega desetletja novega tisočletja, ko je uporaba internetnih tehnologij in storitev začela bistveno vplivati na življenja posameznikov in gospodarskih subjektov ter za mnoge postala nezamenljiv, če ne že nepogrešljiv del vsakdana. Ta premik mnogi imenujejo Splet 2.0 (angl. Web 2.0) in bi ga težko označili za revolucionarnega v tehničnem smislu, vendar je vsekakor treba

priznati, da je igral ključno vlogo pri splošnem dojemanju spleta ter hkrati usmerjal njegov (tehnični) razvoj. Ključni koncept Spleta 2.0 spreminja vlogo uporabnika spletne strani (aplikacije) iz pasivnega prejemnika informacij (opazovalca) v aktivnega (so)ustvarjalca spletne vsebine, ki s spletno aktivnostjo širi svoj socialni krog. Tovrstna spletna aktivnost vedno bolj vpliva tudi na njegovo realno življenje, naj si gre za izbiro prenočišč v sklopu daljšega potovanja (CouchSurfing), izbor glasbe v avtomobilu (SoundCloud), kuhanje večerje (Kulinarika.net) ali povečano športno aktivnost ob udeležbi spletnega tekmovanja (Garmin Izziv).

Potenciala tovrstne uporabe spleta se zaveda vedno večje število gospodarskih subjektov, katerih cilj je povečanje povpraševanje po njihovih izdelkih ali storitvah. Razvoj teh aplikacij je večinoma tržno naravnan. Pogosto se razvoj prične tako, da neko podjetje (naročnik aplikacije) dobi idejo za internetno storitev (aplikacijo), s katero bi povečalo svojo prepoznavnost, okrepilo blagovno znamko, pridobilo dodatne osebne podatke uporabnikov za učinkovitejše trženje, tržilo oglasni prostor ali spodbudilo potencialne stranke k nakupu svojih izdelkov ali storitev.

Problematika diplomskega dela bo specifičnost razvoja komercialnih spletnih aplikacij, katerih razvoj je pogosto podvržen mnogim omejitvam, med katerimi prednjačijo finančne omejitve naročnika in zelo kratki časovni roki za izdelavo. Opisali bomo uporabljena orodja, metode dela oz. metodologijo, ki smo jo dopolnjevali skladno s splošnimi priporočili dobre prakse za razvoj spletnih aplikacij. Uporabili bomo znanje iz razvoja informacijskih sistemov, ki smo ga pridobili med študijem, postavljenega v okvire finančnih, časovnih in drugih omejitev (posebnosti), s katerimi se srečujemo na spletu.

3 GLAVNI DEL

3.1 Specifičnost javnih spletnih aplikacij napram intranet in ekstranet aplikacijam

3.1.1 Kaj so spletne aplikacije?

Delovanje takih aplikacij deluje po principu odjemalec - strežnik. Del programske kode se izvede na strežniku (npr. PHP koda), del pa na odjemalcu (npr. Javascript koda). Odjemalec z uporabo HTTP protokola na strežnik pošlje zahtevek HTTP(S), strežnik zahtevek sprejme, ga obdela, in odjemalcu pošlje odgovor HTTP(S). Za dostop do spletnih aplikacij torej ne potrebujemo namestiti posebne programske opreme, ampak uporabljamo običajen spletni brskalnik, ker prenos podatkov poteka preko HTTP(S) protokola.

Na strežniški strani potrebujemo (vsaj) naslednje vire oz. resurse:
- strojno opremo (strežnik, omrežno opremo), ki je lahko nameščena znotraj organizacije ali v ustreznem datacentru (kolokacija, gostovanje, virtualni stroj v oblaku)
- operacijski sistem (tipično Linux/Unix ali Windows Server)
- programsko opremo za spletni strežnik (Apache, Nginx, IIS)
- programsko opremo za sistem za upravljanje s podatkovno bazo (MySQL, PostgreSQL, Oracle, MS Access)
- interpreter ali prevajalnik programske kode spletne aplikacije (PHP, Python, Ruby)

Pogosto se kot del operacijskega sistema pojavijo še storitve za elektronsko pošto (standard POSTFIX), do katerih spletna aplikacija dostopa preko vmesnikov, ki so del programskega jezika.

Na strani odjemalca potrebujemo, kot že rečeno, le spletni brskalnik. Spletni brskalnik je lahko nameščen v operacijskem sistemu klasične delovne postaje (PC-ja), prenosnega računalnika, tabličnega računalnika ali pametnega telefona.

Predstavil bom specifičnost razvoja komercialnih spletnih aplikacij napram ostalim, predvsem intranet in ekstranet aplikacijam (informacijskim sistemom). Osredotočil se bom na način razvoja spletnih aplikacij za naročnike, ki imajo zelo omejene finančne in kadrovske zmožnosti (vodenje projekta iz strani naročnika), časovni roki razvoja so zelo omejeni (največ 2 meseca), aplikacije pa relativno kompleksne za izdelavo - ne gre za tipične spletne strani, ampak za aplikacije s specifično poslovno logiko, torej za informacijske sisteme v malem. Poslovna logika se bo v idealnem primeru nadgrajevala in dopolnjevala večkrat na leto, skladno s potrebami uporabnikov naročnika in skladno z naročnikovimi poslovnimi cilji.

3.1.2 Spletne intranet aplikacije

V eni izmed preteklih študentskih zaposlitev (leta 2000) sem se srečal z intranet portalom, na katerem so bile zbrane vse nujno potrebne informacije, ki smo jih potrebovali za delo, poleg tega pa smo na portalu akumulirali znanje (v obliki vprašanj in odgovorov) ter s tem večali učinkovitost našega dela.
Leta 2004 sem za potrebe turistične agencije razvil spletni informacijski sistem za vodenje prijav na turistične aranžmaje in osnovno računovodstvo. Zaposleni in vodstvo so iz lokalnega okolja (lokalne mreže) v aplikacijo vnašali študente, dijake, prijave, aranžmaje, izdajali račune, jih tiskali, beležili plačila za prijave, tiskali položnice, itd.
Čeprav gre v osnovi za dve tehnično precej različni orodji, lahko vseeno najdemo skupni imenovalec, ki velja za veliko spletnih intranet aplikacij, ki tečejo v manjših do srednje velikih podjetjih.

DEFINIRANJE ZAHTEV IN NAČRTOVANJE

- Aplikacijo se razvije skladno z zelo dobro znanimi zahtevami (in potrebami), predviden način delovanja je natančno opisan. Cilji, katere naj bi organizacija dosegla z razvojem te aplikacije, so jasno določeni.

RAZVOJ

FINANCIRANJE

- Za financiranje razvoja take aplikacije se organizacija odloči, ko oceni, da bo z njenim delovanjem izboljšala svoje poslovanje, optimizirala svoje poslovne procese oz. dosegla jasno zastavljene (vnaprej določene) cilje. Financiranje razvoja načeloma ne predstavlja posebnega bremena za organizacijo (v primeru internega razvoja). Če gre za zunanji razvoj, se zanj praviloma odloči šele, ko ga lahko financira (sooča se s povečanjem obsega poslovanja, ki (deloma) pokriva stroške razvoja).

ČASOVNI ROKI

- Časovni roki praviloma niso zahtevni, aplikacijo se razvija skladno z zmožnostmi za to usposobljenega kadra (v primeru internega razvoja).

ENOSTAVNOST UPORABE

- Enostavnost uporabe (angl. “usability”) intranet aplikacije ne predstavlja pomembnega dejavnika pri razvoju, saj bo aplikacijo uporabljalo omejeno število ljudi, ki se lahko uporabe priučijo (v to so praktično prisiljeni), ob morebitnih težavah jih pomagajo sodelavci. Pogosto obstajajo tudi navodila za uporabo.

IZGLED

- Investicija v izgled (oblikovanje) bi praviloma pomenila nepotreben strošek in porabo časa. Izgled se zato ne smatra za pomembnega, estetski učinek take aplikacije je zanemarljiv.

HITROST DELOVANJA OZ. ODZIVNOST

- Do aplikacije dostopa omejeno število uporabnikov, ki je pogosto znano vnaprej. Zato se dovolj dobra odzivnost pogosto lahko doseže brez velikega vložka v programsko in strojno optimizacijo.

MOŽNOST ZLORAB

- Možnost zlorab je praviloma nizka, saj je določena s kontrolo dostopa (uporabniškimi pravicami) za posameznega uporabnika, sledljivost dostopov je enostavna.
- Praviloma uporabniki aplikacije (zaposleni zunaj IT oddelka) nimajo dovolj tehničnega znanja, da bi aplikacijo lahko zlorabili, niti za to nimajo nobenega razloga.

DOSTOPNOST S SPLETNIMI ISKALNIKI

- Dostopnost aplikacije s spletnimi iskalniki je pogosto nemogoča in seveda nezaželena, zato optimizacija za iskalnike ne igra prav nobene vloge.

UPORABA

VPELJAVA

- Za strojne zahteve (strežnik, omrežne naprave) praviloma poskrbi podjetje samo (IT oddelek), zato ni stroškov gostovanja.
- Za namestitev oz. vzpostavitev dostopa do aplikacije iz posamezne delovne postaje praviloma poskrbi nekdo od zaposlenih (v večjih organizacijah zaposleni v IT oddelku), ki uporabnika hkrati seznani z njeno uporabo.

VZDRŽEVANJE

- Za vzdrževanje že razvite aplikacije pogosto skrbi IT oddelek podjetja, ki se le občasno obrne na razvijalca po pomoč. Odpravljanje tipičnih napak oz. težav je praviloma hitro (s pomočjo IT oddelka), saj lahko uporabniki, ki aplikacijo pogosto uporabljajo, ostalim nudijo določen nivo podpore.

NADGRADNJE

- Nadgradnje so praviloma redke (občasne) in se večinoma izvajajo za povečanje nabora funkcionalnosti, ne pa izgleda oz. uporabnosti.

3.1.3 Spletne ekstranet aplikacije

Naslednja skupina aplikacij tipične organizacije so t.i. ekstranet aplikacije, ki so še vedno zaprtega tipa (niso povsem javno dostopne), vendar njihova uporaba ni omejena le na zaposlene, ampak tudi na poslovne partnerje, nekatere posameznike in organizacije (npr. stranke, dobavitelji, logistika, oglaševalske agencije).

Tipična spletna ekstranet aplikacija bi lahko služila razporejanju delovnih sredstev podjetja (npr. gradbene in transportne mehanizacije) po različnih lokacijah, upravljanje z urniki zaposlenih (upravljavcev te mehanizacije), in najem delovnih sredstev pri podizvajalcih, ko se za to pojavijo potrebe. Do aplikacije lahko dostopajo vsi zaposleni (upravljavci razporejajo resurse, zaposleni sproti preverjajo lokacije in urnike dela), vključno s podizvajalci, ki tako ostanejo na tekočem s potrebami podjetja po najemu njihovih delovnih sredstev.

Za spletne ekstranet aplikacije manjših podjetji v splošnem velja naslednje:

DEFINIRANJE ZAHTEV IN NAČRTOVANJE

- Aplikacijo se razvije skladno z dobro znanimi zahtevami, predviden način delovanja je precej natančno opisan, vseeno pa obstaja možnost, da se pri zajemu zahtev “pozabi” na določene funkcionalnosti. Tu mislim predvsem na tiste dele sistema, ki se razvijejo za potrebe poslovnih partnerjev, ki načeloma pri takem razvoju ne sodelujejo neposredno, zato svojih potreb ne izrazijo neposredno, ampak jih v njihovem imenu definira naročnik sam.

RAZVOJ

FINANCIRANJE

- Za financiranje razvoja take aplikacije se organizacija odloči, ko oceni, da bo z njenim delovanjem izboljšala svoje poslovanje, optimizirala svoje poslovne procese oz. dosegla jasno zastavljene (vnaprej določene) cilje. Financiranje razvoja načeloma ne predstavlja posebnega bremena za organizacijo (v primeru internega razvoja). Če gre za zunanji razvoj, se zanj praviloma odloči šele, ko ga lahko

financira (sooča se s povečanjem obsega poslovanja, ki (deloma) pokriva stroške razvoja).

ČASOVNI ROKI

- Časovni roki praviloma niso zahtevni, aplikacijo se razvija skladno z zmožnostmi za to usposobljenega kadra (v primeru internega razvoja) oz. zaposlenih, ki so nosilci projekta (zunanji razvoj zahteva projektnega vodjo iz strani naročnika, pogosto vključuje tudi vodstveni kader).

ENOSTAVNOST UPORABE

- Enostavnost uporabe take aplikacije ne predstavlja pomembnega dejavnika pri razvoju, saj bo aplikacijo uporabljalo omejeno število ljudi, ki se lahko uporabe priučijo (v to so praktično prisiljeni). Pogosto obstajajo tudi navodila za uporabo, ki se pošljejo poslovnih partnerjem preden pričnejo z uporabo. V primeru slabo izdelanega uporabniškega vmesnika je podjetje seveda primorano nuditi določen nivo podpore, kar mu predstavlja posreden strošek.

IZGLED

- Izgled take aplikacije ni preveč pomemben, ker je funkcionalnost na prvem mestu. Kljub temu se pogosto poskrbi za prepoznavnost podjetja (naročnika) z logotipom, obliko pisave in barvno shemo, da je podjetju ustrezna in prepoznavna za poslovne partnerje.

HITROST DELOVANJA OZ. ODZIVNOST

- Do aplikacije dostopa omejeno število uporabnikov, ki ni popolnoma znano vnaprej, vseeno pa lahko število uporabnikov vnaprej ocenimo. Odzivnost oz. hitrost delovanja aplikacije ne igra velike vloge, ker jo uporablja omejeno število uporabnikov (klientov), seveda ob predpostavki, da gre za manjše podjetje. Če imajo poslovni partnerji težave pri uporabi (zaradi dostopov iz počasnejšega zunanjega omrežja), se lahko omeji število hkratnih dostopov do aplikacije oz. določi časovne razpone za njeno uporabo. S podobnimi ukrepi lahko

začasno rešimo problem in brez velikega pritiska poskrbimo za optimizacijo programske kode aplikacije ali nadgradnjo strojne opreme, na kateri teče.

MOŽNOST ZLORAB

- Možnost zlorab je načeloma nizka, saj je določena s kontrolo dostopa (uporabniškimi pravicami) za posameznega uporabnika, sledljivost dostopov je enostavna znotraj organizacije in nekoliko težja pri zunanjih dostopih.
- Praviloma uporabniki aplikacije nimajo dovolj tehničnega znanja, da bi aplikacijo lahko zlorabili. Vseeno obstaja možnost, da si želi poslovni partner (zunanji dostop) pridobiti poslovne skrivnosti organizacije (zabeležene v sistemu, ki zanj niso dostopne) in za to “najame” nekoga z dovolj tehničnega znanja, da vdre v aplikacijo.

DOSTOPNOST S SPLETNIMI ISKALNIKI

- Dostopnost aplikacije s spletnimi iskalniki je pogosto nemogoča in seveda nezaželena, zato optimizacija za iskalnike ne igra prav nobene vloge. Poslovnim partnerjem, ki potrebujejo dostop, se dostopne podatke sporoči, ko jih potrebujejo.

UPORABA

VPELJAVA

- Za strojne zahteve (strežnik, omrežne naprave) praviloma poskrbi podjetje samo (IT oddelek), zato ni stroškov gostovanja.
- Za namestitev oz. vzpostavitev dostopa do aplikacije je potrebno pripraviti natančna navodila oz. poslovnega partnerja najprej seznaniti, kako lahko do nje dostopa (pogosto le preko e-pošte oz. telefona). Dostop mora biti torej enostaven in ne sme zahtevati posega na lokaciji partnerja.

VZDRŽEVANJE

- Za vzdrževanje že razvite aplikacije pogosto skrbi IT oddelek podjetja, ki se le občasno obrne na razvijalca po pomoč. Odpravljanje tipičnih napak oz. težav je praviloma hitro (s pomočjo IT oddelka), saj lahko uporabniki, ki aplikacijo pogosto uporabljajo, ostalim nudijo določen nivo podpore tako za zaposlene, kot za poslovne partnerje.

NADGRADNJE

- Nadgradnje so praviloma redke (občasne) in se večinoma izvajajo za povečanje nabora funkcionalnosti, ne pa izgleda. V kolikor imajo uporabniki (poslovni partnerji) velike težave pri uporabi (potrebujejo pomoč) in če nudenje pomoči podjetju predstavlja opazen strošek, se lahko investira tudi v izboljšanje uporabnosti.

3.1.4 Javne spletne aplikacije

Tretja skupina aplikacij organizacije so javno dostopne (internet oz. spletne) aplikacije. V grobem jih lahko razdelimo na dva dela:
- informacijski sistemi, ki zadovoljujejo potrebe kupcev, komitentov, državljanov in strank.
- Spletne aplikacije tržnega (komercialnega) namena, s katerimi podjetje poveča svojo prepoznavnost, prepoznavnost določene blagovne znamke, pridobiva osebne podatke uporabnikov, trži oglasni prostor, spodbuja potencialne stranke k nakupu svojih izdelkov in storitev, itd.

INFORMACIJSKI SISTEMI ZA VEČJE NAROČNIKE

V razvoj informacijskih sistemov praviloma investirajo večja podjetja (npr. veliki trgovci - spletne trgovine, banke - spletno upravljanje z računom, logistične službe - sledenje naročil in pošiljk, državne ustanove - eŠtudent, mobilni operaterji, ponudniki internetnega dostopa - SiOL servisne strani, itd.). Razvoj teh sistemov je časovno in finančno zelo zahteven - pogosto traja več kot 6 mesecev, lahko tudi več let. Stroški razvoja so ogromni in se merijo v 100.000 €, včasih tudi v milijonih €. Za razvoj teh

sistemov obstajajo preverjene in učinkovite metodologije (npr. EMRIS), ki zagotovijo razvoj robustnega in varnega informacijskega sistema glede na natančno definirane zahteve naročnika.

(KOMERCIALNE) SPLETNE APLIKACIJE

Za drugi sklop aplikacij - za komercialne spletne aplikacije - veljajo precej drugačna pravila. Za pisanje obsežne dokumentacije v času načrtovanja (UML) pogosto ni niti časa, niti denarja, ker so časovni roki zelo kratki. Glede na osebne izkušnje pri izdelavi spletne aplikacije za trženje marketinških idej (delo pri podjetju OpenAd d.o.o.) se pri razvoju takih aplikacij lahko soočamo tudi s pomanjkanjem ustreznega kadra na strani naročnika, ki bi razumel in potrjeval UML dokumentacijo.

DEFINIRANJE ZAHTEV IN NAČRTOVANJE

- Aplikacijo se razvije glede na pogosto nejasne oz. ohlapne zahteve, cilji, ki naj bi jih aplikacija dosegla, niso vedno jasno opredeljivi. Pogosto je potrebno zahteve dodatno opredeliti skupaj z naročnikom.

RAZVOJ

FINANCIRANJE

- Financiranje razvoja praviloma predstavlja večje breme za naročnika, saj posebnega prihodka od delovanja spletne aplikacije v začetnem obdobju ni za pričakovati. Zato je smotrno, da uporabimo obstoječe dobre rešitve za klasične probleme.

ČASOVNI ROKI

- Časovni roki so kratki, zato je potrebno poskrbeti, da določenih delov aplikacije ne razvijamo od začetka, ampak da za splošne probleme uporabimo obstoječe učinkovite rešitve. Kratkim časovnim rokom moramo prilagoditi tudi razvoj, kar na primer pomeni, da ob potrditvi

prototipov zaslonskih mask iz strani naročnika vzporedno poteka programiranje in oblikovanje.

