Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Aleš Bešter

Razvoj spletnega modula za objavljanje in pregled kmetijskih proizvodov

diplomska naloga

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: doc. dr. Mira Trebar

Ljubljana, februar 2012

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00174/2011
Datum: 04.10.2011

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

Kandidat: **ALEŠ BEŠTER**

Naslov: **RAZVOJ SPLETNEGA MODULA ZA OBJAVLJANJE IN PREGLED KMETIJSKIH PROIZVODOV**

DEVELOPMENT OF A WEB MODULE TO ADVERTISE AND REVIEW AGRICULTURE PRODUCTS

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

V diplomskem delu predstavite različne pristope in nekaj aktualnih tehnologij za razvoj interaktivnih spletnih aplikacij, ki omogočajo tako upravljanje kot tudi pregled informacij shranjenih v podatkovni bazi. Izdelajte modul Tržnica za spletno aplikacijo, ki omogoča objavljanje in pregledovanje informacij o kmetijskih proizvodih v spletni aplikaciji Kmetija.

Mentor:
Mira Trebar
doc. dr. Mira Trebar

Dekan:
prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Aleš Bešter, z vpisno številko **63080260**, sem avtor diplomskega dela z naslovom:

Razvoj spletnega modula za objavljanje in pregled kmetijskih proizvodov

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Mire Trebar,
- 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 7. februarja 2012 Podpis avtorja:

Zahvala

Zahvalil bi se mentorici doc. dr. Miri Trebar, za pomoč pri izdelavi diplomske naloge in Juretu Demšarju za nasvete pri razvoju spletnega modula. Obenem pa bi se zahvalil tudi strašem in dekletu za pomoč ter podporo med študijem.

Kazalo

Uporabljene kratice in simboli

AJAX - *(angl.) asynchronous javascript*; asinhroni javascript

ASP - *(angl.) active server pages*; aktivne spletne strani

CSS - *(angl.) cascading style sheet*; kaskadne stilske predloge

HTML - *(angl.) hypertext markup language*; jezik za označevanje nadbesedila

SHA - *(angl.) secure hash algorithm*; varni razprševalni algoritem

SQL - *(angl.) structured query language*; strukturiran poizvedovalni jezik

XHTML - *(angl.) extensible hypertext markup language*; razširljiv jezik za označevanje nadbesedila

XML - *(angl.) extensible markup language*; razširljiv označevalni jezik

XSLT - *(angl.) extensible stylesheet language transformations*; razširljiv stilski jezik transformacij

URI - *(angl.) uniform resource identifier*; univerzalni vir identifikatorja

WWW - *(angl.) world wide web*; splet

Povzetek

Spletne aplikacije so zasnovane z namenom, da omogočajo hkratno delo velikega števila uporabnikov. Uporabljajo se lahko tako za objavljanje kot za pregledovanje podatkov. Pri tem pa je treba poskrbeti za ustrezno načrtovanje in razvoj aplikacij, ki vključujejo različne pristope in tehnologije. Predstavljenih je nekaj osnovnih modelov in tehnologij, ki so uporabljeni pri razvoju enega od modulov za spletno aplikacijo Kmetija. Osnovni cilj modula je omogočiti predstavitev kmetij in kmetijskih proizvodov z nekaterimi dodanimi funkcionalnostmi.

V diplomski nalogi je bil razvit in implementiran spletni modul Tržnica, ki omogoča dodajanje, urejanje ter pregledovanje kmetij, kmetijskih pridelkov in proizvodov v obliki malih oglasov. Neregistrirani uporabniki bodo imeli možnost pregledovanja, registrirani uporabniki pa bodo lahko tudi dodajali in urejali informacije v povezavi s kmetijo. Vsak oglas bo vseboval podatke, po katerih je možno filtrirati in prikazovati zahteve uporabnika. Poleg tega bo možno prikazovati podatke o točno določenih izdelkih, ki so na voljo za prodajo. V modul je vključena tudi statistika, ki omogoča pregled trenutne povprečne cene in zaloge izdelkov, in sicer po različnih obdobjih.

Ključne besede:

Splet, tehnologija, kmetija, proizvodi.

Abstract

Web applications are designed with the intention to allow the simultaneous work of a large number of users. They can be used for both publishing and reviewing data. It is necessary to provide adequate planning and development of the applications that involve different approaches and technologies. There are a few basic models and technologies that are used in developing one of the modules for online application Kmetija presented in the thesis.

The basic goal of the module is to allow the presentation of farms and agricultural products with some additional functionality. There has been developed and implemented the online module Tržnica in the thesis. The online module allows adding, editing and reviewing of farms, agricultural produce and products in the form of small ads. Unregistered users will have the possibility of reviewing; the registered users will have beside this also the possibility of adding and editing information about farms. Each ad will include information, which would allow to filter and to display the requirements of a user. It is also possible to display information about specific products that are available for sale. The module includes also the statistics that provide the overview of the current average price and of the stocks of the products at different periods.

Keywords:

Web, technology, farm, products.

Poglavje 1

Uvod

S pojavom računalnikov se srečamo s pojmom računalniška aplikacija. Vsak izmed nas se je najverjetneje prvič srečal z računalniško aplikacijo, ko je pritisnil gumb za zagon računalnika. Zagnal se je operacijski sistem - aplikacija, ki je namenjena upravljanju računalnika. Dandanes si navaden dan težko predstavljamo brez aplikacij. Vse prostoročno pisanje, računanje in spletne nakupe nadomeščajo aplikacije, izdelane za določene vrste opravil. Priljubljenost tovrstnih aplikacij narašča iz dneva v dan, in sicer zaradi lahkega upravljanja s podatki. Aplikacije za omenjena opravila delimo med namizne aplikacije (angl. desktop applications) in spletne aplikacije (angl. web applications). Prednost slednjih je, da niso nameščene neposredno na trdem disku (lokalno), ampak na oddaljenem strežniku, do katerega dostopamo preko interneta, ki je v poslovnem in domačem okolju vsesplošno razširjen. Spletne aplikacije lahko tečejo na različnih računalnikih in niso odvisne od operacijskega sistema, ki ga uporabljamo. Uporabljajo jih podjetja, ki želijo na čim enostavnejši način nuditi svoje usluge potrošnikom oziroma uporabnikom. Primer poslovne spletne aplikacije je Bolha.net, ki omogoča pregled in vnašanje podatkov vseh vrst oziroma podatkov različnih uporabnikov. Ker je v današnjih časih umetno pridelane hrane vse več, prehrambna industrija vedno bolj stremi k tako imenovanim ekološkim izdelkom, ljudje pa želimo pridobiti čim več podatkov o izvoru hrane. Največ take hrane v Sloveniji pri-

delajo prav na kmetijah; od tu ideja narediti spletno aplikacijo, ki omogoča objavo kmetij in njihovih izdelkov ter kontakta lastnika kmetije. Modul bo imel možnost objavljanja in pregledovanja malih oglasov o izdelkih. Za vse registrirane uporabnike bo podpiral vnos kmetij, njihovih podatkov in slik. Vsaki kmetiji bo možno dodati izdelke z njihovimi podatki in slikami. Oglasi bi bili prikazani pregledno in na kratko predstavljeni na eni strani, ob kliku pa bi se prikazali vsi podatki in slike. Lahko bi bili prikazani na različne načine in sicer: iskanje po različnih vnosnih podatkih izdelka, filtriranje po tipih, lokaciji in pridelavi. Nenazadnje pa tudi sortiranje vseh prikazanih izdelkov po pomembnejših podatkih. Modul bi vseboval tudi statistični pregled, ki bi prikazoval trenutno povprečno ceno (v evrih) in trenutno povprečno zalogo vseh izdelkov, združenih po posameznih enotah. Spremembe cen in zalog izdelkov bi bile hranjene v zgodovini, s pomočjo katere bi se grafično ali tabelarično prikazalo kako sta se spreminjali količina in cena. Prav tako, kot pri Tržnici bi statistični pregled imel možnost izbiranja izdelkov in posledično bi bilo moč spremljati zgodovino za vsako vrsto izdelka posebej, njegovo ceno in količino.

Poglavje 2

Razvoj spletnih aplikacij

Pri razvoju spletnih aplikacij poznamo več različnih pristopov. Ker razvoj ni samo programska koda, ampak tudi logistika razporejanja delovnih procesov oziroma razvojnih procesov, so se izoblikovali različni modeli. Poleg pristopov je zelo pomembna arhitektura, kjer ločimo različne plasti, izmed katerih ima vsaka določeno funkcionalnost oziroma namen. Z več plastmi tudi skrijemo strukturo spletne aplikacije pred uporabnikom.

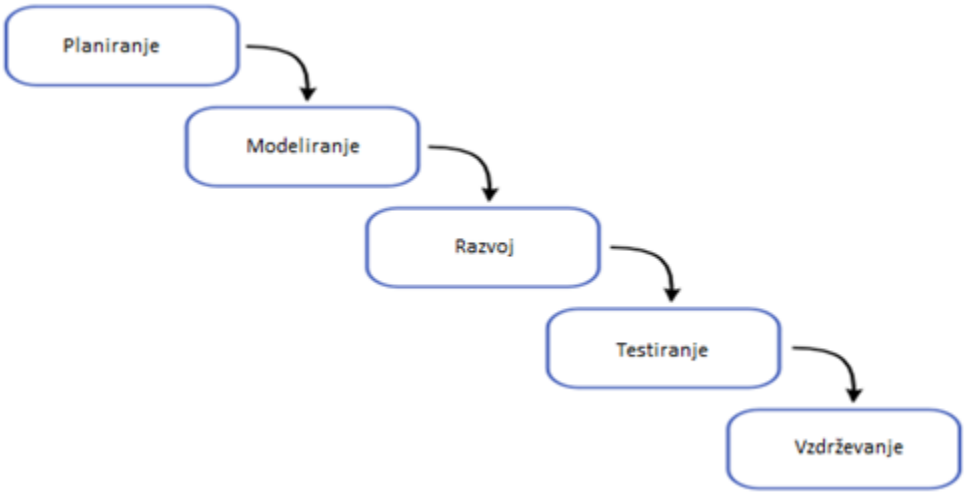
2.1 Splet

Splet (angl. web ali WWW) je sistem za hranjenje in upravljanje informacij, ki omogoča izmenjavo in dostop do podatkov preko računalniškega omrežja, imenovanega internet [3].Podatki so lahko predstavljeni kot običajna spletna stran, kjer so vključeni posamezni dokumenti s statično informacijo. V primeru uporabe dinamičnih podatkov, ki se obdelujejo in shranjujejo na strani različnih ponudnikov storitev, pa je potrebno vključiti spletno programiranje.

2.2 Načrtovanje razvojnih procesov

Načrtovanje procesov, uporabljenih pri razvoju aplikacije, je z vidika uspešnega programiranja korak naprej. Večja aplikacija brez načrtovanja procesov ne

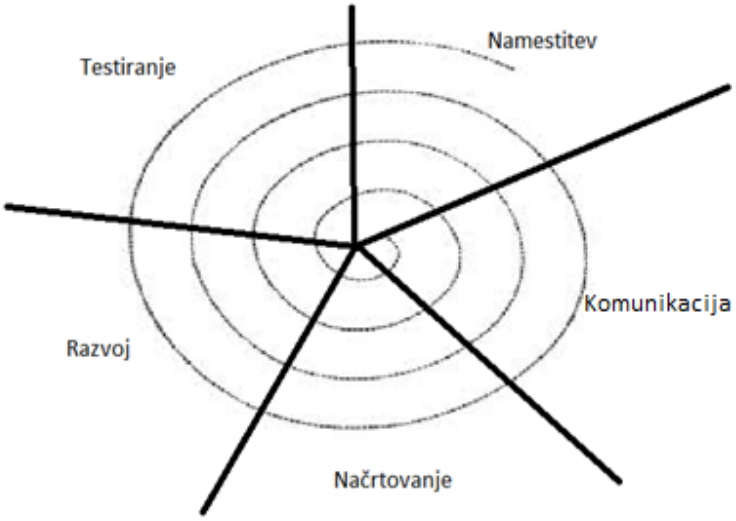
more biti razvita optimalno. Tudi če je programska koda v skladu z zahtevami, je število porabljenih ur pri razvoju aplikacije brez definiranega načrtovanja veliko večje kot pri aplikacijah z definiranim načrtovanjem, kar s poslovnega vidika predstavlja problem. Tovrstnemu programiranju pravimo ekstremno programiranje. Poznamo več različnih modelov, po katerih so definirani procesi razvoja programske opreme. Eden prvih je tako imenovani slapovni model (angl. the waterfall model), ki vključuje razvojne procese, kot so planiranje, modeliranje, razvoj, testiranje in vzdrževanje (prikazano na sliki 2.1). Celotna aplikacija je narejena v eni iteraciji in poteka skozi naštetih pet procesov. Vendar se je sčasoma pokazalo, da slapovni model pri večjih projektih ne zadostuje za opis realnih situacij. Za kaj takega bi, da bi dosegli želeno, potrebovali več iteracij skozi posamezen proces razvoja. Primer je projekt, katerega predviden čas za izdelavo je daljši od dveh let. Za tako dobo je težko vnaprej predvideti, kaj vse bomo potrebovali in kako bomo vsako funkcionalnost realizirali.



