UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Rok Carl

Spletna aplikacija za komuniciranje med profesionalnim fotografom in stranko

DIPLOMSKO DELO NA UNIVERZITETNEM ŠTUDIJU

Mentor: prof. Saša Divjak, PhD
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: 01722/2010
Datum: 01.12.2010

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

Kandidat: **ROK CARL**

Naslov: **SPLETNA APLIKACIJA ZA KOMUNICIRANJE MED PROFESIONALNIM FOTOGRAFOM IN STRANKO**

A WEB APPLICATION FOR COMMUNICATION BETWEEN PROFESSIONAL PHOTORGRAPHER AND CUSTOMER

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Razvijte spletno aplikacijo, ki naj omogoča komunikacijo med profesionalnim fotografom in strankami. Analizirajte morebitne sorodne rešitve, Pri samem razvoju uporabite tehnologije, kot so PHP, MySQL, Javascript in CSS. Uporabite primerno PHP ogrodje, ki naj omogoči hiter razvoj in spremembe v sami aplikaciji.

Mentor:

prof. dr. Saša Divjak

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:

prof. dr. Nikolaj Zimic

I Z J A V A O A V T O R S T V U

diplomskega dela

Spodaj podpisani/-a **Rok Carl**,

z vpisno številko 63020017,

sem avtor/-ica diplomskega dela z naslovom:

Spletna aplikacija za komuniciranje med profesionalnim fotografom in stranko

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal/-a samostojno pod mentorstvom

  Prof. PhD Saša Divjak

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

- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne 16.5.2011 Podpis avtorja:

Zahvala

Na tem mestu bi se rad zahvalil vsem, ki so mi stali ob strani in pomagali pri pisanju diplome in snovanju aplikacije.

Kazalo

Povzetek

Trenutna informacijska podpora profesionalnih fotografov na trgu je nepopolna. V tem diplomskem delu želimo zapolniti eno luknjo v tej podpori: komunikacijo med fotografom in stranko.

Aplikacija, ki je končni rezultat, omogoča fotografu objavo fotografij projekta na spletu, kjer si lahko stranka v miru ogleda fotografije in izbere najboljše, ki jih bo uporabila v svoji naslednji reklamni kampanji, knjigi, prospektu ali čem podobnem. Fotografijam lahko doda komentarje, npr. navodila fotografu za obdelavo. Ko stranka konča z izborom, lahko fotograf uvozi izbor v aplikacijo za izbor fotografij, obdela fotografije ter zaključi projekt.

Spletno aplikacijo smo naredili skoraj popolnoma od začetka s tehnologijami PHP, MySQL, Javascript, CSS, itn. Uporabili smo lastno PHP ogrodje, ki nam je omogočalo hiter razvoj in spremembe v aplikaciji.

Aplikacija je nastala na pobudo fotografa in je v uporabi že nekaj mesecev. Ker podobna rešitev na trgu še ne obstaja, si želimo razširiti uporabo aplikacije preko raznih predstavitev interesiranim fotografom.

Ključne besede

PHP, MySQL, fotografija, spletna aplikacija

Abstract

Current information support on the market for professional photographers is incomplete. We try to fill this void: the communication between a photographer and a client.

The final result, the application, allows the photographer to publish his photographs on the web, where the client can look at them and make a selection of the best photos, which she will use in her marketing campaign, book, brochure or something similar. She can add comments to photos, e.g. instructions for the photographer for retouching. When the client finishes choosing the photos, the photographer can import the selection into an application, where he can retouch photos and finish the project.

We made a web application almost from scratch with the following technologies: PHP, MySQL, Javascript, CSS, etc. We used our own PHP framework, which allows as fast development and changes in the application.

The application was created for one photographer and has been in use for a few months. A similar solution is nonexistent on the market so we wish to expand the use of this application through various presentations to interested photographers.

Keywords

PHP, MySQL, photography, web application

1 Uvod

Rezultat diplomskega dela je spletna aplikacija, ki pomaga pri komunikaciji med fotografom in njegovo stranko; bolj točno pri izbiri fotografij, ki jih naj fotograf uporabi v končnem izdelku, npr. katalogu.

Najprej bomo spoznali tipični potek dela fotografa na projektu, potem pa raziskali problem v tem poteku in kako ga aplikacija rešuje.

1.1 Tipični potek dela fotografa

Vzemimo za primer, da se je fotograf dogovoril s stranko, ki prodaja ročne izdelke, za izdelavo kataloga, ki v najlepši luči prikaže izdelke. Potek dela bi izgledal nekako tako: fotograf pride na dogovorjeno lokacijo (npr. galerijo) z vso fotografsko opremo, ki jo potrebuje. S stranko postavita sceno, kjer bo fotograf slikal izdelke. Naredi nekaj sto fotografij, nato pa se odpravi domov, kjer jih naloži v svoj najljubši program za hranjenje in obdelavo fotografij.

Ko ima fotografije v svojem programu, najprej pobriše vse neprimerne fotografije (slaba osvetlitev ali kadriranje itd.). Nato se loti osnovne obdelave osvetlitve, ostrine, kadra, beline in drugih lastnosti.

Sedaj je fotografu ostalo mnogo fotografij, potrebuje pa jih le malo, morda nekaj deset fotografij. Včasih stranka ali fotograf želi, da je stranka del procesa izbire končnih fotografij. Tako pride do dileme: kako naj izmed ostalih fotografij stranka izbere najboljše in kako se naj ta izbor znajde v fotografovem programu?

Rešitve, ki obstajajo, so nerodne. Fotograf lahko pošlje fotografije po navadni pošti na DVD-ju, stranka pa mu sporoči imena izbranih fotografij. Lahko jih pošlje po e-pošti, če jih ni preveč za poštne strežnike, ki navadno omejujejo velikost sporočila. Po tem procesu pa jih mora fotograf še označiti v svojem programu, kar spet vzame čas.

Rešitev, ki jo ponuja aplikacija, ki je tema te diplomske naloge, je, da fotograf najprej naloži fotografije v aplikacijo, stranki pošlje povezavo, stranka izbere fotografije, fotograf pa iz aplikacije dobi skripto, ki jo požene in mu s tem označi fotografije v njegovem programu.

2 Življenjski cikel aplikacije frame.si

Razvoj aplikacije poteka (z daljšimi premori) že več kot eno leto, bila pa je v uporabi že prvi mesec, vendar v malce drugačni podobi. Razvoj ni potekal preveč premišljeno, ampak organsko – aplikacija je rasla in se spreminjala s potrebami glavnega uporabnika.

2.1 Želje in potrebe fotografa

Aplikacija je nastala na željo fotografa, ki se s fotografijo ukvarja že vrsto let, zadnji dve leti pa tudi profesionalno. Želja po aplikaciji se je rodila v njegovi glavi iz potrebe po rešitvi konkretnega problema – želel je lažjo komunikacijo s strankami pri izbiri fotografij za končni izdelek.

2.2 Rojstvo aplikacije

Po pogovoru s fotografom smo začeli z delom. Prvotne želje so bile (glede na končno različico) preproste:

- imamo samo enega fotografa, ki uporablja aplikacijo,
- dovolj je, če imamo samo en aktiven fotografski projekt in
- stvar lahko teče na fotografovem računalniku.

To je bila seveda *ad hoc* rešitev, kjer ni bilo veliko načrtovanja popolne aplikacije, ampak je bil moto bolj v smislu "samo da bo delalo, pa čimprej".

2.3 Prva različica

Že prva različica je bila sestavljena iz dveh ključnih modulov: AppleScript-a in spletne aplikacije. Smo verzirani PHP programerji, vendar AppleScript-a nismo poznali. Neznan nam je bil tudi Apple-ov operacijski sistem os xOS X.

Na srečo pa je AppleScript preprost skriptni jezik, namenjen upravljanju programov in njihovih notranjih podatkovnih struktur (le tistih, ki jih program objavi). Ta njegova preprostost je omogočila hitro rešitev in nas hkrati frustrirala, saj se ne drži že znanih in preverjenih paradigm iz drugih programskih in skriptnih jezikov, kar otežuje programiranje, še posebej, ko kaj ne dela. Kljub temu pa je v danih okoliščinah bil dobra izbira.

Tako je bila narejena skripta, ki se je povezala z Aperture-jem (izbranim fotografskim programom za urejanje fotografij), vprašala fotografa, kateri album naj izvozi, nato pa pognala izvoz fotografij na lokacijo na fotografovem disku, kjer je tekla spletna aplikacija. Ta pa je potem naredila pomanjšane slike, prikazala stranki spletno stran, preko katere je lahko izbrala fotografije za končni izdelek. Ko je končala z izbiro, je spletna aplikacija pognala drugo AppleScript skripto, ki je pregledala označene fotografije in v programu Aperture označila fotografije, ki jih je stranka izbrala.

2.4 Zaplet

Prva različica je sicer delovala super, ampak kot vedno, ima uporabnik dodatne želje, kaj vse bi še lahko aplikacija znala. Nekaj lastnosti je bilo treba nadgraditi.

Aplikacija je tekla na fotografovem računalniku. To je pomenilo, da je moral računalnik biti prižgan med uporabo (tudi čez noč – za vsak slučaj), hkrati pa je bila administracija sistema na nepravem mestu: fotograf namreč ni sistemski administrator in kadar se npr. strežnik sesuje, moramo potem mi reševati težave, kar pomeni fizični obisk pri fotografu. Boljša rešitev je strežnik na spletu, ki ga upravljajo strokovnjaki in je dostopen 99,9% časa.

S staro omejitvijo enega projekta se sicer da živeti, ni pa idealna. Želeli bi imeti nekakšen arhiv vseh preteklih projektov. S tem bi tudi omogočili strankam ogled slik kasneje, ko smo projekt že zdavnaj zaključili. Pogosto pa se tudi zgodi, da imamo naenkrat aktivna dva ali več projektov, zato je bilo potrebno to omejitev odstraniti.

2.5 Druga različica

Ko je padla odločitev po nadgradnji, nam je bilo nekaj kristalno jasno: potrebno bo preprogramiranje nekaterih ključnih delov aplikacije. Shema baze namreč v prvi različici ni bila dobro dodelana, baza pa je bila v datoteki na disku v SQLite bazi. Zakaj tega nismo naredili že na začetku? Odgovor je, da bi bilo z danimi zahtevami (v tistem času) nesmiselno zapravljati čas na nepotrebni zahtevnejši arhitekturi ali če pouzamemo z eno angleško besedo: *overkill!*

Tako smo se spravili na delo bolj neprobojne arhitekture: za podatkovno bazo smo uporabili MySQL, kot bazo kode pa ogrodje, ki smo ga gradili in izpopolnjevali že tri leta, ter ga uporabili v projektih v produkciji – oboje tako bolj zreli in primerni rešitvi.

Pri tej različici je tudi ostalo: dovolj je robustna, da je preživela test časa in nam omogoča dodajanje funkcionalnosti brez t.i. *hack-ov* ali *workaround-ov.*

Za rešitev drugega problema (“naj aplikacija ne teče na lokalnem računalniku”) pa smo uporabili zasebni virtualni strežnik pri ponudniku Dreamhost, na katerem teče aplikacija. To dodatno omogoča še redundančne podatke, saj se hranijo urne, dnevne in tedenske varnostne kopije podatkov – v našem primeru kopije fotografij in kode aplikacije.

2.6 Žrtev druge različice

Na žalost pa je bilo potrebno pri prenovi žrtvovati eno lastnost prve različice: v prvi so se fotografu takoj označile vse fotografije v programu Aperture, ko je stranka izbrala željene fotografije. To je bilo mogoče, ker je spletna aplikacija tekla na fotografovem računalniku in je tako imela dostop do celotnega sistema, kjer je lahko izvajala skripte za označevanje.

Sedaj, ko aplikacija teče na oddaljenem strežniku, pa ta strežnik seveda ne more direktno poganjati ukazov na fotografovem računalniku. Tako je bilo potrebno uvesti še en korak: ko stranka izbere fotografije, sname fotograf s spletne aplikacije skripto, ki jo požene lokalno in mu označi izbrane fotografije.

To lastnost je bilo vredno žrtvovati za ostale izboljšave.

3 Obstoječe rešitve

Aperture je program za obdelovanje in hrambo fotografij in je zelo popularen med profesionalnimi fotografi, ki uporabljajo Apple-ove računalnike. Odločili smo se narediti aplikacijo za ta profil fotografov, tako da je tudi nabor obstoječih rešitev omejen.

Na trgu presenetljivo niso obstajale prave oz. zadovoljive rešitve. Še tista kakšna redka, ki jo je bilo moč najti, ni povsem zadovoljila potreb.

3.1 Publish for Approval

Edina namenska rešitev na tržišču je Publish for Approval [2]. To je vtičnik za Aperture, ki omogoča, da fotograf slike albuma preko tega vtičnika izvozi nekam na disk, kjer teče spletni strežnik. Ta potem vsebuje spletno aplikacijo, ki nato teče na fotografovem računalniku. Tja se stranka tudi poveže, izbere fotografije, te pa postanejo označene v Aperture-ju.

Tako je to delovalo. Nato je februarja 2010 izšla tretja verzija Aperture-ja, na kateri ta vtičnik ni več deloval. Razni poskusi kontaktiranja avtorja vtičnika, da bi ga popravil, niso obrodili sadov. Tudi stran je že praktično zamrla (http://automator.us/aperture/publish/), saj deluje le še osnovna stran, vse povezave pa ne delujejo več.

Tako ta možnost ni bila več uporabna.

3.2 Elektronska pošta

Ker namenske rešitve ni, se fotografi pogosto zatečejo k svojim rešitvam problema. Eden je uporaba elektronske pošte. Fotograf najprej izvozi vse fotografije na svoj trdi disk v JPEG formatu, pri čemer močno oklesti velikost fotografij in kvaliteto. Dandanes imajo fotografi slike v velikosti dvanajst do dvajset milijonov pik, kar v dobri JPEG kvaliteti pomeni okrog štiri do osem megabajtov prostora. Če pošljemo stranki nekaj sto fotografij, znese velikost paketa nekaj gigabajtov. Praktično vsi ponudniki elektronske pošte imajo omejeno velikost priponke, Google-ov Gmail ima dvajset megabajtov. To pri dvesto fotografijah pomeni, da je vsaka lahko velika le sto kilobajtov, kar je zelo malo, če želimo kvaliteto.

Poleg tega pa mora stranka ročno vpisovati izbrane fotografije nazaj v email, npr. »izbrali smo fotografije sha_001.jpg, sha_002.jpg, sha_013.jpg, ...« Za tem mora pa še fotograf za vsako napisano fotografijo najti pravo v svojem programu in jo tam označiti. To je dolgotrajen in neprijeten postopek, tako da je ta rešitev daleč od idealne.

3.3 Facebook

Fotografije lahko naložimo tudi na naš Facebook profil in določimo, da ima do teh fotografij dostop le naša stranka. V tem primeru mora tudi stranka imeti Facebook profil, česar pa morda ne bi želeli zahtevati od stranke. Na naloženem albumu lahko stranka izbere »všeč mi je« na izbranih slikah.

Podobno kot pri elektronski pošti je tudi tukaj veliko dela in fotografije so nizke kvalitete.

3.4 Druge spletne storitve

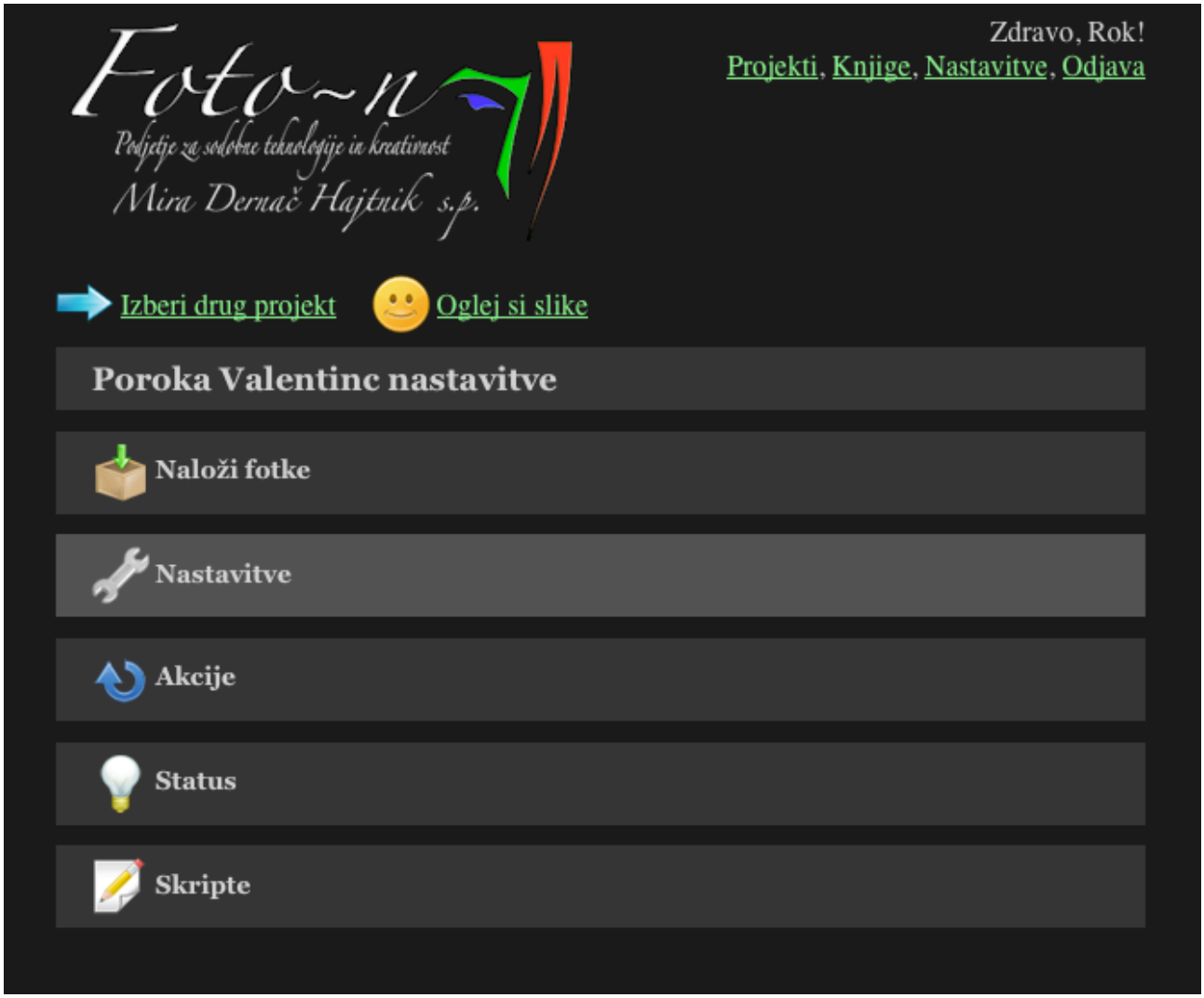
Obstaja mnogo strani, na katerih se da fotografije objaviti (Flickr, Photobucket, ...), vendar

vsem manjka pohitren prenos izbora v fotografov program, nekaterim tudi izbor fotografij s strani stranke. Tako nismo našli primerne rešitve, ki bi lahko nadomestila našo.