ENOSTAVNOST UPORABE

- Enostavnost uporabe take aplikacije je pogosto na prvem mestu in je tesno povezano z oblikovanjem ali celo “pomanjkanjem oblikovanja” (npr. Google-ov iskalnik). Navodila za uporabo aplikacije ne obstajajo niti ne smemo pričakovati, da bodo potencialni uporabniki pripravljeni vložiti veliko truda v seznanjanje z njeno uporabo. Aplikacija mora biti za uporabo intuitivna (angl. “Don’t make me think!”), uporabniški vmesnik enostaven.

IZGLED

- Izgled spletne aplikacije je zelo pomemben in mora biti opravljen profesionalno, skladno z namenom uporabe. Slabo je, če izgled definira naročnik - naročnik priskrbi svojo barvno shemo in logotip, za oblikovanje pa poskrbi profesionalni spletni oblikovalec.

HITROST DELOVANJA OZ. ODZIVNOST

- Do aplikacije bo dostopalo neznano število uporabnikov, lahko ga sicer ocenimo vnaprej, vendar vedno obstaja možnost nenadnega povečanja obiska. Spletna aplikacija mora torej biti (vsaj do neke mere) pripravljena na nenadna povečanja (konice), kar dosežemo z optimizacijo za hitrost.

- Odzivnost oz. hitrost delovanja aplikacije pomembno vpliva na njeno uporabnost ter na t.i. Google PageRank. Najprej moramo poiskati programsko ali sistemsko rešitev, ki ne zahteva nadgradnje v strojno opremo. Kadar smo izkoristili vse ostale možnosti, je potrebno razmisliti o zamenjavi. V najslabšem primeru je potrebno aplikacijo prestaviti v okolje, ki ji bo zagotavljajo dovolj sistemskih resursov.

MOŽNOST ZLORAB

- Možnost zlorab vedno obstaja, zato je potrebno varnosti in zlorabam posvetiti posebno pozornost. Na spletu so npr. pogost pojav programske skripte, ki samodejno objavljajo nezaželene (angl. spam) komentarje oz. ustvarijo nov uporabniški račun povsod, kjer jim tega ne preprečimo.

- Na spletu vedno obstajajo uporabniki z dovolj tehničnega znanja, da zlorabijo aplikacijo. Ranljive točke spletne aplikacije je potrebno identificirati, specificirati možne načine zlorab in za njih poiskati primerne in učinkovite rešitve.

DOSTOPNOST ZA SPLETNE ISKALNIKE

- Dostopnost aplikacije je globalna, aplikacija je javno dostopna preko spletnega naslova (URL-ja). Spletni naslov mora biti zapomljiv, posebno pozornost je potrebno nameniti optimizaciji take aplikacije za iskalnike.

UPORABA

VPELJAVA

- Za strojne zahteve pogosto poskrbi podjetje, ki je razvilo rešitev oz. poišče / priporoči primernega ponudnika gostovanja. Gostovanje spletne aplikacije je praviloma plačljivo in predstavlja dodaten strošek za naročnika.
- Dostop do aplikacije je javen, preko spletnega brskalnika - dostop do aplikacije ne sme zahtevati posebnih navodil, ampak le poznavanje spletnega naslova (angl. URL). Vseeno je potrebno testno verzijo aplikacije, ki je praviloma tudi dosegljiva preko spleta, ustrezno zaščititi pred nepooblaščenimi dostopi.

VZDRŽEVANJE

- Za vzdrževanje že razvite aplikacije skrbi izvajalec skladno s potrebami, ki jih zazna naročnik pri svojih uporabnikih oz. po priporočilih izvajalca.

Vsaka spletna aplikacija se mora nadgrajevati, v nasprotnem primeru hitro zastara.

- Za odpravljanje napak poskrbi izvajalec spletne aplikacije. Običajno naročnik in izvajalec podpišeta vzdrževalno pogodbo, v kateri definirata odzivne roke za odpravljanje napak in trajanje garancije. Odpravljanje napak v garancijskem roku ne predstavlja dodatnega stroška za naročnika.

NADGRADNJE

- Nadgradnje so pogoste, zato mora biti aplikacija zasnovana tako, da je njena glavna poslovna logika izdelana skladno s priporočili dobre prakse.

3.2 Metodologija razvoja komercialne spletne aplikacije

3.2.1 Obstoječe metodologije razvoja spletnih aplikacij

Spletni sistemi so mešanica založništva in razvoja programske opreme, trženja in računalništva, interne komunikacije in zunanjih povezav ter mešanica umetnosti in tehnologije.[1]

Za razvoj spletnih aplikacij obstaja precejšnje število potencialno uporabnih metodologij, ki jih bom omenil v tem poglavju. Vseeno je glede na raziskavo “A survey of multimedia and web development techniques and methodology uses” (vir: http://ir.library.nuigalway.ie/xmlui/bitstream/handle/10379/270/A%20Survey%20of.pdf?sequence=1) razvidno, da večina podjetji uporablja svojo metodologijo, ki je zasnovana na ohlapni različici strukturnega razvoja informacijskih sistemov, pri čemer igrajo veliko vlogo prototipi. Podjetja večino obstoječih tehnologij razvoja programske opreme smatrajo za preobsežne in preveč nerodne. Vredno je omeniti tudi, da je praktična uporaba metodologije vedno tesno povezana z resursi, ki jih ima izvajalec na voljo. V primeru zelo majhne razvojne ekipe je potrebno še tako primerno metodologijo prilagoditi dejanskemu stanju v podjetju in pripravljenosti ekipe in naročnika, da se držita določenih pravil.

Metodologija, ki smo jo razvili v našem podjetju, sledi zgornji ugotovitvi in je prilagojena za razvojno ekipo, sestavljeno iz treh ljudi: vodje projekta, programerja in oblikovalca (zaslonskih mask). Je mešanica ohlapne različice strukturnega razvoja in rapidnega razvoja aplikacij - Rapid Application Development (RAD).

RAD je metodologija, ki promovira uporabo prototipov in (spletnih) programskih ogrodji za hiter razvoj programske opreme in s čimprejšnjo vključenostjo naročnika. V polje RAD metodologij spada kar nekaj metodologij, ki vsaka na svoj način predstavljajo najboljše prakse za razvoj programske opreme, ki je kvalitetno napisana, promovira ponovno uporabo obstoječe kode in hiter, efektiven razvoj.

Prva od teh je agilni razvoj. Osnovni principi agilnega razvoja, objavljeni na http://agilemanifesto.org/ so sledeči:

Posamezniki in sodelovanje napram procesom in orodjem
Delujoča programska oprema napram popolni dokumentaciji
Sodelovanje naročnika napram pogodbenim pogajanjem
Odzivanje na spremembe napram striktnemu sledenju načrta

Čeprav imajo besede na desni določeno vrednost, imajo večjo vrednost besede na levi.

Vseeno pa pri prenosu teorije v prakso utegnemo naleteti na kakšno težavo. Agilni razvoj močno promovira Test Driven Development in sodelovanje v parih [4]. Test driven development pomeni, da je vsak kos kode najprej podvržen testiranju (najprej napišemo test, nato funkcionalnost), kar zaradi izjemno kratkih časovnih rokov v našem primeru načeloma ni bilo izvedljivo. Podobno velja za programiranje v parih, ker je pri projektih sodeloval samo en programer.

Določene razvojne smernice smo si “sposodili” od Vitkega razvoja programske opreme (Lean Software development), ki tudi spada v področje RAD. Promovira princip, da se čim prej razvije manjši del funkcionalnosti, ki so seveda podvržene uporabi v praksi. Osnovni vidiki “Vitkega razvoja” zajemajo:
- odstranite vse, kar je odvečno
- spodbujajte učenje
- odločitve sprejemajte čim kasneje (ko boste imeli več informacij)
- rezultat imejte čim prej
- razvojna ekipa naj ima določen vpliv na potek projekta
- razvijajte z integriteto, kar vključuje sprotno refaktroriranje programske kode
- na sistem glejte, kot na celoto

Vseeno so tudi RAD metodologije napisane precej splošno in za projekte z daljšimi razvojnimi cikli, kot v našem primeru (kjer nekaj tedenski razvojni cikel pomeni že konec projekta). Naslednja slaba stvar obstoječih metodologij je, da veljajo v

splošnem za razvoj programske opreme in ne ponujajo konkretnih rešitev za razvoj spletnih aplikacij, kjer so pravila razvoja malce drugačna.

Kot je pokazala praksa, sta prototipiranje in razvoj zaslonskih mask ključna pri predstavitvi pravilnega razumevanja naročnikovih zahtev samemu naročniku. Zaslonske maske predstavljajo otipljive in vizualno učinkovite pol-izdelke, ki so narejeni hitro in poceni.

3.2.2 Definiranje zahtev

Cilji naročnika tovrstne spletne aplikacije so povečanje poslovnega uspeha iz naslova primarne dejavnosti, povečanje prepoznavnosti podjetja oz. blagovne znamke ali informacijska (spletna) podpora določenim poslovnim procesom (optimizacija poslovanja ali nadgradnja poslovnega modela). Finančne zmožnosti so pogosto zelo omejene, časovni roki za izdelavo izjemno kratki, pričakovanja pa velika.

NAROČNIKOVE IDEJE IN PRIČAKOVANJA

Naročniki ideje za funkcionalnosti spletne aplikacije praviloma dobijo sami, ob opravljanju svoje primarne dejavnosti, podobno aplikacijo lahko zaznajo pri konkurenci oz. do navdiha pridejo med brskanjem po spletu. Osnovno idejo, kaj naj bi aplikacija počela, torej imajo. Ker se večina naročnikov z razvojem (spletne) programske opreme “na ključ” srečuje prvič, imajo pogosto napačno predstavo o časovni in finančni zahtevnosti takih projektov. Omejitev, ki jih ima svetovni splet, se ne zavedajo, sočasno seveda tudi niso seznanjeni z vsemi možnostmi, ki jih ta ponuja. Večina se jih sicer zavedajo, da mora biti spletno mesto dobro “optimizirano za iskalnike”, vendar to smatrajo kot problem, ki je popolnoma rešljiv na tehničnem nivoju, o vsebinskih metodah optimizacije pa ne vedo ničesar.

Pogosto gre za projekte, za katere niti sam naročnik ni prepričan, ali se bodo “prijeli” med uporabniki, kakšno bo zanimanje zanje, kako hitro se bo pojavila konkurenčna spletna aplikacija (morda je celo že v razvoju?). Pričakovanja so seveda velika; upajo, da bo spletna aplikacija že nekaj dni po javni objavi postala (sama od sebe) zelo obiskana oz. priljubljena, njihov posel pa bo v nekaj tednih dobesedno zacvetel. Da

je to daleč od resnice oz. prej izjema, kot pravilo, verjetno ni potrebno posebej poudarjati.

OMEJITVE RAZVOJA

V splošnem velja, da gre za finančno majhne vložke in kratke časovne roke izvedbe. Še posebej, kadar so naročniki spletnih projektov manjša podjetja z omejenimi kadrovskimi in finančnimi resursi, ki si želijo povečati prepoznavnost svojih produktov in storitev. Vedno se mudi, da bi bili z neko storitvijo prvi na trgu oz. dohiteli ali celo prehiteli konkurenco. V svojem diplomskem delu se bom osredotočil na projekte, ki morajo biti dokončani (aplikacija mora biti javno objavljena in delujoča) v roku 4 - 6 tednov, strošek celotne storitve, ki seveda vključuje tudi razvoj spletne aplikacije, pa se giblje med 5000€ in 15.000€.

ZAJEM ZAHTEV, PRVI KORAKI NAČRTOVANJA IN PRIPRAVA PONUDBE

Naročnikove zahteve izvajalec lahko zajame na več načinov. Potrebno je seveda izbrati primernega ter se do določene mere prilagoditi naročnikovim pričakovanjem. V primeru, da je naročnik projekta večje podjetje ali organizacija, njihova interna pravila pogosto določajo in “vsiljujejo” določene način dela, ki se mu morajo prilagoditi vsi potencialni izvajalci v času pridobivanja ponudb.

Najbolj osnovne zahteve naročnik pogosto posreduje med telefonskim pogovorom s potencialnim izvajalcem, čemur lahko sledi tudi zapis v pisni obliki, npr. po e-pošti. Na podlagi teh najbolj osnovnih informacij izvajalec lahko oceni, ali je za izvedbo projekta sploh primeren kandidat (ima vsa potrebna znanja, razvojno strojno opremo in dovolj časa za izvedbo).

Ob pozitivnem odgovoru izvajalca sledi prvi sestanek, na katerem sta prisotni vsaj obe odgovorni osebi za izvedbo projekta (projektni vodji na strani izvajalca in naročnika), pogosto pa tudi del tehnične oz. strokovne ekipe (npr. predvideni vodja razvoja iz strani izvajalca in vodja IT oddelka iz strani naročnika). Pri manjših projektih, na katere se osredotočam v tem diplomskem delu, se večkrat zgodi, da je

naročnik projekta hkrati v vlogi lastnika podjetja, direktorja, projektnega vodje in brez ustreznega tehnično podkovanega kadra.

Na prvem sestanku naročnik podrobno predstavi svoje želje in ideje za projekt. Del funkcionalnosti, ki naj bi jih aplikacija podpirala, razdela in predstavi zelo podrobno, določen del pa ostane le na nivoju bolj ali manj definiranih idej. Neredko gre za dobre, potencialno odlične ideje, ki jih naročnik zaradi pomanjkanja tehnološkega znanja in nepoznavanja aktualnih trendov in smernic ni uspel natančneje umestiti v projekt.

Po predstavitvi popolnih zahtev naročnika izvajalec poda svoje strokovno mnenje na podlagi izkušenj, strokovne izobrazbe, priporočil dobre prakse, zakonodaje in aktualnih trendov, ki veljajo za splet kot medij oz. platformo. Svoje mnenje poskuša uskladiti z naročnikom in natančneje definirati popolne funkcionalnosti sistema (spletne aplikacije), pri čemer se ne ozira na morebitno povečanje kompleksnosti in s tem povezane časovne in finančne zahtevnosti projekta. Glede na poslovne cilje naročnika se določi celoten nabor funkcionalnosti spletne aplikacije, ki naj bi te cilje (sčasoma) dosegle.

Projekt se torej najprej definira v celoti, skladno s poslovnimi cilji naročnika, ne glede na (trenutne) omejitve, s katerimi se je potrebno spopasti v naslednjem koraku. V realnem gospodarskem (poslovnem) okolju smo vedno podvrženi množici omejitev pri dejanski izvedbi projekta, ne glede na našo vlogo (naročnik ali izvajalec). Bistvene omejitve seveda postavi naročnik. Tem omejitvam se mora izvajalec prilagoditi, vendar le tako, da določene omejitve definira tudi sam.

Omejitve, ki jih postavi naročnik, praviloma vključujejo:

- (prekratek) časovni rok za izvedbo projekta
- (prenizka) finančna sredstva, ki jih je pripravljen vložiti v projekt
- del funkcionalnosti, ki jih smatra za nujne že v prvi fazi razvoja (vključene v prvo verzijo aplikacije)

Omejitve, ki jih glede na naročnikove omejitve lahko postavi izvajalec, vključujejo:

- omejen nabor funkcionalnosti sistema, ki jih je mogoče izvesti v zahtevanem roku glede na predvidena finančna sredstva
- določitev tehničnih pogojev (razvojne platforme) za pravočasno in stroškovno učinkovito izvedbo
- določitev dinamike potrjevanja vmesnih korakov razvoja (prototip, izgled) in specificiranje nekaterih projektnih rokov, kot npr. pravočasna dostava vsebin in multimedijskega gradiva iz strani naročnika, določitev časovnega okna, potrebnega za uporabniško testiranje aplikacije

Izvajalec in naročnik zato skoraj vedno stopita v proces “pogajanja”. Ta proces se lahko prične že na prvem sestanku, kadar je naročnik pripravljen takoj razkriti svoje omejitve glede časovne izvedbe in financiranja, kar se ne zgodi preveč pogosto. Praviloma si naročnik izbere več potencialnih izvajalcev, od katerih pričakuje, da bodo sami oblikovali ponudbo brez poznavanja njegovih omejitev.

V tem primeru izvajalec predstavi ponudbo, v kateri poda finančno in časovno oceno za razvoj projekta v celoti, ki je za naročnika - ker so določene funkcionalnosti aplikacije popolnoma nove in obstoječe funkcionalnosti podrobneje definirane in realno stroškovno ocenjene -, pogosto nesprejemljiva. Zato je priporočljivo, da se razvoj spletne aplikacije razdeli na več faz, pri čemer prva faza vključuje vse tiste funkcionalnosti, za katere izvajalec oceni, da podpirajo optimalen nabor najpomembnejših poslovnih ciljev.

Uspešnemu dogovoru sledi priprava in podpis pogodbe, v kateri se natančno definira nabor osnovnih (prvotnih) funkcionalnosti spletne aplikacije, določi celoten nabor izvedenih storitev, definira časovnica, plačilna dinamika, garancijsko obdobje za razvito programsko opremo (brezplačno odpravljanje hroščev in ostalih težav, za katere je odgovoren izvajalec) in določila avtorskih pravic.

3.2.2 Načrtovanje spletne aplikacije

Načrtovanje spletne aplikacije, ki je omejena z zelo kratkimi časovnimi roki in

finančnimi sredstvi, predstavlja določen izziv. Kljub temu, da gre za bolj ali manj kompleksen spletni informacijski sistem, je odveč pričakovati, da se bomo lahko držali metodologij, ki pritičejo razvoju večjih informacijskih sistemov (npr. EMRIS). Razlogov za to je seveda več. Med bistvenimi sta seveda časovna in finančna omejitev projekta, poleg teh pa seveda tudi dejstvo, da naročniki projekta (posamezniki, zaposleni v manjših podjetjih) praviloma ne razumejo niti se niso pripravljeni naučiti razvojne dokumentacije, kot je npr. UML (angl. Unified Modelling Language). Zato se je za pravočasno in učinkovito načrtovanje spletnih aplikacij potrebno poslužiti primernih metod in kompromisov.

Osnovne vsebinske in funkcionalne zahteve

Glede na naročnikova pričakovanja lahko aplikacijo razdelimo na več “standardnih” sklopov oz. gradnikov, ki se, kot sčasoma opazimo, precej ponavljajo. Večina spletnih aplikacij tako zaobjema naslednje vsebinsko-funkcionalne sklope:

Statične strani

Pri statičnih straneh seveda ne mislimo na to, da je njihova vsebina, ko jo enkrat postavimo, popolnoma statična oz. časovno nespremenljiva, ampak na koncept, ki določa, da je prikazana natanko ena (zadnja) verzija vsebine. Priporočljivo je seveda, da sistem shranjuje tudi stare različice vsebine, kljub temu pa imajo uporabniki spletne aplikacije dostop le do zadnje verzije.
Tipični primeri statičnih strani so:

Podporne strani:

- Pogoji poslovanja
- O podjetju
- Kontakt oz. spletna vizitka
- Politika zasebnosti (ta je še posebej pomembna, saj opisuje, katere osebne podatke spletna aplikacija zbira, kako jih obdeluje in definira ostale namene njihove uporabe)

Vsebinske strani, ki podrobno opisujejo delovanje spletne aplikacije. Količina teh strani je seveda od aplikacije do aplikacije lahko različna.

Meta oz. navidezne strani, ki so lahko brez lastne vsebine, ker praviloma vsebino oz. določene funkcionalnosti le povezujejo med seboj. Tipičen primer take strani je npr. prva stran spletne aplikacije, ki pogosto služi kot agregat posameznih vsebinskih sklopov, npr. prikazuje obrazec za prijavo v aplikacijo, zadnjih N objav na blogu, polje za iskanje po vsebini in prijavo na e-novice.