Slika 2.1: Slapovni model.

Kot rešitev slednjega problema se je pojavil spiralni model (angl. the spiral model), ki se v osnovi ne razlikuje kaj dosti od slapovnega modela, vendar vsebuje več iteracij skozi posamezne procese razvoja. Spiralni model

prikazuje slika 2.2.



Slika 2.2: Spiralni model.

V modulu, ki je predstavljen v diplomskem delu, je uporabljen spiralni model s sledečimi procesi: načrtovanje, razvoj, testiranje, namestitev in komunikacija. V začetku je bilo največ iteracij skozi načrtovanje, saj se je po začetnem razvoju pokazalo, kaj je treba spremeniti. Skozi večino projekta so se ponavljale iteracije načrtovanja in razvoja. Po vsaki razviti funkcionalnosti je bila le-ta testirana in po potrebi popravljena. Proti koncu pa je bilo več korakov skozi proces namestitve in komunikacije, ker je bil modul objavljen na strežniku, kjer se je pokazala njegova dejanska zmožnost - v smislu hitrosti delovanja in obnašanja na spletnem strežniku. V primeru kakršnih koli popravkov je bil ponovno uporabljen proces komunikacije.

2.3 Večnivojska arhitektura

Velike aplikacije, kot so portali in aplikacije za e-trgovanje, so izpostavljene velikemu številu sočasnih zahtev. V ta namen mora biti zagotovljen visok nivo zmožnosti zagotavljanja storitve. Modularna, večnivojska arhitektura ponuja okolje, v katerem so lahko različne aplikacijske komponente zaradi

zagotavljanja takih zahtev podvojene. Arhitekture večjih spletnih aplikacij so po navadi razdeljene na tri nivoje:

- Predstavitvena plast (angl. the presentation layer) služi zbiranju zahtev odjemalca. Omogoča dinamično kreiranje HTML-strani, prikazane uporabniku, in sicer glede na podatke, ki jih vnese odjemalec, nato se prožijo zahteve različnih skript na nižjem nivoju - to je aplikacijska logika.

- Plast aplikacijske logike (angl. the application logic layer) je najpomembnejša plast; vsebuje celotno logiko spletne aplikacije. Odjemalec na višjem nivoju kliče metode, definirane v plasti aplikacijske logike. Ta izvaja operacije s podatki, pridobljenimi iz še enega nivoja nižje, kjer se le-ti nahajajo.

- Plast za upravljanje virov (angl. the resource management layer) omogoča dostop do podatkovnih baz in pregled želenih podatkov.

Za poganjanje večnivojskih arhitektur potrebujemo okolje, ki to podpira. Temu služijo aplikacijski strežniki (angl. application servers), s pomočjo katerih se učinkovito izvajajo procesi med spletnim strežnikom in plastjo za upravljanje virov [1].

2.4 Varnost spletnih aplikacij

Uporabniki imajo lahko različne dostope do podatkov modula. Nekateri lahko le pregledujejo podatke, medtem ko jih drugi lahko urejajo in brišejo. Da ločimo različne tipe uporabnikov, je potrebna identifikacija. Identifikacija je proces za preverjanje uporabnikovega dostopa do podatkov. Protokol HTTP ima že vgrajeno podporo za osnovno identifikacijo, kjer se uporabniško ime in geslo prenašata preko identifikacijske glave (angl. authentication header). Uporabniško ime in geslo se prenašata kot niz znakov; tako tovrstno prenašanje ni varno, razen če poteka preko varne povezave (angl. secure connection). Večina aplikacij ima razvito svojo shemo za prenos uporabniškega

imena in gesla, ki teče v ozadju osnovne HTTP-identifikacije [3].
Zaradi zagotavljanja večje varnosti se gesla kriptirajo. V ta namen je Agencija za nacionalno varnost (angl. Nacional Security Agency) leta 1993 v Združenih državah Amerike objavila algoritem za kriptiranje, imenovan SHA-1 (Secure Hash Algorithm) [9], katerega izhodna velikost je 160 bitov, največja velikost sporočila pa je 2^{64} - 1 bitov. Možne operacije nad algoritmom SHA-1 so logični in, ali, ekskluzivni ali, rotacija in dodajanje. Možnost uspešnega napada je 1:251 [8]. Naslednik algoritma SHA-1 je SHA-2, ki vsebuje dva algoritma, z različnima dolžinama blokov. Znana sta pod imenoma SHA-256 (uporablja 256 bitov) in SHA-512 (uporablja 512 bitov).

2.5 Podatkovna baza

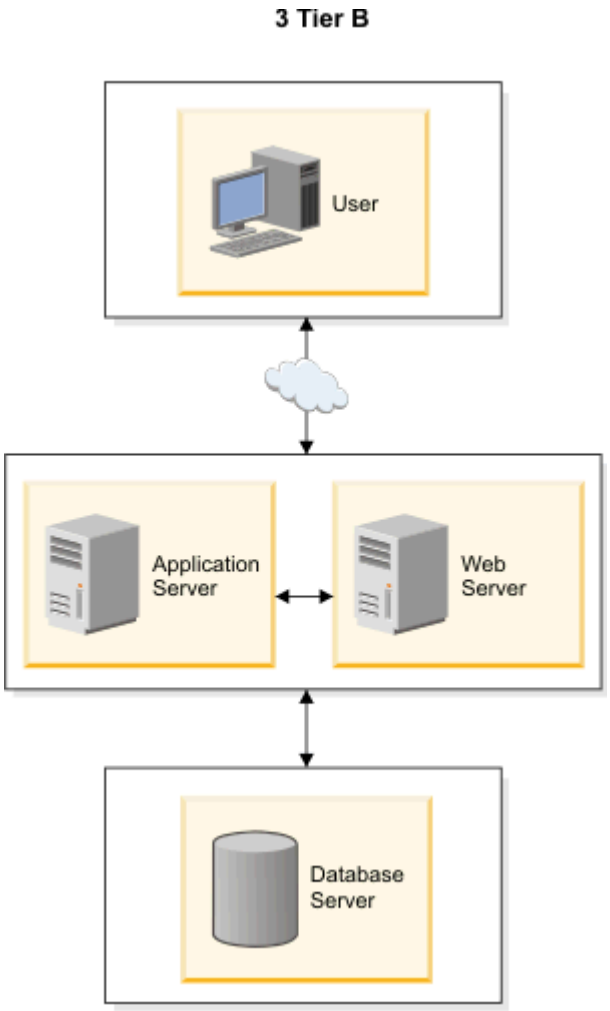
V razvoju projekta so bila uporabljena različna orodja. Za implementacijo in postavitev baze sem uporabil Microsoftovo orodje Microsoft SQL Server Management Studio 2008, katerega osnovna jezika sta T-SQL in ANSI-SQL in ki podpira relacijske modele podatkovnih baz. Vsebuje vse potrebne funkcionalnosti za vzpostavitev podatkovne baze, vnašanje tabel, relacij med tabelami in navsezadnje za vnos podatkov v podatkovno bazo. Dostop do baze je implementiran s trinivojsko arhitekturo (slika 2.3), s katero dejansko strukturo podatkovne baze skrijemo uporabnikom.

Tri-nivojska arhitektura se deli na [4]:

- odjemalca: tu se izvaja uporabniški vmesnik in pošilja zahteve odjemalca,
- aplikacijski strežnik: ta je zadolžen za procesiranje,
- podatkovni strežnik: ta shranjuje podatke, izvaja shranjene procedure in skrbi za integriteto in varnost podatkov.

Prednost tovrstne arhitekture, ki je prikazana na sliki 2.3, je, da uporabnik lahko prilagodi pogled podatkov, ne da bi s tem vplival na arhitekturo;

poleg tega pa uporabnik ne vidi fizične zgradbe podatkovne baze, saj je odjemalec na povsem drugem nivoju kot sama implementacija baze. Fizično zgradbo vidi samo administrator, ki jo lahko po potrebi spreminja, pri čemer pa ne spremeni pogleda uporabnikov.



Slika 2.3: Tri-nivojska arhitektura (vir: [11]).

2.6 Programsko okolje

Programsko kodo sem napisal v okolju Microsoft Visual Studio 2008, v programskem jeziku ASP .NET C#. Uporabljenih je tudi nekaj Telerikovih [10] gradnikov AJAX.
V okolju Microsoft Visual Studio je integrirano razvojno okolje (IDE) podjetja Microsoft. Uporablja se za razvoj konzolnih in grafičnih uporabniških vmesnikov, spletnih strani, storitev in drugega. Zasnovan je tako, da ima podobo spletne aplikacije in je ločen od programske kode, ki določa funkcionalnost. To je doseženo z dvema različnima dokumentoma, kjer je v enem samo programska koda, ki definira podobo, v drugem pa samo logistika, ki teče na strežniku. Omogoča pisanje v več programskih jezikih. Poleg uporabljenega C# imamo integrirane VisualBasic .NET, Visual F# in C/C++. Programski jeziki, kot so Microsoftov programski jezik M, Phyton in Ruby, pa so na voljo v jezikovnih paketih. Podprti so tudi jeziki, kot so XML/XSLT, HTML/XHTML, JavaScript in CSS. Ima že veliko uporabnih gradnikov (gradnik za vnos besedila, sezname, gumbe ...), ki sem vključil v projekt, kjer je bilo potrebno; večina gradnikov pa je Telerikovih. Telerik je korporacija, ki ima na voljo različne plačljive gradnike, temelječe na AJAX-u (Asynchronous JavaScript). V veliki meri sem gradnike Telerika uporabljal za prikaz podatkov, ki predstavljajo zaključeno celoto (npr. prikaz vseh izdelkov - RadListView).

Za branje podatkov iz baze ima okolje Microsoft Visual Studio integrirano Language Integrated Query (LINQ). Uporablja se namesto jezika SQL.

2.7 Spletne tehnologije

Spletne tehnologije (angl. web technologies) odjemalcem omogočajo komunikacijo med brskalnikom in spletnim strežnikom, na katerem je naložena spletna stran. Spletna stran se prenaša preko protokola HTTP, ki sprejema zahteve (angl. request) odjemalca, strežnik pa glede na vsebino zahteve odjemalcu posreduje odgovor (angl. response). Osnovni jezik za oblikovanje vse-

bine spletne strani je HTML, ki definira format, velikost, stil teksta in drugo. Omogoča vključevanje slik in drugih multimedijskih kontrol. Za večjo preglednost programske kode se je pojavil CSS, ki slog prikaza definira ločeno od vsebine. Tako je dosežena večja preglednost nad HTML-dokumentom. Vendar lahko večji HTML-dokumenti, ne glede na ločeno vsebino in stil, postanejo nepregledni. Kot rešitev tega problema se je pojavil XML, ki omogoča strukturiranje podatkov [1].

2.7.1 Upravljalniki URI

Upravljalniki URI so posebni mehanizmi, namenjeni procesiranju HTTP-zahtev in prikazu zahtevanih podatkov. Natančneje, URI je namenjen identifikaciji instance, ki upravlja zahtevo. Instanca upravljalnik URI - posreduje zahtevo za procesiranje. Rezultat zahteve je nato posredovan nazaj spletnemu strežniku, ta pa posreduje podatke uporabniku [2].

2.7.2 Active Server Page (ASP)

ASP ali klasični ASP (angl. classic ASP) je prvi Microsoftov skriptni pogon na strani strežnika za generiranje dinamičnih spletnih strani. Spletne strani s končnico .asp brskalnik interpretira drugače kot običajne spletne strani s končnicama, kot sta .html in .htm. Ob zahtevi za ASP-stran se najprej izvede koda na strani strežnika, kar pa ni vidno v brskalniku. Ko je koda za prikaz zgenerirana, to odjemalec vidi v brskalniku [5]. ASP-strani ne morejo biti zagnane na katerih koli strežnikih, ampak mora biti tovrstna stran posredovana preko posebnega Microsoftovega strežnika, ki omogoča ASP. Privzet programski jezik za ASP je VBScript, vendar lahko spremenimo jezik z ukazom *<script language="language" runat="server" >*, kjer je parameter *language* programski jezik v kateremu želimo pisati [6].