4 Delovanje aplikacije

4.1 Administracijski del

4.1.1 Urejanje projekta

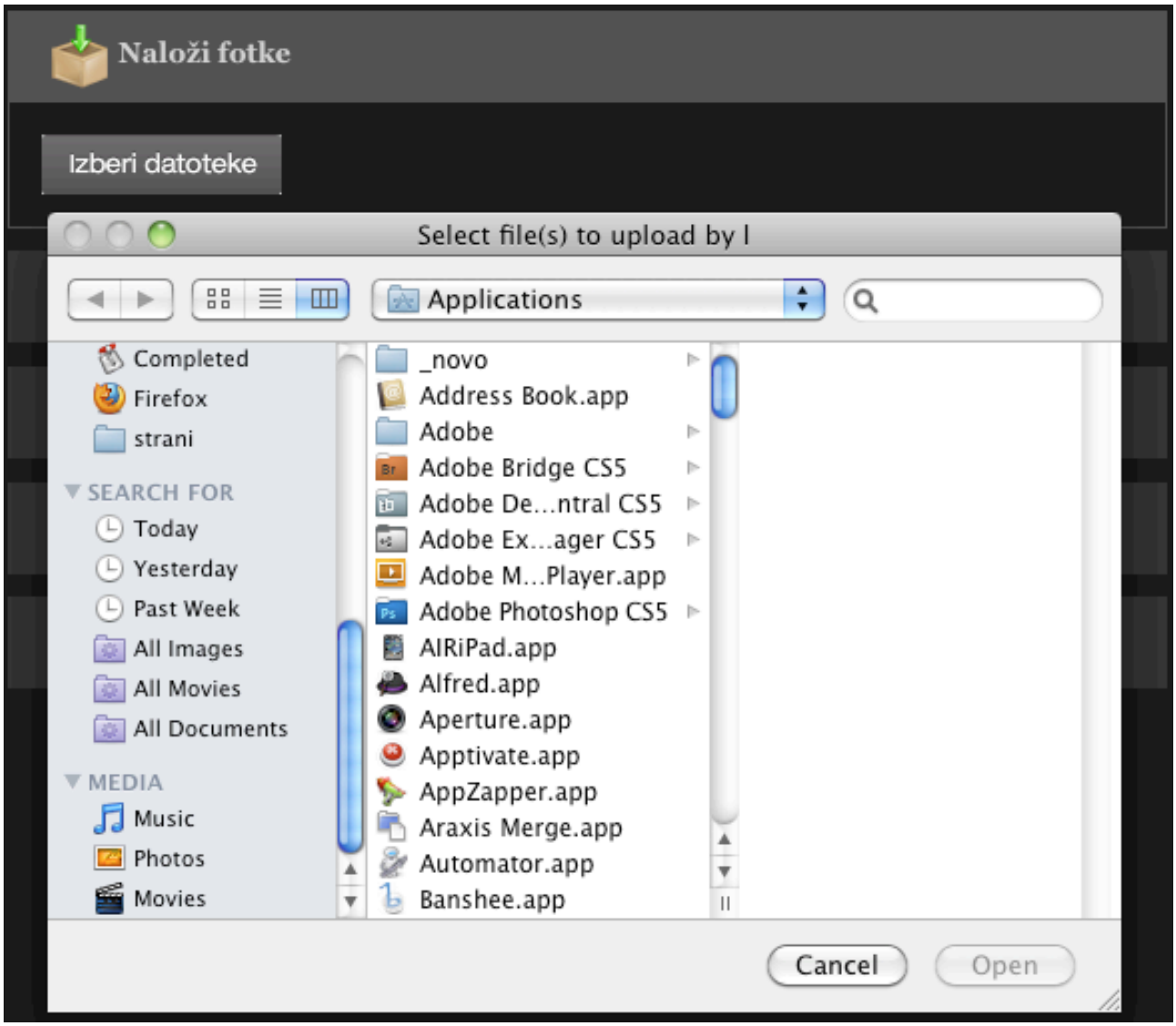


Slika 1 - administracijski del aplikacije

Administracijski del je sestavljen iz dveh delov. V glavi imamo logotip fotografa, ki ga lahko sam naloži (spremeni), in povezave do seznama projektov, foto knjig, nastavitev aplikacije ter odjave.

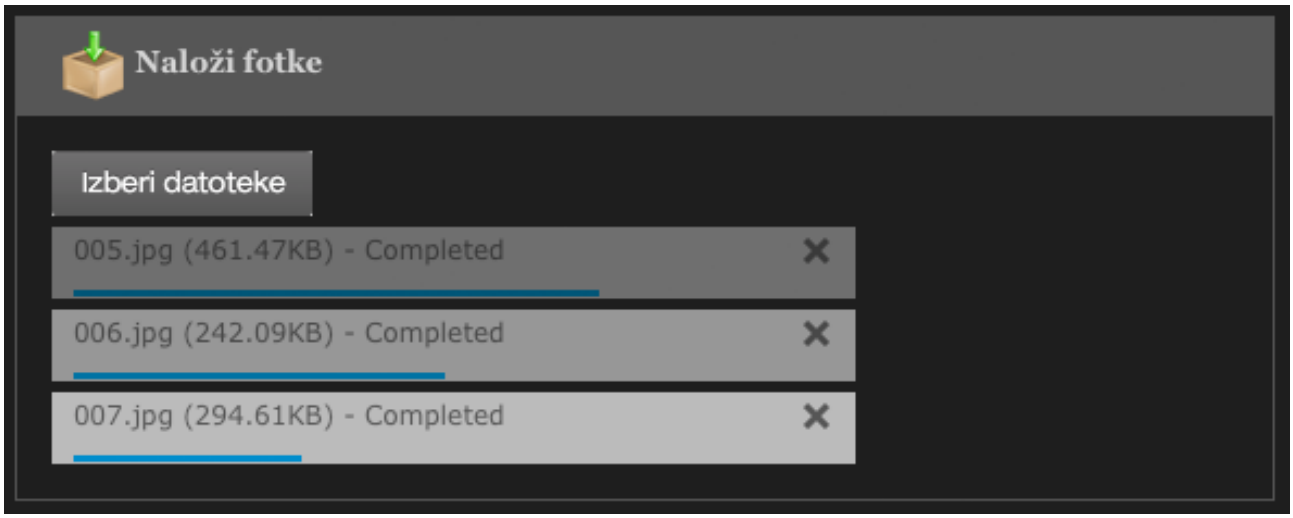
V glavnem delu strani imamo povezavi do seznama vseh projektov (ponovno) in do ogleda fotografij. Za tem sledi vsebinski del urejanja projekta, ki je sestavljen iz petih delov. Če kliknemo na naslov katerega od teh delov, se nam odkrije njegova vsebina. Pa poglejmo vsakega posebej.

4.1.1.1 Nalaganje fotografij



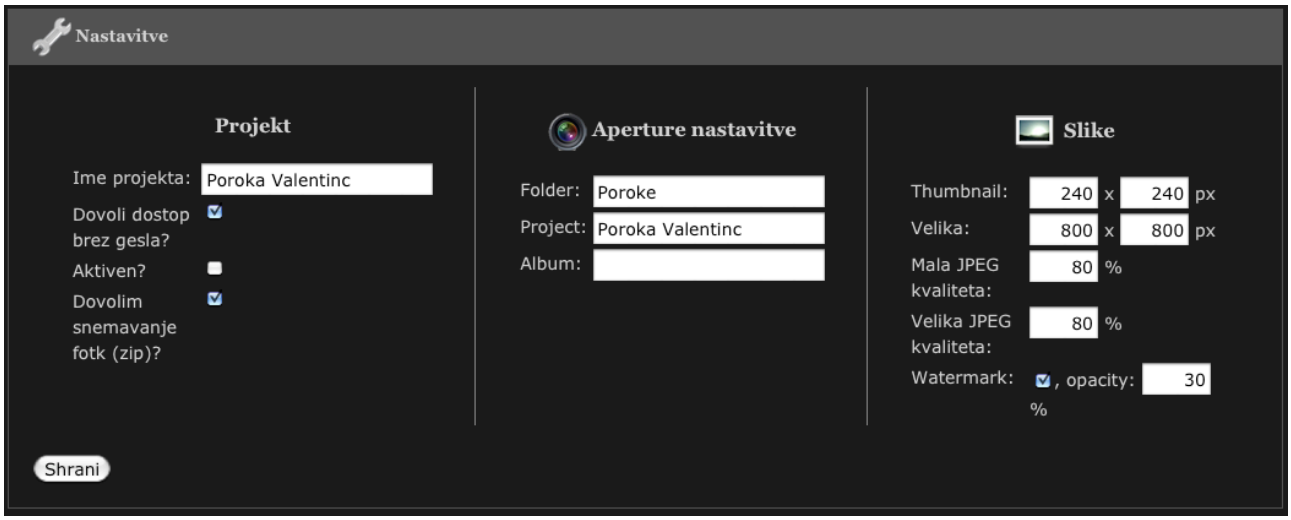
Slika 2 - nalaganje fotografij

V delu nalaganja fotografij imamo gumb z imenom Izberi datoteke. Za tem gumbom se skriva Flash skripta, ki omogoča hkratno nalaganje več fotografij naenkrat, kar je sicer izredno težko v HTML-ju (no, v zadnjem času so se začele pojavljati ajax rešitve, ki to počnejo brez Flash-a). Ko izberemo fotografije, se začnejo vse hkrati nalagati na strežnik. S tem fotograf doda fotografije projektu.



Slika 3 - proces nalaganja fotografij na strežnik

4.1.1.2 Nastavitve projekta

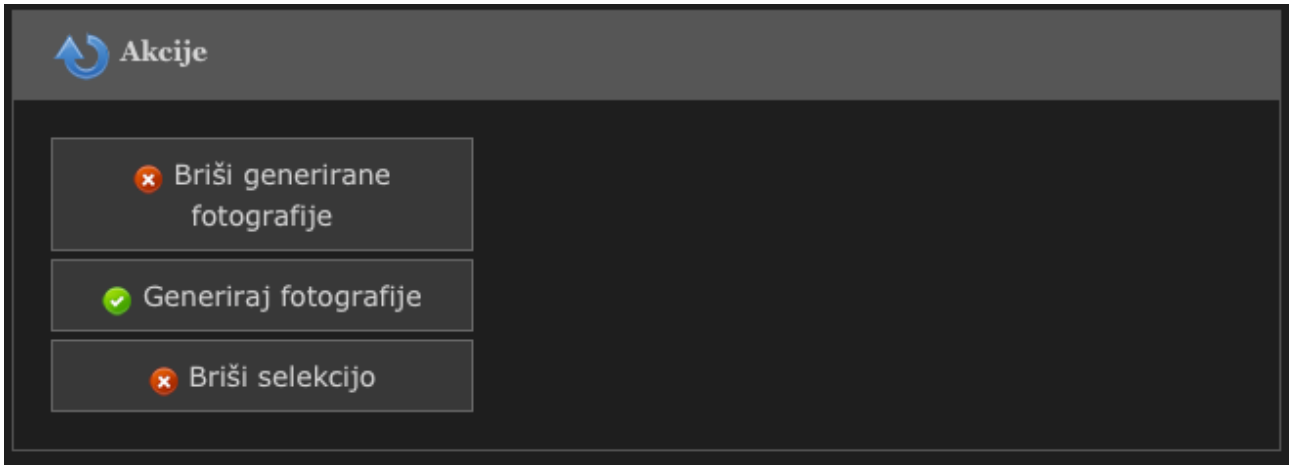


Slika 4 - nastavitve projekta

V delu nastavitve projekta lahko pregledujemo in spreminjamo nastavitve za projekt. Urejamo lahko nastavitve za:

- Projekt - ime projekta, ali dovolimo dostop do izbire fotografij brez gesla, ali je projekt še aktiven (če dovolimo označevanje) in ali dovolimo uporabnikom snemavanje vseh fotografij v Zip formatu.
- Aperture – nastavimo lahko ime mape, projekta in albuma v Aperture-ju, kjer želimo, da so fotografije označene.
- Slike – nastavimo lahko velikost in kvaliteto velikih ter malih fotografij in ali velike fotografije uporabljajo vodni tisk ter prozornost tega tiska.

4.1.1.3 Akcije projekta



Slika 5 - možne akcije projekta

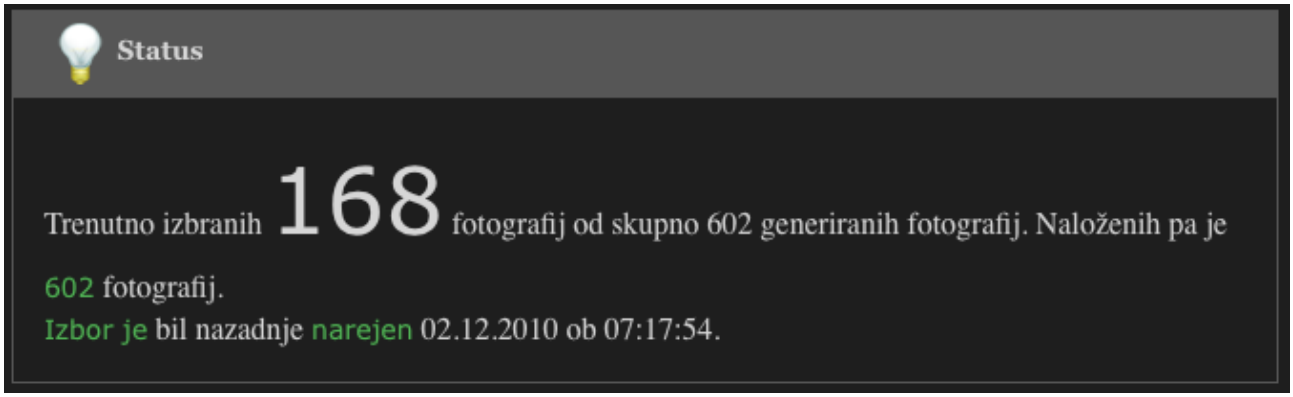
Imamo tri akcije, ki jih lahko izvedemo nad projektom:

Briši generirane fotografije – s tem brišemo vse generirane fotografije (velike in majhne). To je uporabno takrat, kadar spremenimo velikosti velikih ali majhnih fotografij in bi želeli, da se ponovno naredijo v pravi velikosti.

Generiraj fotografije – začne generirati pomanjšane fotografije. To nam pride prav, ko naložimo fotografije na strežnik in bi želeli, da se takoj generirajo manjše fotografije.

Briši selekcijo – s tem lahko fotograf briše selekcijo fotografij, ki je bila narejena na strani ogleda projekta, tako da lahko uporabnik začne znova.

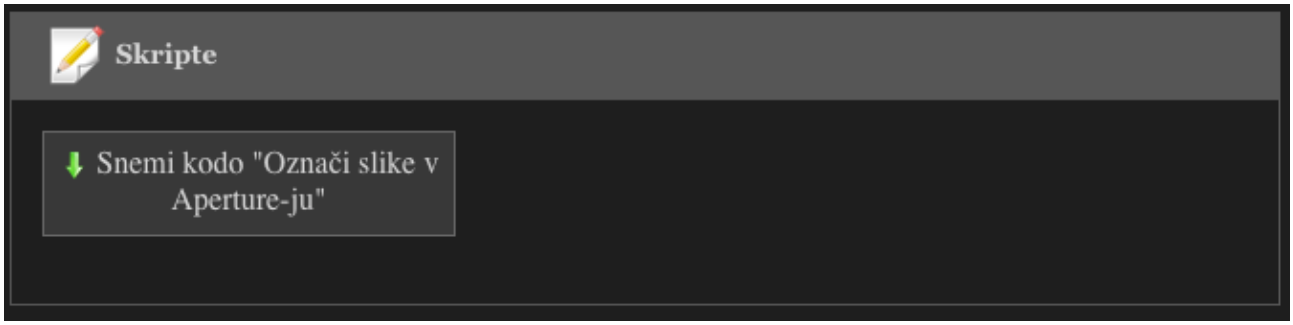
4.1.1.4 Stanje projekta



Slika 6 - stanje projekta

V tem delu vidimo trenutno stanje projekta. Vidimo lahko, koliko fotografij imamo, ali je izbor že narejen in koliko fotografij ima ter kdaj je bil izbor nazadnje narejen.