Skupno vsem statičnim stranem je naslednje:

- Vsaka statična stran je dosegljiva preko enoličnega spletnega naslova (URL-ja), npr:
  - http://aplikacija.com (navidezna stran, ki služi kot domača stran)
  - http://aplikacija.com/o-nas/pogoji-poslovanja (podporna stran na 2. nivoju)
  - http://aplikacija.com/hitra-predstavitev/ (vsebinska stran na 1. nivoju)

- Po vsaki vsebinski statični strani (razen meta strani), je mogoče iskanje, njena vsebina je indeksirana
- Vsaki strani (tudi meta strani) je mogoče določiti meta informacije, ki služijo spletnim iskalnikom (ključne besede oz. meta-keywords in opis oz. meta-description)
- Vsaki strani je mogoče določiti pozicijo v hierarhičnem drevesu strani (angl. sitemap), pri čemer ima lahko vsaka stran natanko 1 nadstran (starša) in več podstrani (otrok). Seveda obstajajo tudi strani, ki nimajo staršev in otrok, npr. domača stran (ki je vedno prva stran na prvem nivoju hierarhične lestvice).
- Taksonomije (kategorizacije, značk, itd.) pri statičnih straneh praviloma ni
- Datum objave praviloma ni pomemben (pomembnejša je zadnja verzija vsebine)

Dinamične strani

Pod tem izrazom razumemo vsebinske sklope, pri katerih velja, da se vsebino dodaja pogosteje in določi taksonomijo, ki to vsebino med seboj povezuje. Tipičen primer dinamičnih vsebin so denimo novice ali zapisi spletnega dnevnika (bloga), večji vsebinski sklopi, kot npr. pomoč uporabnikom, ipd. Praviloma ima vsaka objava svojo predhodno objavo (razen seveda prve objave), datum objave in datum zadnje

spremembe, kategorijo in morebitne ostale označbe. Za spletno aplikacijo seveda ni nujno, da vsebuje tudi dinamične strani.

Skupno vsem dinamičnim stranem je naslednje:

- Vsaka dinamična stran je dosegljiva preko enoličnega spletnega naslova (URL-ja), npr:

  - http://aplikacija.com/novice/dosegli-smo-mejo-1000-uporabnikov/ (označba “novice” tu ne nakazuje na hierarhijo v drevesu strani, ampak na kategorijo dinamične vsebine)

- Po vsaki dinamični objavi je mogoče iskanje oz. je njena vsebina ustrezno indeksirana
- Vsaki objavi je mogoče določiti meta informacije (ključne besede in opis)
- Posamezna dinamična objava ni vključena v drevo strani (angl. sitemap)
- Za posamezen sklop dinamičnih vsebin obstaja arhiv (npr. arhiv vsebin z enako kategorijo, označbo)
- Vsaki objavi je mogoče določiti taksonomijo, v kar spada kategorizacija (spet s svojim hierarhičnim drevesom nad- in pod- kategorij) in značke (ključne besede). S taksonomijo organiziramo dinamično vsebino v podobne in sorodne vsebinske sklope.

Potrebno je poudariti, da npr. arhiv novic, dosegljiv na spletnem naslovu http://aplikacija.com/novice ne predstavlja dinamične vsebine kot take, ampak gre praviloma za statično meta stran, podobno kot osnovno stran, ki združuje (agregira) vsebino iz drugih vsebinskih sklopov.

Ponavljajoče oz. standardne vsebinske in funkcionalne zahteve

Pri večini spletnih aplikacij se določeni tipi dinamičnih vsebin ponavljajo, med katerimi so koledar dogodkov, novice, zapisi na blogu, povezave na zunanje spletne strani, pogosta vprašanja in odgovore, aktualna (nujna) sporočila in obvestila. Prav tako se ponavljajo določeni moduli oz, funkcionalnosti, ki ne sodijo med aplikativne strani (poslovno logiko), ampak so del osnovne funkcionalnosti spletne strani: iskanje

po vsebinah, optimizacija za iskalnike, fotogalerije, deljenje vsebin s prijatelji (Facebook, Twitter in ostala družbena omrežja), ipd. Te funkcionalnosti so iz naročnikovega vidika “pričakovane same od sebe” in ne smejo bistveno podražiti projekta oz. ne smejo predstavljati večje časovne zahtevnosti za implementacijo.

Aplikativne strani oz. strani s specifično poslovno logiko

Aplikativne strani so jedro spletne aplikacije. Tipične aplikativne strani predstavljajo npr. stran za prijavo, registracijo, spremembo gesla, urejanje in prikaz uporabniškega profila ter vse strani, ki so potrebne za implementacijo poslovne logike spletne aplikacije. Gre seveda za funkcionalno najbolj kompleksne sklope spletne aplikacije, ki jih bolj kot z drevesom strani definiramo s prototipi.

Drevo strani

Definiranju statičnih, dinamičnih in aplikativnih strani najprej sledi izgradnja drevesa strani (angl. sitemap). V drevesu strani določimo hierarhijo med posameznimi statičnimi vsebinami in definiramo povezave med vsebinskimi sklopi oz podrobneje določimo način navigacije po strani (meniji, glava in noga strani).
Najpomembnejše strani (na 1. nivoju) pogosto postanejo del glavne navigacije, ki se lahko pojavi v glavi (angl. header) strani oz. na robu strani (angl. sidebar). Tipično gre za navigacijo, ki je dostopna povsod po strani. V ta del navigacije pogosto spadajo tudi določene aplikativne strani, kot so prijava, registracija, ogled profila in podobno. V nogi strani se ponavadi pojavijo tipične podporne strani, kot so Politika zasebnosti, Pogoji poslovanja in kontaktni podatki.
Določitev navigacije, ki je enostavno razumljiva in intuitivna za uporabo, je del dobre prakse načrtovanja spletnih aplikacij, ki spada na področje uporabljivosti (angl. Usability).

Prototipiranje

Proces izdelave prototipov vseh tipičnih zaslonskih mask je verjetno najpomembnejši korak pri načrtovanju spletne aplikacije. Prototip namreč služi kot osnova za naslednja dva koraka, oblikovanje in razvoj. Izdelava prototipa nam omogoča, da stranki aplikacijo predstavimo na vizualen način, kot ne-oblikovane zaslonske maske.

Primera orodji, ki omogočata prototipiranje, sta Axure (http://www.axure.com/) in iPlotz (http://iplotz.com/). V podrobnejši razlagi orodji se ne bom spuščal, obe orodji pa omogočata, da na hiter in učinkovit način definiramo ključne dele spletne aplikacije v obliki zaslonskih mask, po katerih lahko uporabnik (prosto) navigira s pritiskanjem na povezave in gumbe. Tak način predstavitve delovanje spletne aplikacije je učinkovit, ker je odpravljanje napak na prototipih hitro, enostavno in poceni. Pri izdelavi prototipov sodelujeta oba, naročnik in izvajalec. Iz strani izvajalca ponavadi sodeluje celotna projektna ekipa (oblikovalec, programer in načrtovalec), saj lahko vsak od njih enostavno poda svoje pripombe. Naročnik na podlagi tako izdelanih prototipov ugotovi, ali so vključeni vsi vsebinski in funkcionalni sklopi, ki jih potrebuje in si “plastično” predstavlja delovanje aplikacije, še preden je izdelana.

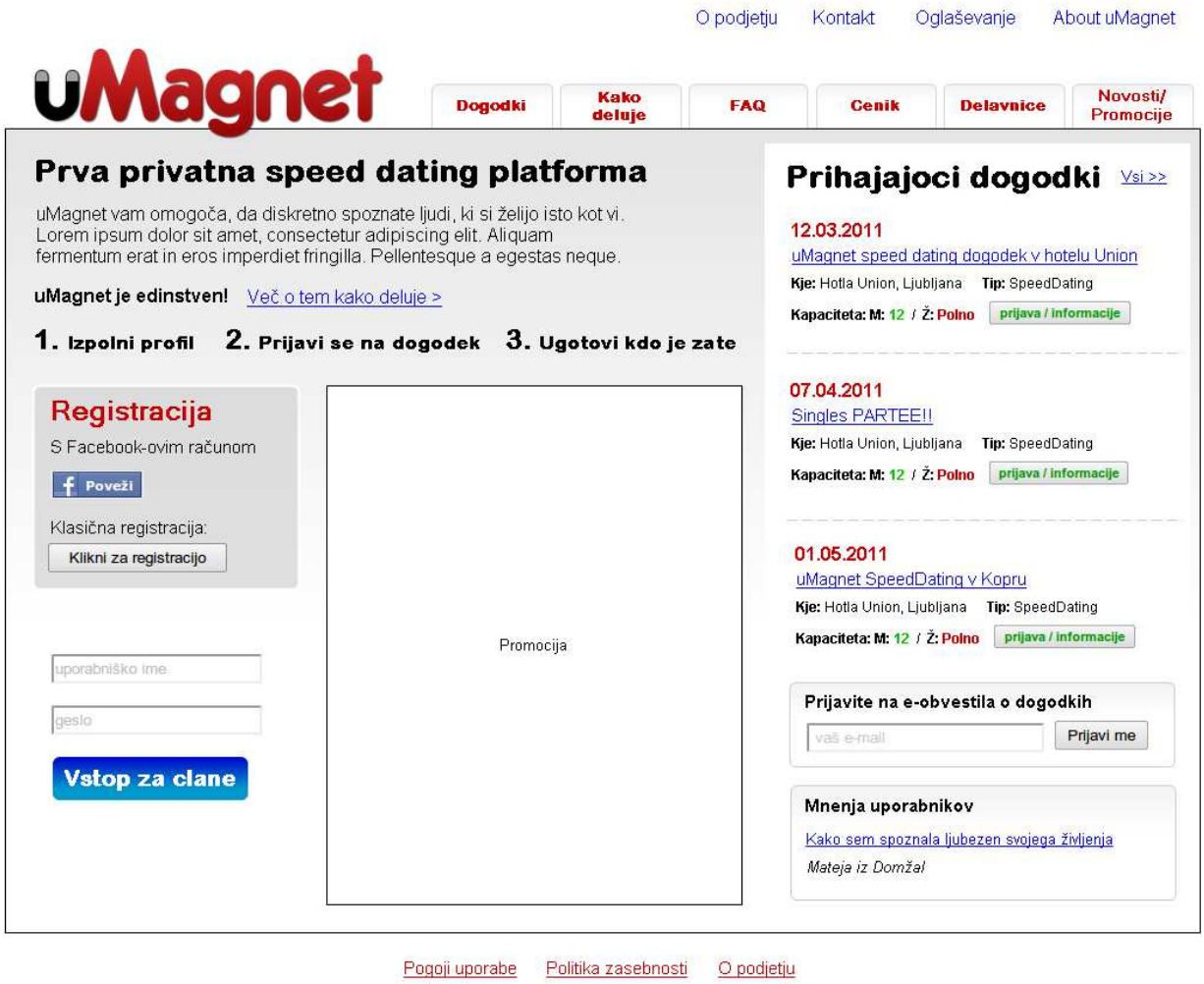
Večina orodji omogoča, da se zaslonske maske avtomatično izvozi na tak ali drugačen način, najbolj uporaben se je seveda enostaven format HTML, ki naročniku omogoča interakcijo s prototipom (klikanje na povezave in gumbe, ki izvedejo vnaprej definirane akcije).

Natančen razvoj prototipa je splošno koristen korak, saj izdelava dobrega prototipa glede na zajete zahteve ustvari pol-izdelek, ki služi kot osnova za nadaljnje korake razvoja. Izdelan prototip je zadnji korak procesa načrtovanja in zajema zahtev, ki lahko predstavlja neko zaključeno celoto. Na podlagi prototipa se določi končna cena oblikovanja vseh potrebnih (tipičnih) zaslonskih mask, časovni rok, ter končna cena za razvoj spletne aplikacije.

Ko ima naročnik izdelano drevo strani in prototip za spletno aplikacijo, se lahko odloči tudi za zamenjavo izvajalca, če je to potrebno. Pri večjih spletnih agencijah je to pogost način dela, saj na podlagi tako izdelanega prototipa lahko poiščejo zunanjega oblikovalca in pod-izvajalca (programerja), ki bosta skupaj izdelala končni izdelek. Hkrati prototip služi kot “varovalo” za izvajalca, da se lažje spopade z morebitnimi zahtevami naročnika, ki se pojavijo kasneje in ki niso bile zajete v prototipu.

Z razvojem prototipov in drevesa strani se zaključi faza načrtovanja. Seveda se postavlja vprašanje, ali oblikovanje, eden od naslednjih korakov v procesu, sodi v

sklop načrtovanja ali v sklop razvoja. Odločil sem se, da sodi v sklop razvoja zato, ker oblikovanje spletne aplikacije zaradi časovne omejenosti projekta poteka vzporedno z razvojem podatkovnega modela in programskih modulov (programiranjem).



Slika1: Primer prototipa

3.2.3 Razvoj

Ko imamo končan in potrjen prototip, se pogosto zgodi, da naslednje korake v razvoju izvajamo. Na podlagi prototipa se lahko prične s pripravo podatkovnega modela aplikacije in programiranjem poslovne logike (brez zaslonskih mask) ter

oblikovanjem, ki mu zaporedno sledi razvoj zaslonskih mask in dokončna integracija zaslonskih mask v spletno aplikacijo.

Oblikovanje

Oblikovanje spletne aplikacije predstavlja, čeprav se morda za izvajalca iz tehničnega stališča zdi trivialno, za naročnika vedno zelo pomemben korak. Podrobnosti o oblikovanju seveda presegajo problemsko področje diplomskega dela, vseeno pa velja omeniti nekaj bistvenih stvari.
Spletno aplikacijo se oblikuje glede na vnaprej znane funkcionalne zahteve (kaj bo aplikacija počela), drevo strani (kako bo aplikacija strukturirana - navigacija) in prototipe (kako bo aplikacija to počela) [5]. Prehod iz faze prototipa v končno obliko ne pomeni le apliciranja ustreznih grafičnih elementov (slik), tipografije in barvne sheme na obstoječ prototip, ampak pogosto zajema določene spremembe v pozicijah elementov. Prototip torej definira, kateri elementi se bodo prikazali na posamezni zaslonski maski, oblikovanje pa natančno določa pozicijo in obliko teh elementov.

Rezultat faze oblikovanja je praviloma vektorska datoteka tipa .ai (Adobe Illustrator) ali .pdf / .psd datoteka (Adobe Photoshop).
Pomembno je omeniti še, da se oblikovanje za splet močno razlikuje od klasičnega oblikovanja (npr. tiskovin), saj ima splet kot medij določene funkcionalne omejitve, uporabniki spletnih aplikacij pa določene navade in pričakovanja glede samih interakcij s spletno aplikacijo. Tipičen primer je omejitev širine in višine zaslona računalniškega monitorja uporabnika spletne aplikacije (npr. 1280 x 1024 točk).
Dobra praksa oblikovanja spletnih aplikacij in spletnih strani zapoveduje, da se mora spletna stran oblikovati tako, da večina uporabnikov (tipično 90% in več) ni prikrajšana za njeno uporabo. To pomeni:

- potrebno je določiti najmanjšo ločljivost zaslona, pri kateri so vse bistvene informacije dostopne na vidnem polju zaslona (angl. above the fold), kar pomeni, da uporabniku in potrebno premikati drsnikov spletnega brskalnika gor in dol
- pogosto je bila taka ločljivost 1024x768 točk, pri čemer je potrebno upoštevati še širino drsnika na različnih spletnih brskalnikih, zato je

dejanska omejitev širine spletne aplikacije, optimizirane za to ločljivost, približno 970 točk, kar je seveda za oblikovanje precej bistven podatek
- najmanjšo širino zaslona se določi glede na preteklo analitiko ali glede na najnovejše trende (pomik tipičnega zaslona iz 1024 x 768 na 1280 x 1024)

Za oblikovanje spletnih aplikacij lahko uporabljamo le omejen nabor pisav (tipografije), če želimo, da se stran na enak način prikaže v več različnih brskalnikih. V primeru, ko stran vključuje specifično obliko pisav (moderni brskalniki s podporo za CSS3 vključujejo direktivo @font-face, s katero lahko vključimo licencirane TrueType pisave oz. pisave Otf), moramo seveda poskrbeti, da imamo vse potrebne pravice za njihovo uporabo. Hkrati se vedno lahko zgodi, da uporabnik zaradi takšnih ali drugačnih omejitev ne bo videl izbrane oblike pisave, zato mora biti aplikacija funkcionalna tudi s sistemsko (privzeto) pisavo, ki jo določimo, če uporaba specifične pisave ni mogoča.
Oblikovanje spletnih aplikacij nujno vključuje tudi različne obrazce, tabelarično prikazane podatke in interakcije (z gumbi, z obrazci, drsniki, slikami, itd.). Za načrtovanje teh obstajajo priporočene prakse, ki jih morajo spletni oblikovalci seveda zelo dobro poznati, npr:

- lažjo berljivost podatkov v tabeli dosežemo, če parne vrstice obarvamo drugače, kot neparne
- izpolnjevanje spletnih obrazcev je praviloma podvrženo tudi nepredvidljivim situacijam oziroma napakam (npr. obvezna polja niso izpolnjena, napake pri validaciji), zato je potrebno tudi sporočila o napakah ustrezno oblikovati
- sporočila o napakah pri posameznih poljih (npr. pri validaciji podatkov) naj bi se izpisovala tik ob polju, in ne nad ali pod celotnim obrazcem

Oblikovanje za splet je zelo obsežno področje, ki presega vsebino tega diplomskega dela. Želel sem le poudariti, kako pomembno je, da spletno aplikacijo zaradi specifičnosti medija oblikuje izkušen spletni oblikovalec.

Slika 2: Primer oblikovane zaslonske maske na podlagi prototipa (Slika 1)

Razvoj zaslonskih mask

Pod razvojem zaslonskih mask razumemo transformacijo datoteke, ki je rezultat oblikovanja v sklop datotek, ki jih razume spletni brskalnik (HTML, CSS, JS in grafične datoteke). Razvoj zaslonskih mask postaja vedno bolj kompleksno področje, saj vedno novi standardi HTML-ja in CSS-ja povečujejo fleksibilnost in uveljavljajo metode dobre prakse.

Tipičen primer je postavitev elementov na strani s pomočjo tabele (pozicijo elementa na zaslonu določa celica v tabeli, ki se razteza čez cel zaslon) v primerjavi s sodobnim standardom spletnega oblikovanja, ki elemente veliko natančneje razporedi z uporabo enostavnih HTML značk ( ), katerim določimo ustrezne CSS razrede, npr: ( ).

Dobra praksa zapoveduje, da naj bodo HTML datoteke same po sebi brez neposrednih oblikovnih direktiv (angl. inline styling), vsebino naj se ločuje s klasičnimi HTML elementi (<h1> za naslov, <p> za odstavek, za vsebinski blok, za posebno dodatno označbo vsebine, itd.).