2.7.3 Seja

Seja predstavlja povezano zaporedje HTTP-zahtev med uporabnikom in spletnim strežnikom [2]. Ko se seja kreira, se ji določi ID-seje, s katerim se uporabnik identificira na vsaki podstrani. Z uporabo seje preprečimo, da bi bilo potrebno vpisovanje uporabniškega imena in gesla na vsaki podstrani spletne strani oziroma aplikacije, in sicer v primeru, da gre za stran, ki potrebuje prijavo.
Ker je vsaka podstran spletne strani instanca sama zase, nimamo globalnih spremenljivk, ki bi bile na voljo čez celotno spletno aplikacijo oziroma stran, lahko pa uporabimo sejo kot način posredovanja podatkov med podstranmi. Vrednosti, ki so shranjene v seji, lahko preberemo, kadarkoli je seja aktivna. Ko se seja uniči, se z njo uničijo tudi vsi podatki in spremenljivke, shranjeni v njej. Trajanje same seje je odvisno od potreb spletne strani oziroma aplikacije; lahko traja, dokler se uporabnik ne odjavi s spletnega mesta, ali pa poteče po določenem času, ki so ga nastavili razvijalci.

Seje v ASP .NET
Spremenljivke seje so shranjene v zbirki SessionStateItemCollection, do katere dostopamo preko HttpContext.Session. V zbirko lahko shranjuje podatke po principu ključ - vrednost ali indeks - vrednost [7].
Primer:

```
Session["key"] = value.
```

Podatke, ki so shranjeni v seji, je treba ob branju pretvoriti v pravilen tip spremenljivke, ki ga podatki predstavljajo. Če smo v sejo shranili spremenljivko tipa Integer, moramo tudi podatke shraniti v spremenljivko tega tipa; če je na primer v seji celoten ArrayList, pa jih moramo shraniti v obliki ArrayList.
Primer:

```
ArrayList sessionValues = (ArrayList) Session["key"].
```

2.7.4 Piškotki (angl. cookies)

Piškotki so majhne tekstovne datoteke, ki hranijo informacije strežnika. Nahajajo se na odjemalčevem računalniku. Zapisane so v obliki ključ - vrednost. Prenašajo se preko glave dokumenta HTTP-zahteve. Piškotke lahko uporabljamo celoten čas njihove življenjske dobe, ko pa ta poteče, do vsebine piškotka ni več možno dostopati. Poznamo tri tipe piškotkov:

- Sejni piškotki (angl. session cookies) se lahko uporabljajo samo med eno sejo. Ko se ta konča, se uniči tudi piškotek. Ob vnovičnem obisku strani pa se generira nov, z novo vsebino.
- Začasni piškotki (angl. temporary cookies) imajo določeno življenjsko dobo in so na voljo le do njenega preteka. Doba se določi ob generiranju in se zapiše v piškotek. Ko piškotek preteče, se izbriše iz odjemalčevega trdega diska. Dobe so lahko na primer dan, teden, mesec, leto ali več.
- Stalni piškotki (angl. Permanent Cookies) nimajo življenjske dobe in se nikoli ne izbrišejo avtomatsko (lahko jih izbrišemo ročno), zato jih lahko uporabljamo nedoločen čas.

Prednost uporabe piškotkov je prenos ID-seje (angl. session ID) med odjemalcem in strežnikom na uporabniku povsem neopazen način. Poleg tega niso niti prostorsko niti časovno potratni za prenos, saj se prenaša le majhna količina podatkov [2].

Poglavje 3

Tržnica

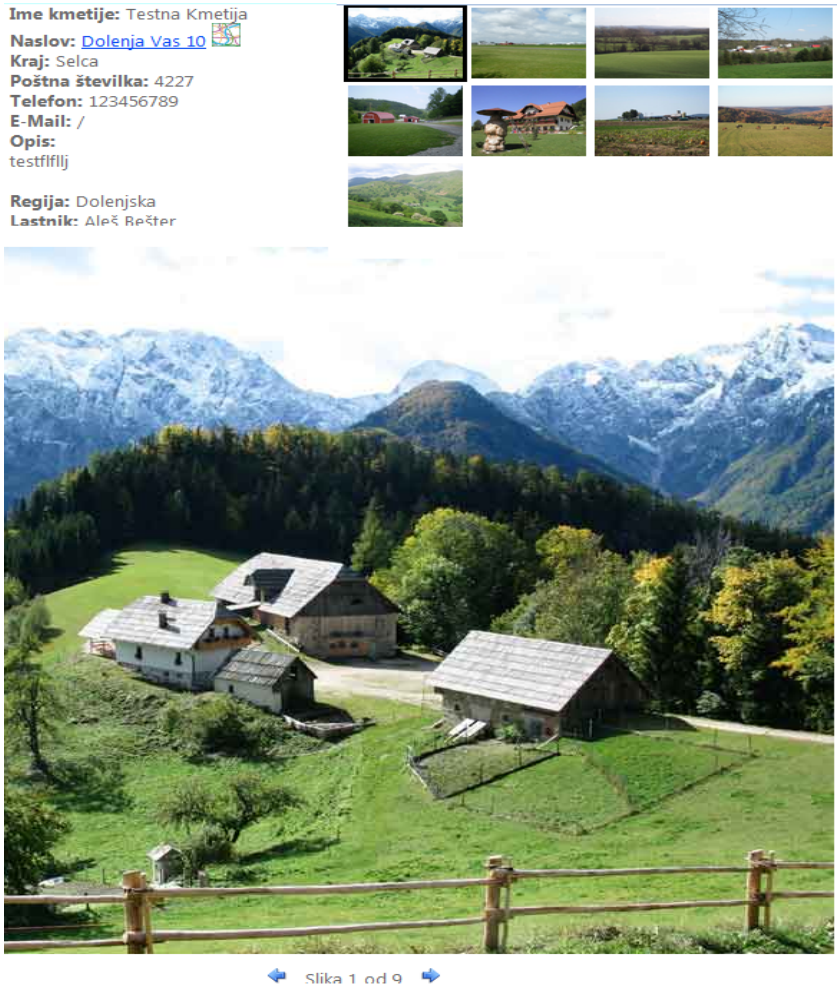
Tržnica je modul spletne aplikacije Kmetija, ki je namenjen oddajanju in pregledovanju malih oglasov o kmetijskih izdelkih. Omogoča več različnih uporabnikov, in sicer: neregistriran uporabnik, registriran uporabnik in administrator. Vsaka skupina uporabnikov ima različne dostope do podatkov v podatkovni bazi. Prikaz podatkov o kmetiji je na sliki 3.1. Vsebuje tudi statistični pregled izdelkov, in sicer po različnih podatkih, ki so prikazani v tabelarični ali grafični obliki.

Neregistrirani uporabniki lahko pregledujejo podatke o kmetijah, izdelkih in statistični pregled, kjer je prikazana zgodovina sprememb za posamezne tipe izdelkov. Spremembi, ki se beležita v zgodovini, sta cena in zaloga. Glede na spremembe se nato pri statističnem pregledu izriše graf. Pregled izdelkov omogoča filtriranje po tipu izdelka, pridelavi in regiji. Filtriranje je lahko po posameznem atributu, lahko pa je tudi po več atributih hkrati. Na primer: če uporabnik izbere filtriranje po sadju, gorenjski in ekološko, bo prikazalo samo izdelke, ki so tipa sadje, prihajajo iz gorenjske regije in so pridelani ekološko. Imamo tudi možnost iskanja po ključnih besedah, vpisanih v vnosno polje poleg gumba iskanje.

Registrirani uporabniki imajo poleg vseh funkcionalnosti pregledovanja še možnosti dodajanja, urejanja in brisanja kmetije ter izdelkov kmetij. Vsaki kmetiji in izdelku je možno dodati tudi več slik in le-te odstraniti. Med

registrirane uporabnike spada tudi administrator.

Administrator ima dostop do vseh funkcionalnosti aplikacije. Ne glede na to, da ni lastnik nobene kmetije, ima dostop do strani za urejanje in brisanje. Poleg funkcionalnosti registriranih uporabnikov pa administrator lahko pri filtrih dodaja, ureja in briše možnosti, in sicer v kombiniranih izvlečnih listah. Obenem ima vpogled v podatke uporabnikov, ki jih lahko vse tudi ureja - z izjemo gesla; tega pa lahko ponastavi na novo začetno vrednost.



Slika 3.1: Prikaz podatkov o kmetiji [13].

3.1 Podatki

Namen Tržnice je objava in predstavitev kmetij, izdelkov in njihovih slik v lasti registriranih uporabnikov. Uporabniki morajo biti lastniki določene kmetije, ki mu izdelek dodajajo. Vsak uporabnik je lahko lastnik ene ali več kmetij. Kmetije so predstavljene z naslednjimi podatki:

- ime kmetije,
- naslov,
- kraj,
- poštna številka,
- telefonska številka,
- e-pošta,
- opis,
- regija,
- lastnik.

Obvezna so vsa polja, razen e-pošte. Vsak izdelek pa je opisan z naslednjimi podatki:

- tip izdelka (sadje, zelenjava, suhomesni izdelki...),
- način pridelave (ekološka, konvercionalna, integrirana ali ostalo),
- regija (Gorenjska, Notranjska, Dolenjska, Koroška...),
- način prodaje (na veletržnici Vižmarje, Plemljeva 2; na zalogi pri ponudniku; po dogovoru in po prednaročilu).

Naštete podatke se lahko izbere iz že vnaprej pripravljenih možnostih.

- opis izdelka,

- cena,
- datum objave oglasa,
- datum izteka oglasa (ko se oglas izteče, izdelek pri prikazu ni več viden),
- zaloga.

Sledeče pa vnesemo sami preko vnosnega polja.

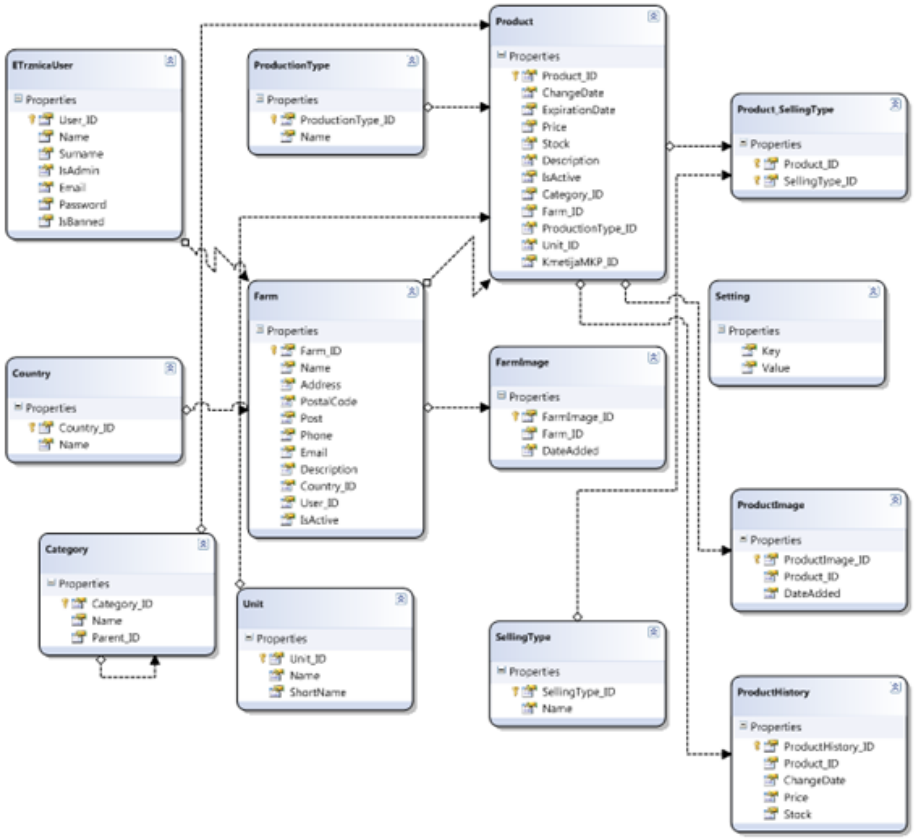
Podatkovna baza je realizirana v programu Microsoft SQL Server Management Studio 2008. Orodje je zelo uporabno, namenjeno je razvijanju spletnih aplikacij v ASP.NET, saj omogoča direkten izvoz podatkovne baze v dokument tipa .dbml (prikazano na sliki 3.2), ki ga lahko uvozimo v naš projekt. Ko so ustvarjene potrebne tabele in relacije med njimi, jih izvozimo v datoteko tipa .dbml in vključimo v projekt ASP.NET. Najprej kreiramo datoteko LinqToSQLClasses tipa .dbml, kjer je predstavljena podatkovna baza oziroma tabele, ki jih sami dodamo. Da dodamo tabelo, moramo definirati naslov strežnika, na katerem se podatkovna baza nahaja. To storimo v Raziskovalecu projekta (angl. solution explorer), kjer vnesemo naslov za dostop in geslo podatkovne baze. Ko je povezava vzpostavljena, se prikažejo vse tabele, ki se so na voljo in jih je moč povleči (t. i. drag and drop) v dokument .dbml. Relacije med njimi se generirajo same, glede na relacije v podatkovni bazi. Ko so tabele v dokumentu .dbml, so pripravljene za uporabo. Do podatkovne baze dostopamo preko DataBaseDataContext-a. Primer vnosa podatkov za povezavo do podatkovne baze prikazuje slika 3.3.