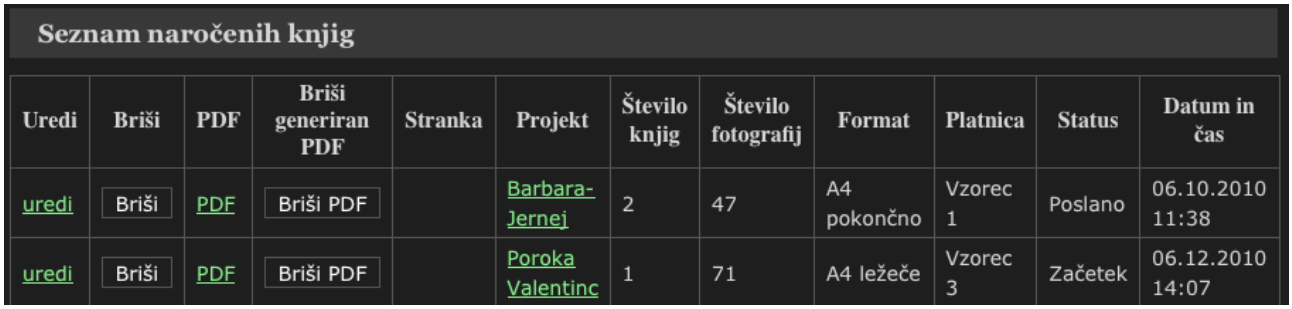
4.1.1.5 Skripte



Slika 7 - skripte

Pod skriptami najdemo skripto, s katero, če jo fotograf požene na svojem računalniku, označi izbrane fotografije v Aperture programu.

4.1.2 Pregled naročenih knjig

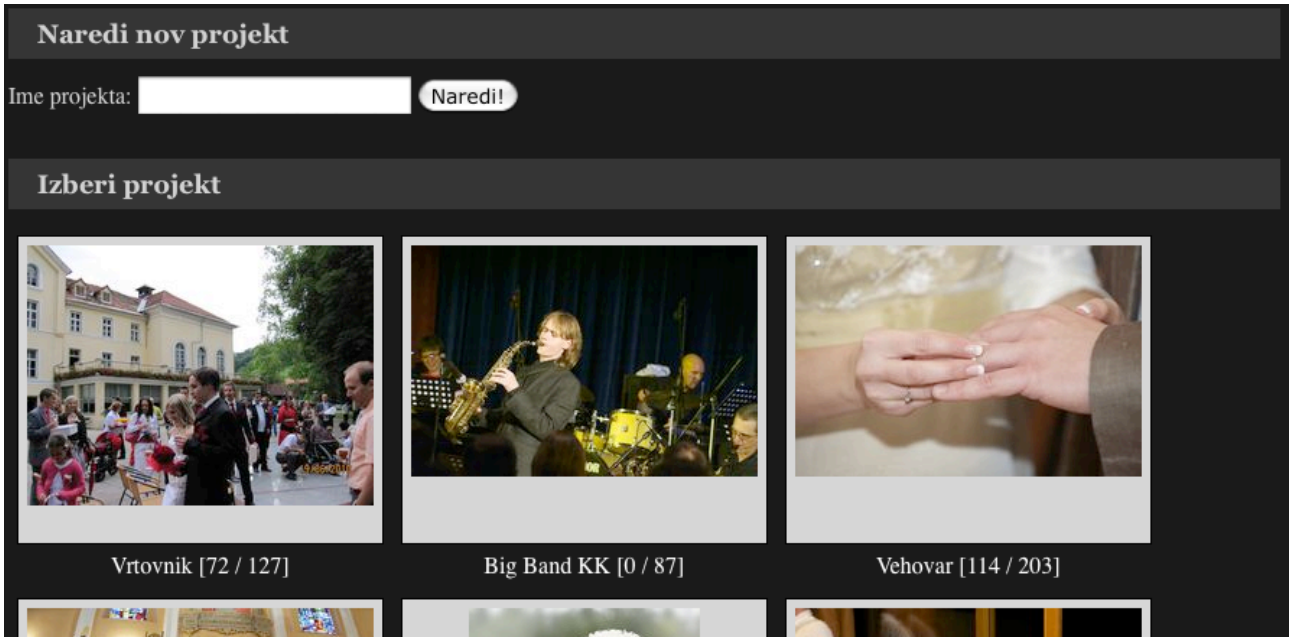


Uredi	Briši	PDF	Briši generiran PDF	Stranka	Projekt	Število knjig	Število fotografij	Format	Platnica	Status	Datum in čas
uredi	Briši	PDF	Briši PDF		Barbara-Jernej	2	47	A4 pokončno	Vzorec 1	Poslano	06.10.2010 11:38
uredi	Briši	PDF	Briši PDF		Poroka Valentinc	1	71	A4 ležeče	Vzorec 3	Začetek	06.12.2010 14:07

Slika 8 - seznam naročenih knjig

Na tej strani lahko pregledujemo vse naročene knjige. Vidimo lahko podatke o številu knjig, številu izbranih fotografij, formatu itd. Glavna stvar na tej strani je izvoz knjige v PDF formatu. Tako lahko fotograf izvoženo knjigo le še pošlje v tiskarno.

4.1.3 Izbira projekta

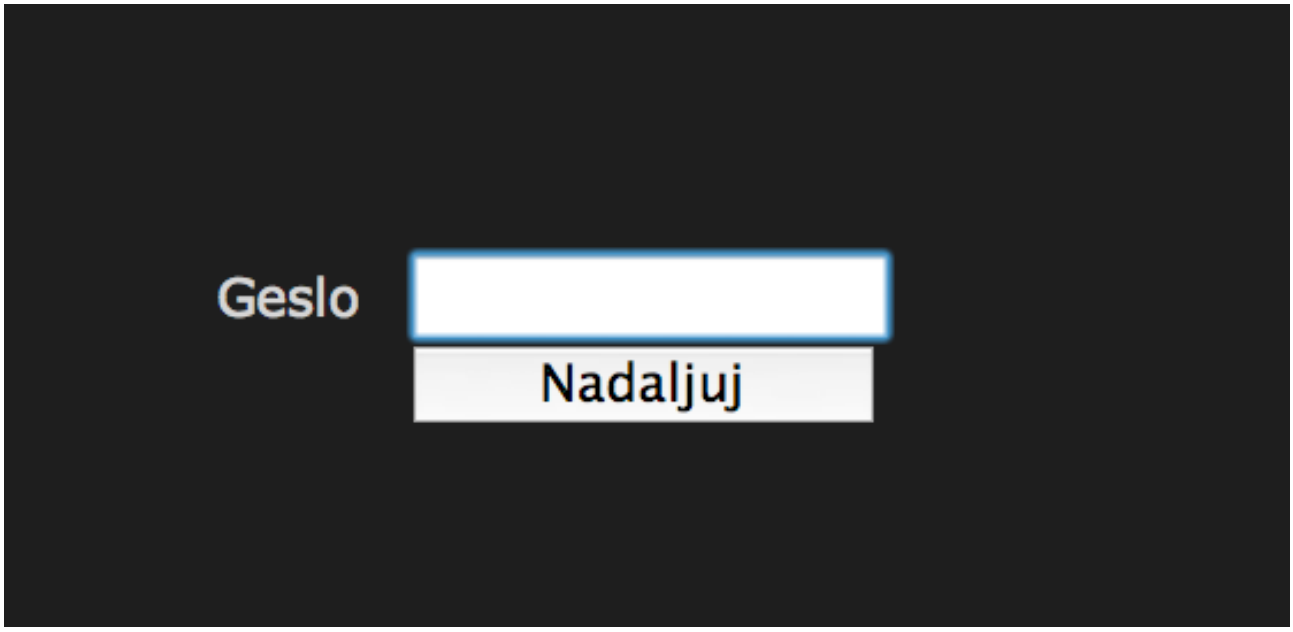


Slika 9 - izbira projekta

Na tej strani ima fotograf možnost dodajanja novega projekta in pregleda vseh projektov. Prikaže se seznam vseh projektov z naslovno sliko, tako da ga lažje identificira. Pod sliko se nahaja ime projekta, kakor tudi število izbranih fotografij ter število vseh naloženih fotografij. Klik na sliko ali ime popelje fotografa na stran urejanja projekta.

4.2 Uporabniški del

4.2.1 Vpis gesla za dostop do projekta

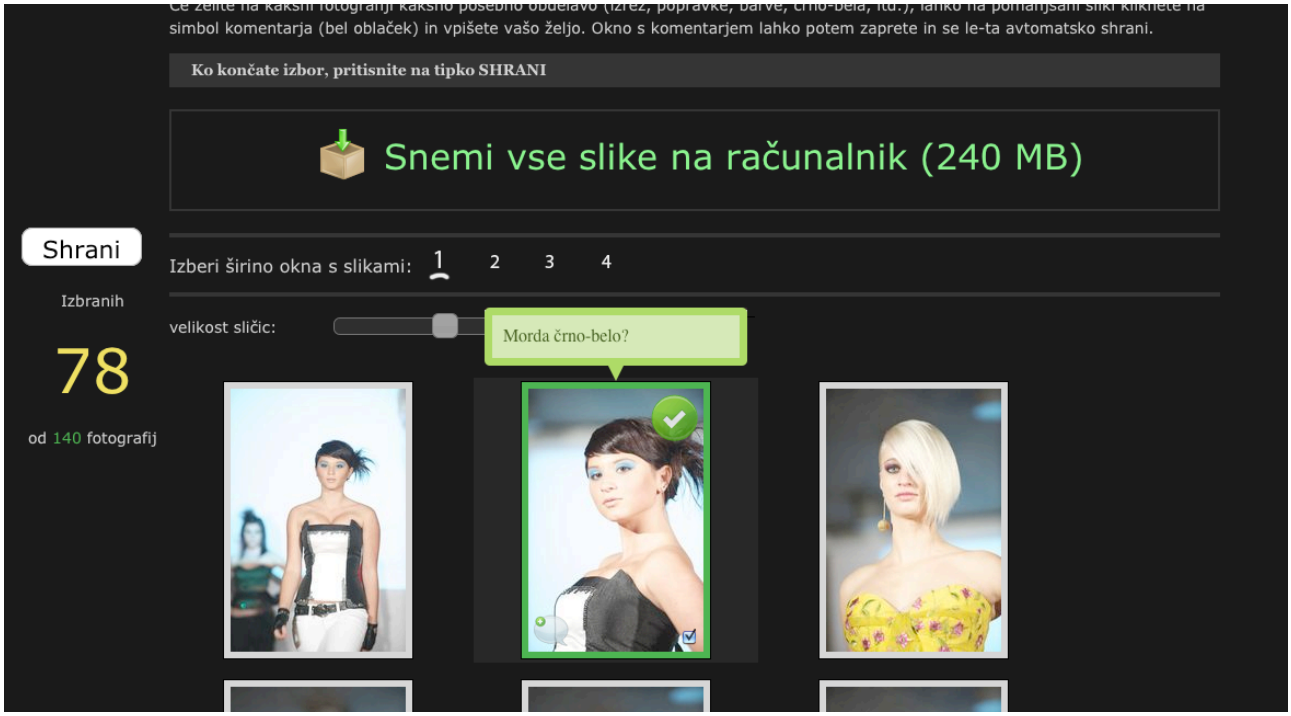


Slika 10 - prijava v aplikacijo

Če v naslovno vrstico brskalnika vpišemo le URL aplikacije (brez izbranega projekta) ali

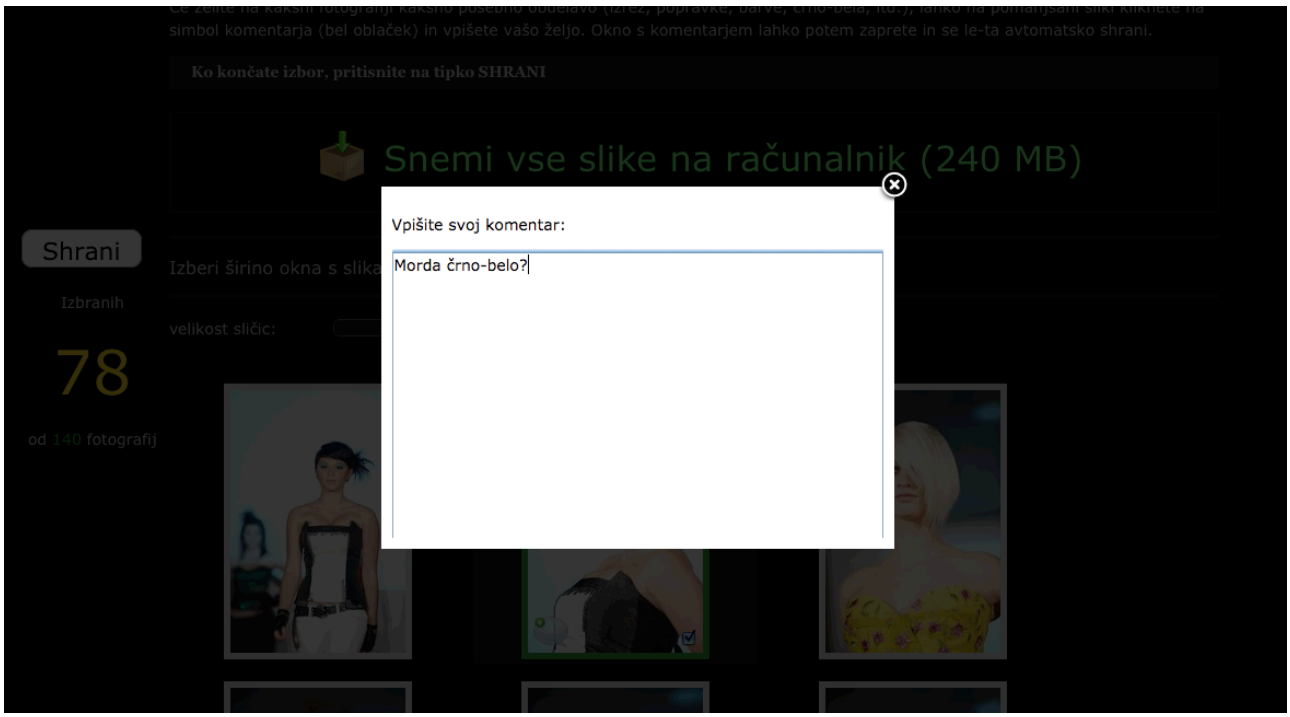
URL do projekta, ki zahteva geslo za dostop, nas pričaka vnosno polje, kamor lahko vpišemo geslo za dostop. Vsak projekt, ki zahteva geslo, ima unikatno geslo. Če je vpisano geslo pravilno, lahko preko gesla aplikacija ugotovi, kateri projekt je uporabnik želel videti in ga preusmeri na izbrani projekt.

4.2.2 Ogled projekta in izbira fotografij



Slika 11 - ogled projekta in izbira fotografij

Stran ogleda projekta najprej postreže z nekaj teksta, ki uporabniku razloži delovanje strani. Glavni del je seznam vseh fotografij projekta. Če se z miškinim kazalcem postavimo nad fotografijo, se nam prikaže komentar fotografije (če ta obstaja) ter dve dodatni kontroli. Prva je bel oblaček, ki ima zelen plus, če ima fotografija že dodan kak komentar. Če kliknemo nanj, se nam prikaže vnosno polje za vnos komentarja.



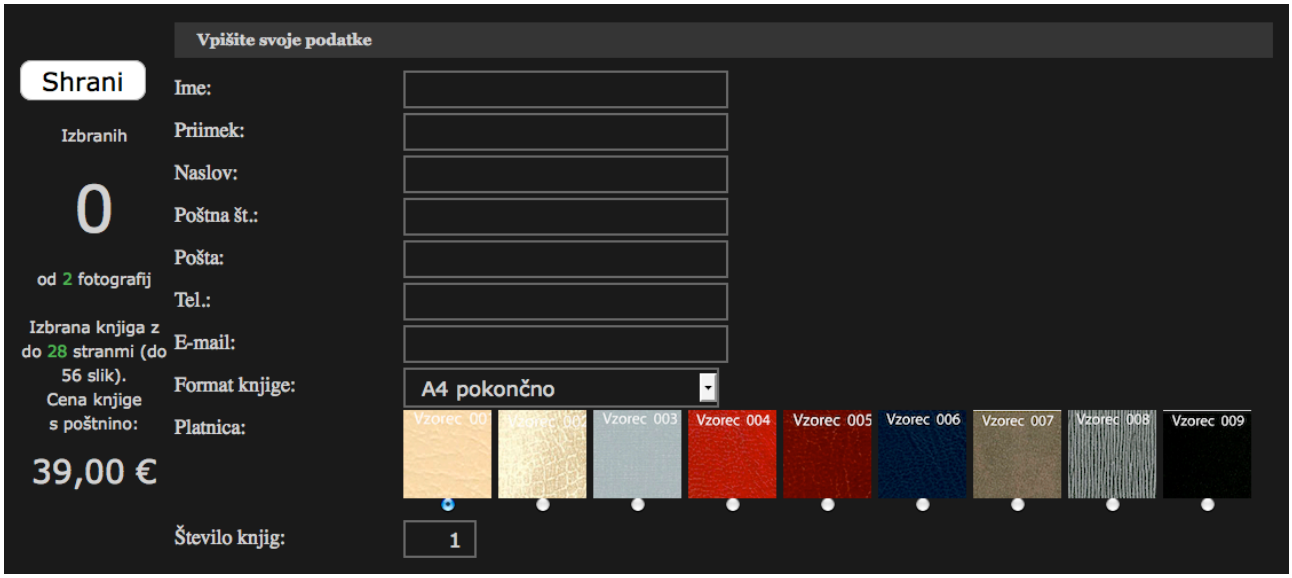
Slika 12 - urejanje komentarja

Druga kontrola pa je izbirno polje, ki ima kljukico, če je fotografija že izbrana. Če kliknemo nanjo, s tem izberemo fotografijo. Namesto klika na izbirno polje lahko pritisnemo tipko a. Ko izberemo fotografijo, se na njej nariše zelena kljukica, ki ponazarja izbrano fotografijo.

Na levi strani ekrana vedno vidimo število izbranih fotografij ter gumb *Shrani*, na katerega kliknemo, ko smo zadovoljni z izborom.

4.2.2.1 Knjiga

Izbor fotografij za foto knjigo je za uporabnika povsem enak kot navaden izbor, le da ima uporabnik pred seznamom fotografij vnosni obrazec, kjer izpolni svoje podatke in parametre knjige, ki jo želi.