Prednosti uporab te metode je veliko, med najpomembnejšimi so:

- ločitev oblikovanja in vsebine, kar omogoča vzporednost razvoja spletne aplikacije (več o tem v nadaljevanju) in olajša vzdrževanje aplikacije v prihodnosti
- hitrejši čas nalaganja strani, ker spletni brskalniki (ob ustreznih nastavitvah spletne aplikacije) CSS datoteko shranjujejo lokalno (predpomnilnik)
- povečanje kompatibilnosti med brskalniki

Slednje je pri razvoju zaslonskih mask še kako pomembno. Vsak spletni brskalnik namreč uporablja točno določen algoritem prikazovanja (WebKit za Applov Safari in Google Chrome, Gecko za Firefox, itd.). Če je spletna aplikacija oblikovana glede na spletne standarde, npr. HTML5, XHML, HTML 4.1 transitional, je verjetnost, da se bo enako prikazala v vseh spletnih brskalnikih, veliko večja. Kljub temu obstaja črna ovca med sodobnimi spletnimi brskalniki (Internet Explorer 6), ki se spletnih standardov ne drži in že več let razvijalcem zaslonskih mask povzroča nemalo sivih las. Hkrati delež uporabnikov, ki ga uporabljajo, neustavljivo pada, zato bo podpora spletnemu brskalniku Microsoft IE 6 sčasoma postala stvar preteklosti.
Končni rezultat razvoja zaslonskih mask so torej vsaj trije tipi datotek:

- grafične datoteke, ki smo jih izrezali iz oblikovne datoteke (.psd) in optimizirali za splet (npr. glede velikosti in transparence - uporaba .png datotek, kjer je to potrebno)
- datoteke HTML, ki definirajo vsebinske sklope (bloke) skupaj z vzorčno vsebino (“lorem ipsum”), ki je statično zapisana v datotekah
- datoteke CSS (oz. datoteka - glej poglavje o optimizaciji), ki definirajo oblikovne razrede stilskega lista

- če uporabljamo specifične oblike pisav (@font-face direktiva), tudi ustrezne (licenčne!) .ttf ali .otf datoteke

Kaj pa datoteke Javascript? Ali razvoj interakcij, ki jih omogoča Javascript z manipulacijo DOM (Document Object Model), spada v razvoj zaslonskih mask ali sodi v kasnejšo fazo (razvoj programskih modulov). Enostavnega odgovora ni oziroma je odgovor odvisen od izvedbene ekipe. Običajno ima oblikovalec, ki poskrbi tudi za razvoj zaslonskih mask, omejeno znanje Javascript-a, zato vsaj del "bremena" praviloma pade na programerja.
To vedno velja za tiste dele aplikacije, kjer se uporablja asinhron XML-HTTP zahtevek (angl. XML-HTTP request), ki je skupni imenovalec AJAX aplikacij. AJAX se praviloma uporablja takrat, ko odjemalčev del spletne aplikacije asinhrono komunicira z zaledjem (strežniškim delom). Klasičen primer uporabe AJAX-a je obveščanje uporabnika o zasedenosti trenutno vpisanega uporabniškega imena, še preden uporabnik pritisne na gumb za dokončanje registracije in seveda brez osveževanja celotne strani.
Vendar se Javascript lahko uporablja za tudi veliko bolj preproste zadeve, npr. za rotacijo ali povečevanje slik na spletni strani, oziroma za splošno upravljanje dokumentnega objektnega modela. Če ima razvijalec zaslonskih mask dovolj potrebnih znanj, lahko poskrbi za del Javascript funkcionalnosti tudi sam in dodatno olajša delo programerju.

Sledenje še posebej drži, ker so se pojavila kvalitetna in dobro dokumentirana Javascript programska ogrodja, kot so Prototype, MooTools, JQuery in Dojo. S tovrstnimi ogrodji postane razvoj interakcij veliko enostavnejši, ker:

- poenostavi način pisanja Javascript funkcij in upravljanja z dokumentnim objektnim modelom tako, da preko programskega vmesnika nudi določen nivo abstrakcije in programsko knjižnico funkcij za najbolj pogoste naloge
- končna Javascript koda, ki jo razume spletni brskalnik, upošteva vse specifike različnih interpretacij, ki jih uporabljajo spletni brskalniki (napisana koda naj bi delovala enako na vseh brskalnikih)
- specifične Javascript kode je manj, zato je lažje razumljiva in berljiva (enake naloge lahko namesto z dvesto vrsticami kode rešimo z dvema)

- veliko funkcionalnosti lahko rešimo z uporabo vtičnikov, zato včasih razvoj sploh ni potreben, ne vzame dodatnega časa in ne predstavlja dodatnega stroška za naročnika

Glede izbire najbolj ustreznega Javascript programskega ogrodja se morata programer in razvijalec zaslonskih mask predhodno uskladiti. Tehnično je seveda mogoče uporabiti več ogrodji naenkrat (za prikaz iste zaslonske maske), vendar se s tem povečuje kompleksnost kode in obremenitev spletnega strežnika, saj mora namesto ene naložiti dve ali več Javascript datotek (knjižnic). Slednje seveda vpliva na čas nalaganja strani, ki mora biti čim krajši (več o tem v razdelku o optimizaciji).

Razvoj sistema (programiranje)

Oblikovanje in Razvoj zaslonskih mask sta koraka, na katera glede časovne in finančne zahtevnosti nimamo prav velikega vpliva. Ostajata pretežno enaka za vse spletne projekte, tudi za običajne spletne strani. Časovna in finančna zahtevnost obeh korakov je relativno enostavno določljiva; največji vpliv ima izbira oblikovalca (oblikovalskega studia oz. agencije) in število (pod)strani spletnega projekta (različnih tipičnih zaslonskih mask), ki jih je potrebno oblikovati in razviti.

Nasprotno pa izbira načina razvoja, v kar sodi tudi izbira programskih ogrodji, bistveno vpliva na celotno časovno in finančno zahtevnost spletnega projekta, pa naj si gre za enostavno spletno stran ali za kompleksno spletno aplikacijo.

Izbira načina razvoja komercialnih spletnih aplikacij je vedno podvržena določenim omejitvam in kot sem že omenil, se bom v tem diplomskem delu osredotočil na način razvoja komercialnih spletnih aplikacij, ki so podvržene naslednjim omejitvam:

- dokončane morajo biti v izjemno kratkih časovnih rokih (4-6 tednov), vključno z oblikovanjem in razvojem zaslonskih mask
- finančna sredstva so zelo omejena (tipično ne presežejo 10.000€, v kar je seveda všteto oblikovanje in razvoj zaslonskih mask)
- tako razvit sistem mora biti enostaven za vzdrževanje in primeren za obsežnejše nadgradnje, če se za to pojavi potreba

Seveda to niso edine zahteve, predstavljajo le najbolj osnovne omejitve, postavljene s strani naročnika. Izkušen izvajalec se seveda zaveda, da mora biti sistem hkrati:

- enostaven za uporabo (dobra uporabniška izkušnja za obiskovalce)
- enostaven za upravljanje (uporaba administrativnega vmesnika za upravljanje vsebine in specifičnih procesov)
- dobro optimiziran za iskalnike
- čas nalaganja posameznih strani mora biti čim krajši
- poskrbljeno mora biti za spletno analitiko (spremljanje obiska, uporabnikov, spremljanje načinov uporabe aplikacije in odkrivanje morebitnih težav pri uporabi)
- ustrezati mora določenim zakonskim zahtevam (npr. glede zbiranja, hrambe in uporabe osebnih podatkov)
- onemogočiti je potrebno nepravilno uporabo sistema in o tem ustrezno obvestiti uporabnika (validacija podatkov tako na strani odjemalca, kot na strani strežnika)
- poskrbljeno mora biti za varnost pred morebitnimi vdori in ostalimi zlorabami (več o tem v poglavju o varnosti in zlorabah)
- naročniku je potrebno nuditi podporo pri vseh procesih in nalogah, ki zaradi določenega razloga (še) niso programsko podprte (izvedljive z uporabo administrativnega vmesnika)

Poleg naštetega pa je v domeni izvajalca praviloma tudi:

- konfiguracija podpornih storitev projekta, kot so registracija domen, domenskih zapisov, e-poštnih naslovov, sistema za pošiljanje množičnih e-sporočil, FTP dostopov, ipd.
- avtomatično spremljanje obremenjenosti, ki jo sistem povzroča na strežniški infrastrukturi in odkrivanje morebitnih napak v delovanju ter ozkih grl (v skrajnem primeru selitev v zmogljivejše strežniško okolje - namenski strežnik ali virtualna infrastruktura v oblaku)
- skrb za varnost pred izgubo podatkov v primeru programske ali strojne napake (in posledično povrnitev sistema v delujoče stanje)

- avtomatično spremljanje pravilnega delovanja sistema in ukrepanje ob izrednih situacijah (npr. s pomočjo zunanje storitve, kot je Wormly monitoring: www.wormly.com)
- navodila za uporabo sistema

	VODJA PROJEKTA	PROGRAMER	OBLIKOVALEC
1	zajem zahtev	natančnejše definiranje zahtev	
2	načrtovanje	priprava sistemskega okolja, konfiguracija domene	izdelava prototipa
3	potrjevanje prototipa	priprava programskega okolja, namestitev programskih ogrodji in konfiguracija	popravki prototipa
4		priprava osnovne strukture spletne aplikacije, instalacija vtičnikov	oblikovanje
5	potrjevanje oblikovanja	razvoj programskih modulov	popravki oblikovanja
6		razvoj programskih modulov	razvoj zaslonskih mask
7		razvoj programskih modulov	implementacija zaslonskih mask
8		razvoj interakcij in funkcionalnosti uporabniškega vmesnika (AJAX)	implementacija zaslonskih mask
9	vodenje testiranja	odpravljanje programerskih napak	odpravljanje oblikovnih napak
10	pomoč pri vnosu vsebin, potrditev in plačilo	podporne storitve, priprava produkcijskega okolja in objava aplikacije	priprava navodil za uporabo

Tabela 1: Primer razvojnega cikla

Praktični vidiki razvoja in ustvarjanje dodane vrednosti

Sistem, ki ga bo izvajalec razvil s podanimi omejitvami in zahtevami, se bo v primeru (komercialne) uspešnosti razvijal naprej, nadgrajeval in spreminjal svojo prvotno (oblikovno) podobo. Število uporabnikov bo naraščalo, pojavljale in odpravljale se bodo nove, še neodkrite napake. Robustnost in kvaliteta programske kode se bo

sčasoma povečevala. Dodajale se bodo funkcionalnosti, da podprejo nove poslovne procese, ki so se na začetku morda zdeli nepomembni, a so se kasneje izkazali za koristne.
Sočasno se bo izvajalec lotil drugih, novejših projektov. V njegovem poslovnem interesu je, da postopoma povečuje svojo učinkovitost in zmanjšuje obseg dela, ki ga potrebuje za razvoj (podobnih) komercialnih spletnih aplikacij, hkrati pa se kvaliteta razvitih rešitev povečuje.

To lahko doseže tako, da predvidene funkcionalnosti novega projekta razdeli na dva glavna sklopa:

1. Sklop funkcionalnosti bo rešil s prilagoditvami sistema za upravljanje z vsebinami (najosnovnejše), zato, da bo projekt razvit hitreje in ceneje
    1. del funkcionalnosti je podprt že z jedrom sistema (osnovno verzijo) oz. z ustreznimi prilagoditvami osnovnih funkcionalnosti
    2. del funkcionalnosti je podprt z vtičniki, s katerimi lahko nadgradimo sistem
    3. za določen del funkcionalnosti bi bilo najbolj smotrno, da razvijemo nove vtičnike (manjše funkcionalnosti)

2. Sklop funkcionalnosti bo razvil z razvojem rešitve na ključ - ta del funkcionalnosti predstavlja jedro projekta -, zato mora biti rešitev razvita tako, da omogoča enostavno vzdrževanje in nadgrajevanje. Tudi ta sklop lahko izvajalec dodatno razdeli:
    1. učinkovite rešitve je že razvil v preteklosti (uporabil bo svoje obstoječe rešitve)
    2. učinkovite rešitve so deloma že razvite (uporabil bo del kode iz obstoječe rešitve, ki jo bo hkrati optimiziral - sproti bo opravil pregled kode)
    3. učinkovite rešitve še niso razvite
        1. zahtevana funkcionalnost je lahko koristna tudi za druge projekte (funkcionalnost bo zasnoval generično in za razvoj porabil več časa)

2. zahtevana funkcionalnost je uporabna le za konkreten projekt (funkcionalnost bo zasnoval specifično in za razvoj porabil manj časa)

Spodaj so podani primeri funkcionalnosti izmišljene aplikacije, ki ustrezajo tej razdelitvi. Naročnik aplikacije je podjetje, ki povezuje investitorje s podjetniki (podjetniškimi idejami). Na spletni strani (vsebinskem delu spletne aplikacije) so objavljene novice, pretekli projekti in prihodnji dogodki (srečanja). Aplikativni del (jedro spletne aplikacije) omogoča registracijo podjetnikov, ki na predpisan način pošiljajo predloge svojih podjetniških idej. Tako poslane ideje ocenjujejo investitorji (uporabniki s posebnim statusom) in najperspektivnejše podjetnike povabijo na uradne predstavitve njihovih projektov. Najboljše med njimi financirajo oz. zagotovijo semenski kapital za njihovo izvedbo.

1. Vsebinski del spletne aplikacije:
    1. Na podstrani “Predstavitve” so prikazane vse objave tipa “Predstavitve”, ki jih lahko dodatno filtriramo glede na lokacijo in mesec v letu
    2. Vsebinski del mora biti optimiziran za iskalnike, za kar obstaja nabor kvalitetnih vtičnikov, ki dobro rešujejo ta problem - uporabimo enega ob obstoječih
    3. Vsakič, ko objavimo novo vsebino s kategorijo “Investicija”, se mora po e-pošti obvestiti ustrezne osebe iz PR agencije, ki je zadolžena za kliping celotne mreže podjetji, katere del je tudi naročnik projekta
2. Spletna aplikacija registriranim uporabnikom omogoča pošiljanje idej za projekte, ideje nato ocenjuje interna ekipa in nagradi najboljše.
    1. Uporabili bomo že razvit modul “uporabniki”, ki omogoča registracijo, aktivacijo računa, prijavo, pozabljeno geslo, urejanje in ogled profila uporabnika
    2. Razvili bomo modul “admin”, ki omogoča ocenjevanje prijavljenih projektov. Nadgradili bomo obstoječ modul “admin”, ki trenutno omogoča zgolj pregled vseh prijavljenih uporabnikov
    3. Na novo bomo razvili

1. modul “ocene” za ocenjevanje prijavljenih del kot generičen modul za ocenjevanje abstraktnih entitet iz strani privilegiranih uporabnikov
2. modul “projekti”, ki bo uporabnikom prikazal specifične obrazce za vnos vsebine in datotek glede na vrsto projekta

Seveda se izvajalec ob prvem projektu, pri katerem bo uporabil opisano metodologijo razvoja, sooča le s sklopom 1 in s sklopom 2.c (razvoj na “ključ” se izvede brez možnosti uporabe obstoječih modulov). Zato je prvi v seriji takih projektov za izvajalca najmanj donosen, ker je časovno in tehnično najbolj zahteven. Vendar si bo izvajalec, ob doslednem upoštevanju podane metodologije v daljšem obdobju, ustvaril zbirko modulov in knjižnic (zbirko znanja oz. rešitev), s katerimi bo postopoma večal dodano vrednost svojega razvoja in izboljševal kvaliteto svojih rešitev. Vsak nov projekt bo zaradi ponovne uporabe obstoječih (preverjenih in optimiziranih) modulov kvalitetnejši, razvoj bo hitrejši in za izvajalca enostavnejši. Stremeti je potrebno k temu, da se z vsakim novim projektom rešuje le tiste probleme, za katere še ne obstajajo kvalitetne rešitve. Če nam podana metodologija omogoča, da svoj čas posvetimo predvsem tem problemom (in se ne ukvarjamo s tistimi, ki so že rešeni), bodo naše rešitve kvalitetnejše, projekt v celoti pa bo veliko hitreje razvit.

Konkurenčnost razvitih rešitev

Hitrejši in enostavnejši razvoj komercialnih spletnih aplikacij nam hkrati omogoča tudi večjo konkurenčnost. Povečuje razpon med ceno v naši ponudbi in našimi dejanskimi stroški dela - ker se stroški manjšajo, se naš razpon veča. Spodnja meja, pri kateri je izvedba projekta finančno še vedno smotrna, se niža. To pomembno vpliva na naslednje dejstvo: v poslovnem svetu je pogosto tako, da velika podjetja postavljajo pravila igre, ki so jih manjša podjetja primorana upoštevati, če želijo z njimi sodelovati. Oboji se namreč zavedajo, da izvedba projekta za manjše podjetje nikakor ne pomeni le plačila (za izvedbo), temveč služi predvsem kot referenca, ki lahko bistveno vpliva na njegovo nadaljnjo uspešnost (pridobivanje novih projektov, ugled, izpostavljenost v medijih, itd.). Zato so velika podjetja v položaju, ko lahko postavijo prenizko (nerealno) ceno izvedbe, ker se zavedajo, da je prava vrednost pri sodelovanju z njimi drugje. Izkoriščajo svoj položaj in se zavedajo, da bo izbrano podjetje pripravljeno delati z projekt z relativno izgubo glede na vloženo delo. Z

uporabo podane metodologije bo naša izguba veliko manjša, kljub temu pa bo aplikacija razvita pravočasno in, kar je še pomembnejše, kvalitetno. Verjetnost, da bo naročnik z našo izvedbo zadovoljen, je zato veliko večja. Zadovoljstvo velikega naročnika pa v poslovnem svetu prinese veliko prednosti. Veliki naročniki so primorani za ohranitev svojega (vodilnega) položaja na trgu izvajati več (večjih) projektov. Njihova izpostavljenost je večja, zato slabe poslovne odločitve nosijo veliko večje posledice, kot pri manjših podjetjih. Podobno velja za uspešne projekte - pozitivna izkušnje iz preteklosti (uspešna izvedba projekta) bistveno vpliva na izbiro izvajalca naslednjega projekta. Zato bomo kot potencialni izvajalec naslednjega projekta lahko postavili nekoliko višjo (realnejšo) ceno izvedbe napram ostalim kandidatom in kljub temu ohranili konkurenčnost naše ponudbe - velik naročnik je skoraj vedno pripravljen plačati nekoliko višjo ceno za izvedbo, ki je preverjeno kvalitetna. Kar obenem pomeni, da mora biti naša ponudba bistveno nižja, če se z nami sreča prvič, obenem pa se za izvedbo projekta poteguje tudi izvajalec, s katerim ima pozitivne izkušnje iz preteklosti.

Arhitektura Model Pogled Krmilnik (MPK)

Arhitektura Model Pogled Krmilnik ali po angleško Model View Controller (MVC) loči poslovno logiko aplikacije od predstavitve in hrambe podatkov. Poslovno logiko aplikacije praviloma upravlja krmilnik (če se zgodi to, potem naredi to).

Model služi za upravljanje s podatki (skrbi za hrambo podatkov v podatkovnem skladišču oz. podatkovni bazi) in zagotavlja določeno abstrakcijo dostopa do podatkov (npr. med krmilnikom in podatkovnim skladiščem). Večkrat je veliko poslovne logike, ki je povezana s podatki, tudi v Modelu, saj je pogosto smiselno, da so Krmilniki čim enostavnejši.

Pogled služi predstavitvi podatkov v zaslonskih maskah. Krmilnik torej pridobi ustrezne podatke od Modela, na podlagi podatkov sprejme “odločitve” (poslovna logika) in sproži prikaz ustreznega Pogleda (zaslonske maske), kateremu posreduje tudi podatke, če jih potrebuje.

Arhitektura MPK omogoča lažje nadgrajevanje aplikacije v prihodnosti, zato je

smiselno, da je jedro aplikacije razvito s programskim ogrodjem, ki MPK arhitekturo omogoča oz. vzpodbuja. Če nam programsko ogrodje MPK dopušča modularnost razvitih rešitev, lahko že razvite MPK rešitve (module) uporabimo večkrat in vsakič znova dodatno optimiziramo. S tem se izognemo ponovnemu razvoju rešitev, ki smo jih že razvili in delujejo dobro.