Na Tržnici je za to uporabljen kodirni algoritem SHA-1, ki se uporablja za kodiranje uporabnikovih gesel. SHA-1 je naslednik predhodnega algoritma, imenovanega SHA-0, s katerim so odpravili napako, zaradi katere se ni SHA-0 nikoli dobro uveljavil. V nasprotju s svojim predhodnikom se je SHA-1 zelo razširil in je uporabljen v mnogih aplikacijah na svetovnem spletu.

Primer kriptiranja z algoritmom SHA-1:

```
Vhod: geslo123
```

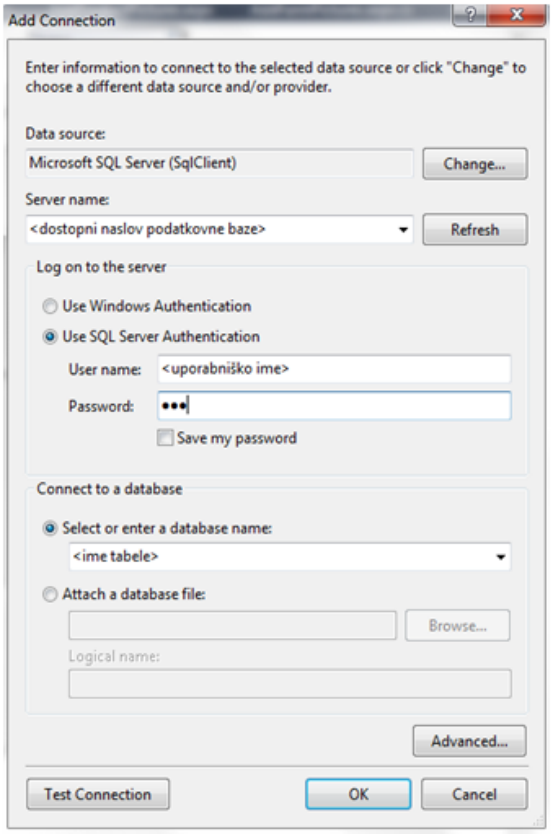
```
Izhod: 4135b080fefb09efd61b7ccf07e8d81e5027a202
```



Slika 3.2: Datoteka .dbml, z vsemi tabelami na Tržnici.

3.2 Funkcionalnosti modula Tržnica

Osnovne funkcionalnosti Tržnice so pregled, urejanje, dodajanje in brisanje podatkov. Ker pa so neregistrirani uporabniki potencialni registrirani uporabniki, je pregled razvit na dva načina, da so pregledneje predstavljeni kmetije in izdelki.



Slika 3.3: Vnos podatkov za povezavo na podatkovno bazo.

3.2.1 Pregled

Osnovna funkcionalnost modula, ki temelji na objavah, je seveda pregled. Vsaka kmetija ali izdelek, ki ga doda uporabnik, mora biti nekje prikazan, če ne vsem uporabnikom pa vsaj administratorju. Tako kot je v navadi, imata neregistriran uporabnik in administrator nekoliko ločena pogleda na objave izdelkov in kmetij. Slednji lahko ureja vse podatke o izdelkih, kmetijah, pripadajočih slikah in tako naprej, medtem ko neregistriran uporabnik nima dostopa do strani za manipulacijo s podatki. Nekje vmes pa je registriran uporabnik, ki ima (lahko) le nekaj administratorskih pravic, in sicer če je lastnik kmetije. V tem primeru lahko ureja kmetijo in vse njene pripadajoče izdelke.

Pregled podatkov je prikazan na dva načina:

- celovit, kjer so prikazane vse kmetije oziroma izdelki brez pripadajočih slik,
- podroben, kjer so prikazani vsi podatki o kmetiji.

3.2.1.1 Celovit način

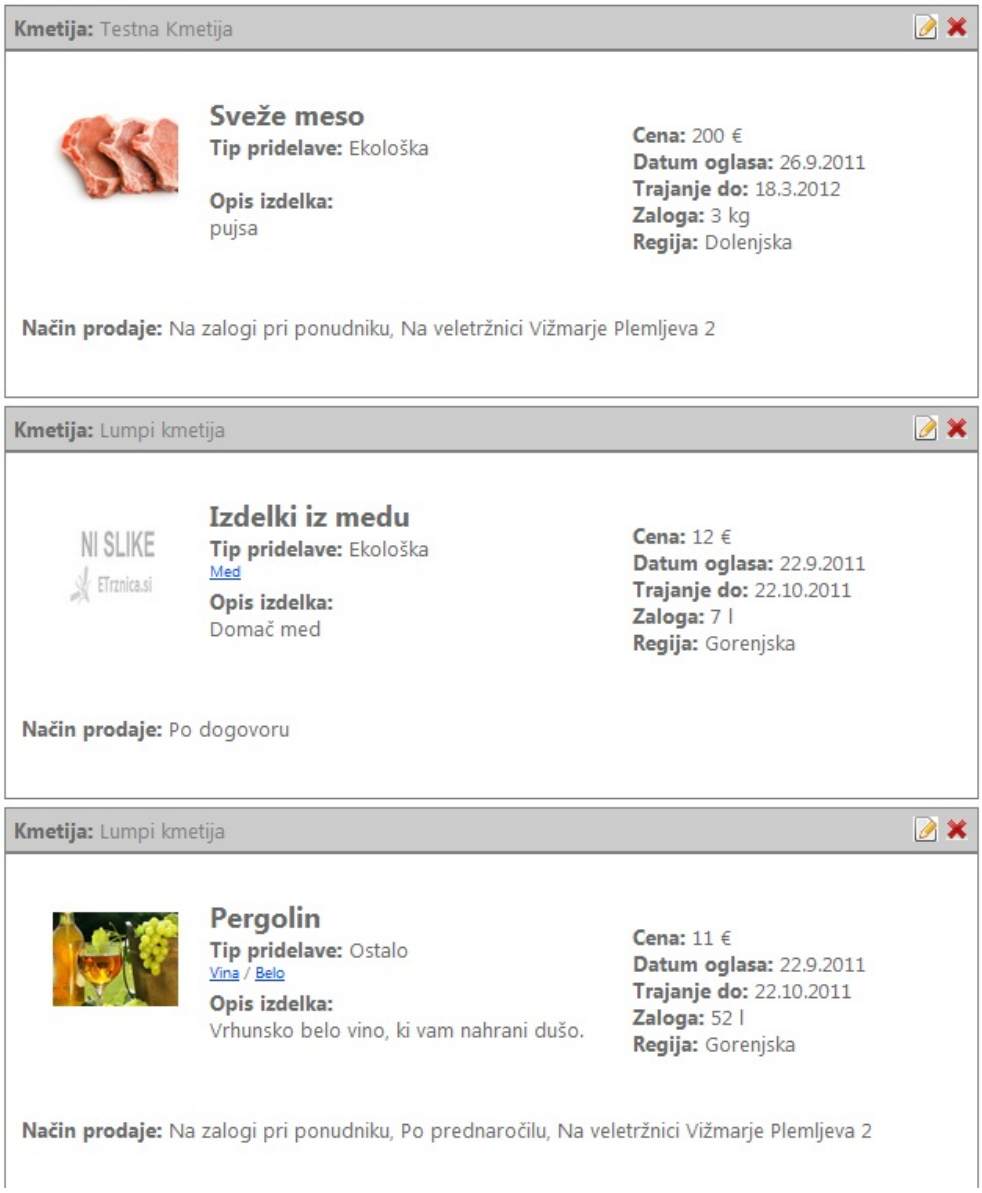
Celovit način prikazuje več instanc različnih kmetij in le nekaj osnovnih podatkov o njih. Zraven je prikazana še slika, ki se izbere naključno izmed vseh slik, ki pripadajo določeni kmetiji. Če nima nobene naložene slike, se prikaže vnaprej pripravljena; ta je skupna za vse kmetije, ki nimajo dodanih slik. Ta del je realiziran s pomočjo Telerikovega gradnika RadListView, ki je neke vrste tabela in ima v vsaki celici eno izmed kmetij. RadListView deluje tako, da je vključen v našo stran .aspx, kjer določimo podobo in potrebne parametre za prikaz podatkov. RadListView sestavlja predstavitvena predloga (angl. layout template), kjer določimo ploščo oziroma površino, na katero nameravamo postaviti gradnike in kontrole. Plošči se lahko med drugim določi upravitelja strani (angl. RadDataPager), ki služi za število prikazanih elementov na posamezni strani RadListView-a. Naslednja in ob enem najbolj

pomembna komponenta RadListView-a je podatkovna predloga (angl. item template), s katero so določeni podatki prikazani uporabniku. Na Tržnici so to oznake, gumbi, slike, hiperpovezave in drugo. V podatkovni predlogi je definirana tudi podoba oziroma način postavitve elementov - kakor so elementi oblikovani, tako so prikazani. Oblikovanje elementov je določeno s kodo CSS: v ločeni datoteki ali pa neposredno v podatkovni predlogi RadListView-a. Zadnji od pomembnejših elementov je prazna predloga (angl. empty data template). V njej je določen in oblikovan prikaz, če v RadListView-u ni podatkov za to. V primeru Tržnice je to samo ena oznaka, ki ji je določen primeren tekst. Celovit način je prikazan na sliki 3.4.

Ko je vse potrebno za prikaz podatkov pripravljeno, se lahko v kodi ozadja (angl. code behind) realizira funkcionalnost. To so dokumenti s končnico .aspx.cs. Prva stvar pri RadListView-u je določitev vira podatkov (angl. RadViewListDatasource) s poizvedbo v jeziku Linq. Primer nastavljanja vira z Linqom je prikazan na sliki 3.5:

RadListView ima lastnost OnNeedDataSource, ki je določen v kodi .aspx, potrebuje pa ime metode v ozadju, v kodi .aspx.cs.
Primer:
`OnNeedDataSource="rlvFarms_NeedDataSource"`.

Da se vrnemo na poizvedbo v Linq-u. V metodi se nastavi vir na poizvedbo (*_db.Farms.Where(f =>f.IsActive).OrderBy(o =>o.Name)*), kjer je _db instanca podatkovne baze, ki je tipa DataBaseDataContext in je globalna spremenljivka. Z _db.Farms se določi za katero tabelo na katero tabelo v bazi se sklicuje. Stavek Where ima enak pomen, kot pri jeziku SQL in sicer, določa pogoj, za izbor podatkov. Na Tržnici je to IsActive, kar pomeni, da bodo v poizvedbo vključene samo vrstice iz podatkovne baze, ki imajo stolpec IsActive nastavljen na vrednost *true*. Podobno kot Where ima tudi OrderBy enak pomen, kot ga ima v jeziku SQL. Gre za razvrščanje podatkov po različnih stolpcih.



Slika 3.4: Celovit način prikaza, ki uporablja RadListView.

```
protected void rlvFarmPictures_NeedDataSource(object sender,
RadListViewNeedDataSourceEventArgs e)
{
    rlvFarmPictures.DataSource = _db.FarmImages.Where(f => f.Farm_ID == _farmID);
}
```

Slika 3.5: Primer nastavljanja vira podatkov gradniku RadListView.

Ko je vir podatkov pripravljen, je možno definirati prikaz podatkov v podatkovni predlogi. Podatki se pobirajo iz vira podatkov eden za drugim, in sicer preko metode OnItemDataBound, ki se kliče tolikokrat, kolikor je različnih vrstic v poizvedbi. V njej se določi oznake, slike, hiperpovezave, in drugo na želene vrednosti. Da se ve, za katero vrstico gre in kateri podatki so izpisani, je treba nastaviti spremenljivko tipa RadListViewDataItem na argumentno spremenljivko metode (e), in sicer na e.Item. V njej se ob vsakem klicu metode zamenja vrednost; kazalec se prestavi na naslednji zapis v viru RadListView.

Ko je vrstica, ki bo uporabljena, pripravljena, se naredijo še instance kontrol, definiranih v kodi .aspx - slika 3.6 prikazuje kodo .aspx, s katero se deklarira kontrola tipa asp Image:

```
<asp:Image ID="imgFarm" FarmID='<%# Eval("Farm_ID") %>'
runat="server" Width="160px" Height="120px" />
```

Slika 3.6: Deklariranje kontrole asp:Image.