Slika 13 - nastavitve knjige

5 Opis strežnika in programskega sklada

5.1 Strežnik

Strežnik, na katerem teče produkcijska različica aplikacije, imamo zakupljen pri Dreamhost-u, pri katerem se je zgodba zadovoljstva začela že pred približno petimi leti. Imeli so kar nekaj lastnosti, ki so nam izboljšale vsakdan in jih je bilo pri drugih ponudnikih težko dobiti.

Naj opišemo te lastnosti, saj so druge precej standardne.

5.1.1 Vse neomejeno

Dreamhost ponuja skoraj vse možnosti, ki so pri drugih ponudnikih navadno omejene, v neomejenih količinah. To seveda ne pomeni neskončno, saj so računalniški resursi količinsko omejeni. Pomeni pa, da v praksi ne boste nikdar dosegli meje, če se boste le držali ne preveč omejevalnih pogojev uporabe (kot npr. da prostora ne uporabljate za zasebne varnostne kopije – recimo kopije vaših slik z dopusta).

Neomejena ponudba:

- količina prostora na disku,
- mesečni prenos (količina podatkov, ki se vsak mesec pretoči z naših spletnih strani preko HTML dokumentov, slik, videov itd.),
- gostujočih domen in poddomen,
- uporabnikov in email računov,
- MySQL podatkovnih baz.

Zelo malo je ponudnikov, ki dandanes ponujajo vse te lastnosti v neomejenih količinah. Leta 2005 pa je bil kontrast še večji. Takrat sicer Dreamhost ni ponujal neomejeno količino prostora in prenosa, pa vendar neko noro količino – 500 GB prostora in 1 TB prenosa, ostalo pa neomejeno, medtem ko so drugi ponudniki močno omejevali vsako od omenjenih lastnosti v upanju, da bodo zaslužili na strankah, ki bodo želele več od zelo omejene ponudbe.

Dreamhost je ubral drugo pot: služijo na količini uporabnikov. Trenutno (december, 2010) pri njih gostuje že več kot milijon spletnih domen [3]. Očitno se takšna ponudba obrestuje za stranke in ponudnika storitev.

5.1.2 Specializirana administracijska plošča

Izdelava strani in aplikacij mora biti zabavna. To pomeni, da tudi ne sme biti neprijetna, seveda. Menimo, da to pomeni, da ne uporabljamo cPanel administracijske plošče. cPanel je de facto standard, a ne preveč posrečena generična rešitev po principu ena velikost za vse.

V letu 2011 je situacija na tržišču že zelo spremenjena – veliko ponudnikov gostovanja ima svoje nadzorne plošče, nekateri se lotevajo tudi povsem drugačnega pristopa k postavitvi strani in aplikacij (omembe vredno: heroku.com). Še vedno pa marsikako podjetje zateka k takšnim narejenim rešitvam.

Že samo dejstvo, da so Dreamhost sami naredili nadzorno ploščo, priča o tem, da so strastni do svojega dela, imajo veliko razvijalskega talenta (kar je zelo pomembno pri psihološki povezavi s strankami, ki so tudi razvijalci) in da želijo za svoje stranke narediti najboljšo možno uporabniško izkušnjo.

Ta administracijska plošča ima kak dodatno prednost:

- Dostop do prošenj za podporo: ogledamo si lahko zgodovino preteklih prošenj, oddajamo novo prošnjo za pomoč ali vprašanje, na katero nam (ponavadi približno v roku dveh, treh ur) odgovori zaposleni strokovnjak.
- Glasovanje za nadgradnje – Dreamhost-ove stranke lahko glasujejo za različne programske in strojne nadgradnje sistema, kot npr. "nadgradite SQLite na različico 3.6.16+". Vsaka nadgradnja dobi težavnost (od tri do pet), stranke pa glasujejo za njim najbolj pomembne nadgradnje. Ti glasovi močno vplivajo na to, kaj bodo Dreamhost-ovi inženirji nadgradili naslednjič.

5.1.3 Subversion strežnik

Dreamhost omogoča gostovanje Subversion skladišč *(repository-jev)* na svojih strežnikih. O Subversion-u bomo govorili kasneje, tukaj pa naj povemo le, da imamo lahko vso kodo svojih projektov (in starih različic le-teh) shranjeno pri Dreamhost-u, kar pomeni dvoje:

- Najprej nam ni potrebno skrbeti za varnostne kopije naših projektov. To pomeni, da lahko stavek "Sesul se mi je disk z vsemi projekti, z vsem delom, z aplikacijami strank itd." nadaljujemo z "Ampak ni problema, imam kopijo *v oblaku*."
- Imamo centralno skladišče kode, do katerega dostopamo preko interneta, kar je nadvse priročno. Že neštetokrat smo zapustili svoj stacionarni računalnik brez misli, da bi želeli kasneje svoje delo nadaljevati kje drugje, potem pa smo nekje drugje sedli za drug računalnik (npr. prenosnik) in preprosto nadaljevali z delom.

5.2 Ogrodje ali CMS

Različne spletne aplikacije in strani pogosto vključujejo mnogo podobnih problemov: registracija in prijava, preprost dostop do podatkov v podatkovni bazi, generiranje pomanjšanih slik itd. Ogrodje (ang. *framework*) je skupek kode, ki rešuje te pogoste probleme in je ključen element pri ustvarjanju spletnih aplikacij. To je podlaga, na kateri začnemo graditi aplikacijo. V praksi to pogosto pomeni, da preden začnemo s pisanjem prve vrstice kode, skopiramo vsebino ogrodja v mapo našega projekta, potem pa začnemo z delom.

5.2.1 Morda CMS?

Poleg ogrodja pa lahko za gradnjo spletne aplikacije uporabimo tudi CMS – sistem za urejanje vsebin. Nekaj znanih CMS-ov iz sveta PHP-ja je: Wordpress, Drupal in Joomla. Wordpress bi lahko opisali kot pravo izbiro, če bi želeli narediti blog ali spletno stran z blogovsko funkcionalnostjo (poudarek torej na člankih). Drupal je dobra izbira, če želimo kompleksen CMS z veliko fleksibilnostjo: torej za dobre programerje. Joomla pa je delno zadovoljiva izbira, če želimo CMS z veliko funkcionalnosti in mislimo, da je MVC paket piškotov ter da je AJAX pred uporabo potrebno zmešati z vodo.

Kljub temu, da CMS-i vsebujejo veliko pomembne funkcionalnosti, pa so to, kot že ime pove, sistemi za urejanje vsebin in primarno niso namenjeni gradnji spletnih aplikacij, zato v našem

primeru niso prišli v izbor.

5.2.2 Torej ogrodje

Ogrodij za ustvarjanje spletnih aplikacij je ogromno, a ker smo se odločili uporabljati PHP, je ta izbira malce manjša. Naj tu omenimo tri znana ogrodja v PHP-ju.

Prvo je Zend, ki nosi ime motorja, na katerem teče PHP *(Zend Engine)*, v praksi pa je bolj skupek razredov in funkcij, kot pa celotno ogrodje, ki vodi gradnjo aplikacije.

Drugi je CakePHP, ki je povzet po znanen ogrodju, ki je v zadnjem času v vzponu: Ruby on Rails. Uporablja enake programske vzorce: ActiveRecord za dostop do baze ter MVC za ločitev uporabniškega vmesnika ter aplikacijske logike.

Tretji pa je CodeIgniter, ki je od omenjenih najmanj ogrodje, ampak preprosto okolje, v katerem lahko hitro ustvarjamo spletne aplikacije. Njegova vrlina pa je hitrost in zanjo ga je pohvalil tudi oče PHP-ja Rasmus Lerdorf (http://blogs.sitepoint.com/2008/08/29/rasmus-lerdorf-php-frameworks-think-again/).

5.2.2.1 Kaj izbrati?

Katerega izmed naštetih smo izbrali za našo aplikacijo? Nobenega, uporabili smo svojega. Gre za ogrodje, ki ga ustvarjamo, nadgrajujemo in spreminjamo že več kot tri leta.

Zakaj bi kdo izdelal lastno ogrodje in izumljal toplo vodo, če so ogrodja že narejena? Sliši se absurdno, saj so to že preverjene rešitve. A vsaka od teh je bila narejena, ko so druge že obstajale. Tako je bilo tudi s to – med obstoječimi rešitvami avtor ni našel prave in je zato začel pisati svojo. Poleg tega se starejša ogrodja zelo počasi razvijajo, revolucije v kodi so redke, kljub temu, da se spletne tehnologije spreminjajo z ogromno hitrostjo.

V to ogrodje smo vključili rešitve za vse pogoste probleme pri gradnji spletnih aplikacij, na katere smo kadar koli naleteli. Še posebej omembe vredni lastnosti pa sta olajšan dostop in urejanje podatkov v bazi ter ločenje aplikacijske logike ter pogleda po principu MVC s pomočjo motorja za predloge Smarty.

6 Koda, ki teče na strežniku

Koda, ki teče na strežniku (ang. *server-side code*) je koda, ki teče na oddaljenem strežniku in je uporabnikom aplikacije nevidna. Zadolžena je za hrambo vseh podatkov, ki jih aplikacija potrebuje (slike, podatki o projektih, podatki o uporabnikih itd.), za obdelavo ter za prikazovanje teh podatkov.

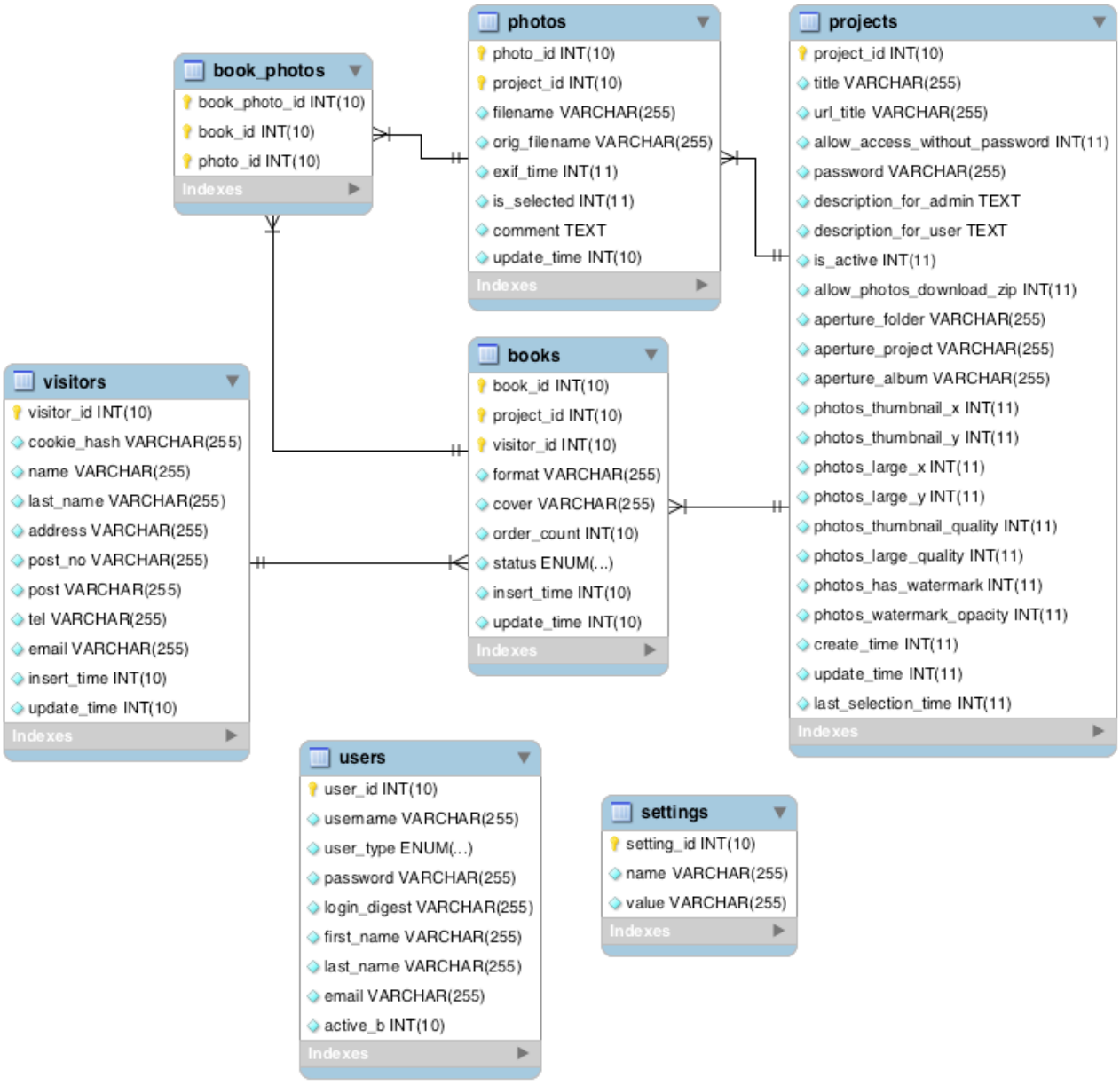
Na strežniku tečeta dva ključna dela aplikacije: MySQL kot podatkovna baza, ki hrani vse podatke, in PHP ter ostala koda, ki poganja celotno spletno stran.

6.1 MySQL – podatkovna baza in podatki

Če želimo razumeti delovanje aplikacije, je ključno spoznati podatkovni model in podatke. Za podatkovno bazo v naši aplikaciji uporabljamo MySQL, ki je relacijska podatkovna baza. Podatkovna baza pomeni neko hrambo podatkov, ki jo najprej napolnimo, kasneje pa pridobimo željene podatke iz nje. Relacijska baza pa pomeni, da so entitete (npr. fotografije, osebe, dogodki) med seboj povezane v relacije, npr. oseba ima lahko mnogo projektov, vsak projekt ima lahko mnogo fotografij – te fotografije pa tako pripadajo točno določenem projektu.

6.1.1 Podatkovni model

Podatkovni model bomo najlažje razumeli s pomočjo spodnje slike, razlaga pa sledi takoj pod njo.



Slika 14 - shema podatkovnega modela aplikacije

Vidimo, da je shema podatkovne baze izredno preprosta. Sedem entitet, ki ga sestavljajo:

- **Tabela users**: tu hranimo podatke o vseh uporabnikih strani, ki imajo privilegiran dostop – administrator in uporabnik(i), ki dodaja v spletno aplikacijo nove projekte, slike itn.
- **Tabela settings**: večina nastavitev aplikacije je interne narave in uporabniki ne potrebujejo dostopa do njih (npr. lokacija na disku, kamor se shranjujejo slike). Nekatere nastavitve pa so toliko specifične in spremenljive, da želimo, da ima fotograf dostop do njih, kot npr. logotip ali pa email naslov, kamor mu naj aplikacija pošilja email sporočila. Temu služi tabela settings, saj v njej hranimo vse take nastavitve, ki veljajo za celotno aplikacijo.
- **Tabela projects**: tu imamo shranjene vse projekte. Projekt v aplikaciji predstavlja

resničen projekt fotografa, ta povezava pa je fotografu zelo naravna, saj takšno nomenklaturo uporablja že v drugih fotografskih programih. Pri vsakem projektu se shranjuje naziv projekta, velikost pomanjšanih slik za prikaz in še druge podatke.

- **Tabela photos**: vsaka fotografija, ki jo fotograf naloži na strežnik, se shrani v to tabelo. No, v resnici se ne shrani fotografija, ampak le njeni metapodatki, torej podatki o sliki. Dejanska fotografija se shrani na trdi disk strežnika, v bazo pa ime datoteke na disku, tako da jo znamo potem najti. Za shranjevanje slik na trdi disk (in ne v bazo v binarni obliki - BLOB) smo se odločili zaradi učinkovitosti in hitrosti spletnega strežnika, saj nam tako za vsako prikazano sliko ni potrebno poganjati PHP-ja, ki bi potem še klical MySQL in prenašal podatke preko TCP protokola, ampak le Apache prebere binarne podatke slike z diska. To je tudi najbolj razširjena praksa pri delu s slikami v bazi. Iz sheme je razvidno, da vsaka fotografija pripada natanko enem projektu (slike tako ne morejo obstajati brez projekta), projekt pa lahko vsebuje nič, eno ali več fotografij.
- **Tabela books**: *book* predstavlja knjigo ali album fotografij. Aplikacija namreč omogoča obiskovalcem (strankam) tudi izdelavo knjige ali albuma, v katerem lahko sami izberejo fotografije. V tej tabeli pa shranimo format knjige, ki jo stranka želi (A4, 21x21 cm, ...), platnico itn. Vsaka knjiga pripada natanko določenem projektu, torej vsebuje slike iz natanko enega projekta fotografa, medtem ko si iz enega projekta uporabniki lahko ustvarijo več knjig.
- **Tabela book_photos**: to je povezovalna tabela (ang. *many to many*), ki poveže med seboj fotografije in knjige. Povezovalna je zato, ker imamo lahko eno fotografijo v več knjigah, ena knjiga pa ima lahko več fotografij.
- **Tabela visitors**: visitor je obiskovalec strani in ga shranjujemo v podatkovni bazi, kadar želi naročiti foto knjigo. Tako vsaka knjiga pripada točno določenem obiskovalcu. O obiskovalcu si shranimo njegove osebne podatke, tako da fotograf ve, kam naj pošlje knjigo.

6.1.1.1 Nekaj zanimivih stolpcev

6.1.1.1.1 update_time

Nekaj tabel ima stolpec *update_time*, kjer hranimo čas, ko je bil zapis v tabeli nazadnje urejen (t.j. da smo spremenili kak atribut, npr. ime projekta). Tega podatka sicer ne prikazujemo v aplikaciji, služi pa kot dodatna varnost, ki bi prišla prav, če bi nas fotograf spraševal, kaj se je zgodilo s tem ali onim podatkom.

Takrat mu lahko povemo, kdaj je bil nazadnje posodobljen in to v večini primerov reši težavo. Lahko bi hranili tudi celotno zgodovino vseh verzij podatkov (primer: fotograf spremeni ime projekta, hkrati pa v zakulisju shranimo še kopijo prejšnjega imena projekta in datum spremembe), ampak to zaenkrat ni potrebno.