Varnost komercialnih spletnih aplikacij in možnosti zlorab

S problemom istovetnosti se na spletu pogosto srečujemo. Preprečevanje morebitnih zlorab sistema je kompleksen proces, ki ga je potrebno rešiti na več nivojih. To seveda še najbolj velja za sisteme, ki omogočajo registracijo uporabnikov in hranijo bolj ali manj občutljive osebne podatke. Opisanih je nekaj tipičnih možnih zlorab, za katere bom v nadaljevanju (Poglavje 3.3 - Razvoj v praksi) opisal tudi rešitve, ki jih ponujata obe programski ogrodji.

Lastnik podatkov seveda želi zagotoviti, da so vpisani podatki pravilni in ne zgolj rezultat spletnih robotov oz. skript, ki poskusijo zlorabiti sistem, npr. v obliki navideznih komentarjev s povezavami do sumljive vsebine. Če takih komentarjev ne preprečimo, nam lahko spletni iskalniki avtomatično znižajo “oceno” in nas celo “kaznujejo” do te mere, da se naša spletna aplikacija med iskalnimi zadetki sploh ne pojavi (t.i. Google Ban). Poleg tega je zloraba komentarjev in npr. uporabniških računov zelo moteča tudi za ostale uporabnike in negativno vpliva na splošno oceno kvalitete našega spletnega mesta.

Za preprečevanje ustvarjanja navideznih uporabniških računov (npr. iz strani spletnih skript), lahko uporabimo procesne rešitve, npr. tako, da mora vsak uporabnik, ki opravi registracijo, svoj račun aktivirati preko unikatne spletne povezave, ki jo po e-pošti pošljemo na vpisan e-naslov. Za uporabnike, ki registracijski postopek uspešno opravijo, moramo seveda poskrbeti, da imajo dostop samo do omejenega dela spletne aplikacije.

Napadi z vrinjenimi SQL stavki (angl. SQL injection attacks) so pogost način zlorab spletnih obrazcev. Morebitni napadalec na tak način poskuša izkoristiti površnost programerja, ki ni ustrezno poskrbel za saniteto vpisanih podatkov. Vsak vpisan

podatek je namreč potrebno, preden ga vstavimo v SQL poizvedbo, ustrezno prečistiti, kar pomeni, da vse znake, ki imajo poseben pomen v SQL sintaksi, dodatno označimo (med te znake sodi npr. enojni narekovaj).

Skriptni napadi (angl. Cross-side scripting attacks oz. XSS) so eden od najbolj perečih problemov spletnih aplikacij. Ime nakazuje na tipe zlorab, ki so se pojavili najprej, to je izvajanje Javascript kode v varnostnem kontekstu druge aplikacije. Se pravi, napadalec naloži stran A v kontekst strani B (skupaj s škodljivo Javascript kodo, ki na primer odstrani validacijo vpisanih podatkov), vpiše škodljive podatke in vse skupaj pošlje nazaj do strežnika na strani A v procesiranje. Če na strežniku ni dodatno poskrbljeno za zaščito pred tovrstnimi napadi, se lahko zgodi katastrofa. Večina strokovnjakov za spletno varnost deli XSS napade na dve osnovni vrsti: 1) ne-obstojne, kjer se zloraba aktivira takoj in 2) obstojne, kjer se zlorabljena koda shrani v podatkovno bazo in izvede vsakič, ko se uporabniki sprožijo neko akcijo, npr. ogled uporabnikovega profila.

Kraja seje (angl. Session stealing) je naslednji v vrsti pogostih napadov, pred katerim ni enostavne obrambe. Vsakič, ko pokličemo PHP funkcijo sessions_start(), PHP ustvari naključen identifikator seje in uporabniku pošlje Set-Cookie direktivo (angl. HTTP header) nazaj uporabniku, ki ustvari piškotek, s katerim se identificira pri vsaki nadaljnji HTTP komunikaciji s strežnikom [2]. Privzeto ime piškotka je PHPSESSID. Ker lahko potencialni napadalec s prestrezanjem podatkov med računalnikoma A (odjemalec) in B (strežnik) enostavno ugotovi naključno ustvarjen niz podatkov, ker je privzeto ime piškotka vnaprej znano, je priporočljivo, da namesto PHPSESSID uporabimo nek naključen niz znakov in vsakič, ko se uporabnik uspešno prijavi, ponovno ustvarimo nov indikator seje (PHP funkcija session_regenerate_id()).

Vrst različnih zlorab je seveda zelo veliko, opisal sem samo najbolj pogoste. Še tako dobri varnostni mehanizmi ne morejo nadomestiti rednega nadzorovanja aplikacije in uporabniških podatkov. Metoda, ki jo za ta namen uporabljamo, so avtomatično pošiljanje e-sporočil (angl. SGE - System Generated Emails) na tistih delih aplikacije, pri katerih lahko prihaja do možnih zlorab, npr. pri registraciji novega uporabnika ali pri izvedbi zahtevka, za katerega trenutno aktivni uporabnik nima potrebnih pooblastil

in bi lahko pomenili zlorabo sistema.

3.2.4 Testiranje

Testiranje osnovnih funkcionalnosti aplikacije poteka sočasno z razvojem. Zelo priporočljivo je, da se za čim večji del aplikacije, ki zajema poslovno logiko, napiše avtomatične teste enote (angl. Unit Tests), ki preverjajo pravilno delovanje posameznih razredov oz. metod. V nasprotnem primeru nikoli ne vemo, ali ne aplikacija delujejo pravilno, ko smo nad določenim delom kode izvedli popravke. Ročno testiranje je zamudno in praktično nemogoče je pokriti vse možne scenarije, ki se lahko zgodijo. Zato s testi enot poskrbimo, da vsaka komponenta zase deluje pravilno in po pričakovanjih.

Uporabniški test (angl. User Acceptance Test) je najpogosteje sestavljen iz testiranja “na roke”, vendar v povezavi s tipičnimi scenariji. Npr. registracija uporabnika, sprememba gesla, prijava, sprememba podatkov profila, testiranje napačnih vnosov, itd... Uporabniški oz. uporabniški potrditveni test je namenjen potrditvi naročnika, da aplikacija deluje po pričakovanjih in je pogosto pogoj za poplačilo stroškov projekta v celoti. Za vodenje seznama napak in pomanjkljivosti se lahko uporablja kar preglednico s stolpci, v katerih so med drugim:
- ime osebe, ki je zaznalo napako
- ime in opis napake
- okolje (brskalnik, operacijski sistem)
- čas zaznanja napake
- ime osebe, ki je odgovorno za napako
- status napake (nova, odprta, zaprta, rešena)

3.2.5 Optimizacija

Optimizacija časa nalaganja strani

Optimizacija časa nalaganja strani je večplastna, ker je čas nalaganja strani odvisen od velike vrste različnih dejavnikov. Seveda velja, da mora biti sama vsebina na

strani (velikost slik, itd.) primerno pripravljena, slike ustrezno pomanjšane in stisnjene (npr. JPEG stiskanje).

Sama optimizacija programske kode presega okvire tega diplomskega dela, čeprav lahko v določenih primerih igra ključno vlogo pri času nalaganja strani - pri neskončnih zankah ne pomaga nobena druga oblika optimizacije in splet tukaj ni nobena izjema.

Za tiste dele aplikacije, do katerih se najpogosteje dostopa, je smiselno uvesti mehanizem predpomnjenja (angl. caching). Predpomnjenje se lahko izvaja na samem nivoju podatkovne baze (predpomnjenje poizvedb MySql: http://dev.mysql.com/doc/refman/5.1/en/query-cache.html)
ali z uporabo programske opreme, kot je Memcached (http://memcached.org/), s katerim v delovni pomnilnik računalnika shranimo rezultate časovno najbolj potratnih poizvedb.

Na optimizacijo pomembno vplivajo tudi HTTP meta značke, s katerimi določimo, kdaj je bila določena stran nazadnje osvežena in brskalniku povemo, kako naj ravna s predpomnjenjem. Velik prihranek lahko dosežemo tudi s stiskanjem prenesenih podatkov (angl. compression), kar dosežemo z aktivacijo določenih modulov na spletnem strežniku (npr. mod_deflate na strežniku Apache).

Število poizvedb HTTP, ki jih sproži posamezna stran, mora biti čim manjše. Poizvedbe HTTP se uporabljajo za vključevanje datotek CSS in Javascript. Splošno pravilo je, da naj bi bile vse različne datoteke CSS na strežniku združene v eno samo, strežnik pa naj bi to datoteko še dodatno stisnil, preden jo pošlje brskalniku (odjemalcu). Podobno velja za datoteke Javascript - čim manj, tem bolje. Vključene naj bodo čim kasneje (nikakor takoj na vrhu strani) in minimizirane (angl. minified). Minimizirane datoteke Javascript so praktično neberljive, zato se uporabljajo le v produkcijskem okolju.

Za spletne aplikacije, razvite z namenom, da do njih dostopajo uporabniki iz celega sveta, je smiselno uporabiti CDN (angl. Content Delivery Network). Fizična oddaljenost odjemalca (brskalnika) oz. uporabnikovega računalnika od spletnega

strežnika opazno vpliva na hitrost nalaganja strani. CND poskrbi za to, da se slike, datoteke flash, CSS in Javascript naložijo iz tiste lokacije, ki je trenutnemu uporabniku geografsko najbližje. Aplikacija mora biti temu seveda prilagojena (zahtevki HTTP do teh lokacij morajo sprožiti prejem podatkov iz CDN, ne lokalnega strežnika).

Enega od večjih javnih CDN-jev vzdržuje spletni gigant Google. Enostaven primer uporabe CDN lahko ponazorimo s tem, da programsko ogrodje Javascript JQuery vključimo iz Google CDN-ja, namesto iz lokalnega strežnika:

```
<script type="text/javascript"
      src="http://ajax.googleapis.com/ajax/libs/jquery/1.4.0/jquery.min.js">
</script>
```

Ne samo, da se bo obremenitev naše infrastrukture na ta način zmanjšala, uporabniki bodo do datoteke dostopali hitreje (glede na lokacijo). Pogosto prenos datoteke niti ne bo potreben, ker bo njihov spletni brskalnik zaznal, da so enako datoteko prenesli že prej (med obiskom neke druge spletne strani) in uporabil svojo lokalno kopijo.[3]

Optimizacija za iskalnike

Optimizacija za iskalnike je preobsežno poglavje in jo bomo v okviru diplomskega dela zgolj omenili. Pri optimizaciji za iskalnike gre za določene vsebinske in tehnične pristope, ki omogoča, da spletni iskalniki lažje in boljše kategorizirajo vsebino. Prvo pravilo, ki ga uporablja Google-ov iskalnik je, da višje na seznamu iskalnih zadetkov prikaže tista spletna mesta, ki so za večino uporabnikov najbolj relevantna (do njih vodi največ povezav iz drugih spletnih mest). Gre za podobno tehniko, kot je citiranje znanstvenih člankov. Seveda tak način odpira ogromno možnosti za zlorabe, zato je Google PageRank zapletena formula, ki se jo kar naprej izboljšuje zato, da se prepreči nove zlorabe. Poleg relevantnosti posameznega članka na razvrstitev seveda vplivajo še drugi parametri.

Višina prikaza na seznamu zadetkov je vedno odvisna od vpisanih ključnih besed. Za vpisane ključne besede se možnost dobre pozicije v iskalnikih poveča, če se te pojavijo:

- v imenu domene
- v spletnem naslovu strani (URL-ju)
- v naslovu strani (označba <title> v glavi strani)
- v naslovu vsebine (označba <h1> v vsebini strani)
- kot (odebeljen) del vsebine strani

Na dobro uvrstitev vpliva še veljavna sintaksa, ki je skladna s spletnimi standardi (npr. XHTML 1.0, XHTML 1.1, HTML 4.0) in CSS 2, CSS 3. Za razvoj teh standardov skrbi spletni konzorcij World Wide Web Consortium (http://www.w3.org/).

3.2.6 Objava spletne aplikacije: beta faza, vzdrževanje in nadgradnje

Ko se komercialno spletno aplikacijo objavi, se pravo delo pogosto šele začne. Sama metodologija razvoja namreč zagotavlja, da se najmanjši sprejemljivi produkt (angl. minimum viable product) objavi čim prej in na ta način dobi prve odzive potencialnih uporabnikov spletne aplikacije. Skladno z njihovimi željami se določi prioritete nadgradenj, za katere ocenimo, da so smiselne in določi dinamiko razvojnih ciklov (npr. objava nove verzije vsakih 14 dni). Zato je veliko spletnih aplikacij v t.i. permanentni beta fazi, ki lahko traja več let, preden se končni izdelek popolnoma izoblikuje glede na dejanske potrebe uporabnikov in tržišča. Na spremembe mora biti pripravljena razvojna ekipa in naročnik sam.

Nadgrajevanje in objavljanje popravkov praviloma sledi enaki metodologiji, kot je bila uporabljena za razvoj spletne aplikacije s to izjemo, da se obstoječe prototipe popravlja le takrat, ko se za to pokaže dejanska potreba (večje spremembe na zaslonskih mask ali spremembe v navigaciji). Seveda je uporaba prototipov zaslonskih mask nujna, kadar nadgradnja zajema nove zaslonske maske in morebitne nove poslovne procese.

Navodila za uporabo sistema

Kadar se zamenja upravljavec sistema (na strani naročnika), je zelo koristno, da ima novi upravljavec na voljo navodila za uporabo sistema. V nasprotnem primeru je

izvajalec po nepotrebnem obremenjen s podporo, ki jo je primoran nuditi novemu upravljavcu. Priprava ustreznih (razumljivih!) navodil ob splavitvi projekta je za izvajalca, ki je ravno takrat pod največjo obremenitvijo, pogosto nepotrebna dodatna zadolžitev, ki jo lahko enostavno reši s praktičnim izobraževanjem o uporabi sistema. Tovrstno izobraževanje je vedno potrebno pred primopredajo spletne aplikacije, vendar se (v obdobju nekaj let) pogosto izkaže, da bi s pripravo ustreznih navodil porabili manj časa, kot smo ga porabili pri sprotni podpori.

Potrebno se je namreč zavedati, da se z razvojem tovrstnih spletnih aplikacij premo-sorazmerno povečuje tudi nabor informacij, s katerimi mora upravljati izvajalčeva projektna ekipa. Tudi če ta več let ostane nespremenjena, ji bodo specifične rešitve in načini uporabe sčasoma ušle iz spomina.
Razlogov je zato več kot dovolj, da izvajalec ob zaključku projekta (samoiniciativno!) priskrbi kvalitetna navodila za uporabo (v pisni obliki in s slikami zaslona, na katerih so ustrezno označene pomembne akcije), čeprav priprava takih navodil ni bila vključena (niti ovrednotena) v projektni pogodbi. Neredko se namreč zgodi, da pripravo navodil “zavrne” naročnik sam, saj mu predstavljajo dodaten strošek, ki se mu lahko izogne (ker ni prepričan v komercialno uspešnost projekta).

3.3 Razvoj v praksi

Za praktično izvedbo opisane metodologije razvoja je bilo seveda potrebno izbrati najprimernejša orodja. Če povzamemo, potrebujemo torej CMS sistem za izgradnjo tipičnih vsebin spletnega mesta, ki bo enostaven za uporabnika, enostavno nadgradljiv in ki bo imel veliko bazo uporabnikov ter razvijalcev. Nadgradljivost sistema mora biti izvedena v obliki vtičnikov, baza obstoječih vtičnikov mora biti ustrezno velika. Dobro mora biti poskrbljeno za optimizacijo za iskalnike, rešitve tipičnih zahtev (npr. fotogalerije), XML kazala strani, lovljenje nezaželenih komentarjev, ipd.

Glede na zgornje zahteve smo izbrali open-source CMS sistem WordPress, katerega razvoj se je začel leta 2003 in ga danes uporablja že več kot 20 milijonov spletnih mest (vir: http://en.WordPress.com/stats/). Ker je bil WordPress prvotno razvit kot platforma za osebno založništvo (angl. personal publishing) oz. kot podpora pisanju spletnih dnevnikov (bloganju), smo pri izdelavi kompleksnejših zaslonskih mask naleteli na velik prostor za izboljšave in ga elegantno zapolnili z ogrodjem Carrington za WordPress (http://carringtontheme.com/). Ogrodje Carrington ni samo podoba (angl. theme), temveč razvojno ogrodje, ki poenostavi razvoj in uporabo WordPress-a za spletna mesta z različnimi strukturami vsebine.

Hitro smo ugotovili, da sta WordPress in Carrington idealna rešitev za praktično vsa spletna mesta, vendar imata seveda omejitve, povezane z razvojem kompleksnejših spletnih aplikacij. Skupni imenovalec tovrstnih aplikacij so praviloma uporabniški profili, družabna omrežja in specifična poslovna logika, ki jo je potrebno učinkovito nadgrajevati in razvijati. Čeprav obstaja projekt BuddyPress, ki v WordPress vključi uporabniške račune, se zanj nismo odločili, ker smo želeli uporabniške račune in z njimi povezano poslovno logiko razviti kot ločen sistem, ki bo na najbolj enostaven in učinkovit način reševal specifične probleme - izogniti smo se želeli prilagajanju naših rešitev samemu sistemu.
Predvidevali smo, da bodo uporabniški profili zelo tesno povezani s specifično poslovno logiko, ki se bo spreminjala od projekta do projekta. Želeli smo razviti generičen modul, ki bo tovrstno funkcionalnost lahko podprl tudi pri vseh bodočih

projektih. Da bo razvoj modula in specifične poslovne logike čim hitrejši, cenejši, razvit produkt pa čim bolj kvaliteten, smo se odločili uporabiti preizkušeno, robustno, varno in dobro dokumentirano odprto-kodno (angl. open-source) programsko ogrodje Zend Framework.

Čeprav za razvoj sistema uporabljamo dva različna programska paketa, je potrebno zagotoviti, da se sistem iz stališča uporabnika obnaša enotno - uporabnik ne sme opaziti, da gre za dve ločena sistema. Zato je bilo potrebno poskrbeti za določeno mero integracije in tako iz obeh sistemov potegniti le najboljše (enostavnost vzdrževanja vsebin z WordPress-om in kvalitetno razvito poslovno logiko z Zend Framework-om). Rezultat je integracija dveh preizkušenih, dobro podprtih in kvalitetnih ogrodji, ki omogoča hiter, enostaven in robusten razvoj komercialnih spletnih aplikacij.

Vsa orodja bom podrobneje vpisal v nadaljevanju in predstavil rešitve za določene probleme, ki sem jih izpostavil v prejšnjih poglavjih. Na ta način bom prikazal smotrnost uporabe tovrstne rešitve za razvoj komercialnih spletnih aplikacij, pri katerem smo zelo omejeni s časovnimi in finančnimi zmogljivostmi.

3.3.1 CMS sistem WordPress in ogrodje Carrington

WordPress je open-source orodje za upravljanje spletnih vsebin (CMS - Content Management System), ki je zaradi odlične zasnove pridobil izjemno veliko število privržencev po vsem svetu. Kot že omenjeno, se je iz orodja, v prvi vrsti namenjenega piscem spletnih dnevnikov (t.i. blogerjem), z razvojem določenih funkcionalnosti (v prvi vrsti vpeljavi Strani (angl. Pages) z verzijo 1.5 in Galerij (angl. Galeries) z verzijo 2.5.[6]

Verzija 3 je med drugim vpeljala še eno močno orodje: generično podporo za posebne tipe objav. Tovrstna funkcionalnost je bila pred verzijo 3 na voljo samo v obliki vtičnikov, vključenost te funkcionalnosti v sam sistem pa jasno nakazuje, da se WordPress premika iz blogerske platforme v sistem za upravljanje s spletnimi vsebinami.