Spodnji primer prikazuje kodo .aspx.cs, ki v kodi ozadja deklarira instanco kontrole tipa asp Image:

```
Image imgFarm=(Image)item.FindControl("imgFarm").
```

Gre za isto kontrolo in ko je enkrat nastavljena (s pomočjo metode FindControl (stringcontrolName)), se lahko določi še ostale lastnosti, ki se jih v .aspx kodi ni. To so npr. tekst, pot do slike, povezava hiperpovezave, itd., kot prikazuje slika (Slika 3.7).

Prav tako kot za nastavljanje vira je treba za metodo ItemDataBound v kodi .aspx določiti, na katero metodo naj se sklicuje lastnost ItemDataBound. Primer: `OnItemDataBound="rlvFarms_ItemDataBound"`.

3.2.1.2 Podroben način

Podroben način je način, kjer so izpisani vsi podatki o kmetiji, poleg pa so prikazane tudi slike, ki pripadajo kmetiji, če so le-te na voljo. Podatki o kmetiji so realizirani z oznakami (asp Label) in eno hiperpovezavo (asp

```
protected void rlvFarms_ItemDataBound(object sender, RadListViewItemEventArgs e)
{
    RadListViewDataItem item = (RadListViewDataItem)e.Item;
    Literal ltrDescription = (Literal)item.FindControl("ltrDescription");
    Image imgFarm = (Image)item.FindControl("imgFarm");
    int farmID = Convert.ToInt32(imgFarm.Attributes["FarmID"]);
    //random image
    int skipCount = GetRandomNumber(_db.FarmImages.Where(i => i.Farm_ID == farmID).Count());
    List<FarmImage> imgList = _db.FarmImages.Where(f => f.Farm_ID == farmID).ToList();
    if (imgList.Count == 0)
        imgFarm.ImageUrl = "~/Resources/Images/FarmIcon.png";
    else
    {
        FarmImage frmImage = imgList[skipCount];
        imgFarm.ImageUrl = "~/Files/FarmImages/" + farmID + "/" + frmImage.FarmImage_ID + ".jpg";
    }
    Farm farm = _db.Farms.SingleOrDefault(f => f.Farm_ID == farmID);
    if(farm != null)
        ltrDescription.Text = farm.Description.StripHtmlAndShortenString(200);
}
```

Slika 3.7: Metoda ItemDataBound.

HyperLink). Ker je tukaj prikazana samo ena instanca kmetije in je prikazana natanko ena kmetija, nam ni treba uvajati RadListView-a, ampak lahko vse oznake, ki jih potrebujemo, naredimo v kodi .aspx. Pozneje lahko v kodi .aspx.cs oznakam določimo pripadajočo vrednost.

Slika 3.8 prikazuje definiranje oznake in hiperpovezave v kodi .aspx

```
<b>Ime kmetije:</b>
<asp:Label ID="lblName" runat="server"/>
<b>Naslov:</b>
<asp:HyperLink ID="HyperLink1" runat="server"/>
```

Slika 3.8: Definiranje oznake in hiperpovezave v kodi .aspx.

Nastavitev vrednosti v kodi .aspx.cs prikazuje ukaz spodaj:

```
lblName.Text="farm.Name"
```

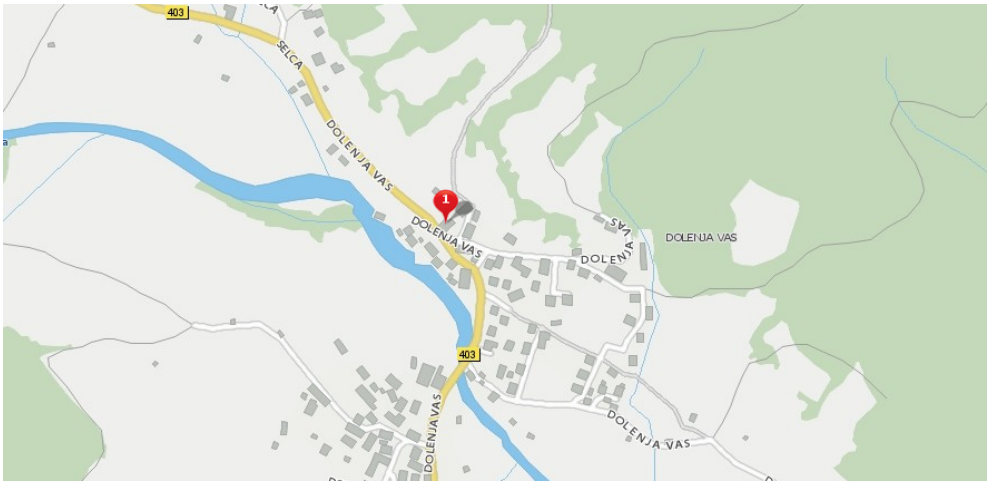
V spremenljivki farm je shranjena kmetija, katera se izpiše. Definirana je z Linq poizvedbo:

```
Farm farm = _db.Farms.SingleOrDefault(f => f.Farm_ID == _farmID).
```

Zgornja poizvedba vrača kmetijo z določenim ID v spremenljivki _farmID.

Hiperpovezava naslov kmetije, vsebuje povezavo na zemljevid Najdi.si (http://zemljevid.najdi.si). Ob kliku na naslov kmetije nas brskalnik preu-

smeri na zemljevid in prikaže točno lokacijo kmetije. Primer: slika 3.9, naslov Dolenja vas 49, 4227 Selca, se odpre povezava.



Slika 3.9: Lokacija kmetije na zemljevidu Najdi.si.

Povezava, ki nas preusmeri na določeno lokacijo na zemljevidu Najdi.si izgleda takole: http://zemljevid.najdi.si/search_maps.jsp?q=+Dolenja+Vas+10+4227. Sestavljena je iz naslova zemljevida Najdi.si ter naslova in poštne številke kmetije, vendar moramo namesto presledkov v nazivu kraja in poštne številke vstaviti plus ('+'). Slika 3.10 prikazuje kodo .aspx.cs, ki nastavi povezavo.

```
lblRegion.Text = farm.Country.Name;
lblUser.Text = farm.ETrznicaUser.Name + " " + farm.ETrznicaUser.Surname;
ltrDescription.Text = farm.Description;

string najdisiUrlS = "http://zemljevid.najdi.si/search_maps.jsp?q=";
string farmAddress = farm.Address.Replace(" ", "+");
string farmPostalCode = farm.PostalCode.Replace(" ", "+");
```

Slika 3.10: .aspx.cs koda, ki sestavi URL za prikaz lokacije na zemljevidu Najdi.si.

Slika 3.11: Prikaz vseh slik kmetije.

Prikaz slik na desni strani (slika 3.11) pa je realiziran z RadListView-om, ker je lahko za isto kmetijo več slik. RadListView ima v svoji podatkovni predlogi samo eno kontrolo, in sicer sliko (asp Image), ki se nastavlja preko lastnosti in metode OnItemDataBound. Podobno je prikazano tudi pri predstavitvi vseh kmetij. Razlika pri RadListView-u pri slikah pa je, da se lahko med slikami premikamo tudi s smernimi tipkami tipkovnice: ob kliku na določeno sliko se prikaže večja slika (velikosti 800×600 slikovnih točk), črna obroba pa se premakne na izbrano sliko. Izbira slik s smernimi tipkovnicami je realizirana s programom v JavaScriptu, kjer se preverja, ali je prišlo do vnosa iz tipkovnice; če je, se proži dogodek click() slikovnega gumba ibtnBack ali ibtnForward (aspImageButton). V dogodku click je sprogramirana logika za premikanje naprej in nazaj po vseh slikah - preverjanje, ali je izbrana slika

v novi vrstici ali je izbrana slika na koncu. Potem se obroba izbrane slike prestavi na prvo sliko v RadListView-u in v kontroli večje slike se prikaže prva slika. Postopek je obraten za premik s prve na zadnjo sliko.
Koda Javascript za premikanje med slikami s tipkovnico (slika 3.12):

```
document.onkeyup = keyCheck;

function keyCheck(e)
{
    if (e.keyCode == 37)
        document.getElementById('<%= ibtnBack.ClientID %>').click();
    else if (e.keyCode == 39)
        document.getElementById('<%= ibtnForward.ClientID %>').click();
}
```

Slika 3.12: Koda Javascript za premikanje po slikah s tipkovnico.

Lastnost keyKode se sklicuje na šifre tipk in šifri za gumb na tipkovnici. Torej, naprej in nazaj imata šifri 37 in 39.

Za premikanje po slikah brez tipkovnice sta realizirana tudi dva gumba, en za naprej in en za nazaj, in sicer pod veliko sliko. Ob kliku na en ali drug gumb se zgodi isto kot pri pritisku smernega gumba na tipkovnici - kliče se metoda click().

Pri podrobnem pregledu se spodaj, pod kontrolo za veliko sliko, nahajajo vsi izdelki, ki pripadajo izbrani kmetiji. Podoba pregleda izdelkov določene kmetije je enaka podobi izdelkov na prvi strani in ima tudi možnost urejanja po istih podatkih kot na prvi strani. Ne vsebuje pa filtriranja po tipu pridelka, tipu izdelave in regije, prav tako nimamo možnosti izbire načinov prodaje in iskanja po ključnih besedah.

Prikaza kmetij in izdelkov sta si zelo podobna. Pri obeh se prikaz deli na strnjen in podroben način in pri obeh gre za enak princip prikazovanja podatkov, le da se izpisujejo drugi podatki, in sicer glede na poizvedbe.

3.2.2 Dodajanje

Tako pregled kmetij kot pregled izdelkov sta si med seboj dokaj podobna, in sicer zato, ker sta oba realizirana na posebnih straneh, ki vsebujeta vnosna polja, preko katerih se vnese podatke o posamezni kmetiji. Vsako obvezno polje, tako kot pri urejanju, vsebuje preverjevalnike za obvezna polja in preverjevalnik za pravilen vnos. Razlika med urejanjem in dodajanjem nove kmetije ali izdelka je v bistvu samo pri zapisu v bazo in izpisu vnosnih polj. Pri dodajanju so vsa vnosna polja, kombinirane izvlečne liste in drugo prazni, za razliko od strani za urejanje, kjer se vnosna polja nastavijo na vrednosti, zapisane v bazi. Za zapis v podatkovno bazo shranimo (v instanco tipa Farm) že obstoječo kmetijo, ki ji določimo nove vrednosti. Po spremembi podatkov shranimo še nove vrednosti in urejanje je končano.
Primer dodajanja kmetije:

Definiranje nove insatance kmetije: `Farm farm = new Farm()`.
Dodajanje nove kmetije v podatkovno bazo:
`_db.Farm.InsertOnSubmit(farm)`.

Pri dodajanju novega zapisa v podatkovno bazo je treba določiti tabelo, v katero bo dodan nov zapis (Farms), nato pa še določiti, katera instanca se dodaja (farm). Ko sta tabela in instanca določeni, je treba potrditi spremembe v bazi.

Prav tako kot pri urejanju se ob uspešnem vnosu v podatkovno bazo na koncu izpiše statusna oznaka, ki pove, ali je bil vnos uspešen ali ne.

3.2.3 Urejanje

Urejanje podatkov o kmetijah se ne razlikuje dosti od urejanja podatkov o izdelkih. Do strani za urejanje lahko uporabniki, ki so lastniki kmetij ali administratorji, dostopajo preko kontrole slikovni gumb (asp ImageButton), ta pa ni vidna uporabnikom, ki niso lastniki ali administratorji. Urejanje deluje

tako, da se vsi podatki, vneseni v tekstovna polja, oziroma podatki, izbrani v kombiniranih izvlečnih listah, zapišejo v bazo. Ne preverja se, ali je bil podatek spremenjen, ampak se avtomatično prepiše stare podatke. Preverjanje podatkov sledi samo, če gre za spremembo pri zalogi ali ceni izdelka, saj se ti dve lastnosti izdelkov beležita v posebni tabeli v podatkovni bazi. Potrebujemo jih namreč pri realizaciji grafov, ki prikazujejo spreminjanje lastnost izdelkov na statističnem pregledu izdelkov. Obvezna polja imajo preverjevalnike polj (angl. field validators), ki nam v primeru nevnesenih ali napačno vnesenih podatkov preprečujejo shranitev v podatkovno bazo. Poznamo več različnih preverjevalnikov: preverjevalnike za prazna polja, preverjevalnike za pravilnost vnesene vsebine in druge. Na Tržnici se uporabljata le navedena. Vsak preverjevalnik mora imeti določeno skupino preverjanja. V isti skupini preverjevalnikov morajo biti vsa polja pravilno vnesena, drugače nam ni dovoljeno nadaljevati. Lastnost ErrorMessage določa tekst, ki se izpiše ob napačnem ali praznem vnosu v vnosno polje. Treba je dodati še kontrolo, na katero se preverjevalnik nanaša.
Primer definiranja preverjevalnika za obvezna polja, ki ga prikazuje slika 3.13:

```
<asp:RequiredFieldValidator runat="server" ID="rfvCategoryName"
ControlToValidate="rtbCategoryName" ErrorMessage="Vnesi ime kategorije."
 ValidationGroup="vgAddFile" CssClass="error"/>
```

Slika 3.13: .aspx koda preverjevalnika za obvezna polja.