6.1.1.1.2 exif_time

Naslednji zanimiv stolpec je *exif_time* v tabeli photos, kjer hranimo datum in čas, ko je bila fotografija narejena. Vse fotografije, ki pridejo iz fotoaparata, vsebujejo t.i. Exif metapodatke, kjer so zapisani podatki o fotografiji (čas nastanka, zaslonka, čas, ISO itd.). Čas nastanka iz Exif zapisa shranimo, da lahko fotografije urejamo po času pri prikazu, saj se ne moremo zanašati na datum zadnje spremembe datoteke ali imena datoteke, saj ni nujno, da bi ta vrstni red soupadal z vrstnim redom časa nastanka iz Exif zapisa.

6.1.1.1.3 password

Kaj pa bi bilo lahko posebno zanimivega v stolpcu *password* tabele users, notri pač shranimo geslo uporabnika, kajne? Raje ne! Lahko se zgodi, da nekdo nepovabljen pride do podatkov v bazi. Za to obstaja več načinov: lahko vdre v spletno aplikacijo (npr. SQL injeciranje), morda vdre v spletni strežnik (varnostne luknje, napačna dovoljenja na deljenem strežniku itd.) ali pa se dokoplje do naših varnostnih kopij podatkov (morda takrat, ko jih prenašamo s spletnega strežnika).

Ko enkrat ta oseba pride do podatkov v bazi, lahko vsa gesla prebere, če le-ta niso kriptirana. To najprej pomeni, da lahko dostopa do aplikacije kot nek fotograf in mu briše vse projekte. Še večja škoda pa nastane, ker ljudje pogosto radi žrtvujemo nekaj varnosti za lenobo in uporabljamo isto geslo za različne spletne storitve. Torej eno geslo za dostop do fotografij, socialnih omrežij (Facebook, Twitter), e-pošte in celo spletnega bančništva! Takšno početje močno odsvetujemo in v ta namen predlagamo LastPass storitev, kjer si zapomnimo samo eno glavno geslo, program pa shrani različna gesla za spletne storitve in nas tudi prijavi s temi dostopnimi podatki.

Ko enkrat nepridiprav pride do gesla naše stranke, lahko naredi veliko škode. Zato je potrebno gesla naših strank dobro varovati. Ena mera varnosti je, da vsa gesla kriptiramo z enosmernim kriptiranjem (ang. *one-way hash*), kar pomeni, da lahko le kriptiramo geslo, odkriptirati (t.j. dobiti originalno geslo) pa ga ne moremo oz. to ni preprosto izvedljivo. Iz gesla *geslo* dobimo s pomočjo SHA1 algoritma kriptiran tekst e8cc564a5e9320d6c22647c5e6dab55005bf1e68. Kako to dobimo in zakaj se iz kriptiranega teksta praktično skoraj ne da dobiti originalno geslo, je izven obsega te diplomske naloge, naj pa tukaj le povemo, da za tem stoji kar nekaj diskretne matematike.

Kaj pa naj počnemo s takim kriptiranim geslom, če ga še sami ne znamo odkriptirati? A ni to neuporabno? Sploh ne. V bazi hranimo le kriptirano geslo. Ko se uporabnik želi prijaviti v sistem, vpiše svoje geslo, ki ga sistem nato kriptira in primerja s tistim v bazi. Če se ta dva ujemata, potem je matematično ogromna verjetnost, da je uporabnik vpisal pravilno geslo. Obstaja sicer teoretična možnost, da je vpisal nek tekst, ki daje isti kriptiran tekst, ampak je v praksi to zanemarljivo.

Super, sedaj smo varni, kajne? Ne še. Vzemite zgornji kriptiran tekst gesla *geslo*, odsurfajte na spletno stran passcracking.com, potem pa ga prilepite v vnosno polje in kliknite na gumb *DoIT*. In zgodilo se bo "praktično nemogoče": dobili boste odkriptirano geslo. Programerjem strani to uspe tako, da shranijo pogosto uporabljana gesla in njihove kriptirane oblike v veliko podatkovno bazo, ki ji rečemo mavrčina tabela. Ko jih vprašamo za geslo, ki je skrito za nekim kriptiranim tekstom, program ne začne z dekriptiranjem (saj bi to lahko trajalo leta), ampak le pogleda v bazo, ali ima shranjeno primerno geslo, ki nam ga vrne v manj kot sekundi.

Kaj nam je storiti? No, čeprav se zdi položaj težaven, je rešitev izredno preprosta: geslo začinimo (ang. *salt*). Preden geslo kriptiramo, mu dodamo neko naključno zaporedje znakov, recimo ghsdjfh48tjadfownje28q9, ki je vedno enako. Tako postane geslo bolj kompleksno in zagotovo ne bo shranjeno v kakšni mavrični tabeli, pa še dodatno težje ga bo odkriptirati.

6.2 PHP ter ostala koda – možgani aplikacije

Vsakič, ko se spravimo k razumevanju kakšne spletne aplikacije, uporabljamo pristop z vrha navzdol (ang. *top-down approach* [4]) – začnemo povsem pri vrhu aplikacije (kaj to pomeni, bomo kmalu izvedeli), potem pa nadaljujemo vedno globlje v detajle. Naš pristop je podoben,

kot je pristop razhroščevalnika (ang. *debugger-ja*). Začnemo na začetku aplikacije in počasi pregledujemo kodo v istem vrstnem redu, kot se tudi izvaja. Ko pridemo do funkcije, nas čaka enaka odločitev kot pri razhroščevalniku: ali naj vstopimo v funkcijo in delovanje le-te, ali pa jo kar preskočimo, če se nam zdi nezanimiva ali pa po njenem imenu sklepamo, kaj počne.

Na tak način bomo spoznali delovanje aplikacije, prikazali pa bomo le pomembne in / ali zanimive dele.

6.2.1 Apache nastavitve

Najprej bomo spoznali posebne pomembne nastavitve Apache-ja, ki niso nastavljene na privzete vrednosti. Za spremembe nastavitev uporabljamo t.i. *.htaccess* datoteko. Za nastavitve Apache-ja lahko uporabljamo httpd.conf, ki je glavna konfiguracijska datoteka, vendar bi to želeli pustiti na miru. Razlog za to je, da želimo, da naša aplikacija deluje tudi na strežnikih, ki jih delimo z drugimi strankami ponudnika gostovanja, torej da nismo sami na strežniku. V tem primeru ne moramo spreminjati httpd.conf datoteke, saj ta vpliva na vse uporabnike in bi s tem spreminjali nastavitve vsem uporabnikom.

Rešitev, ki obstaja, je .htaccess datoteka, ki jo shranimo v mapo naše aplikacije in s tem damo prioriteto tem nastavitvam nad tistimi v httpd.conf datoteki.

Pa si poglejmo zanimive dele s te datoteke.

6.2.1.1 mod_rewrite

mod_rewrite je modul za Apache. Apache moduli so dodatki, ki jih lahko posebej vključimo in izključimo. Eden bolj popularnih je mod_rewrite [5] in je močno orodje za prepisovanje (spreminjanje) URL-jev. Poenostavljeno povedano nam omogoča, da spremenimo URL ".../project.php?title=flowers" v ".../project/flowers". Poleg tega je zmožen še veliko, saj lahko uporabljamo regularne izraze, spremenljivke strežnika in okolja itn.

Pa poglejmo delček .htaccess datoteke, ki uporablja mod_rewrite.

```
RewriteEngine On
RewriteRule
!(\.zip|\.php|\.html|\.htm|\.doc|\.pdf|\.ico|\.gif|\.jpe?g|\.png|\.ico|\.css|\.js|\.swf|\.
mp3|\.htc|\.xml)$ index.php [nocase,last]
```

V prvi vrstici vklopimo delovanje mod_rewrite motorja, potem pa pride preprost del. Vsakič, ko se URL ne konča z eno izmed končnic v oklepaju, prepišemo URL v index.php. To pomeni, da se URL "project/telovadnica" prepiše v "index.php", medtem ko se URL "img/bg.png" ne prepiše, saj se konča s končnico v oklepaju.

V oklepaju so končnice, za katere Apache ne bo poganjal PHP-ja, ampak bo le vrnil datoteko. Vsak drug URL pa bo prepisan v index.php in bo tako šel čez PHP. To pa pomeni dvoje:

- Najprej nam omogoči, da neprijazne URL-je zamenjamo za bolj prijazne, kot je primer malo višje. Tako so URL-ji, ki jih uporabniki vidijo v naslovni vrstici v brskalniku bolj berljivi in lažje se jih vpiše, če jih kdaj želimo ročno vpisati (ko ne sledimo kakšni povezavi). Lepši URL-ji pa so pozitivni tudi s stališča optimizacije strani za iskalnike (Google, Najdi.si), vendar nam to pri takšnem projektu ni toliko pomembno, ker večina strani sploh ni dostopnih iskalnikom.
- Druga stvar pa je morda še bolj pomembna. Na ta način imamo eno vstopno točko v aplikacijo: vsaka stran, ki potrebuje PHP, se začne z isto datoteko: index.php. Kako to vpliva na delovanje, pa bomo spoznali kasneje. Če ne bi uporabljali tega sistema, pa bi

bilo tako, da bi za vsak URL uporabljali drugo PHP skripto.

6.2.1.2 Predpomnjenje

Predpomnjenje (ang. *caching*) spletnih resursov lahko upravljamo s HTTP glavo (ang. *headers*). Dober vodič za predpomnjenje z Apache-jem lahko najdemo na spletu [6]. Predpomnjenje pomeni, da brskalnik (Firefox, Safari, ...) shrani nekatere resurse (slike, stile, Javascript datoteke, ...) na lokalnem disku in mu jih pri naslednjem ogledu strani ni več potrebno ponovno sneti s strežnika, kar je seveda super, saj se tako strani prej naložijo.

V HTTP glavo lahko v ta namen dodamo različne ukaze, ki upravljajo delovanje tega predpomnjenja. Imamo *Expires*, *ETag*, *Last-Modified*, *Pragma*, *Cache-Control,* ... Teh je veliko in delo z njimi hitro postane kompleksno, zato nam lahko pomaga Apache modul *mod_headers*, s katerim nam je to olajšano.

Najprej nastavimo privzeto vrednost, da vsi resursi potečejo po treh dneh (259.200 sekund) od zadnjega ogleda.

```
ExpiresActive On
ExpiresDefault A259200
```

Takoj zatem nastavimo potek resursov en mesec od zadnjega obiska za vse resurse, ki se redko oz. nikdar ne spremenijo: slike, glasba itd.

```
<FilesMatch "\.(ico|gif|jpg|jpeg|png|flv|pdf|swf|mov|mp3|wmv|ppt)$">
  ExpiresDefault A2419200
  Header append Cache-Control "public"
</FilesMatch>
```

Za dokumente, ki se pogosto spreminjajo (.css, .js, .html, ...), zahtevamo, da se bolj pogosto ponovno naloži najnovejša različica s strežnika – vsaki dve uri.

```
<FilesMatch "\.(xml|txt|html|js|css)$">
  ExpiresDefault A7200
  Header append Cache-Control "private, must-revalidate"
</FilesMatch>
```

6.2.1.3 Ostale malenkosti

Večina modernih brskalnikov zna brati kompresirane resurse z gzip algoritmom. Namesto, da strežnik (Apache) pošlje kak resurs, npr. HTML dokument, ga najprej kompresira z gzip algoritmom, potem pa pošlje ta kompresiran dokument. Tu žrtvujemo nekaj procesorskega časa (kompresiranje) za nekaj prenosne širine, kar se večinoma splača. Tako lahko s pomočjo Apache-ja stisnemo razne tipe dokumentov s preprostim ukazom.

```
AddOutputFilterByType DEFLATE text/plain
```

Ta ukaz stisne vse dokumente, katerih tip dokumenta (t.i. MIME tip) je preprost tekst. Podobno nastavimo še za mnogo drugih tipov: text/html, text/css, text/javascript itd.

V datoteki .htaccess lahko nastavimo tudi nekatere PHP nastavitve, ki jih pravtako ne moramo spreminjati v datoteki, kjer so vse PHP-jeve nastavitve (php.ini). Tako nekaj ključnih stvari spremenimo kar preko .htaccess datoteke. Vsaka naslednja vrstica vsebuje nastavitev, ki jo lahko najdemo tudi v konfiguracijski datoteki php.ini.

```
php_flag magic_quotes_gpc off
php_flag display_errors on
php_flag magic_quotes_runtime off
php_flag register_globals off
php_value default_charset "utf-8"
```

6.2.2 Shema direktorijev in datotek na disku

Ena izmed želj pri programiranju ogrodja je bila to, da je shema podatkov na disku podobna, kot bi bila, če ne bi uporabljali nobenega ogrodja. Tako želimo, da se .css datoteka ne nahaja globoko v poddirektoriju. Primer poti .css datoteke v Wordpress sitemu je

```
wp-content/themes/themename/css/main.css.
```

Želeli bi krajšo pot:

```
css/main.css.
```

Tako je naša shema direktorijev zelo poznana tudi tistim, ki niso programerji, ampak "le" spletni dizajnerji (tisti, ki naredi HTML in CSS datoteke).

```
_app
css
img
js
projects
index.php
```

Kot lahko vidimo, imamo svoje stile, slike ter fotografije in Javascript kodo na istem mestu, kot je to navada. Edina stvar, ki je malce skrita in bolj kompleksna, je HTML koda, saj se ta ne nahaja na standardnih mestih (/o-nas.html, ...), ampak malo globlje v direktoriju /_app. To je tudi direktorij, v katerem se nahaja vsa programska koda aplikacije in vse drugo, kar naj ne bo dostopno zunanjemu svetu (izjema tukaj je le index.php, ki se ne nahaja v /_app, ampak kar v /).

Vidimo lahko tudi direktorij projects, v katerem imamo vse fotografije projektov.

Razen /_app direktorija je vse drugo jasno, pa si poglejmo še vsebino tega direktorija.

```
classes
db
logs
pages
session_data
setup
smarty
temp
templates
config.php
functions_common.php
functions_additional.php
```

Razložimo še vsakega od teh posebej.

- **classes** – tu hranimo vse PHP razrede, ki jih potrebujemo, recimo HTMLPurifier za čiščenje HTML-ja, ki ga je uporabnik vnesel (odstranimo nevarne oznake ali ang. *tag-e* in atribute), Mysql za dostop do MySQL podatkovne baze, PHPMailer za pošiljanje

e-poštnih sporočil itn.

- **db** – tu imamo shranjene varnostne kopije podatkovne baze, SQL ukaze, ki so spreminjali podatkovno shemo in podobno.
- **logs** – vse dogodke, kar se dogaja z aplikacijo in se nam zdijo pomembni, shranimo v ta direktorij v dnevnike. Tako imamo dnevnik za dogodke s podatkovno bazo, dnevnik za napake in dnevnik za čase izvajanja posameznih strani.
- **pages** – tu se skriva vsa glavna logika aplikacije. Datoteka /index.php je le vstopna točka, tu pa so vse ostale .php datoteke, ki upravljajo delovanje. Tu najdemo datoteke, kot je admin.php (za administrativni del) in prijava.php, ki nas prijavi v aplikacijo.
- **session_data** – HTTP je protokol brez stanja (ang. *stateless*). To pomeni, da je zanj vsak zahtevek (ang. *request*) ločena in neodvisna stvar. Vsaka aplikacija pa želi hraniti sejo – le tako si lahko zapomnimo različne uporabnike. PHP nam to omogoča preko piškotov, kamor shrani identifikator za sejo, na lokalni disk pa shrani podatke, ki jih povezujemo z uporabnikom. Tako lahko spremljamo aktivnosti uporabnikov. Pri privzetem delovanju PHP-jevih sej pa je le manjša težava: podatke o uporabniku PHP shranjuje v /tmp direktorij (na *nix operacijskih sistemih), ki ga seveda lahko berejo vsi uporabniki na strežniku, to pa pomeni, da vsi uporabniki na deljenem strežniku lahko berejo podatke naših obiskovalcev, kar predstavlja varnostno luknjo. To zakrpamo na preprost način, da podatke o seji shranjujemo v direktorij, ki ga lahko le mi beremo s pomočjo PHP-jeve nastavitvene spremenljivke session.save_path.
- **setup** – tu imamo shranjene nastavitvene PHP skripte, ki so del ogrodja. Vsaka aplikacija ima drugačne potrebe. Nekatere potrebujejo uporabnike z možnostjo različnih vlog, v nekaterih potrebujemo hierarhične strani itd. Da se izognemo ogrodju s veliko neke funkcionalnosti, ki je aplikacija ne bo potrebovala, shranimo le-te v setup direktorij v obliki skript, ki, ko jih poženemo, naredijo vse potrebno, da začne dodatek delovati: doda tabele v podatkovno bazo, doda .php skripte ter HTML predloge.
- **smarty** – Smarty je knjižnjica za delo s predlogami. To so tekstovni dokumenti, kjer imamo vnešene, kjer pač želimo, spremenljivke, kontrolne strukture (foreach, if, ...) itn. Mi jo uporabljamo za implementacijo pogleda (ang. *view*) v MVC (Model-view-controller) arhitekturnem vzorcu. V direktoriju smarty imamo celotno knjižnjico in njene vtičnike, ne pa tudi predlog (te so spravljene v direktoriju templates).
- **temp** – včasih želimo hraniti nekatere datoteke le začasno, npr. slike, ki jih je uporabnik naložil in jih sistem še ni obdelal. Vsake toliko v tem direktoriju pobrišemo stare datoteke.
- **templates** – v tem direktoriju spravljamo vse predloge (poglede). Lahko si jih zamišljamo kot bolj napredne HTML datoteke. Vsebujejo namreč lahko spremenljivke, kontrolne strukture. Tu najdemo datoteke, kot so 404.tpl (manjkajoča stran), prijava.tpl, index.tpl itd.
- **config.php** – to je PHP skripta, v kateri imamo shranjene nastavitve za celotno aplikacijo.
- **functions_common.php** in functions_additional.php – v prvi datoteki se nahajajo funkcije, ki so zelo pogosto v uporabi pri spletnih straneh in aplikacijah in jih nimamo v razredih, npr. `check_filename` za varnostno preverjanje veljavnosti imena datoteke (da nam uporabnik ni vstavil kakšnih poti, ki bi razkrile naš sistem), `encrypt_password`

za kriptiranje gesla itn. V datoteki functions_additional.php pa najdemo dodatne funkcije, ki ne pridejo zelo pogosto prav, npr. `resize_photo` za spremembo velikosti slike ali fotografije.