Sama osnova WordPressa odlično pokriva tipične funkcionalnosti, ki jih potrebuje vsaka spletna stran (naj gre za kompleksnejšo spletno aplikacijo ali ne). Statične strani ustvarjamo in hierarhično razporejamo kot Strani (angl. Pages), ki jim določimo oblikovno predlogo, starša, ime, vsebino in dodatne parametre, ki jih potrebujemo. Dinamično vsebino (novice, blog, obvestila) ustvarjamo znotraj razdelka Objave (angl. Posts). Dinamično vsebino dodatno kategoriziramo in dodajamo značke (angl. Tags). Za vso dinamično vsebino se avtomatično ustvarja arhiv, vsaka objava ali stran podpira tudi uporabniške komentarje (če seveda to želimo). Funkcije za iskanje po vsebini, ustvarjanje navigacije, prikazovanje arhiva in prikazovanje kategorij so v WordPress vključene avtomatično in zelo enostavne za uporabo.

Povezave do ostalih (zunanjih) strani dodajamo znotraj Povezav (angl. Links), uporabnike in uporabniške pravice določamo v razdelku Uporabniki (angl. Users). Različni nivoji uporabnikov nam omogočajo, da tiste dele administrativnega vmesnika, ki so v prvi vrsti namenjeni razvijalcem (Nastavitve, Orodja, Vtičniki in Izgled) skrijemo pred ostalimi (dejanskimi) uporabniki - upravljavci vsebin, ko je razvoj končan ter tako skrijemo odvečne možnosti pred bodočim upravljavcem sistema (naročnikom).

Med najmočnejše plati WordPress-a sodijo nedvomno vtičniki, ki omogočijo podporo za praktično vsako funkcionalnost, ki si jo lahko zamislimo. Spletna aplikacija Garmin Izziv (www.garmin-izziv.si), ki smo jo razvili na način, opisan v tem diplomskem delu, vključuje 18 aktivnih vtičnikov, ki rešujejo celo vrsto različnih problemov, med katerimi so:

- optimizacija za iskalnike
- upravljanje z oglasnimi pasicami
- izboljšave administrativnega vmesnika WordPressa
- periodično ustvarjanje XML mape strani
- nadgradnje generičnih WordPress funkcij za prikaz arhiva vsebin določene kategorije
- spletne ankete
- FTP klient
- spletni obrazci

- ... itd.

V času pisanja diplomskega dela obstaja približno 15.000 različnih vrst WordPress vtičnikov, ki so dostopni na uradni strani (http://WordPress.org/extend/plugins/). Vtičnike razvijajo programerji iz celega sveta in s tem pomembno vplivajo na popularnost WordPress-a kot platforme za urejanje vsebin, s katero je mogoče ustvariti praktično karkoli.

Ogromna izbira že razvitih vtičnikov in enostavnost razvoja lastnega vtičnika bistveno vplivata na čas in ceno razvoja komercialne spletne aplikacije. Za veliko večino tipičnih zahtev oz. funkcionalnosti naročnika že obstajajo vtičniki, ki dobro delujejo in s katerimi imamo razvijalci dobre izkušnje. Vtičnike, ki jih razvojna ekipa pogosto potrebuje (vtičnik za integracijo Zend Framework-a v WordPress, opisan v Prilogi C), pa, ko jih enkrat razvijemo, vedno znova lahko uporabimo pri bodočih projektih. V osnovno instalacijo WordPress-a naša razvojna ekipa tipično vključi med 10 in 12 vtičnikov, s katerimi imamo izjemno dobre izkušnje in ki odpravljajo nekatere pomanjkljivosti osnovne verzije WordPress-a, kot na primer odsotnost vgrajenega mehanizma za predpomnjenje (angl. caching).

Določene omejitve WordPress-a smo zaznali na področju oblikovnih predlog (angl. Templates). V večini primerov se namreč zgodi, da je osnovni sistem za razvoj predlog preveč enostaven, da bi elegantno podprl večje število tipičnih podstrani. Seveda je mogoče kompleksnejša spletna mesta ustvariti tudi s privzetim sistemom za razvoj predlog, vendar obstaja orodje, ki ta problem rešuje več kot odlično - Carrington Theme Framework.

Kot pravijo avtorji ogrodja, so Carrington razvili zato, da bi si poenostavili in olajšali postavljanje kompleksnejših predlog za WordPress. Rezultat njihovega razvoja so nato ponudili tudi širši javnosti pod licenco GPL - General Public Licence (http://carringtontheme.com/2009/08/what-is-carrington-the-qa/).

Kadar želimo v WordPress-u prikazati neko vsebino le pod določenimi pogoji, je potrebno pogoj določiti znotraj if () …. else stavka v ustreznem delu predloge.

WordPressov sistem za razvoj oblikovnih predlog je razbit na posamezne .php datoteke, ki predstavljajo posamezne bloke v končnem prikazu strani, na primer:

- archive.php (prikaz arhiva določene kategorije, ne glede na kategorijo)
- footer.php (prikaz noge strani, tako glavne, kot vseh podstrani)
- header.php (prikaz glave strani, tako glavne, kot vseh podstrani)
- index.php (prikaz prve strani)
- page.php (prikaz posamezne podstrani)
- search.php (prikaz iskalnih zadetkov)
- sidebar.php (prikaz levega ali desnega stolpca poleg glavne vsebine)
- single.php (prikaz dinamične vsebine, npr. novice ali blog zapisa, ne glede na kategorijo)

  ...

Z ogrodjem Carrington pa isti problem rešimo veliko bolj elegantno tako, da znotraj strukture oblikovne predloge ustrezno poimenujemo datoteke in s tem določimo logiko za prikaz. Ogrodje Carrington namreč zagotavlja atomarnost vsebine glede na določene pogoje, ki jih definira ime datoteke. Podobno, kot so osnovne WordPress predloge sestavljene iz datotek header.php, footer.php, search.php, … tako je ogrodje Carrington sestavljeno iz map z ekvivalentnimi imeni (header, footer, search). Znotraj map ustvarimo datoteke, ki pritičejo bloku kode (mape) in z imeni določimo, v katerih primerih naj se vključijo v končno oblikovno podobo posamezne strani.

- **content**
  - cat-dogodki.php (prikaz bloka dinamične vsebine s kategorijo dogodki)
  - page.php (prikaz bloka vsebine posamezne strani)
  - izdelki.php (prikaz bloka vsebine strani Izdelki)
- **header**
  - header-default.php (prikaz glave strani - velja za vse strani, ki ne zadostijo drugemu pogoju)
  - home.php (prikaz glave strani na prvi strani)
  - cat-blog.php (prikaz glave strani pri objavah s kategorijo blog)
- **comment**
  - user-gregor.php (prikaz komentarja uporabnika z uporabniškim imenom gregor)
- **sidebar**
  - sidebar-default.php (prikaz privzetega stolpca ob glavni vsebini)

- search.php (prikaz stolpca strani, kadar so na glavni vsebini rezultati iskanja)
- author-janez.php (prikaz stolpca strani, kadar je avtor objave uporabnik janez)
- tag-linux.php (prikaz stolpca strani, kadar ima vsebina značko “linux”)
- …

Primer: v desnem stolpcu (sidebar.php) strani želimo prikazati določene informacije. Prikaz informacij je odvisen od tega, na kateri strani se nahajamo. Če se nahajamo na strani O nas, Kontakt ali Politika zasebnosti, naj se na desni strani prikažejo kontaktne informacije podjetja. Na osnovni strani (Domov), se desni stolpec ne prikaže, na strani z novicami pa želimo dodati polje za prijavo na e-novice.

Z običajno WordPress predlogo bi problem rešili tako, da bi v datoteki wp-content/themes/ime-predloge/sidebar.php naredili ustrezno vejitev s stavkom IF … ELSE:

```
<?php
if (!is_home()) { // Stolpec se prikaže samo, če nismo na domači strani
	if (is_page(‘O nas’) || is_page(‘Kontakt’) || is_page(‘Politika zasebnosti’))
		printContactInfo();
	elseif (is_archive()) {
		if (hasCategory(‘novice’)) // Napisati moramo funkcijo, ki preveri,
	ali prikazujemo arhiv objav s kategorijo “novice”
			printSubscribeToNews();
	}
}
?>
```

Z ogrodjem Carrington isti problem rešimo veliko bolj elegantno.

- **sidebar**
  - contact_info.php (znotraj te datoteke pokličemo funkcijo printContactInfo())
  - subscribe_to_newsletter.php (znotraj te datoteke pokličemo funkcijo printSubscribeToNews())
- **pages**
  - o_nas.php

    - politika_zasebnosti.php
    - pogoji_poslovanja.php

      V katerih s funkcijo cfct_template_file('sidebar', 'kontaktni_podatki') prikličemo ustrezen stolpec.

- **loop**
    - cat-novice.php (v tej datoteki s funkcijo cfct_template_file('sidebar', 'subscribe_to_newsletter’') pokličemo ustrezen stolpec)

Vidimo, da na prvi pogled razvoj z ogrodjem Carrington izgleda nekoliko bolj zapleten, vendar je potrebno poudariti, da z razbijanjem predloge na manjše, lažje obvladljive kose in z izkoriščanjem mehanizma za logiko prikazovanja (abstrakcijo pogojev na nivoju posameznih datotek) zelo kmalu dosežemo točko hitrejšega razvoja pri kompleksnejšem spletnem mestu.

Po našem mnenju ogrodje Carrington uspešno odpravi eno ob večjih pomanjkljivosti WordPress-a, saj postane koda predloge veliko bolj pregledna. Taka preglednost igra bistveno vlogo pri vzdrževanju aplikacije, ko se pojavi zahteva za dodajanje novih zaslonskih mask ali spreminjanje načina prikaza obstoječih. Z ogrodjem Carrington je WordPress korak bližje tipičnemu CMS sistemu kot blogerski platformi, ob tem pa ohrani vse svoje prednosti, ki so botrovali njegovemu bliskovitemu razvoju.

Seveda pa ogrodje Carrington ne more rešiti popolnoma vseh specifik, ki jih želi imeti naročnik na posamezni strani. Za izvedbo poljubnega prikazovanja vsebine na posamezni strani so zato pogosto potrebne t.i. poljubne zanke (angl. Custom Loops), s katerimi se vključimo v WordPress Loop - mehanizem, ki pri prikazovanju strani poskrbi za nastavitev vseh potrebnih parametrov in objektov, ki jih potrebujemo za prikaz ter omogoči uporabo določenih funkcij, kot je npr. funkcija za prikaz naslova trenutne objave `the_title()` (vir: http://codex.WordPress.org/The_Loop).
Poleg uporabe poljubnih zank pri prikazu vsebine pa so nadvse koristna tudi poljubna polja (angl. Custom Fields), ki jih za vsak tip objave lahko nastavimo v obliki parov ključ-vrednost (angl. key-value pairs).
Podan je dejanski primer kombinacije Carrington ogrodja, poljubne zanke in poljubnih polj pri portalu Garmin Izziv, ki smo ga razvili z opisano metodologijo in tehnologijo.

Stran Garmin Trase smo znotraj WordPress-a ustvarili kot statično stran brez vsebine. S tem je postala del običajne strukture strani. Z ogrodjem Carrington smo določili način prikaza strani - glavna datoteka, ki služi za prikaz strani, se nahaja znotraj mape `loop`, ime datoteke je `trase.php`. WordPress, Carrington in poljubne zanke nam omogočajo, da na strani Garmin Trase prikazujemo dinamično vsebino - objave s kategorijo Garmin Trase. Torej stran Garmin Trase služi kot vsebnik (angl. container) za objave z enako-imensko kategorijo.

Vsaki od objav smo določili vrsto poljubnih polj (Razdalja, Višinska razlika, Zahtevnost, Disciplina,...) in z vtičnikom za FTP prenos datotek naložili XML datoteko s podatki o trasi (pridobljeno iz Garmin tekaške ure). Rezultat poljubne zanke je prikazan na Sliki 3 (programska koda je na voljo v Prilogi B).

Trase GARMIN ekipe

Predloge za GARMIN trase lahko vsi registrirani uporabniki pošljete na e-naslov: marketing@garmin.si. Če bo trasa izbrana in dodana kot GARMIN trasa, bomo uporabnika izbrane trase nagradili s praktično nagrado Garmin.

Legenda:

- Lahka trasa
- Srednje težka trasa
- Težka trasa

▼ Trening ženske kolesarske ekipe Klub Polet GARMIN Prikaži

Razdalja	75,9 km
Višinska razlika	1429 m
Zahtevnost	●

Prenesi traso

Poglej podrobnosti »

▶ Trasa 6 urnega treninga moške kolesarske kontinentalne ekipe Zheroquadro Radenska Prikaži

▶ Jutranji krog (tek) Anice Živko Prikaži

▶ Kolesarski trening Aje Opeka, članice kolesarske ekipe Klub Polet GARMIN Prikaži

▶ Garmin triatlon Bled Anje Križnik Tomažin Prikaži

▶ Trasa 7 urnega treninga moške kolesarske kontinentalne ekipe Zheroquadro Radenska Prikaži

▶ Luče – Logarska dolina, tekmovalni odsek Ultramaratona Celje – Logarska dolina 2010 Prikaži

Slika 3: Prikaz rezultata poljubne zanke WordPress-a za prikaz tras v aplikaciji Garmin Izziv

3.3.2 Zend Framework

Opisana kombinacija WordPress-a in ogrodja Carrington lahko odlično služi za postavitev običajne spletne strani ali t.i. vsebinskega dela komercialne spletne aplikacije, v katerem praktično ni poslovne logike oz. je ta zelo enostavna in omejena na npr. pošiljanje spletnih obrazcev.

Poslovna logika oz. “jedro” informacijskega sistema (spletne aplikacije) mora biti razvita na način, ki bo zagotavljal veliko robustnost kode in ločil hrambo, poslovna pravila ter predstavitev podatkov - razvoj mora striktno slediti arhitekturi Model - Pogled - Krmilnik. Za izvedbo poslovne logike smo se odločili uporabiti programsko ogrodje Zend Framework, ki teče na zelo razširjenem skriptnem jeziku PHP 5. Ogrodje zadosti vsem naštetim pogojem [7] in na nek način vsiljuje dobre prakse objektnega programiranja. Hkrati nudi sprejemljiv način licenciranja (licenca BSD za uporabnike, individualni ali korporativni CLA za soustvarjalce ogrodja), dobro je poskrbljeno tudi za dokumentacijo. Velika in hitro rastoča skupnost uporabnikov ter razvijalcev (http://framework.zend.com/community/resources) nam lahko pomaga pri težavah, na katere naletimo, komercialna uspešnost velikih spletnih sistemov, ki temeljijo na Zend Framework-u pa kaže na zrelost projekta in same programske kode (npr. sistem Magento za e-poslovanje: http://www.magento.com, Fox Interactive Media: http://www.ign.com, IBM: http://services.alphaworks.ibm.com/qedwiki/ in drugi). Ogrodje Zend Framework je imelo do tega trenutka več kot 10 milijonov prenosov.

Seveda se lahko bralcu porodi vprašanje, zakaj v tem primeru nismo zgradili celotnega sistema z uporabo Zend Framework-a. Odgovor na to vprašanje je jasen: razvoj administrativnega vmesnika sistema za upravljanje z vsebinami in vseh dodatnih funkcionalnosti, kot so fotogalerije, spletne ankete, optimizacije za iskalnike, orodja za predpomnjenje, itd. bi trajal preveč časa. Stranke bi morale v tem primeru dejansko investirati v razvoj nečesa, kar je že razvito (angl. “reinveinting the wheel”), cene projektov bi poskočile in postali bi nekonkurenčni. Zato smo se odločili, da “na ključ” razvijemo samo tiste dele sistema, ki podpirajo specifično poslovno logiko, za vse ostale klasične probleme pa uporabimo obstoječe, dobro delujoče rešitve in si s tem zagotovimo konkurenčnost, stranki pa maksimalno kvaliteto razvite rešitve.

Zend Framework nam tako služi kot zbirka šibko povezanih (angl. loosely coupled) komponent, od katerih v aplikacijo vključimo samo tiste, ki jih potrebujemo (npr. Zend_Mail za pošiljanje e-sporočil), obenem pa podpira MPK arhitekturo aplikacije. Ker je vseh komponent enostavno preveč, se bomo omejili na tiste komponente, ki služijo arhitekturi Model-Pogled-Krmilnik.

Komponente, ki podpirajo MPK arhitekturo

Zend_Application, Zend_Application_Bootstrap, Zend_Application_Module in Zend_Application_Resource

Zagon MVC aplikacije je kompleksen proces, pri katerem je potrebno zagotoviti pripravo izvedbenega okolja, to je vzpostavitev povezave s podatkovno bazo, določitev parametrov za izgled, registracijo pomožnih (angl. helper) razredov na nivoju pogleda in krmilnika, registracijo seje (angl. session), itd. S komponentami Zend_Application si olajšamo zagon modularno razvitih aplikacij. Komponente poskrbijo za avtomatično preverjanje odvisnosti med posameznimi razredi, pripravo PHP okolja in avtomatično nalaganje modulov (angl. autoloading). Vzpostavijo delovno okolje za zagon posameznih modulov znotraj aplikacije in omogočajo specifikacijo različnih okolji glede na posamezen modul. Ko se zagonski proces konča, sprožijo glavni krmilnik pogleda (Zend_Controller_Front).

Zend_Controller_Front, Zend_Controller_Action, Zend_Controller_Dispatcher, Zend_Controller_Plugin, Zend_Controller_Router

Zend_Contoller je srce MPK sistema v Zend Framework-u. Zend_Controller_Front oz. glavni krmilnik pogleda prestreže vse zahtevke znotraj aplikacije in glede na prestrežene zahtevke (definirane z URL-ji) sproži ustrezne akcijske krmilnike (angl. Action Controllers). Zend_Controller_Action je abstraktni razred, ki služi za implementacijo krmilnikov znotraj posameznih modulov. Vsak modul ima nabor akcijskih krmilnikov (Action Controllers), vsakemu od akcijskih krmilnikov lahko določimo posamezne akcije, ki se izvedejo, ko sprožimo zahtevek na določen naslov (URL). Za sprožitev in preslikavo teh zahtevkov skrbita Zend_Controller_Dispatcher in Zend_Controller_Router (določita in sprožita ustrezen modul, akcijski krmilnik,

akcijo in morebitne HTTP GET parametre glede na HTTP zahtevek npr. /uporabniki/janez/profil/). Zend_Controller_Plugin je abstraktna komponenta, ki jo uporabimo za pisanje lastnih vtičnikov (na nivoju modula ali globalno). Primer tovrstnega vtičnika je na primer vtičnik, ki preveri, ali ima trenutno prijavljen uporabnik administrativne pravice, ali ne.

Zend_Form

Zend_Form poenostavi upravljanje s spletnimi obrazci in z zbiranjem uporabniških podatkov. Avtomatično poskrbi za filtriranje, saniteto vpisanih podatkov, prikaz obrazcev, dinamično dodajanje validatorjev (npr. dolžina niza, ali je vpisan veljaven e-naslov), grupiranje in razvrščanje obrazcev in preverjanje, ali je poslan obrazec veljaven. Če obrazec ni veljaven (obvezna polja niso izpolnjena), avtomatično poskrbi za prikaz sporočil o napakah. Uporaba komponente Zend_Form lahko služi tudi za razvoj prototipov v okviru RAD (Rapid Application Development), saj je dodajanje elementov v spletni obrazec izredno enostavno, za saniteto vpisanih podatkov ni potrebno skrbeti (ker vse opravi komponenta sama), izpis še tako kompleksnega obrazca (z vsemi napakami) pa je mogoč z eno samo vrstico kode.