Primer definiranja preverjevalnika za pravilno vnešeno vsebino je prikazan na sliki 3.14:

```
<asp:RegularExpressionValidator ID="regEmail" Display="Dynamic"
ErrorMessage="Napačna oblika E-mail naslova" CssClass="error"
ValidationGroup="vgReg" ControlToValidate="rtbEmail" runat="server"
ValidationExpression="^[a-zA-Z][\w\.-]*[a-zA-Z0-9]@[a-zA-Z0-9][\w\.-]*
[a-zA-Z0-9]\.[a-zA-Z][a-zA-Z\.]*[a-zA-Z]$"/>
```

Slika 3.14: .aspx koda preverjevalnika za pravilno vnešeno vsebino.

Sledeči ima tudi lastnost ValidationExpression, ki definira pravilen vnos

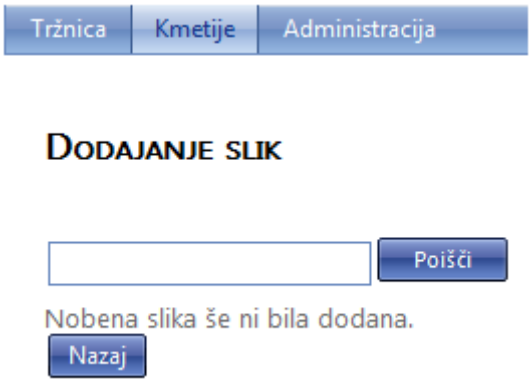
v vnosno polje. V zgornjem primeru je definirano pravilo, da mora biti v vnosu nujno znak '@'in pika '.'. Pred tema znakoma pa so lahko vse male in velike črke angleške abecede - brez posebnih znakov.

Če so vsi podatki vneseni pravilno, se izvede zapis v podatkovno bazo; v instanco kmetije oziroma izdelka se zapišejo vsi podatki v vnosnih poljih, nato pa z ukazom db.SubmitChanges() podatke v bazi nastavimo na novo vnesene zapise. Pri vsaki spremembi se na strani spodaj prikaže skrita oznaka (asp Label), ki sporoča, ali je bila sprememba uspešno shranjena ali ne.

3.2.4 Upravljanje s slikami kmetij in izdelkov

Dodajanje slik je realizirano na princip nalaganja slik na strežnik; pred shranjevanjem obrežemo na velikost 800×600 slikovnih točk. Obrezano sliko pa na koncu shranimo na strežniku v posebno mapo, rezervirano za slike.

Slike se nalagajo preko posebne Telerikove kontrole RadAsyncUpload, ki je namenjena nalaganju datotek. RadAsyncUpload je nastavljen tako, da se da dodajati samo slike s končnicami .jpg, .jpeg, .png, .bmp in .gif. Ko pritisnemo gumb poišči, se nam datoteke v trenutni datoteki filtrirajo, tako da se prikažejo samo slike z navedenimi končnicami. Podoba kontrole za nalaganje slik je prikazana na sliki 3.15.



Slika 3.15: Izgled kontrole za nalaganje slik.

Slika 3.16 prikazuje deklariranje nalagalnika datotek v .aspx kodi:

```
<telerik:RadAsyncUpload ID="ruAddPictures" runat="server"
 Localization-Select="Poišči" AllowedFileExtensions=
 ".jpg,.jpeg,.png,.bmp,.gif"
MultipleFileSelection="Disabled" MaxFileSize="5242880"
OnClientFileUploaded="ajaxRequestOnFileUploaded"/>
```

Slika 3.16: Deklariranje nalagalnika datotek v kodi .aspx.

Kot je vidno na sliki 3.16, ima nalagalnik omejitev izbire datotek, in sicer na eno datoteko, ker se takoj po naloženi sliki kliče metoda na strani klienta ajaxRequestOnFileUploaded, v kateri se proži dogodek na strani strežnika, imenovan ajaxRequest z argumentom FileUploaded.

Slika 3.17 prikazuje metodo ajaxRequestOnFileUploaded:

```
function ajaxRequestOnFileUploaded(sender, e)
{
    $find('<%= rapAddPictures.ClientID %>').
    ajaxRequest('FileUploaded');
}
```

Slika 3.17: Metoda ajaxRequestOnFileUploaded.

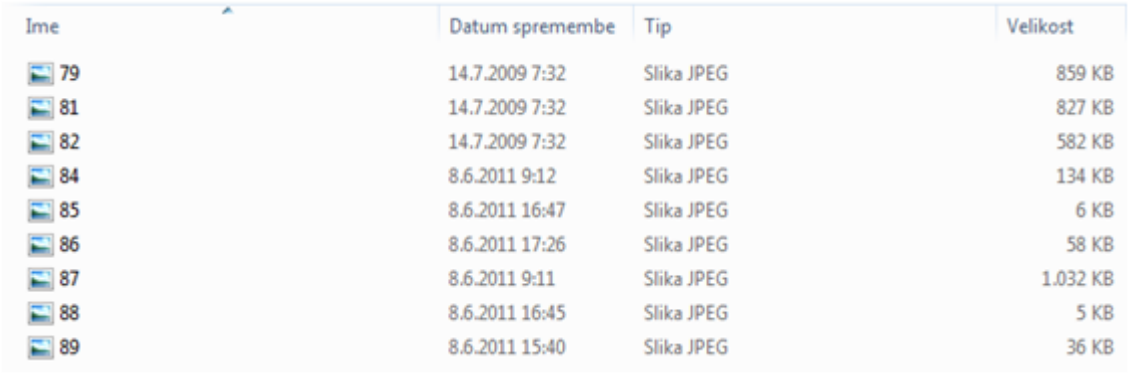
Z ukazom $find se poišče kontrolo, ki je prožila dogodek naložene datoteke. To je kontrola nalagalnika slik, imenovanega rapAddPictures. Na slednjem dogodku pa prožimo nov dogodek, imenovan ajaxRequest, pri katerem se na strani strežnika shrani obrezano sliko. Da modul ve, katerim kmetijam in katerim izdelkom pripadajo slike, se v podatkovno bazo zapiše nov zapis, ki določa ID-slike, ID-kmetije oziroma izdelka ter čas in datum nalaganja datoteke - to je prikazano na sliki 3.18.

	FarmImage_ID	Farm_ID	DateAdded
1	79	13	2011-08-09 14:09:53.800
2	81	13	2011-08-09 14:10:50.953
3	82	13	2011-08-09 14:11:01.970
4	84	13	2011-08-12 12:31:07.593
5	85	13	2011-08-12 12:31:12.283
6	86	13	2011-08-12 12:31:16.300
7	87	13	2011-08-12 12:31:21.663
8	88	13	2011-08-12 12:31:28.640
9	89	13	2011-08-12 12:31:33.720
10	96	22	2011-08-16 10:53:48.117
11	97	22	2011-08-16 10:58:33.573
12	98	20	2011-08-16 13:08:32.677
13	99	20	2011-08-19 10:34:33.963

Slika 3.18: Tabela za beleženje slik o kmetijah.

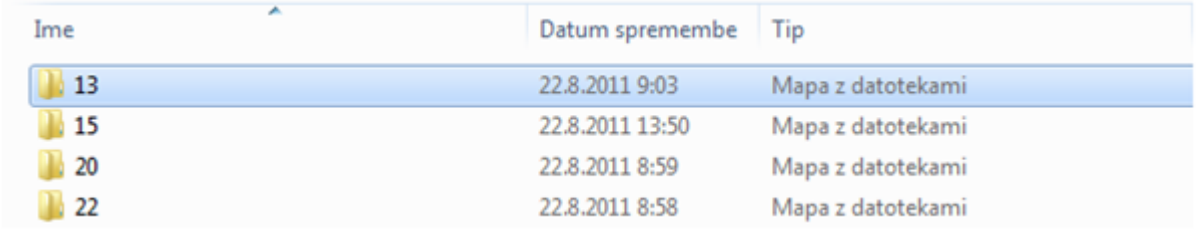
Sama datoteka - slika je shranjena v mapi na strežniku, iz katere se ob zahtevi prikažejo. Pot do posamezne slike na strežniku je: *~/Files/ProductImages/productID/productImageID.jpg*, kjer productID v zavitih oklepajih predstavlja ID izdelka in ob enem ime mape na strežniku (prikazano na sliki 3.19). Za vsako kmetijo oz. izdelek je rezervirana mapa, kjer se hranijo slike, katerih imena so lastni ID-ji iz podatkovne baze (kot prikazuje slika 3.20). Na koncu je dodana še končnica datoteke.

Podobno je pri slikah kmetij, kjer se ID izdelka zamenja z ID kmetije, ID slike izdelka pa z ID slike kmetije. Pot do slik kmetij je tako sledeča: *~/Files/ProductImages/farmID/farmImageID.jpg*.



Ime	Datum spremembe	Tip	Velikost
79	14.7.2009 7:32	Slika JPEG	859 KB
81	14.7.2009 7:32	Slika JPEG	827 KB
82	14.7.2009 7:32	Slika JPEG	582 KB
84	8.6.2011 9:12	Slika JPEG	134 KB
85	8.6.2011 16:47	Slika JPEG	6 KB
86	8.6.2011 17:26	Slika JPEG	58 KB
87	8.6.2011 9:11	Slika JPEG	1.032 KB
88	8.6.2011 16:45	Slika JPEG	5 KB
89	8.6.2011 15:40	Slika JPEG	36 KB

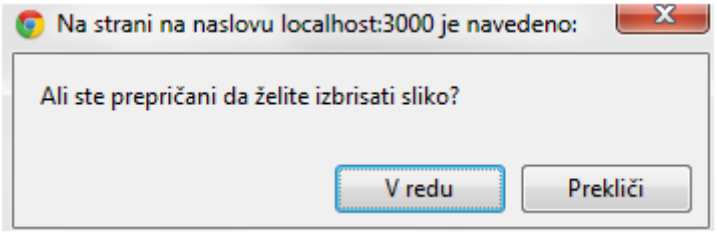
Slika 3.19: Slike v strežniški mapi izdelka.



Ime	Datum spremembe	Tip
13	22.8.2011 9:03	Mapa z datotekami
15	22.8.2011 13:50	Mapa z datotekami
20	22.8.2011 8:59	Mapa z datotekami
22	22.8.2011 8:58	Mapa z datotekami

Slika 3.20: Mape na strežniku za posamezne kmetije.

Brisanje slik je mogoče takoj po dodajanju nove slike, ko se poleg slike prikaže ikona, tako imenovani slikovni gumb (aspImage:Button). Slikovni gumb ob kliku prikaže dialog, ki opozarja o brisanju slike (prikazano na sliki 3.21). S klikom na potrditveni gumb se slika najprej izbriše iz strežniške mape, nato pa se izbrišejo še podatki o sliki v podatkovni bazi.



Slika 3.21: Potrditveno okno za izbris slike.

Slika 3.22 prikazuje metodo za brisanje v okolju ASP .NET v jeziku C#. Deluje tako, da se v spremenljivko tipa FileInfo naloži datoteka, s katero želimo manipulirati; če datoteka obstaja, se lahko tudi izbriše. Nujno pa je treba preverjati, ali instanca tipa FileInfo nima vrednosti null, saj je lahko pot do datoteke napačna ali pa je bila spremenjena. Če preverjanje uspe in instanca tipa FileInfo vsebuje podatke o sliki, se lahko z njo izvajajo različne operacije - ena izmed njih je brisanje. Datoteko se izbriše z ukazom .Delete(). Po izbrisu datoteke na strežniku se izbriše še zapis v podatkovni bazi, in sicer z ukazom db.FarmImages.DeleteOnSubmit(farmImage), pri čemer je farmImage instanca, ki predstavlja zapis slike v podatkovni bazi.

```
protected void ibtnRemoveImage_Click(object sender, EventArgs e)
{
    ImageButton ibtnRemoveImage = (ImageButton)sender;
    farmImageID = Convert.ToInt32(ibtnRemoveImage.Attributes["FarmImageID"]);
    FileInfo file = new FileInfo(Server.MapPath("~/Files/FarmImages/"
    + _farmID + "/" + farmImageID + ".jpg"));
    if (file.Exists)
    {
        file.Delete();
        FarmImage farmImage = _db.FarmImages.SingleOrDefault
            (i => i.FarmImage_ID == farmImageID);
        if (farmImage != null)
        {
            _db.FarmImages.DeleteOnSubmit(farmImage);
            _db.SubmitChanges();
        }
        rlvFarmPictures.Rebind();
    }
}
```

Slika 3.22: Metoda za izbris slike lokalno in v podatkovni bazi.