6.2.3 config.php

Prva pomembna stvar, ki jo naredimo, je to, da naložimo konfiguracijo. Tu uporabljamo funkcijo *define*, s katero definiramo konstanto, npr.:

```
define("DB_ACCESS_REQUIRED", true);
```

Ta konkretni izraz določa, da v tem primeru potrebujemo dostop do podatkovne baze, v nasprotnem primeru ogrodje ne bi vzpostavljalo povezave s podatkovno bazo.

Namesto funkcije define bi seveda lahko uporabljali tudi spremenljivke ali eno spremenljivko, v kateri bi hranili vse naše nastavitve. To bi bilo priročno, saj bi lahko nastavitve hranili v hierarhični obliki, takole:

```
$settings = array(
  "database" => array(
      "username" => "root",
      "password" => "some_password",
  ),
  default_email => "rok@ddesign.si",
  ...
);
```

To bi bilo sicer lepo in berljivo, je pa problem, da bi bila spremenljivka `$settings` spremenljivka in ne konstanta, kar v svetu PHP-ja, za razliko od mnogih drugih programskih jezikov, pomeni, da ni avtomatsko dostopna znotraj funkcij brez uporabe besede global. Vsaka funkcija, ki bi uporabljala nastavitve, bi tako morala ali prejeti spremenljivko `$settings` v parametrih funkcije ali pa uporabiti besedo global:

```
function connect_to_database() {
  global $settings;
  // dostopamo do $settings spremenljivke
  ...
}
```

Zaradi tega razloga uporabljamo funkcijo define.

6.2.3.1 Nastavitve za razvojni in produkcijski strežnik

Razvoj aplikacije se ne ustavi v trenutku, ko jo damo v uporabo / *v produkcijo*. Še vedno jo želimo *razvijati*, zato imamo dva strežnika: razvojni in produkcijski. Razvojni je običajno kar računalnik razvijalca, na katerem lahko nadgrajuje aplikacijo, ne da bi motil delovanje strani v produkciji.

Produkcijski strežnik pa je zakupljen strežnik enega izmed ponudnikov gostovanja ali pa celo strežnik, ki ga podjetje samo vzdržuje (brez posrednika). Pri nas smo izbrali prvo rešitev – zakupljen strežnik. Na tem strežniku teče aplikacija, ki jo fotograf uporablja.

Obstaja še tretja vrsta strežnika, ki ga tukaj ne uporabljamo, in mu v angleškem jeziku pravimo *staging server* – nekakšen testni strežnik, kjer lahko uporabniki testirajo nove funkcionalnosti, preden pridejo v produkcijo. Lahko si ga predstavljamo kot strežnik, na katerem teče beta različica. Te vrste strežnik pride v poštev šele pri večjih operacijah in ga

tukaj, kot rečeno, ne uporabljamo.

Torej medtem, ko fotograf uporablja različico 1.23, lahko programer na razvojnem strežniku že dela na različici 1.24 ali 1.28. Tudi, če naredi kakšno veliko napako na razvojnem strežniku, se mu ni treba bati, da bo naredil kakšne probleme in aplikacija ne bo delovala. Ko so dodatki narejeni in delujoči, le nadgradi aplikacijo na produkciji na novo verzijo.

Ker gre za dva različna strežnika, potrebujeta vsak svoje nastavitve. Produkcijski strežnik na Dreamhost-u uporablja dostopne podatke do baze, do e-poštnega sistema in še marsikaj.

To lahko naredimo na preprost način:

```
if ($_SERVER['HTTP_HOST'] == "localhost") {
  // nastavitve za razvojni strežnik
}
else if (strpos($_SERVER['HTTP_HOST'], "frame.si") !== false) {
  // nastavitve za produkcijski strežnik
}
```

Nastavitve, ki so drugačne na različnih strežnikih, vpišemo v ta dva bloka kode, v prvega damo nastavitve za razvojni strežnik, v drugega pa za produkcijski. Ta del kode temelji na tem, da brskalnik za vsak zahtevek pošlje v HTTP glavi tudi ime strežnika v spremenljivki *Host*, takole:

```
GET /projects/flowers/ HTTP/1.1
Host: frame.si
...
```

Vrednost Host shrani v superglobalni spremenljivki (takšna, ki je avtomatsko nastavljena in povsod dostopna) `$_SERVER`. Ker aplikacijo razvijamo na lokalnem strežniku, vemo, da gre za razvojni strežnik, kadar je v Host spremenljivki vrednost localhost in produkcijski, kadar je v tej spremenljivki vrednost frame.si.

6.2.4 Vhod v spletno aplikacijo (index.php)

6.2.4.1 Začetne nastavitve

Kot rečeno, se vsak dostop do spletne strani začne v eni PHP skripti: index.php. Najprej, kot smo videli, naložimo konfiguracijo, ki jo imamo shranjeno v datoteki config.php.

```
require '_app/config.php';
```

Nato naložimo razreda za lažji dostop in spreminjanje podatkov v MySQL podatkovni bazi.

```
if (DB_ACCESS_REQUIRED) {
  require '_app/classes/Generic_mysql_retriever.php';
  require '_app/classes/Mysql.php';
}
```

Za tem sledi nalaganje funkcij, ki jih pogosto uporabljamo.

```
require '_app/functions_common.php';
```

Samo še nekaj začetnih nastavitev, pa bomo pripravljeni na bolj pomembne stvari.

```
session_start();
header('content-type: text/html; charset=utf-8');
```

```
error_reporting(ERROR_REPORTING_LEVEL);
set_error_handler("error_handler");
set_exception_handler('exception_handler');
```

V zgornji kodi začnemo sejo s klicem `session_start` – s tem PHP-ju povemo, naj preko piškotov spremlja uporabnika ter naj shrani vse stvari, ki jih damo v superglobalno spremenljivko `$_SESSION`, nekam, kjer bodo dosegljive tudi v naslednjih dostopih do strani.

Za tem nastavimo množico znakov na utf-8, s čimer omogočimo skoraj brezskrbno uporabo šumnikov. S klicem `error_reporting` upravljamo, katere tipe napak naj PHP sporoča nazaj uporabniku. Nato sledita le še klica `set_error_handler` ter `set_exception_handler`, ki nastavita, kateri funkciji se naj poženeta, če pride do napake ali izjeme. Tako lahko uporabniku prikažemo lepo stran s preprostim in uporabniku prijaznim opisom napake, pravi razlog pa shranimo v dnevnik in jo morda tudi pošljemo po e-pošti, če se nam zdi kritična.

6.2.4.2 Priprave na odpremo (ang. dispatch)

Že prva vrstica kode v tem drugem delu je zanimiva.

```
$actions = get_url_directories(APP_RELATIVE_URL);
```

Kot vidimo, dobimo neke akcije s pomočjo funkcije `get_url_directories`. Kaj so te akcije? Če si zamislimo preprost URL zahtevek (napisano brez protokola in strežnika) /users/rokcarl, potem je spremenljivka `$actions` takšne oblike:

```
Array (2)
(
  [0] => users
  [1] => rokcarl
)
```

Tudi v primeru, da je URL zahtevek /users/rokcarl?show_info=1&update=0, ima naša spremenljivka isto vsebino. Če smo pozorno spremljali, vemo, da nas zadnji URL zahtevek odpelje na index.php, ker smo tako nastavili v naši .htaccess datoteki. Brez te nastavitve pa bi Apache iskal skripto z imenom rokcarl v direktoriju users, kateri bi podal dva parametra: show_info in update. Funkcija `get_url_directories` tako vrne virtualne direktorije, ali, kakor so v ogrodju poimenovani, akcije.

To je zelo priročno, saj lahko hitro ugotovimo, katero stran je uporabnik obiskal oz. katero akcijo želi pognati. Parametri te akcije pa so normalno dostopni preko `$_GET` avtospremenljivke.

PHP kodo, ki opravlja dejansko delo, imamo shranjeno v /_app/pages v datotekah z imenom, ki je vezano na akcije. Tako bo URL zahtevek /users kmalu pognal skripto /_app/pages/users.php, če ta obstaja. Ta povezava med zahtevkom in PHP skripto pa se opravi na koncu v datoteki /index.php. Poglejmo kako. Sedaj, ko imamo akcije v spremenljivki `$actions`, lahko vzamemo prvo vrednost iz seznama (če le-ta obstaja) in poženemo primerno PHP skripto. Če prva vrednost ne obstaja, poženemo /_app/pages/index.php, če pa je prva vrednost enaka admin, poženemo /_app/pages/admin.php. Pred tem seveda preverimo, da ni v tem URL zahtevku (in s tem v našem seznamu) kakšnih neprijetnih kombinacij znakov, ki bi zlonamernemu uporabniku omogočile izvajanje poljubnih skript na strežniku.

```
if (file_exists("_app/pages/{$parsed_action}.php")) {
  require("_app/pages/{$parsed_action}.php");
```

```
}
```

Ker pa želimo, da imajo projekti čim krajši URL (za lažje ročno vpisovanje), ne vsebujejo dodatne akcije. Primer URL-ja za projekt z imenom NekaRevija dec-10 je le /NekaRevija-dec-10 in ne /projekt/NekaRevija-dec-10, zato v primeru, da datoteka z imenom akcije ne obstaja, poiščemo po bazi, ali obstaja kakšen projekt s takim imenom in ga prikažemo. Če tudi tak projekt ne obstaja, sporočimo uporabniku, da takšna stran ne obstaja.

6.2.5 Ogled projekta

6.2.5.1 Najdemo pravi projekt

Ogled projekta upravlja skripta /_app/pages/index.php. Najprej dobimo podatke o projektu, ki si ga uporabnik želi ogledati. Tako pričakujemo, da imamo ime projekta shranjen v spremenljivki $actions na prvem mestu, s tem imenom pa preprosto dobimo celoten projekt iz baze.

```
$project = $mr->srq("select * from projects where url_title = '@url_title@' limit 1",
array("@url_title@" => $parsed_action), false);
```

Na tem mestu imamo spremenljivki `$mr` instanco razreda `Generic_mysql_retriever`, funkcija srq pa je okrajšava za funkcijo `single_restricted_query`, besede v imenu funckije pa pomenijo:

- single – funkcija naj vrne le en zapis iz podatkovne baze,
- restricted – parametre SQL, ki jih vnesemo v parametre funkcije, naj funkcija omeji (ang. *restrict*) in naj tako prepreči zlonamerne znake v parametrih, s čimer se zavarujemo pred napadi in
- query – *query* je angleška beseda, ki pomeni poizvedbo, v našem primeru poizvedbo v podatkovni bazi.

6.2.5.2 Če ta projekt ne obstaja ali če ni projekt določen

Lahko se zgodi, da projekt, ki ga je uporabnik zahteval preko brskalnika, ne obstaja. Lahko se tudi zgodi, da uporabnik pride do naše spletne aplikacije brez imena projekta v URL-ju (ko je URL le /).

V tem primeru namesto strani s projektom uporabniku prikažemo stran, kjer lahko vpiše geslo za dostop do projekta.

```
if (!$project) {
  $cp = "enter_password.tpl";
  render($cp, $cpv);
}
```

`$cp` pomeni ime datoteke z vsebino strani (content page), `render` pa je funkcija, ki izpiše HTML dokument brskalniku (to funkcijo bomo spoznali kasneje).

Ko geslo vpiše, preverimo v podatkovni bazi, če obstaja kak projekt z danim geslom.

6.2.5.3 Ima uporabnik dovoljenje za dostop?

Ko dobimo podatke od projekta, moramo preveriti, ali ima uporabnik sploh dovoljenje za dostop do tega projekta. Fotograf namreč lahko nastavi, da so nekateri projekti zaščiteni z

geslom. Zato moramo dovoljenje preveriti.

Uporabnik lahko vstopi v projekt preko štirih načinov:

- projekt lahko gledamo tudi brez gesla (če je fotograf tako nastavil na administratorski strani),
- uporabnik ima v piškotu shranjeno geslo za projekt in to geslo je enako geslu, ki je shranjeno v projektu,
- uporabnik je pravkar vpisal pravilno geslo in
- uporabnik je prijavljen in ima administratorske pravice.

Ti štirje pogoji so lepo vidni v kodi, ki dovoljenje do projekta preverja.

```
$allow_access = false;
if (!$allow_access && $project['allow_access_without_password'] == 1)
  $allow_access = true;
if (!$allow_access && isset($_COOKIE['project_password']) && $_COOKIE['project_password']
== $project['password'])
  $allow_access = true;
if (!$allow_access && isset($_POST['password']) && $_POST['password'] ==
$project['password'])
  $allow_access = true;
if (!$allow_access && isset($_SESSION['logged_user']) &&
$_SESSION['logged_user']['user_type'] == "admin")
  $allow_access = true;
```

V spremenljivki `$allow_access` imamo sedaj shranjeno ali je uporabniku dovoljen dostop. Če je uporabnik tudi pravkar vpisal geslo in je to geslo pravilno, shranimo to geslo v uporabnikov piškot za 30 dni, tako da mu ga naslednjič ni potrebno ponovno vpisovati.

Tukaj visok nivo varnosti ni pomemben, zato imamo geslo shranjeno kar v navadni obliki.

```
if ($allow_access && isset($_POST['password']) && $_POST['password'] ==
$project['password']) {
  setcookie("project_password", $_POST['password'], time()+60*60*24*30, '/');
  $_COOKIE['project_password'] = $_POST['password'];
  add_msg_and_redirect("Prijava uspešna!", "info", $project['url_title']);
}
```

Če je uporabnik vpisal pravilno geslo, ga klic funkcije add_msg_and_redirect preusmeri (HTTP koda 301) na stran projekta.

Sedaj vemo, ali je obiskovalcu dostop dovoljen do izbranega projekta. Če ni, mu le namesto strani s projektom prikažemo stran, kjer lahko vpiše geslo za izbrani projekt.

```
if (!$allow_access) {
  if (isset($_POST['password']))
    add_page_message("Vpisano geslo je napačno.", "error");
  $cp = "enter_password.tpl";
  $cpv['project'] = $project;
  render($cp, $cpv);
}
```

6.2.5.4 Prikaz strani projekta

Vsak projekt ima navadno dodane slike, zato moramo najprej pridobiti seznam vseh slik.

```
$project['photos'] = $mr->uq("select * from photos where project_id =
{$project['project_id']} order by exif_time asc, filename asc");
```

Kot kaže koda, dobimo vse fotografije izbranega projekta in jih sortiramo po atributu exif_time, t.j. po času narejene fotografije, ker želimo, da so na vrhu strani prikazane fotografije, ki so bile najprej narejene.

Za tem sledi zanimiv izraz.

```
generate_images_if_necessary_imagemagick($project, $mr);
```

Preden prikažemo stran projekta, želimo preveriti, če so ustvarjene vse pomanjšane slike, saj je lahko fotograf pred kratkim naložil nove fotografije. Pomanjšane slike potrebujemo, da lahko uporabniku prikažemo več fotografij naenkrat na ekranu. S tem klicem ustvarimo vse potrebne slike, če še niso bile ustvarjene. To počnemo s pomočjo programa ImageMagick.

Fotograf lahko na projektu dovoli, da uporabnik (stranka) sname vse slike projekta na svoj računalnik v obliki zip datoteke. Če je to dovoljeno, moramo tako še izračunati velikost zip datoteke, da jo prikažemo uporabniku (npr. 245 MB).

```
$images_zip_filesize = "/";
if (isset($project['allow_photos_download_zip']) && $project['allow_photos_download_zip'])
  $images_zip_filesize = file_exists("projects/{$project['url_title']}/data/slike.zip") ?
format_filesize(filesize("projects/{$project['url_title']}/data/slike.zip")) : 0;
```

Preverimo, če je dovoljeno snemavanje vseh slik in če zip datoteka obstaja, potem izračunamo velikost te datoteke in formatiramo velikost v človeku prijazno obliko, saj nam funkcija filesize vrača velikost datoteke v bajtih.

Sedaj, ko imamo vse spremenljivke nastavljene, je čas, da izpišemo končni HTML dokument. Tik pred tem pa le še nastavimo vse spremenljivke, za katere želimo, da so dostopne pogledu (*view* del MVC vzorca). Ta del bo postal bolj jasen malce kasneje pri opisu funkcije render.

```
$cpv["images_zip_filesize"] = $images_zip_filesize;
$cpv['actions'] = $actions;
$cpv['project'] = $project;
$cpv['permissions'] = $permissions;
$cpv['creating_book'] = $creating_book;
$cpv['visitor_data'] = $visitor_data;

render($cp, $cpv);
```

Tako smo spoznali prikaz strani projekta, ostalo je le še nekaj funkcij, katerih delovanje morda želimo spoznati.

6.2.5.5 Funkcija render

Funkcija render ima nekaj ozadja, ki ga je potrebno razložiti. Začnimo z MVC vzorcem.

6.2.5.5.1 MVC

MVC je kratica za Model-view-controller in je arhitekturni vzorec v programiranju [7]. Najprej moramo razumeti problem, da bomo razumeli, zakaj se je MVC razvil, to pa najlažje prikažemo s koščkom psevdokode. To bomo zelo poenostavili, saj je pomemben le princip in ne vsak izraz posebej. Ta košček prikazuje možno kodo za prikaz vseh uporabnikov v HTML tabeli.

```
<?php
$database = get_database_object(); // povežemo se na podatkovno bazo
$users = $database->get_rows_by_sql("select * from users");
?>
<table>
  <tr>
    <th>Ime</th>
    <th>Priimek</th>
  </tr>
<?php
foreach ($users as $user) {
  ?>
    <tr>
      <td><?php echo $user['first_name']; ?></td>
      <td><?php echo $user['last_name']; ?></td>
    </tr>
  <?php
}
?>
</table>
```

Kot lahko vidimo iz kode, se povežemo na podatkovno bazo, dobimo podatke o vseh uporabnikih, potem pa odpremo tabelo in nato za vsakega uporabnika izpišemo eno vrstico v tabeli s podatki uporabnika.