Zend_Layout

Zend_Layout implementira dvo-nivojski nivo pogleda (angl. Two Step View pattern), ki razvijalcem omogoča, da izpis, ki ga sproži akcijski krmilnik in njemu pripadajoč pogled, ovijemo v drug, zunanji pogled, ki implementira oblikovno predlogo. V tej oblikovni predlogi sta ponavadi definirana glava in noga strani, ki “ovijeta” izpis trenutne zaslonske maske. Poleg tega Zend_Layout definira ločen doseg (angl. scope) za nekatere spremenljivke, kot so npr. splošen tip HTML dokumenta (angl DocType) in druge. S komponento Zend_Layout lahko definiramo več oblikovnih predlog, kar nam npr. omogoča, da določenim skupinam uporabnikov prikazujemo drugačno oblikovno predlogo (npr. uporabimo večjo pisavo za vse tiste uporabnike, ki so starejši od 65 let) - vsaka datoteka, ki jo uporablja Zend_Layout, lahko vključuje svojo datoteko CSS.

Zend_View, Zend_View_Filter, Zend_View_Helper

Komponente Zend_View služijo predstavitvi Pogleda znotraj arhitekture Model-

Pogled-Krmilnik. Z njimi definiramo predstavitev podatkov za posamezne akcije, ki so definirane v akcijskih krmilnikih. Do akcijskih podatkov iz samega Pogleda seveda ne moremo neposredno dostopati, zato je potrebno znotraj same akcije dostopati do instance Pogleda in tej instanci določiti vsako spremenljivko posebej. Ker so te spremenljivke lahko precej kompleksne (npr. tabela z uporabniškimi podatki), hkrati pa želimo ohraniti preglednost skriptne datoteke Pogleda, lahko definiramo pomožne razrede Pogleda (angl. View Helpers). Znotraj pomožnih razredov pogleda poskrbimo za ustrezno oblikovanje podatkov (npr. izpis tabele, oblikovanje datumov, dinamično ustvarjanje povezav) in rezultat uporabimo v Pogledu. Tako poskrbimo za kodo, ki je lepo berljiva in ki jo je veliko lažje vzdrževati.

Struktura Zend Framework aplikacije (Garmin Izziv):

application (glavna mapa s programsko kodo aplikacije)
- **configs** (konfiguracijske datoteke)
  - application.ini (privzeta konfiguracijska datoteka)
- **controllers** (privzeti akcijski krmilniki)
  - ErrorController.php (akcijski krmilnik za napake)
  - IndexController.php (privzeti akcijski krmilnik)
- **layouts** (vsebinske in grafične predloge)
  - sge (mapa s predlogami za sistemsko ustvarjena e-sporočila)
  - layout.phtml (glavna grafična predloga)
- **modules** (moduli, razviti za potrebe projekta ali preneseni iz ostalih projektov)
  - **competitions** (vsak modul ima enako strukturo)
    - **controllers** (akcijski krmilniki)
      - ...
    - **forms** (obrazci in posebni elementi obrazcev)
      - ...
    - **models** (modeli - hramba)
      - DbTable (hramba v obliki tabele podatkovne baze)
        - ...
    - **views** (pogledi in pomožni razredi pogledov - predstavitev podatkov za vsako akcijo akcijskih krmilnikov)
    - Bootstrap.php (neobvezna zagonska datoteka, specifična za ta modul)
  - **leagues**
    - … (podobno kot zgoraj)

```
            tracks
                  … (podobno kot zgoraj)
            users
                  … (podobno kot zgoraj)
      views (privzeti pogledi)
            scripts (mapa s pogledi)
                  error (mapa s pogledi akcijskega krmilnika index)
                  index (mapa s pogledi akcijskega krmilnika error)
      Bootstrap.php (obvezna glavna zagonska datoteka aplikacije)
cache (podatki za predpomnjenje)
library (Zend in uporabniške zbirke komponent)
      Custom (naša uporabniška zbirka, napisana posebej za ta projekt)
      PhpThumb (zbirka funkcionalnosti za kreiranje pomanjšanih slik)
      Zend (glavna zbirka Zend ogrodja)
      ZendX (pomožna GUI zbirka ogrodja ZendX)
public (mapa, ki jo lahko bere spletni strežnik in je ločen od same aplikacije)
      ...
uploads (mapa z datotekami, ki jih naložijo uporabniki)
      ...
```

Pri sami strukturi aplikacije je morda najbolj zanimiva mapa `modules`, v katerem so posamezni moduli, razviti za potrebe aplikacije. Z modularnim razvojem zagotovimo ločenost funkcionalnosti na posamezne ločene sklope, kar bistveno vpliva na preglednost kode in omogoča lažje vzdrževanje in nadgrajevanje aplikacije z novimi funkcionalnostmi. Z modulom `users` aplikaciji dodamo podporo za uporabniške profile (registracijo uporabnikov, prijavo, spremembo gesla, itd). Prvič smo ga razvili za potrebe projekta UCIRA (University of California institute for research in the arts) in z minimalnimi prilagoditvami ponovno uporabili pri vseh naslednjih projektih in tako zmanjšali potreben čas in stroške razvoja aplikacije.

Kritičnega pomena za delovanje aplikacije je seveda mapa `library`, v kateri so zbirke posameznih komponent. Pomembno je, da našo lastno zbirko (`Custom`) ločimo od ostalih zbirk, še posebej od zbirke Zend Framework-a. Priporočljivo je, da je zbirka Zend Framework-a shranjena izven mape spletnega strežnika (`/var/www/` na operacijskem sistemu Linux). Ker isto zbirko uporabimo za več projektov, lahko pri vsakem projektu ustvarimo simbolično povezavo do zbirke in s tem zagotovimo

enotno kopijo zbirke ogrodja za vse razvite aplikacije:

```
$ ln -s /usr/local/frameworks/zend/library/Zend-1.11.4 /var/www/garmin-izziv.si/app/library/Zend
```

Rešitve nekaterih tipičnih težav z Zend Framework-om

Povečanje varnosti z abstrakcijo SQL poizvedb

Z abstrakcijo SQL stavkov dosežemo neodvisnost poizvedb od vrste podatkovne baze. Zend_Db_Select omogoča objektno usmerjen pristop pri pisanju SQL poizvedb, avtomatično dodajanje narekovajev (angl. quotes) in ostalih meta znakov.

```
class Leagues_Model_DbTable_Leagues extends Zend_Db_Table_Abstract {
	...
public function submitLeague($leagueId) {
	if ($leagueId > 0) {
		$row = $this->fetchRow($this->select()
				->where('id = ?', $leagueId)
				->where('published IS NULL')
				->where('active = ?', 1)
		);
		if($row != null) {
			$row->published = date('Y-m-d H:i:s');
			$row->save();
		}
	}
	return null;
	}
	...
}
```

Abstrakcija poizvedb na podatkovni bazi bistveno vpliva na varnost, ker zmanjšuje možnost zlorab z vrinjenimi SQL stavki (angl. SQL injection attacks).

Avtorizacija dostopa

Avtorizacijo dostopa do posameznih delov aplikacije lahko z ustrezno strukturo aplikacije in njenih akcijskih krmilnikov zagotovimo na enostaven način.

Dostop do administrativnega dela aplikacije (za katerega uporabljamo ločen akcijski krmilnik - `AdminController.php`) omejimo tako, da definiramo standardno funkcijo `init()` v akcijskem krmilniku, ki se izvede prva - ne glede na akcijo krmilnika. Z metodo `init()` preverimo, ali je trenutni uporabnik prijavljen in če ima administrativne pravice (za kar uporabimo pomožni razred akcijskega krmilnika). Če uporabnik ni prijavljen oz. nima administrativnih pravic, ga ustrezno preusmerimo.

```
class Leagues_AdminController extends Zend_Controller_Action {
	...
	public function init() {
		$this->_userId = $this->_helper->authUsers->getCurrentUserId();
		if(!$this->_userId)
		$this->_redirect('/'.Zend_Registry::get('routes')->account_login-
	>route, array('exit' => true));
		if(!$this->_helper->authUsers->isCurrentAdmin())
			$this->_redirect('/'.Zend_Registry::get('routes')-
	>account_profile->route, array('exit' => true));

			...
	}
	…
}
```

S tem pristopom se lahko elegantno izognemo preverjanju avtorizacije uporabnika za vsako akcijo posebej, kar posledično pomeni podvajanje kode, ampak problem rešimo samo enkrat - na nivoju akcijskega krmilnika. Če bi potrebovali tako avtorizacijo na nivoju celotnega modula, bi lahko uporabili zagonsko datoteko (Bootstrap.php) tega modula.

Konfiguracija aplikacije (primer pošiljanja sistemsko ustvarjenih e-sporočil)

Pošiljanje sistemsko ustvarjenih e-sporočil je funkcionalnost, ki jo potrebuje praktično vsaka večja spletna aplikacija. Sistemska sporočila se pošiljajo za potrebe aktivacije uporabniških računov, ob spreminjanju gesel, ob potrjevanju povezav med uporabniki

(grajenju socialnega omrežja), ipd. Ker gre za komponento, ki jo lahko uporablja več modulov, jo dodamo v uporabniško knjižnico in “library/Custom”. Pošiljanje e-sporočil zahteva seveda določene nastavitve, ki jih definiramo v za to namenjeni datoteki:

Datoteka application.ini (nastavitve so dostopne preko Zend_Registry):

```
...
mail.smtp=true
mail.host=smtp.gmail.com
mail.smtpconfig.name=localhost
mail.smtpconfig.port=465
mail.smtpconfig.ssl=ssl
mail.smtpconfig.auth=login
mail.smtpconfig.username=********@gmail.com
mail.smtpconfig.password=*******
mail.from=marketing@garmin.si
…
```

Razred Custom_Mail z uporabo Zend_Registry-a uporabi te nastavitve v konstruktorju:

```
class Custom_Mail {
	public function __construct() {
		...

		$mailConfig  = Zend_Registry::get('configuration')->mail;
		if ($mailConfig->smtp) {
			$transport = new Zend_Mail_Transport_Smtp($mailConfig->host,
	$mailConfig->smtpconfig->toArray());
		}
		else {
			$transport = new Zend_Mail_Transport_Sendmail();
		}
		Zend_Mail::setDefaultTransport($transport);
	}
}
```

Znotraj razreda seveda definiramo javno dostopne funkcije za pošiljanje posameznih

e-poštnih sporočil. Ker so e-poštne nastavitve za produkcijsko okolje pogosto drugačne, kot za razvojno in testno, nam Zend Framework omogoča, da v datoteki application.ini definiramo različne segmente konfiguracije glede na okolje - [production], [testing], [development], - znotraj katerih določimo specifične nastavitve za vsako posamezno okolje. Nato v specifikaciji virtualnega gostitelja (Apache Virtual Host) definiramo:

```
SetEnv APPLICATION_ENV production
```

oz.

```
SetEnv APPLICATION_ENV testing
```

Na tak način dosežemo, da nam ob migraciji aplikacije in testnega na produkcijsko okolje ni potrebno vsakič znova popravljati konfiguracijskih datotek.

Optimizacija za hitrost

Optimizacijo z predpomnjenjem (angl. caching) lahko dosežemo precej enostavno. Za to skrbi razred Zend_Cache_Manager, za katerega nastavitve poskrbimo v zagonski datoteki (`Boostrap.php`). Za predpomnjenje lahko uporabimo več mehanizmov. V konkretnem primeru smo uporabili sistem Memcached (distribuiran objektni sistem za predpomnjenje).

```
class Bootstrap extends Zend_Application_Bootstrap_Bootstrap {
	…
	protected function _initCache() {
		$oBackend = new Zend_Cache_Backend_Memcached(
			array(
				'servers' => array( array(
					'host' => '127.0.0.1',
					'port' => '11211'
				) ),
				'compression' => true
		) );

		$oCacheLog =  new Zend_Log();
		$oCacheLog->addWriter( new Zend_Log_Writer_Stream( 'file:///tmp/pr-
memcache.log' ) );
```

```
		$oFrontend = new Zend_Cache_Core(
			array(
				'caching' => true,
				'cache_id_prefix' => 'garminIzziv',
				'logging' => true,
				'logger'  => $oCacheLog,
				'write_control' => true,
				'automatic_serialization' => true,
				'ignore_user_abort' => true
			) );

		$oCache = Zend_Cache::factory($oFrontend, $oBackend);
		Zend_Registry::set('cache', $oCache);
		return $oCache;
	}
	…
}
```

Za lažje upravljanje z mehanizmom predpomnjenja v aplikaciji definiramo pomožni razred akcijskega krmilnika in v njem funkcijo, ki nudi določeno mero abstrakcije pri uporabi predpomnjenja:

```
class  Custom_Controller_Action_Helper_Memcached extends
	Zend_Controller_Action_Helper_Abstract {

	public static function maybeGetFromCache($cacheId, &$class, $method, $params
= array()) {
		$cache = Zend_Registry::get('cache');
		if (!$cache->test($cacheId)) {
			$dataSet = call_user_func(array($class, $method), $params);
			$cache->save($dataSet, $cacheId);
			return $dataSet;
		}
		return $cache->load($cacheId);
	}
}
```

Pridobivanje podatkov iz predpomnilnika (“cache hit”) in avtomatično shranjevanje v predpomnilnik (ko se zgodi “cache miss”) postane z uporabo pomožnega razreda veliko bolj enostavno.

```
$searchResults = Custom_Controller_Action_Helper_Memcached::maybeGetFromCache(
       'showAllUsers',
       new Users_Model_DbTable_Users(),
       'showAll'
);
```

3.3.3 Integracija WordPress-a in Zend Framework-a v enotno spletno aplikacijo

Bistven element pri praktični uporabi opisane metodologije razvoja predstavlja integracija obeh sistemov v navidezno enotno spletno aplikacijo. Integracijo lahko vzpostavimo na več nivojih, bistveno pa je, da preprečimo podvajanje kode (funkcionalnosti definiramo samo na enem mestu in do njih dostopamo iz obeh okolji).

Datotečna struktura

Predpogoj za integracijo je ustrezna datotečna struktura. Znotraj direktorja, kjer je nameščen WordPress, naredimo novo mapo (`app`), v katerem je koda Zend Framework aplikacije. Struktura mape `/var/www/garmin-izziv.si/` je torej taka:

```
…
app (Zend Framework aplikacija)
wp-admin (WordPress mapa)
wp-content (WordPress mapa)
wp-includes (WordPress mapa)
…
```

Dostopnost

Do obeh okolji seveda ne moremo dostopati s popolnoma enakim spletnim naslovom (URL-jem), ker vsako od okolji uporablja svoj mehanizem za usmerjanje in

procesiranje zahtevkov (angl. routing). Opisali bomo dve potencialni rešitvi.

1) Dostopanje s pod-domeno

V tem primeru do WordPress-a dostopamo na osnovni domeni (npr. garmin-izziv.si), do Zend aplikacije pa na pod-domeni (npr. portal.garmin-izziv.si). Pri tem pristopu potrebujemo dve ločeni specifikaciji Apache virtualnih gostiteljev, eno za WordPress in drugo za Zend Framework. Bistveno je, da določimo isto domeno za piškotke (angl Cookie Domain), ki ju bosta uporabljali obe ogrodji.

Primer nastavitev obeh virtualnih gostiteljev:

```
<VirtualHost *:80>
	DocumentRoot "/var/www/garmin-izziv.si/"
	ServerName garmin-izziv.si
	php_value session.cookie_domain ".garmin-izziv.si"
	…
</VirtualHost>

<VirtualHost *:80>
	DocumentRoot "/var/www/garmin-izziv.si/app/public/"
	ServerName portal.garmin-izziv.si
	php_value session.cookie_domain ".garmin-izziv.si"
	…
</VirtualHost>
```

2) Dostopanje s pod-mapo

Elegantnejša rešitev je, da dostop do Zend Framework-a omogočimo z uporabo pod-mape. V ta namen ustvarimo simbolno povezavo do mape public:

```
$ ln -s /var/www/garmin-izziv.si/app/public /var/www/garmin-izziv.si/skupnost
```

Zend Framework aplikacija je sedaj dostopna na enaki domeni, kot WordPress: garmin-izziv.si/skupnost

Tako je lahko domena za piškotke v obeh okolji enaka (privzeta). Na tak način

zadošča samo ena konfiguracija virtualnega gostitelja strežnika Apache, obe okolji uporabljata enako (privzeto) domeno za piškotke.

Integracija obeh okolji

WordPress funkcije (npr. izpis navigacije po strani) morajo biti dostopne Zend Framework-u in obratno. Dostopnost WordPress funkcij znotraj Zend okolja dosežemo na enostaven način. V Zend aplikaciji odpremo datoteko `public/index.php` in dodamo naslednji vrstici

```
define('WP_USE_THEMES', false);
require('../../wp-load.php');
```

Integracija Zend Framework-a v WordPress je nekoliko bolj zapletena. Potrebno je namreč zagotoviti, da se koda za vzpostavitev (angl. bootstrapping) Zend Framework aplikacije zažene pred WordPressom. Ta proces najlažje izvedemo z uporabo WordPress vtičnika, ki smo ga razvili in opisali v Prilogi C. Če želimo proces izvesti ročno, je potrebno definirati funkcijo `wz_init()` in jo postaviti na vrh vsake datoteke znotraj mape `wp-content/themes/carrington/header` (funkcija se mora izvesti preden začne brskalnik prikazovati kakršnekoli podatke).

```
function wz_init() {
	set_include_path('.' . PATH_SEPARATOR .
$_SERVER['DOCUMENT_ROOT'].'app/library' . PATH_SEPARATOR . get_include_path());
	define('APPLICATION_PATH', 'app/application');

	require_once 'Zend/Loader/Autoloader.php';
	Zend_Loader_Autoloader::getInstance();

	$application = new Zend_Application(
		'production',
		APPLICATION_PATH . '/configs/application.ini'
	);

	$application->bootstrap();
}
```

Integracija dostopa do enotne podatkovne baze

WordPress in Zend Framework uporabljata isto podatkovno bazo, pri čemer uporabljata za ločitev tabel svoje predpone (za WordPress tipično wp_ime_tabele, za Zend Framework npr. zend_ime_tabele). Za dostop do podatkovne baze je potrebno določiti štiri podatke: ime gostitelja, ime podatkovne baze (sheme), uporabniško ime in geslo. Vse vrednosti definiramo le enkrat, dostop do njih mora biti omogočen iz obeh okolji.