3.2.5 Obrezovanje slik

Da velikost slik kmetij ali izdelkov, ki jih naložijo uporabniki, ne bi bila (pre)velika, je realizirano obrezovanje slik. Če je slikovna datoteka velika, je nalaganje strani glede na čas zelo potratno, poleg tega pa je prostor na spletnem strežniku omejen. Obrezovanje slik deluje tako, da program vzame naloženo datoteko, jo prikaže uporabniku, ta pa nato izbere del slike, ki bo prikazan. Velikost nove slike bo natanko 800×600 slikovnih točk. V primeru manjših slik od navedene velikosti, bo prvotna slika povečana do zahtevane velikosti, kar pomeni, da se bo kvaliteta slike poslabšala.

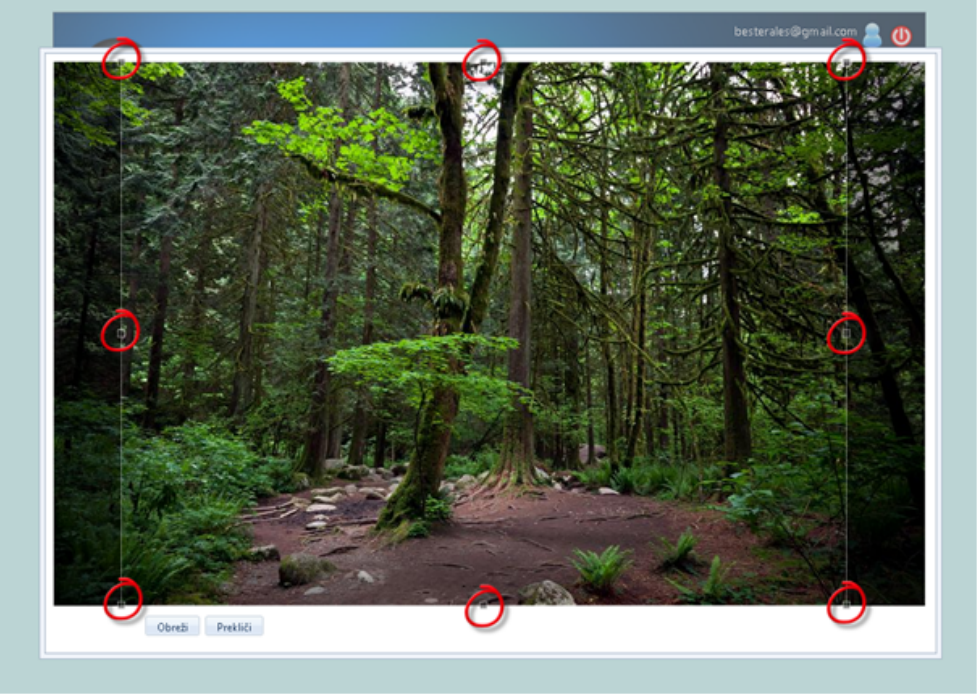
Obrezovalnik slik je realiziran prek neplačljive kontrole, imenovane Asp.net WebCropImageControl. Kontrolo je treba še nastaviti, da shranjuje slike na pravilno mesto, poleg tega pa je nastavljena še velikost največje in najmanjše slikovne datoteke in na koncu še razmerje slik - to vidimo na Tržnici na primeru 4:3.

Primer vpeljave kontrole v .aspx prikazuje slika 3.23, na kateri je najprej definirana prikazana kontrola asp:Image. Ta vsebuje prikazano sliko, name-

njeno za obrezovanje, da lahko kasneje obrezovalniku definiramo, na katero se bomo sklicevali. V nadaljevanju se kliče ukaz cs:CropImage; z atributom Image se mu definira, na katero kontrolo naj se sklicuje. Z ukazom MinSize in MaxSize se določita minimalna in maksimalna velikost obrezane slike. Na Tržnici gre za fiksno velikost (800×600 slikovnih točk), ker sta obe lastnosti enaki.

```
<asp:Image ID="imgCrop" runat="server" />
<cs:CropImage ID="wci1" runat="server" Image="imgCrop"
 MinSize="800,600" MaxSize="800,600" W="800" H="600" />
```

Slika 3.23: Vpeljava kontrole za obrezovanje slik.



Slika 3.24: Prikaz obrezovalnika.

Kot lahko vidimo na sliki 3.24, se na naloženi sliki prikaže pravokotnik (velikosti 800×600 slikovnih točk), katerega površina pokriva del slike, ki se shrani med slike kmetije ali izdelka. Z rdečo so obkrožene kontrole za večanje in manjšanje slike, vendar je, kot že povedano, izhodna slikovna datoteka

fiksne velikosti, tako da velikosti pravokotnika ni možno nastavljati. Lahko pa se premika levo, desno, navzgor in navzdol po sliki, da se določi del, ki ga želimo imeti shranjenega.

Obrezovanje je realizirano v posebnem oknu, ki se odpre nad podstranjo za nalaganje in prikaz slikovnih elementov. Ta je realiziran s Telerikovo kontrolo, imenovano RadWindow. Deluje tako, da se takoj po naloženi sliki (lastnost OnClientFileUploaded) kliče metoda, ki odpre novo okno.
Ukaz .aspx.cs, ki odpre novo okno (RadWindow):

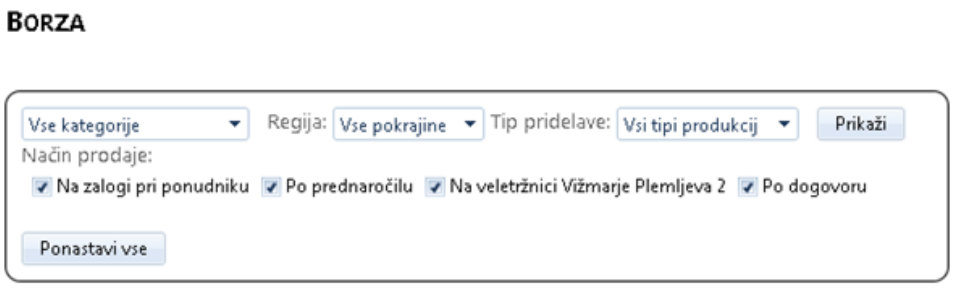
```
string script = "window.radopen('/SubPages/CropImage"+_farmID
 +"/"+imageName+"','rwCropImage').
```

Kot lahko vidimo v navedeni programski kodi, sta ukazu window.radopen dodana še dva argumenta, in sicer pot do slike ter ime okna, ki se bo odprlo. ScriptManager poskrbi še za izvedbo kode, kajti v tem primeru se kliče koda, ki se mora izvesti na strani odjemalca. Kot navedeno ima ukaz za odpiranje okna za enega izmed argumentov navedeno pot do slike, to pa zato, da ve, katero sliko želimo odpreti. Seveda pa se ta slika mora nahajati nekje na strežniku; zato nalagalnik slik deluje tako, da takoj ko je slika uspešno naložena in še preden se odpre novo okno za obrezovanje, sliko shrani na strežnik. Po končanem obrezovanju shrani novo sliko (slikovno datoteko, ki jo je uporabnik obrezal), začasno (naloženo v nalagalnik datotek) pa izbriše iz strežniške mape. Prav tako se ob kliku na gumb obreži zapre okno za obrezovanje - ob kliku na gumb se kliče metoda closeCropWindow().

3.2.6 Statistični pregled ali t.i. borza

Statistični pregled je namenjen pregledu vseh izdelkov, ki so objavljeni na Tržnici, le da v nasprotju z navadnim pregledom na tržnici, ki je že opisan, prikazuje podatke na grafičen ali tabelaričen način. Podatki so prikazani v različnih časovnih obdobjih, ki se izberejo iz kombinirane izvlečne liste. Statistični pregled sestavljajo filtri (podobno kot pri navadnem pregledu - prikaz na sliki 3.25 in sliki 3.26), s katerimi se določa selekcija med izdelki, ki jih želi uporabnik prikazati. Poleg filtrov je definirana tudi plošča (asp:Panel),

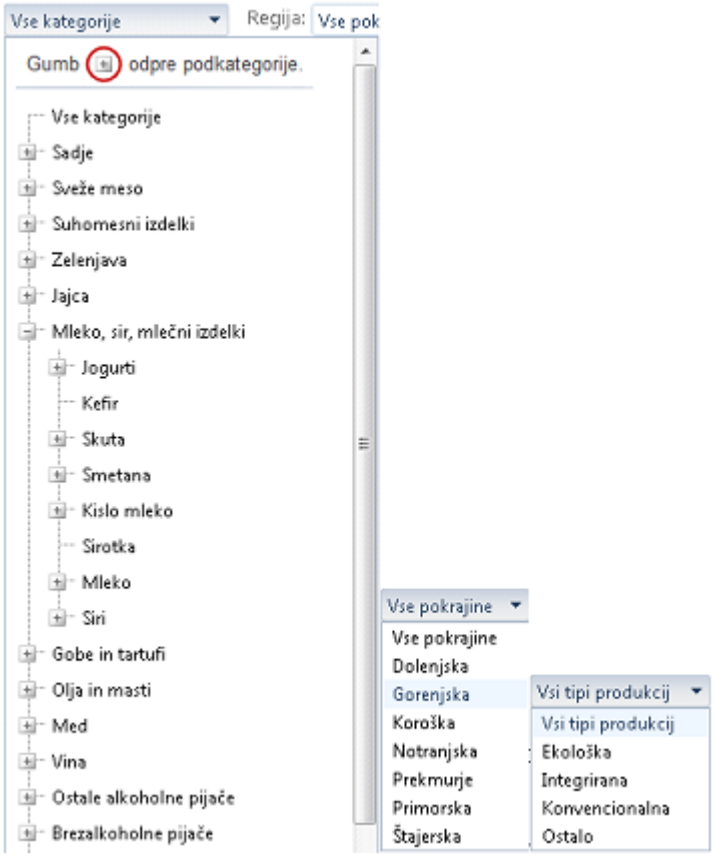
na kateri se prikazujeta grafa oziroma tabeli.



Slika 3.25: Filtri na Borzi.

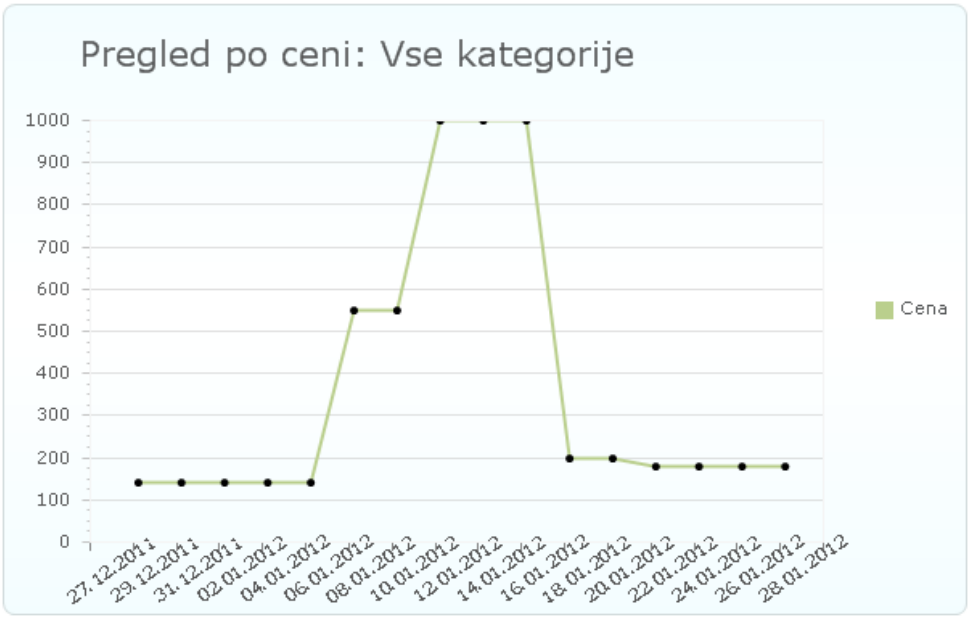
Prikaz podatkov je razdeljen na dva načina. Prvi je grafični, saj prikazuje podatke v dveh grafih. Drugi pa je tabelarični in vsebuje dve tabeli z izpisanimi podatki. Pri **grafičnem načinu** sta realizirana dva grafa; eden je namenjen prikazu spreminjanja povprečne trenutne cene izdelkov, ki so objavljeni. Trenutna povprečna cena se izračuna tako, da se vzamejo vsi izdelki v določenem časovnem obdobju, nato pa se izračuna povprečje. Število časovnih obdobij je različno in odvisno od izbrane možnosti v kombinirani izvlečni listi. Možnosti so sledeče:

- 1 dan (12 časovnih obdobij),
- 1 teden (7 časovnih obdobij),
- 1 mesec (15 časovnih obdobij),
- 6 mesecev (14 časovnih obdobij),
- 1 leto (12 časovnih obdobij),
- 5 let (10 časovnih obdobij) in
- celoten (12 časovnih obdobij).