Že ta koda izgleda grozno! Ampak to je zelo preprost primer. Lahko si zamišljamo, da bolj kompleksni problemi vsebujejo še bolj kompliciran kodo. Velik problem tukaj je, da je pogled (HTML koda) močno povezan (ang. *thightly coupled*) s kontrolerjem in podatkovnim modelom. Taki kodi pravimo tudi špagetasta koda, saj je vse močno prepleteno.

Do problema pride, ko želimo en del spremeniti. Če želimo spremeniti kaj v kontrolerju, moramo spremeniti tudi pogled in obratno. Morda v tem kratkem primeru to ni povsem očitno, ampak v praksi je to zelo pomemben problem. Med MVC deli mora obstajati le šibka povezava, da lahko preprosto spreminjamo posamezno kodo. Kdo pa želi ponovno pisati aplikacijsko logiko, če je naša naloga le spremeniti dizajn aplikacije?

Pri ustvarjanju ogrodja se nismo preveč obremenjevali s povezanostjo med podatkovnim modelom ter kontrolerjem, pomemben pa nam je bil razkorak med pogledom in preostalim delom aplikacije. V ta namen smo uporabili predlogni motor (ang. *template engine*) Smarty. Smarty nam omogoča, da pogled zgornje kode spremenimo v sledečo kodo, če mu le prej priskrbimo `$users` spremenljivko.

```
<table>
  <tr>
    <th>Ime</th>
    <th>Priimek</th>
  </tr>
  {foreach from=$users item="user"}
    <tr>
      <td>{$user.first_name}</td>
      <td>{$user.last_name}</td>
    </tr>
  {/foreach}
</table>
```

Nič preveč bolje kot prejšnja koda, bi si morda kdo mislil. A v večjih in kompleksnejših projektih naredi veliko razliko. Pa še ena zelo pomembna lastnost je, da je sedaj ta koda shranjena v ločeni datoteki, v predlogi. Tako smo dosegli, da imamo pogled ločen od ostale

kode.

6.2.5.5.2 Glavna predloga in vsebinska predloga

S spletnimi stranmi in aplikacijami je tako, da imamo navadno velik del strani, ki se (skoraj) nikdar ne spreminja. Imamo navigacijski meni, glavo in nogo strani. To kodo za pogled moramo tako ponavljati na vsaki strani. Rešitev za to je ločitev kode za pogled strani na glavno predlogo (ang. *master page*) in vsebinsko predlogo (ang. *content page*). Tako je v vsebinski predlogi le koda, ki je na vsaki strani različna.

Idejo za tako ločitev smo prevzeli od Microsoftovega ogrodja za spletne strani in aplikacije ASP.NET.

6.2.5.5.3 Delovanje funkcije render

Spoznali smo že, da imamo glavno in vsebinsko predlogo. Vsaka je shranjena v svoji datoteki (npr. master.tpl in index.tpl) ter ima svoj seznam spremenljivk, ki jih pogled potrebuje (npr. podatki o projektu, o prijavljenem uporabniku). Te štiri stvari so shranjene v primerno poimenovanih spremenljivkah `$master_page`, `$content_page`, `$master_page_variables` in `$content_page_variables`.

```
function render($content_page, $content_page_variables, $master_page = false,
$master_page_variables = false) {
  if ($master_page == false) {
    echo return_rendered_page($content_page, $content_page_variables, true);
    my_exit();
  }
  else {
    $master_page_variables['content'] = return_rendered_page($content_page,
$content_page_variables, false);
    echo return_rendered_page($master_page, $master_page_variables, true);
    my_exit();
  }
}
```

V kodi se pojavlja klic funkcije `return_rendered_page`, ki jo bomo razložili takoj po opisu `render` funkcije, na kratko povedano pa funkcija vrne HTML dokument ali nek del HTML dokumenta na podlagi izbrane predloge in spremenljivk, ki jih ta predloga potrebuje.

Zgornja koda pravi, da če zahtevamo stran z vsebinsko predlogo, potem uporabimo le to vsebinsko stran, v nasprotnem primeru pa je malo težje. Najprej dobimo vsebino vsebinske strani (npr. index.tpl), nato pa to vsebino shranimo v spremenljivko, ki jo nato pošljemo v glavno predlogo, ta pa nam vrne celotno stran. Ta celotna stran je tako sestavljena iz glavnega dela, v njem pa je na pravem mestu še vsebinski del.

Ko imamo tako celotno vsebino strani, jo preprosto izpišemo v brskalnik s pomočjo funkcije `echo`, ki je enaka funkciji `print` iz drugih jezikov.

6.2.5.6 Funkcija return_rendered_page

Ta funkcija sprejme ime datoteke predloge strani in seznam spremenljivk, ki jih ta predloga potrebuje.

Vsebina pomembnih delov te funkcije je sledeča.

```
$smarty = return_configured_smarty_object();

foreach ($page_variables as $var => $value) {
```

```
  $smarty->assign($var, $value);
}

return $smarty->fetch($page_filename);
```

Najprej dobimo pravilno konfiguriran Smarty-jev objekt (pravilno nastavljene vse poti in parametri delovanja). Smarty je motor za predloge. Njegova glavna naloga je, da s pomočjo naših spremenljivk (podatkov) spremeni predlogo v končni dokument, v tem procesu pa poganja vse kontrolne strukture v predlogi (`foreach`, `if`, ...) in zamenja spremenljivke v predlogi s končnimi vrednostmi.

Naslednji korak naše `return_rendered_page` funkcije je, da vsako (`foreach`) spremenljivko, ki smo jo prej nastavili (`$page_variables`), shranimo med spremenljivke, ki so dostopne pogledu (naši predlogi). Nato preprost klic Smarty-jeve funkcije `fetch` vrne končni dokument.

6.2.5.7 Označevanje fotografij

Fotografije lahko stranka oz. obiskovalec označi le, če je projekt aktiven.

```
if (isset($_POST["action"]) && $project['is_active']) {
  // ... koda za označevanje fotografij projekta
```

Kot lahko vidimo iz kode, se označevanje zgodi le v primeru, če je uporabnik izpolnil obrazec (`<form>`), to je urejeno s kodo `isset($_POST["action"])`.

Vse spremembe v podatkovni bazi, ki jih naredimo v tem bloku kode, ovijemo v transakcijo, da ohranimo bazo v konsistentnem stanju.

```
$mr->euq("start transaction");
// ostala koda, ki spreminja podatke v bazi
$mr->euq("commit");
```

$mr je spremenljivka objekta iz ogrodja, ki nam poenostavi delo z bazo. Tu uporabljamo funkcijo euq, ki je okrajšava za execute_unrestricted_query, kar prevedeno pomeni izvedi neomejeno (brez preverjanja za injekcijami SQL) poizvedbo.

Pomembno je vedeti, da v MySQL-u ne moremo uporabljati transakcij z MyISAM hrambenim motorjem. Le-ta bo ob uporabi transakcije “modro” tiho. Zato morajo biti vse tabele, pri katerih uporabljamo transakcije, shranjene z InnoDB motorjem.

6.2.5.7.1 Shranjevanje komentarjev

Vsaki fotografiji lahko uporabnik doda komentar, s katerim lahko fotografu sporoči, kako naj fotografijo še dodatno obdela ali kaj podobnega. Komentarje lahko najdemo v $_POST['comment'] seznamu, saj je vnosno polje za komentar določeno takole.

```
<input name="comment[Licenje-2.jpg]" />
```

Postopek shranjevanja komentarjev je sledeč: najprej izbrišemo vse komentarje. To je potrebno, saj bi v nasprotnem primeru morali za vsako fotografijo, za katero ni prejetega komentarja, preverjati, ali ga morda v bazi ima in v tem primeru komentar izbrisati. Če izbrišemo vse komentarje, je to poenostavljeno.
Nato pa le gremo po vseh komentarjih in če je kaj vpisanega, shranimo to v bazo.

```
$mr->euq("update photos set comment = '' where project_id = {$project['project_id']}");
foreach ($_POST['comment'] as $image_filename => $comment) {
```

```
  if (!empty($comment))
    $mr->erq("update photos set comment = '@comment@' where project_id =
{$project['project_id']} and filename = '@filename@' limit 1", array("@comment@" =>
$comment, "@filename@" => $image_filename));
}
```

Naslednji korak je vsaki fotografiji nastaviti ali je izbrana. To je shranjeno v atributu is_selected. Postopek je zelo podoben shranjevanju komentarjev. Le ime kvadratka za obkljukanje (ang. *checkbox*) je malce drugače sestavljeno: iz identifikatorja slike ter imena datoteke slike, zato moramo v kodi to še razbiti na dva dela.

```
<input type="checkbox" name="selected[4/Licenje 2.jpg]" />
```

Še PHP koda.

```
$mr->euq("update photos set is_selected = 0 where project_id = {$project['project_id']}");
if (isset($_POST['selected'])) {
  foreach ($_POST['selected'] as $image_id_and_filename => $on) {
    $image_id_and_filename_exploded = explode("/", $image_id_and_filename);
    $photo_id = $image_id_and_filename_exploded[0];
    $mr->erq("update photos set is_selected = 1 where project_id =
{$project['project_id']} and photo_id = '@photo_id@' limit 1", array("@photo_id@" =>
$photo_id));
  }
}
```

Ko to naredimo, je glavno opravljeno. Ostane le še par malenkosti, kot je pošiljanje e-pošte fotografu, da so fotografije izbrane, shranimo čas urejanja projekta, generiramo in shranimo Applescript skripto (ki bo fotografu označila fotografije v programu Aperture) itd.

6.2.5.8 Shranjevanje knjige

Včasih bi želeli, da si lahko stranke same ustvarijo knjigo iz izbranih fotografij. Prednost tega je, da je ceneje / preprosteje, saj fotografu ni potrebno ročno izdelovati knjige, ampak mu jo aplikacija generira iz izbranih fotografij v obliki PDF datoteke. To datoteko lahko fotograf pošlje direktno v tiskarno in se s tem izogne delu.

Stran za izdelavo knjige je skoraj ista kot ta, kjer lahko izberemo fotografije za obdelavo. Dodana so le vnosna polja za nekaj dodatnih informacij: kam naj bo knjiga dostavljena, format knjige, platnica in podobno.

Podobno kot pri označevanju fotografij tudi tukaj začnemo s preverjanjem, če uporabnik res želi ustvariti svojo knjigo, potem pa začnemo transakcijo v bazi.

```
$creating_book = isset($actions[1]) && $actions[1] == "knjiga";
if (isset($_POST['action']) && $creating_book) {
  $mr->euq("start transaction");
  // ...
  $mr->euq("commit");
```

Naslednji trije deli kode so kar razumljivi, s prvim shranimo obiskovalca v bazo, z drugim ga shranimo v piškot (da mu ne bo naslednjič ponovno potrebno vpisovati svojih podatkov), s tretjim pa shranimo podatke o knjigi v bazo.

```
$visitor = $_POST['visitor'];
$visitor['cookie_hash'] = sha1(strval(rand(0,microtime(true))) . $visitor['last_name']);
$visitor['insert_time'] = time();
$visitor['update_time'] = time();
```

```
$mr->insert("visitors", $visitor);
$visitor_id = $mr->get_last_insert_id();

$cookie_visitor_data = array("name"=>$_POST['visitor']['name'],
"last_name"=>$_POST['visitor']['last_name'], "address"=>$_POST['visitor']['address'],
"post_no"=>$_POST['visitor']['post_no'], "post"=>$_POST['visitor']['post'],
"tel"=>$_POST['visitor']['tel'], "email"=>$_POST['visitor']['email']);
setcookie("visitor_data", serialize($cookie_visitor_data), time() + 30*24*3600);

$book = $_POST['book'];
$book['project_id'] = $project['project_id'];
$book['visitor_id'] = $visitor_id;
$book['insert_time'] = time();
$mr->insert("books", $book);
```

Za tem sledi le še to, da shranjeni knjigi dodamo v bazo še izbrane fotografije. Koda je podobna, kot smo jo videli že pri označevanju fotografij, zato je tu ne bomo razlagali.

```
if (isset($_POST['selected'])) {
  foreach ($_POST["selected"] as $image_id_and_filename => $on) {
    $image_id_and_filename_exploded = explode("/", $image_id_and_filename);
    $photo_id = $image_id_and_filename_exploded[0];
    $book_photo = array("book_id" => $book_id, "photo_id" => $photo_id);
    $mr->insert("book_photos", $book_photo);
  }
}
```

Tako, s tem smo zaključili opis kode za ogled projekta.

6.2.6 Administratorski del

Administratorski del aplikacije se začne s preverjanjem ali ima uporabnik dovoljenje za dostop do tega dela. Če nima, ga preusmerimo na prijavno stran.

```
if (!isset($_SESSION['logged_user']))
  client_redirect("prijava/?redirect=" . urlencode($_SERVER['REQUEST_URI']));
```

Sedaj lahko dobimo podatke o projektu. Ampak najprej moramo dobiti URL izbranega projekta, če je kakšen bil izbran.

```
$project_url_title = isset($actions[1]) ? $actions[1] : false;
```

URL administratorskega dela projekta Modna revija bi bil /admin/Modna-revija. V zgornji kodi uporabljamo 2. mesto v seznamu `$actions`, saj je na prvem mestu vrednost admin, na drugem pa Modna-revija. Pripravljeni smo, da vzamemo vse podatke o projektu iz baze, kakor tudi podatke o vseh fotografijah tega projekta.

```
$project = false;
if ($project_url_title) {
  $project = $mr->srq("select * from projects where url_title = '@url_title@' limit 1",
array("@url_title@" => $project_url_title), false);
  if ($project)
    $project['photos'] = $mr->uq("select * from photos where project_id =
{$project['project_id']} order by exif_time asc, filename asc");
}
```

Sedaj pride del, ki razdeli kodo na dve strani. Prva je stran s seznamom vseh projektov. Na tej strani lahko fotograf vidi vse projekte in izbere enega, ki bi ga rad urejal. Na drugi strani pa

lahko ureja detajle projekta in dodaja nove fotografije.

```
if (!$project) {
  // koda za stran za izbiro projekta
}
else {
  // koda za urejanje projekta
}
```

6.2.6.1 Izbira projekta

Za stran izbire projekta ni veliko kode. Najprej dobimo vse projekte, potem pa za vsak projekt nastavimo deset malih sličic za hiter pregled delne vsebine projekta (ko uporabnik premika miško po prvi sličici projekta, se mu prikazuje teh deset drugih sličic, da lahko hitro vidi vsebino projekta) in poiščemo število izbranih in število vseh fotografij.

Kodo za male sličice bomo izpustili, ostala koda pa sledi.

```
$projects = $mr->uq("select * from projects order by is_active, create_time desc");
foreach ($projects as &$project) {
  $selected_photos_count = $mr->suq("select count(photo_id) as photos_count from photos
where project_id = {$project['project_id']} and is_selected = 1");
  $project['selected_photos_count'] = $selected_photos_count['photos_count'];
  $all_photos_count = $mr->suq("select count(photo_id) as photos_count from photos where
project_id = {$project['project_id']}");
  $project['all_photos_count'] = $all_photos_count['photos_count'];
}

$cpv['projects'] = $projects;
render($cp, $cpv);
```

6.2.6.2 AJAX zahtevki

AJAX je kratica, ki je na začetku pomenila asinhron Javascript in XML (Asynchronous JavaScript and XML), ta pomen pa je sčasoma prerasla in postala okrog leta 2005 popularna beseda, ki pomeni osveževanje posameznih delov strani brez osvežitve celotne strani [1]. Lahko se osveži okvirček z aktualnimi novicami, okvirček s športnimi rezultati, lahko poženemo kakšno akcijo v ozadju itd. Vse to se zgodi asinhrono, kar pomeni, da si lahko uporabnik še naprej nemoteno ogleduje stran, v ozadju pa se zgodi zahtevek.

AJAX je pomenilo tehnologijo, kjer je Javascript preko objekta *XMLHttpRequest* s pomočjo podatkovnega formata XML komuniciral s strežnikom. V tej aplikaciji namesto XML-a uporabljamo JSON (*Javascript Object Notation*), saj je v Javascriptu veliko bolj preprost za uporabo, pa še lepši je. Mi torej uporabljamo AJAJ, če smo povsem natančni, a bomo kljub temu uporabljali kratico AJAX, saj je bolj poznana.

Gre za štiri različne zahtevke, v oklepaju je napisan parameter, ki sproži to akcijo: brisanje vseh generiranih fotografij (`delete_generated_photos`), generiranje fotografij (`generate_photos`), brisanje seznama izbranih fotografij (`delete_photos_selection`) ter osvežitev aplikacije na zadnjo različico (`update_app`).

Vsi štirje zahtevki pridejo preko HTTP GET akcije, ki ima za parameter nastavljen zahtevek, ki ga želimo izvesti. Če je uporabljen URL /admin?generate_photos, to ugotovimo s kodo `if (isset($_GET['generate_photos']))`, potem pa generiramo fotografije. Koda vseh štirih akcij sledi.

```
// delete_generated_photos
delete_dir("projects/{$project['url_title']}/img-s");
delete_dir("projects/{$project['url_title']}/img-l");
$mr->euq("delete from photos where project_id = {$project['project_id']}");
echo "'okay'";
exit;

// generate_photos
generate_images_if_necessary_imagemagick($project, $mr);
echo "'okay'";
exit;
}

// delete_photos_selection
$mr->euq("update photos set is_selected = 0 where project_id = {$project['project_id']}");
echo "'okay'";
exit;

// update_app
$output_array = array();
$return_value = 0;
exec("svn update 2>&1", $output_array, $return_value);
$response = ($return_value == 0 ? "'okay'" : "'napaka'");
echo $response;
exit;
```

img-s in img-l sta direktorija v direktoriju projekta, kjer hranimo male in velike fotografije. Osvežitev različice aplikacije pa naredimo tako, da poženemo program svn update, ki potegne najnovejšo različico aplikacije in jo zamenja z našo.