Konfiguracijo za dostop definiramo v WordPress na klasičen način, tako, da v datoteki `wp-config.php` določimo štiri konstante:

DB_HOST (ime gostitelja)
DB_NAME (ime podatkovne baze)
DB_USER (uporabniško ime)
DB_PASSWORD (geslo)

Zend Framework je potrebno "prepričati", da ne uporablja svojih nastavitev za podatkovno bazo, ampak nastavitve WordPress. Zato je vse, kar v konfiguracijski datoteki (`application.ini`) Zend Framework aplikacije definiramo, vmesnik za podatkovno bazo:

```
; Database settings
resources.db.adapter = PDO_MYSQL
```

V glavni datoteki `Bootstrap.php` je potrebno prepisati funkcijo `initDbAdapter()` tako, da se parametri naložijo glede na konstante, ki smo jih določili v WordPress-u:

```
class Bootstrap extends Zend_Application_Bootstrap_Bootstrap {
	...
	protected function _initDbAdapter() {
		$rs_db = $this->getPluginResource('db');

		$rs_db->setParams(
			array(
				'host' => DB_HOST,
```

```
                        'dbname' => DB_NAME,
                        'username' => DB_USER,
                        'password' => DB_PASSWORD
                  )
            );
            $db = Zend_Db::factory($rs_db->getAdapter(), $rs_db->getParams());

            $db->query("SET NAMES 'utf8'");
            Zend_Registry::set('db', $db);
            Zend_Db_Table_Abstract::setDefaultAdapter($db);
      }
```

Funkcija, ki opravi avtentikacijo uporabnika glede na vpisano uporabniško ime in geslo, lahko izgleda tako:

```
class  Custom_Controller_Action_Helper_AuthUsers    extends
      Zend_Controller_Action_Helper_Abstract {
      ...
      public function login($username, $password, $username_column='username',
$password_column='password') {
            $db = Zend_Registry::get('db');
            $usersTable = new Users_Model_DbTable_Users();
            $usersTableInfo = $usersTable->info();
            $authAdapter = new Zend_Auth_Adapter_DbTable($db,
      $usersTableInfo['name'], $username_column, $password_column);
            $authAdapter->setIdentity($username)->setCredential($password);

            $auth  = Zend_Auth::getInstance();
            $authResult  = $auth->authenticate($authAdapter);
            if (!$authResult->isValid())
                  return false;

            return $authAdapter->getResultRowObject();
      }
      ...
}
```

Enotna avtentikacija administrativnih uporabnikov

V pomožnem razredu akcijskega krmilnika definiramo funkcijo `isCurrentAdmin()`, ki preveri, ali ima trenutno prijavljen uporabnik administrativne pravice. Administrativne pravice za Zend Framework aplikacijo so lahko ločene od administrativnih pravic za WordPress. Podan je primer, ki uporabnika prepozna kot administratorja Zend Framework aplikacije, če je ali administrator WordPress-a ali pa ima v tabeli "zf_users" (ki jo uporablja Zend Framework aplikacija) ustrezno nastavljeno vrednost stolpca is_admin.

```
class Custom_Controller_Action_Helper_AuthUsers extends
Zend_Controller_Action_Helper_Abstract {
	...
	public function isCurrentAdmin() {
		if($this->isWpAdmin())
			return true;
		if($this->getCurrentUserData()==false) return null;
		return $this->_userData['is_admin'] == 1 ? true : false;
	}


	public function isWpAdmin() {
		if (current_user_can('level_10'))
			return true;
		return false;
	}
}
```

Uporaba funkcij Zend Framework-a v administrativnem vmesniku WordPress

Z opisanim pristopom je možno v administrativni vmesnik WordPressa dodati tudi administrativne funkcije, ki so povezane z Zend Framework okoljem. Podan je primer WordPress vtičnika, ki doda novo sekcijo "Uporabniki portala" v administrativni vmesnik WordPress-a.

```
/*
Plugin Name: uMagnet Admin
Description: Manage Users, Promo Codes, etc.
Author: Gregor Beslič
Version: 1.0
*/

/**
* Displays users registers into Zend Framework application
*/
function zend_admin() {
	$usersTable = new Users_Model_DbTable_Users();
	$allUsersData = $usersTable->getAllUsersData();

	$adminUsers = new Custom_Controller_Action_Helper_AdminUsers();
	foreach($allUsersData as $id => $userData) {
		$allUsersData[$id] = $adminUsers->setupUserData($userData, 10);
	}
	if(count($allUsersData) > 0) {
		$userHelper = new Custom_View_Helper_DisplayUser();
		$userHelper->displayAdminAll($allUsersData);
	}
}

/**
* Hook for adding the plugin into WordPress Admin
*/
function zend_admin_page() {
	add_dashboard_page('Uporabniki portala','Uporabniki portala',
'manage_options', 'uporabniki-portala','zend_admin');
}
add_action('admin_menu', 'zend_admin_page');
```

4 SKLEPNE UGOTOVITVE

Opisana metodologija in tehnološki pristop, ki smo ga uporabili, sta nam v preteklih dveh letih omogočila kvalitetno izvedbo storitev razvoja in vzdrževanja spletnih aplikacij. Naši naročniki so bili različno velika podjetja in institucije, ki jih je gospodarska kriza prisilila v racionalnejše upravljanje s sredstvi. Potrebovali so izvajalca, ki bo sposoben njihovo idejo realizirati hitro in stroškovno učinkovito.

Uporabljena metodologija se od projekta do projekta minimalno spreminja, vseeno pa je potrebno stremeti k temu, da je celotna razvojna ekipa optimalno obremenjena v vseh fazah razvoja. To še posebej velja za majhna razvojna podjetja oziroma podjetja, ki v svoji zagonski fazi del svojih zmogljivosti namenijo izvedbi komercialnih projektov, s tako pridobljenimi sredstvi pa financirajo razvoj lastnih produktov (angl. bootstrapping).

Če primerjamo obe odprto-kodni rešitvi, ki smo jih uporabili pri razvoju, lahko rečemo, da je učenje uporabe WordPress-a neprimerno lažje. Administrativni vmesnik je glede na naše izkušnje izjemno enostaven za uporabo in naročniki z njim praktično nimajo težav. Po drugi strani ima Zend Framework dokaj strmo krivuljo učenja (angl. steep learning curve) in ni najbolj primeren za razvijalce – začetnike.

Nivo integracije obeh sistemov je šele v začetni fazi, zato je tu prostora za izboljšave še dovolj. Izboljšave bi morale biti usmerjene predvsem večji nivo abstrakcije razvitih modulov ZF, komponente zbirke ZF in vtičnikov WP. Na ta način bi še dodatno skrajšali čas, ki bi bil potreben za izvedbo naslednjega projekta.

SEZNAM UPORABLJENIH VIROV

[1] T.A. Powell, Web Site Engineering: Beyond Web Page Design, Prentice Hall, 1998

[2] D. Shafik, B. Ramsey: Zend PHP 5 Certification study guide, Toronto, Kanada: Marco Tabini & Associates, 2006, stran 215

[3] E. Castledine, C. Sharkie: JQuery - Novice to Ninja, Melbourne, Australia: Sitepoint, 2010, stran 9

[4] G. Vossen, S. Hagemann: Unleashing Web 2.0, Burlington Massachusetts: Morgan Kaufman Publishers, 2007, stran 208

[5] T. Parker, P. Tolland, S. Jehl, M. Costello: Designing with progressive enhancement, Berkeley, California: New Riders, 2010, stran 13

[6] A. Brazell: WordPress Bible, Wiley Publishing, Inc., Indianapolis, Indiana, 2010, stran 16

[7] K. Pope: Zend Framework 1.8 Web Application Development, Birmingham, Velika Britanija: Packt Publishing, 2009

PRILOGE

Priloga A. Priprava sistemskega okolja (LAMP)

1) Inštalacija LAMP sklada (Linux, Apache, MySql, PHP)

1.1) Inštalacija strežnika za podatkovno bazo MySql:

```
$ sudo apt-get install mysql-client mysql-server
```

Med inštalacijo določimo MySql root geslo

1.2) Inštalacija spletnega strežnika Apache:

```
$ sudo apt-get install apache2
```

1.3) Inštalacija PHP:

```
$ sudo apt-get install php5 libapache2-mod-php5
```

Nato je potreben restart spletnega strežnika:

```
$ sudo /etc/init.d/apache2 restart
```

Priporočljivo je narediti datoteko `info.php`, s pomočjo katere spremljamo konfiguracijo PHP okolja:

```
$ sudo nano /var/www/info.php, za vsebino datoteke dodamo:
<?php phpinfo(); ?>
```

Preizkusimo, če deluje: `http://localhost/info.php`

Za to, da lahko iz PHP-ja kličemo MySql funkcije, moramo inštalirati še:

```
$ sudo apt-get install php5-mysql
```

Pripročljivo je inštalirati tudi naslednje PHP module:

```
$ sudo apt-get install php5-curl php5-gd php5-idn php-pear php5-imagick php5-imap
php5-mcrypt php5-memcache
```

1.4) Inštalacija PhpMyAdmin (Php aplikacija za upravljanje z MySql podatkovno bazo):

```
$ sudo apt-get install phpmyadmin
```

Med inštalacijo imamo možnost, da se phpMyAdmin avtomatično nastavi za naš spletni strežnik Apache

Med inštalacijo imamo možnost, da avtomatično nastavimo konfiguracijo phpMyAdmin-a (vpišemo root geslo, naslednje geslo pustimo prazno - zgenerira se avtomatično)

PhpMyAdmin je dostopen na: `http://localhost/phpmyadmin/`

2) Konfiguracija (domena je npr. podjetje.com)

2.1) Ustvarjanje domenskega zapisa

```
*.dev  IN    A XXX.XXX.XXX.XXX
```

Preverjanje, ali domenski zapis deluje:

```
$ ping projekt.dev.podjetje.com
```

2.2) Konfiguracija Apache strežnika, da na HTTP zahtevke za projekt.dev.podjetje.com reagira ustrezen virtualni gostitelj

```
$ cd /etc/apache2/sites-available/
$ sudo mkdir websites
```

Z naslednjim ukazom dosežemo, da so vsi virtualni gostitelji znotraj mape websites avtomatično vklopljeni.

```
$ sudo ln -s /etc/apache2/sites-available/websites/ websites
$ sudo cp default websites/projekt.dev.podjetje.com
```

Nastavitve virtualnega gostitelja:

```
<VirtualHost *:80>
       DocumentRoot "/var/www/projekt/"
       ServerAdmin webmaster@localhost
       ServerName projekt.dev.podjetje.com

       <Directory /var/www/projekt/>
```

```
                Options -Indexes FollowSymLinks
                AllowOverride All
        </Directory>

        php_value include_path ".:/var/www/projekt/"

        LogLevel warn
        ErrorLog /var/log/www/projekt/error.log
        CustomLog /var/log/www/projekt/access.log combined
</VirtualHost>
```

Kreiranje ustreznih direktorijev za log datoteke:

```
$ sudo mkdir -p /var/log/www/projekt
```

Vklop ustreznih modulov:

```
Mod_Rewrite: $ sudo ln -s /etc/apache2/mods-available/rewrite.load
/etc/apache2/mods-enabled/rewrite.load

$ sudo /etc/init.d/apache2 reload
```

Preverimo, če je Apache konfiguracija ustrezna:

```
http://projekt.dev.podjetje.com
```

2.4.) MySql konfiguracija

usvarjanje podatkovne baze:

```
$ mysql -u root -p
mysql> create database projekt_dev;
mysql> create user 'projekt_user'@'localhost' IDENTIFIED BY 'projekt123';
mysql> grant all on projekt_dev.* to 'projekt_user'@'localhost';
mysql> flush privileges;
```

Sedaj, ko smo pripravili ustrezno okolje, lahko sledi inštalacija WordPress-a in Zend Frameworka.

Priloga B: Poljubna WordPress zanka

```
...
query_posts('cat=5&orderby=rand');
$first = true;
?>
<h1><?php the_title();?></h1>

<div class="post first">
 <?php the_content();?>

 <p>Predloge  za GARMIN trase lahko vsi registrirani uporabniki pošljete na e-naslov: <a href="mailto:marketing@garmin.si">marketing@garmin.si</a>. Če bo trasa izbrana in dodana kot GARMIN trasa, bomo uporabnika izbrane trase nagradili s praktično nagrado Garmin.</p>

 <div class="legend">
   <h3>Legenda:</h3>
   <img src="/wp-content/themes/carrington/img/icon-bluedot.png" alt="" /> Lahka trasa<br />
   <img src="/wp-content/themes/carrington/img/icon-reddot.png" alt="" /> Srednje težka trasa<br />
   <img src="/wp-content/themes/carrington/img/icon-blackdot.png" alt="" /> Težka trasa
 </div>
</div>

<?php
$panels_open = 1;
$html_files = array();
$i = 0;
while (have_posts()) :
the_post();
       $razdalja = get_post_custom_values('razdalja');
       $visinska = get_post_custom_values('visinska');
       $zahtevnost = get_post_custom_values('zahtevnost');
       $disciplina = get_post_custom_values('disciplina');
       $file = get_post_custom_values('podatki');
```

```
      $permalink = get_permalink();
      $id = get_the_ID();
      $html_file = show_google_map($id);
      if($html_file !== false)
            $html_files[$id] = $html_file;
      ?>
<div class="post togglable">
   <a href="#" class="panel-link" onClick="loadGMap(<?= $id ?>, 'http://<?=
$_SERVER['HTTP_HOST'].'/'.$html_files[$id] ?>')"><span class="icon"></span><?php
the_title();?> <span class="floatright">Prikaži</span></a>
   <?php
     echo $panels_open-- > 0 ?
      '<div class="slide-panel" style="height:auto;overflow:hidden;padding-
top:20px;">' :
      '<div class="slide-panel" style="height:0;overflow:hidden;padding-top:0;">';

     echo '<iframe frameborder="no" scrolling="no" class="floatleft"
id="google_map_'.$id.'"';
     if($i++ <= $panels_open) {
      echo ' src="/'.$html_files[$id].'"';
     }
     echo '></iframe>';
   ?>

     <div class="map-info trase floatright">
       <table>
         <?php if(!empty($razdalja[0])) { ?>
         <tr class="first">
           <td class="cell-title">Razdalja</td>
           <td><?php echo $razdalja[0];?> km</td>
         </tr>
         <?php } if(!empty($visinska[0])) { ?>
         <tr>
           <td class="cell-title">Višinska razlika</td>
           <td><?php echo $visinska[0];?> m</td>
         </tr>
         <?php } if(!empty($zahtevnost[0])) { ?>
         <tr>
           <td class="cell-title">Zahtevnost</td>
           <td>
```

```
            <?php
              if($zahtevnost[0]=='1' || $zahtevnost[0]=='2') echo '<img src="/wp-
content/themes/carrington/img/icon-bluedot.png" alt="" />';
              elseif($zahtevnost[0]=='3' || $zahtevnost[0]=='4') echo '<img
src="/wp-content/themes/carrington/img/icon-reddot.png" alt="" />';
              if($zahtevnost[0]=='5') echo '<img src="/wp-
content/themes/carrington/img/icon-blackdot.png" alt="" />';
            ?>
          </td>
        </tr>
        <?php } ?>
        <tr>
          <td colspan="2">
            <?php
              if($disciplina[0]=='Tek') echo '<img src="/wp-
content/themes/carrington/img/icon-running.png" alt="" />';
              elseif($disciplina[0]=='Kolo') echo '<img src="/wp-
content/themes/carrington/img/icon-cycling.png" alt="" />';
              elseif($disciplina[0]=='Plavanje') echo '<img src="/wp-
content/themes/carrington/img/icon-swimming.png" alt="" />';
              elseif($disciplina[0]=='Tek na smučeh') echo '<img src="/wp-
content/themes/carrington/img/icon-cross_country_skiing.png" alt="" />';
            ?>
          </td>
        </tr>
      </table>
                  </div>
     <label class="btn-wrap floatright"><span><input type="submit" value="Prenesi
traso" onclick="location.href = '/download.php?file=<?=
$file[0]?>'"></span></label>
     <br /><br />
     <div class="trackDetailsLinkWp floatright"><a href="<?php echo
$permalink; ?>"><?php echo TRACK_DETAILS_TXT; ?></a></div>
     <div class="clear"></div>
   </div>
 </div>

<?php endwhile; ?>
...
```

Del kode, kot je npr. Javascript koda za odpiranje - zapiranje posameznih vsebinskih blokov oz. izpis teksta na vrhu strani je bil zaradi boljše preglednosti umaknjen.

Priloga C: Integracija Zend Frameworka z uporabo WP vtičnika

Ker smo Zend Framework integrirali že v več spletnih aplikacij, je bilo smiselno razviti vtičnik, ki poenostavi določene korake. Znotraj vtičnika lahko nastavimo več različnih parametrov (pot do Zend Framework aplikacije, knjižnic, itd.).

```
<?php
/*
Plugin Name: Zend Framework integration
Plugin URI: http://appoteka.com/zend-framework-integration
Description: Makes it easy to integrate Zend Framework application with WordPress
Author: Gregor Beslič
Version: 1.0
*/

/**
* Displays the admin forms and saves values
*/
function zfi_admin() {
	if (isset($_POST) && !empty($_POST['submit-paths']))
		zfi_save_options();

	$zfi_ops = get_option('zfi_settings');
	if($zfi_ops !== FALSE)
		$zfi_ops = unserialize($zfi_ops);
	else
		$zfi_ops = array();
	?>
	<div class="wrap">
		<h2>Zend Framework integration</h2>
		<p>Please set the correct path to your Zend Framework application
resources.</p>
		<form action="" method="post" id="zfi-paths-form">
			<h3><label for="path_project">Path to Zend Framework
project:</label></h3>
```

```
                  <p><input type="text" name="path_project" id="path_project"
value="<?php echo $zfi_ops['path_project'] ?>"/></p>

                  <h3><label for="path_libs">Libraries path:</label></h3>
                  <p><input type="text" name="path_libs" id="path_libs"
value="<?php echo $zfi_ops['path_libs'] ?>"/></p>

                  <h3><label for="path_application">Application
path:</label></h3>
                  <p><input type="text" name="path_application"
id="path_application" value="<?php echo $zfi_ops['path_application'] ?>"/></p>

                  <h3><label for="path_conf_file">Config file path (inside
application folder):</label></h3>
                  <p><input type="text" name="path_conf_file" id="path_conf_file"
value="<?php echo $zfi_ops['path_conf_file'] ?>"/></p>

                  <p class="submit"><input type="submit" name="submit-paths"
value="Update paths &raquo; " /></p>
                  <?php wp_nonce_field('zfi-update'); ?>
            </form>
      </div>
      <?php
}

/**
* Hook for adding the plugin into WordPress Admin
*/
function zfi_admin_page() {
      add_options_page('Zend Framework integration','Zend Framework integration',
'manage_options', 'zend-framework-integration','zfi_admin');
}
add_action('admin_menu', 'zfi_admin_page');

function zfi_save_options() {
      if (check_admin_referer('zfi-update')) {

            $zfi_ops = get_option('zfi_settings');
            if($zfi_ops === FALSE)
                  $zfi_ops = array();
```

```
		else
			$zfi_ops = unserialize($zfi_ops);

		if(isset($_POST['path_project']))
			$zfi_ops['path_project'] = $_POST['path_project'];

		if(isset($_POST['path_libs']))
			$zfi_ops['path_libs'] = $_POST['path_libs'];

		if(isset($_POST['path_application']))
			$zfi_ops['path_application'] = $_POST['path_application'];

		if(isset($_POST['path_conf_file']))
			$zfi_ops['path_conf_file'] = $_POST['path_conf_file'];

		update_option('zfi_settings', serialize($zfi_ops));
		echo '<div id="message" class="updated fade"><p>Options
Saved</p></div>';
	}
}


function init_zf() {
	// Zend Frameworks inits in public/index.php
	if(defined('ZF_LOAD') && ZF_LOAD)
		return;

	if($zfi_ops = get_option('zfi_settings')) {
		$zfi_ops = unserialize($zfi_ops);
		if(empty($zfi_ops['path_project']))
			return;
		if(is_admin())
			$zfi_ops['path_project'] = '../'.$zfi_ops['path_project'];
		else
			$zfi_ops['path_project'] =
$_SERVER['DOCUMENT_ROOT'].$zfi_ops['path_project'];

		$path_libs = empty($zfi_ops['path_libs']) ? 'library' :
$zfi_ops['path_libs'];
		set_include_path('.' . PATH_SEPARATOR .
```

```
$zfi_ops['path_project'].'/'.$path_libs . PATH_SEPARATOR . get_include_path());

            $path_application = empty($zfi_ops['path_application']) ?
'application' : $zfi_ops['path_application'];
            define('APPLICATION_PATH',
$zfi_ops['path_project'].'/'.$path_application);

            require_once 'Zend/Loader/Autoloader.php';
            Zend_Loader_Autoloader::getInstance();

            $path_conf_file = empty($zfi_ops['path_conf_file']) ?
'configs/application.ini' : $zfi_ops['path_conf_file'];
            $application = new Zend_Application('production', APPLICATION_PATH .
'/' . $path_conf_file);
            $application->bootstrap();

            Zend_Session::start();
      }
}

add_action('init','init_zf');
```