6.2.6.3 Nastavitve projekta

Vsakemu projektu lahko nastavimo mnogo parametrov, kot npr. velikost pomanjšanih slik, ali se naj na slikah prikazuje vodni tisk, ime projekta itd.

Vsa imena vnosnih polj v obrazu nastavitve imamo določena tako, kot ta primer za polje ime projekta.

```
<input type="text" name="project[title]" value="" />
```

Tekst title, ki ga vidimo v kodi, ustreza imenu polja v podatkovni bazi. Tako bomo vse vrednosti dobili v kodi v seznamu $_POST['project'], ki bo z imeni polj ustrezal stolpcem v bazi. To močno poenostavi zadeve.

Vse nove podatke shranimo v seznam `$new_project_data`.

```
$new_project_data = $_POST['project'];
```

Vse, kar moramo še narediti, preden spremenimo podatke v podatkovni bazi, je da nastavimo vrednosti za polja, ki jih lahko uporabnik obkljuka ali odkljuka, saj se ta ne bojo avtomatsko pravilno nastavila. Tukaj je prikazan primer za polje is_active – uporabnik lahko obkljuka, če želi, da je projekt aktiven.

```
$new_project_data['is_active'] = isset($new_project_data['is_active']);
```

Na koncu le še osvežimo podatke v bazi z novimi podatki.

```
$mr->update("projects", "project_id", $project['project_id'], $new_project_data);
```

6.2.6.4 Generiranje foto knjige

Uporabniki imajo možnost ustvarjanja foto knjige. To naredijo tako, da se pri ogledu projekta odločijo (s klikom na povezavo), da bi želeli ustvariti knjigo. Delovanje aplikacije je potem za uporabnika enako kot pri izboru fotografij za končni izdelek (kar običajno počne), le da mora najprej izpolniti nekaj podatkov o tej knjigi, kot npr. kam naj bo knjiga dostavljena, kakšno platnico bo imela itd.

Ko uporabnik shrani izbrane fotografije za knjigo, se knjiga shrani v bazo, to knjigo pa se poveže z izbranimi fotografijami.

```
$book = $_POST['book'];
$mr->insert("books", $book);
$book_id = $mr->get_last_insert_id();

if (isset($_POST['selected'])) {
  foreach ($_POST["selected"] as $image_id_and_filename => $on) {
    $image_id_and_filename_exploded = explode("/", $image_id_and_filename);
    $photo_id = $image_id_and_filename_exploded[0];
    $book_photo = array("book_id" => $book_id, "photo_id" => $photo_id);
    $mr->insert("book_photos", $book_photo);
  }
}
```

Za tem pride zanimivi del, ko fotograf v administratorskem vmesniku izbere, da želi izvoziti to knjigo. Kode, ki izvozi izbrane fotografije v PDF formatu je veliko, zato bomo prikazali le izbrane dele.

Generiranje PDF dokumenta nam omogoča PHP knjižnjica TCPDF, zato jo je potrebno najprej naložiti.

```
require_once('_app/classes/tcpdf/config/lang/eng.php');
require_once('_app/classes/tcpdf/tcpdf.php');
```

Iz baze poberemo podatke o knjigi in fotografijah.

```
$book = $mr->srq("select * from books where book_id = @book_id@ limit 1",
array("@book_id@" => $book_id));
$book['photos'] = $mr->uq("select p.* from book_photos bp inner join photos p
using(photo_id) where book_id = {$book['book_id']} order by exif_time, filename");
```

Za tem pridejo nastavitve knjige: širina belega roba okrog vsake fotografije, odmik od roba dokumenta, razmak med dvema fotografijama ter število pik na inčo printane knjige.

```
$border_width = 2;
$margin = 20 + 3; // 2 cm + 3mm porezave
$between_images_margin = 20;
$dpi = 300;
```

Med nastavitve sodi tudi format končne knjige. Poglejmo nastavitev za ležeči A4 format. Ta ima nastavljene dimenzije, poleg tega pa še dva prostora za fotografijo.

```
$page_formats = array(
  "A4-landscape" => array(
    "width" => 303,
    "height" => 216
  ),
  ...
);
```

```
$page_formats['A4-landscape']['placeholders'] = array(
  array(
    "x" => 0 + $margin,
    "y" => 0 + $margin,
    "width" => $page_formats['A4-landscape']['width'] / 2 - $margin - $margin / 2,
    "height" => $page_formats['A4-landscape']['height'] - 2 * $margin
  ),
  array(
    "x" => $page_formats['A4-landscape']['width'] / 2 + $margin / 2,
    "y" => 0 + $margin,
    "width" => $page_formats['A4-landscape']['width'] / 2 - $margin - $margin / 2,
    "height" => $page_formats['A4-landscape']['height'] - 2 * $margin
  )
);
```

Izbira položaja fotografije na strani je kompleksnejša stvar (ali pa vsaj bolj široka z besedami kode), zato bomo raje prikazali izris fotografije na stran.

```
$placeholder = $page_format['placeholders'][$i];

$pdf->Rect($page_photos[$i]['inserted_x'] - $border_width, $page_photos[$i]['inserted_y']
- $border_width, $page_photos[$i]['inserted_width'] + 2 * $border_width,
$page_photos[$i]['inserted_height'] + 2 * $border_width, "F", array(), array(255, 255,
255));
$image_number = str_pad(strval(1 + $counter + $i), 4, "0", STR_PAD_LEFT);
resize_photo_with_dpi_and_mm("projects/{$project['url_title']}/orig/{$page_photos[$i]['ori
g_filename']}", "{$image_cache_dir}{$image_number}.jpg",
$page_photos[$i]['inserted_width'], $page_photos[$i]['inserted_height'], $dpi);
$pdf->Image("{$image_cache_dir}{$image_number}.jpg", $page_photos[$i]['inserted_x'],
$page_photos[$i]['inserted_y'], $page_photos[$i]['inserted_width'],
$page_photos[$i]['inserted_height'], '', '', '', false, $dpi, '', false, false, 0);
```

V spremenljivki `$pdf` imamo objekt iz TCPDF knjižnjice, ki nam daje na voljo mnogo funkcij. Tu uporabljamo `Rect`, s katero izrišemo velik bel pravokotnik, ki deluje kot obroba fotografije, ter `Image`, s katero izrišemo fotografijo.

6.2.6.5 Skripta za uvoz izbora v Aperture

Skripta, ki uvozi izbor fotografij v Aperture, je napisana v Apple-ovem skriptnem jeziku Applescript. Skripta najprej požene Aperture, nastavi ime albuma (ime albuma pride iz administratorskega vmesnika in ga vpiše fotograf), da vsem fotografijam v albumu tri zvezdice, nazadnje pa izbranim fotografijam da pet zvezdic. Tako lahko fotograf naredi nov (pametni) album, ki vsebuje le izbrane fotografije.

```
tell application "Aperture"
  set albumName to "{$project.aperture_album}"
  tell album albumName
  set main rating of every image version to 3
  {foreach from=$image_filenames_without_extensions item="image_name"}
    set main rating of image version named "{$image_name}" to 5
  {/foreach}
  end tell
end tell
```

7 Koda, ki teče na odjemalcu

V tem delu bomo prikazali pomembne in zanimive dele kode, ki teče na odjemalcu (klientu). To vključuje HTML, CSS ter Javascript kodo.

7.1 Izbor fotografij ter dinamično štetje izbranih fotografij

Ko odpremo stran projekta, se nam prikažejo vse slike. Nekatere od teh so že označene (če jih je uporabnik že prej označil in se je izbor shranil v bazo), nekatere pa še bodo. Uporabnik jih označi tako, da premakne miško nad fotografijo, potem pa lahko ali pritisne tipko a na tipkovnici ali pa klikne na kvadratek za obkljukanje, ki se je prikazal, ko je miškin kazalec prišel nad fotografijo.

Ko uporabnik klikne na izbirno polje fotografije, se požene sledeč blok Javascript kode.

```
$("div.image input[type='checkbox']").click(function() {
  if (!can_select)
    return;
  if ($(this).attr('checked'))
    $(this).closest("div.outer").addClass("selected");
  else
    $(this).closest("div.outer").removeClass("selected");
  update_selected_images_count();
});
```

Uporaba spremenljivke `$` nam pove, da (verjetno) uporabljamo Javascript knjižnjico jQuery [8]. To je izredno popularna knjižnjica, ki je postala že skoraj de facto standard. V kodi vidimo, da vsakič, ko uporabnik klikne na izbirno polje, najprej preverimo, če ima dovoljenje izbirati fotografije (`can_select`), nato pa označimo okvir fotografije kot izbran ali neizbran (glede na to, ali je izbirno polje obkljukal ali odkljukal), nato pa osvežimo število izbranih fotografij.

Poglejmo si še to funkcijo, ki uporabniku prikaže novo število izbranih fotografij.

```
function update_selected_images_count() {
  var selected_images_count = $("#form div.selected").length;
  $("#selected_images_count")
    .text(selected_images_count)
    .stop().animate({color: "#eee151"}, 300)
    .animate({ dummy: 1 }, 500)
    .animate({color: "#cccccc"}, 1000);
}
```

Najprej s pomočjo jQuery-ja ugotovimo število vseh izbranih fotografij (t.j. število okvirjev z razredom selected), potem pa osvežimo tekst, ki prikazuje število izbranih fotografij, hkrati pa ga obarvamo z nežno rumeno barvo, ki jo potem nastavimo nazaj na sivo, vse skupaj pa animiramo, da se barve počasi prelijejo.

Poglejmo še CSS kodo za okvir z razredom `selected`.

```
.selected div.image div.status {
  background: url(../img/accepted.png) no-repeat;
  width: 48px; height: 48px;
```

```
  position: absolute; top: 8px; right: 8px;
  opacity: 0.75;
}
```

Kot lahko vidimo, je razred selected le nekakšen preklopnik, ki spremeni (če je nastavljen okvirju) delovanje elementa div.status. Temu doda polprosojno sliko zelene kljukice, ki nakazuje izbor, v desni zgornji kot.

Kot rečeno pa lahko uporabnik izbere fotografijo tudi, če pritisne tipko “a”, ko je z miško nad fotografijo. Postopek v kodi pa je takšen: najprej preko knjižnjice shortcut.js nastavimo, naj se za vsak pritisk tipke “a” požene naša koda. Ta koda najde okvir slike, nad katerim se je nahajal kazalec miške, ko se je pritisk zgodil. Nato nam ostane le še pravilna nastavitev ali odstranitev razreda selected, obkljukanje ali odkljukanje izbirnega polja ter osvežitev teksta s številom izbranih fotografij.

```
shortcut.add("a", function(e) {
  if (!can_select)
    return;
  $el = $(document.elementFromPoint(window.mouseXPos,window.mouseYPos));

  $div_image = $el.filter("div.middle").length ? $el.find("div.image") :
$el.closest("div.image");
  if ($div_image.closest("div.outer").hasClass("selected"))
    $div_image.closest("div.outer").removeClass("selected")
      .find("input[type='checkbox']").attr("checked", "");
  else
    $div_image.closest("div.outer").addClass("selected")
      .find("input[type='checkbox']").attr("checked", "checked");
  update_selected_images_count();
}, {
  'type':'keydown',
  'propagate':true,
  'target':document
});
```

7.2 Komentiranje fotografij

V vsakem okvirju se skriva element span.comment-trigger. Ta se prikaže le, kadar je dovoljen ogled komentarjev.

```
.not-show-comments span.comment-trigger {display: none;}
```

Za ozadje tega elementa je narisan oblaček, ki nakazuje komentar. Če ima ta oblaček še znak plus, potem je nek komentar že dodan.

Postopek vpisovanja komentarjev poteka takole: kliknemo na oblaček in prikaže se nam veliko vnosno polje, vse ostalo pa se zatemni. Ta modalni dialog nam priskrbi knjižnjica Fancybox, ki jo navadno uporabljamo za prikaz povečanih slik, kjer se ozadje zatemni. Hkrati si zapomnimo identifikator elementa, kateremu dodajamo komentar. Ko uporabnik vpiše komentar, najdemo izbran element in shranimo vpisano vrednost v skrito vnosno polje, ki je nato dostopno kodi na strežniku.

```
$("span.comment-trigger").click(function() {
  var $input = $(this).next("input");
  var input_value = $input.val();
  $("#comment_input_id_store").val($input.attr("id"));
```

```
  $.fancybox({
    // parametri za fancybox
  });
});
```

Še parametri za Fancybox.

```
onComplete: function() {$("#fancybox-wrap #comment").val(input_value); $("#fancybox-wrap
#comment").focus();},
onCleanup: function() {
  var new_input_value = $("#fancybox-wrap #comment").val();
  var $div_outer = $input.closest("div.outer");
  var $div_image = $input.closest("div.image");
  var comment_input_id = $("#comment_input_id_store").val();
  var user_entered_comment = $("#fancybox-wrap #comment").val();
  $("#" + comment_input_id).val(user_entered_comment);
  $div_image.attr("title", new_input_value);
  if (user_entered_comment == "")
    $div_outer.removeClass("commented");
  else
    $div_outer.addClass("commented");
}
```

Dva parametra sta ključna. `onComplete` se požene takoj za tem, ko se modalni dialog prikaže, takrat nastavimo vsebino vnosnega polja v dialogu na vrednost komentarja fotografije (če ta obstaja). `onCleanup` se požene, ko se dialog zapre. Takrat pa vzamemo novo vrednost komentarja, ki jo je uporabnik vpisal v vnosno polje, in jo shranimo v polje za komentar od fotografije, nato pa še označimo fotografijo kot označeno, tako da je z ikono vidno, da je fotografija komentirana.

7.3 Poganjanje AJAX akcij

Videli smo že, kakšna je koda na strežniku za poganjanje akcij, sedaj pa bomo videli še kodo na odjemalcu.

Vsak gumb, ki sproži akcijo, ima sledečo HTML kodo. Identifikator je seveda pri vsakem gumbu drugačen.

```
<span class="button" id="delete_generated_photos">...</span>
```

Nato sledi le še Javascript koda, ki ob kliku na ta gumb sproži akcijo.

```
$("#delete_generated_photos").click(function() {
  if (!confirm('Res brišem vse fotografije?!'))
    return;

  $("#delete_generated_photos_status").html(" <img src='/strani/frame.si/img/ajax-
loader.gif' style='vertical-align: middle;' \/> brišem slike...");
  $.ajax({
    url: app_url + 'a/' + project_url_title,
    dataType: 'json',
    data: {'delete_generated_photos': 1},
    success: function(data, textStatus) {
      if (data == 'okay')
        $("#delete_generated_photos_status").html(" <img
src='/strani/frame.si/img/accepted-32.png' style='vertical-align: middle;' \/> slike
brisane");
      else
```

```
        $("#delete_generated_photos_status").html("");},
    error: function() {$("#delete_generated_photos_status").html(" <img
src='/strani/frame.si/img/cancel-32.png' style='vertical-align: middle;' \/> prišlo je do
napake");}
  });
});
```

Najprej vprašamo fotografa, če je res prepričan v sprožitev akcije. Nato pa, če je odgovoril pozitivno, nakažemo, da se akcija izvaja, za tem pa sprožimo AJAX zahtevek, kateremu določimo, da naj ob uspešnem zaključku fotografu sporoči, da je akcija uspela.

8 Sklepne ugotovitve

Aplikacija je narejena in v uporabi že več mesecev, a kljub temu še ni dosegla svojega potenciala. Najprej potrebuje več uporabnikov. Obstaja ogromno fotografov, ki potrebujejo takšno aplikacijo, a ne vedo, da obstaja. Naša želja je, da bi preko predstavitev aplikacijo predstavili fotografom in jih s tem navdušili zanjo.

Več uporabnikov bi pomenilo tudi več idej za izboljšave in več načinov uporabe, ki nam lahko pomagajo napredovati z aplikacijo.

Ugotovili smo, da je za takšne vrste aplikacije obnese metoda razvoja ekstremno programiranje, pri katerem hitro razvijamo posamezne module in nato opazujemo uporabo ter na tak način najdemo ideje za nove izboljšave.

Med ustvarjanjem so se nam porodile še dodatne ideje, ki bi jih želeli v prihodnosti razviti. Želeli bi uvesti možnost javnih projektov, kar bi pomenilo, da bi označeni projekti bili vidni za vse obiskovalce, ki bi videli seznam le-teh. Ti bi služili fotografu kot nekakšen portfelj. Naslednja funkcionalnost je bolj napredna uporaba HTML 5 in CSS 3 pri galeriji, tako da bi s tem naredili bolj privlačno brskanje po fotografijah. Tretji dodatek pa je boljša podpora za izvoz PDF knjig. Tu bi lahko fotograf nastavljal različne parametre knjige, kot npr. število slik na stran, ozadja knjige in podobno.

Literatura

[1] Chris Ullman, Lucinda Dykes, »Beginning Ajax«, Wrox Press, 2007
[2] Publish for Approval v1.1. Dostopno na:
http://automator.us/aperture/publish/
[3] Web Hosting Report for dreamhost.com. Dostopno na:
http://www.webhosting.info/webhosts/reports/total_domains/DREAMHOST.COM
[4] Top Down Design in An Object Oriented World. Dostopno na:
http://www.cs.usfca.edu/~parrt/course/601/lectures/top.down.design.html
[5] Apache Module mod_rewrite. Dostopno na:
http://httpd.apache.org/docs/current/mod/mod_rewrite.html
[6] Caching Guide. Dostopno na:
http://httpd.apache.org/docs/2.2/caching.html
[7] Model-View-Controller Pattern. Dostopno na:
http://www.enode.com/x/markup/tutorial/mvc.html
[8] jQuery. Dostopno na:
http://jquery.com/