Slika 3.26: Možnosti v kombiniranih izvlečnih listah.

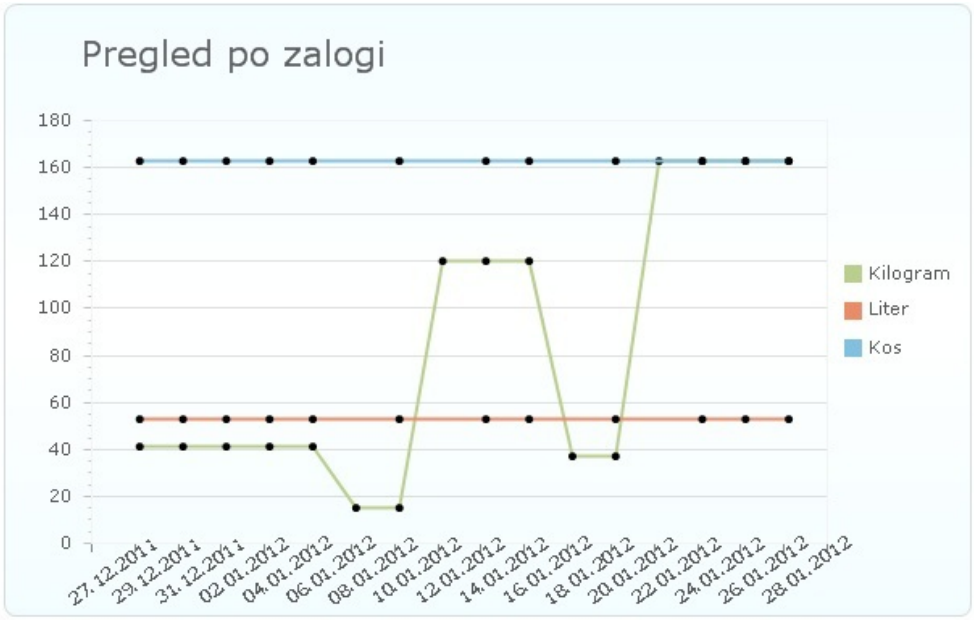
Časovna obdobja so realizirane z namenom, da lahko točno vidimo, od kdaj do kdaj je bila cena na določeni vrednosti in tudi to, kako je padala oziroma naraščala. Na sliki 3.27 je primer grafa za povprečno ceno izdelkov v obdobju zadnjih šestih mesecev. Kot lahko vidimo, je spodaj 14 različnih datumov, ki predstavljajo vmesno časovno obdobje v izbranem obdobju. Na primer: v obdobju med 6. 1. 2012 in 10. 1. 2012 je cena narasla z 550 na 1000 €. Trenutna cena se prikaže ob prehodu miške čez točko na premici.



Slika 3.27: Primer grafa za povprečno ceno.

Drugi prikaz v grafu pa je po povprečni trenutni zalogi (slika 3.28) za določeno mersko enoto (meter, kilogram, liter ...). Tukaj je računanje malce bolj kompleksno kot pri trenutni povprečni ceni, saj je treba združiti vse izdelke istega tipa znotraj posameznega časovnega obdobja ter nato izračunati povprečno vrednost. Graf prikazuje za vsako mersko enoto svojo premico, v legendi samega grafa pa lahko razberemo, katera premica predstavlja katero enoto. Prav tako kot pri povprečni ceni je tudi tukaj možnost izbire med različnimi časovnimi obdobji, ki so razdeljena na enako število podobdobij kot prejšnji graf. Časovna obdobja prikazuje os x, os y pa predstavlja vre-

dnosti posameznih merskih enot oziroma pri grafu trenutne povprečne cene vrednosti cene.



Slika 3.28: Primer grafa za pregled po zalogi.

Grafa sta realizirana s Telerikovo kontrolo RadChart. RadChart deluje lahko na dva načina in sicer, tako, da kontroli podamo t.i. vir podatkov (angl. datasource) in krivulje se izrišejo avtomatsko. Drugi način pa je, da za vsako točko na premici posebej navedemo vrednost na grafu - način, ki je uporabljen na Tržnici. Sledeč način je uporabljen, ker pri avtomatskem izrisu, ni moč izrisati podatkov na tak način, da so navedeni v različnih podobdobjih, v katerih so podatki združeni po različnih zapisih v podatkovni bazi. Zato je bilo potrebno vzeti vsako časovno podobdobje posebej, poiskati vrednosti, ki spadajo vanj, jih združiti po merski enoti in prikazati. Vsako časovno pod obdobje je predstavljen v eni iteraciji zanke while, kjer pa se definira novo točko na premici - določita se ji x in y vrednost.

Načini izpisa datuma na Tržnici so zaradi možnosti izbire časovnega obdobja različni, zato ga ne moremo prikazati povsod na enak način. Na primer: pri

obdobju enega tedna ne bi bilo mogoče razbrati, kdaj se je katera vrednost spremenila, če bi imeli nastavljen enak prikaz kot pri recimo enem mesecu. Načini prikaza po časovnih obdobjih:

- 1 dan - HH:mm (ure:minute),
- 1 teden - ddd (dan v tednu),
- celoten - yyyy (leto),
- 1 mesec, 6 mesecev, 5 let - dd.MM.yyy (dan.mesec.leto)

Tabelarični prikaz pa je realiziran s kontrolo RadGrid (Telerik) s pomočjo RadTableView-a (lastnost RadGrid-a). Delovanje je zelo podobno delovanju grafov. Način računanja je (seveda) enak, le zapis v tabelo je nekoliko drugačen. Za izpis je treba definirati stolpce in njihov način izpisa, kar se določi v kodi .aspx, kar prikazuje spodnja slika 3.29.

```
<telerik:GridBoundColumn DataField="changeDate" DataType="System.DateTime" AllowFiltering="false"
          FilterControlAltText="Filter changeDate column" HeaderText="Datum" DataFormatString="{0:dd.MM.yyyy}"
          SortExpression="changeDate" UniqueName="changeDate" ItemStyle-Width="50px">
</telerik:GridBoundColumn>
```

Slika 3.29: Prikaz definicije enega stoplca v RadMasterView.

Najprej je treba določiti, kakšen način podatkov bo prikazan, kar določimo v lastnosti DataType. Kot vidimo, je tipa System.DateTime. Način izpisa oziroma format določimo v lastnosti DataFormatString. Ena najbolj pomembnih lastnosti je DataField, v njej pa je definirano ime stolpca tabele, iz katere jemlje podatke GridBoundColumn. Ime tabele RadGrid vzame iz vira podatkov, ki je določeno na strežniški kodi (.aspx.cs).

Slika 3.30 prikazuje izpis podatkov v tabelarični obliki.

Datum	Povprečna cena(€)
05.01.2012	140,00
07.01.2012	550,00
09.01.2012	550,00
11.01.2012	550,00
13.01.2012	1000,00
15.01.2012	1000,00
17.01.2012	1000,00
19.01.2012	200,00
21.01.2012	180,00
23.01.2012	180,00
25.01.2012	180,00
27.01.2012	180,00
29.01.2012	180,00
31.01.2012	180,00
02.02.2012	180,00

Datum	Zaloga po enotah
5.1.2012	**Kilogram:** 41, **Kos:** 61, **Liter:** 177
7.1.2012	**Kilogram:** 15, **Kos:** 61, **Liter:** 177
9.1.2012	**Kilogram:** 15, **Kos:** 61, **Liter:** 177
11.1.2012	**Kilogram:** 15, **Kos:** 61, **Liter:** 177
13.1.2012	**Kilogram:** 120, **Kos:** 61, **Liter:** 177
15.1.2012	**Kilogram:** 120, **Kos:** 61, **Liter:** 177
17.1.2012	**Kilogram:** 120, **Kos:** 61, **Liter:** 177
19.1.2012	**Kilogram:** 37, **Kos:** 61, **Liter:** 177
21.1.2012	**Kilogram:** 163, **Kos:** 61, **Liter:** 177
23.1.2012	**Kilogram:** 163, **Kos:** 61, **Liter:** 177
25.1.2012	**Kilogram:** 163, **Kos:** 61, **Liter:** 177
27.1.2012	**Kilogram:** 163, **Kos:** 61, **Liter:** 177
29.1.2012	**Kilogram:** 163, **Kos:** 61, **Liter:** 177
31.1.2012	**Kilogram:** 163, **Kos:** 61, **Liter:** 177
2.2.2012	**Kilogram:** 163, **Kos:** 61, **Liter:** 177

Slika 3.30: Primer prikaza tabel za trenutno povprečno ceno in trenutno povprečno zalogo.

Poglavje 4

Sklepne ugotovitve

Realizirane so vse funkcionalnosti, ki so bile definirane pred začetkom razvoja projekta. To so: pregled, vnos, urejanje, brisanje in statistični pregled. Uporaba funkcionalnosti je omejena glede na pravice registriranih in neregistriranih uporabnikov. Naknadno je bila narejena še administracija, ki je realizirana na posebni strani, s katero ima administrator pregled nad celotnim modulom. Med razvojem pa sta se pojavili ideji o dodatnih razširitvah modula, ki bodo predlagane naročniku, in sicer:

- Možnost ocenjevanja kmetij, s čimer bi bilo doseženo, da bi uporabnik lahko videl zadovoljstvo drugih uporabnikov z izdelkom določene kmetije. Posledično bi kmetije lahko razvrščali po priljubljenosti oziroma glede na cene izdelkov.

- Objavljanje komentarjev izdelkov: uporabnikom bi omogočili pisanje pohval in pritožb za določen izdelek. Tako bi se lahko izoblikovale trgovske znamke najbolj priljubljenih kmetij. Za zdaj Tržnica še ni objavljena v javni obliki, tako da bomo pravo zmogljivost lahko videli šele, ko bo dostopna na spletu.

Po objavi končne različice modula bomo tudi podrobneje spremljali statistični prikaz izdelkov, saj je možno, da bo po več tisoč prikazih izrisovanje grafov nekoliko počasneje od pričakovanega. V tem primeru bo potrebna

optimizacija računanja trenutnih povprečnih cen in zalog. Ne glede na predlagane izboljšave in možnost optimizacije, pa menimo, da bomo s tem lahko prihranili veliko časa uporabnikom, pri iskanju domačih izdelkov, saj bodo združeni na enem mestu.

Slike

Literatura

[1] Sven Casteleyn, Florian Daniel, Peter Dolog, Maristella Matera *Engineering Web Applications*, Berlin Heidenlberg: Springer-Verlag, Nemčija, 2009, pogl. 2.

[2] Gerti Kappel, Birgit Pröll, Siegfried Reich, Werner Retschitzegger, *Web Engineering: The Discipline of Systematic Development of Web Applications*, Berlin Heidenlberg: John Wiley & Sons Ltd., Nemčija, 2003, pogl. 6.

[3] Leon Shklar, Rich Rosen, *Web application architecture, second edition*, West sussex: John Wiley & Sons Ltd., England, 2009, pogl. 1, 3.

[4] Tri-nivojska arhitektura. Dostopno na: http://colos1.fri.uni-lj.si/ERI/RACUNALNISTVO/PODATKOVNE_BAZE/trinivojska_arhitektura.html.

[5] (2012) What are active server pages (Classic ASP). Dostopno na: http://www.webwiz.co.uk/kb/asp-tutorials/what-is-asp.htm.

[6] (2012) Active Server Pages. Dostopno na: http://en.wikipedia.org/wiki/Active_Server_Pages.

[7] (2012) ASP.NET Session state overview. Dostopno na: http://msdn.microsoft.com/en-us/library/ms178581.aspx#Y0.

[8] (2012) SHA-1. Dostopno na: http://en.wikipedia.org/wiki/SHA-1#SHA-1.

[9] (2012) Secure Hash Algorithm. Dostopno na: http://en.wikipedia.org/wiki/Secure_Hash_Algorithm.

[10] (2012) Telerik. Dostopno na: http://www.telerik.com.

[11] (2012) Application Layer – Application Architecture. Dostopno na: http://codingexplorer.wordpress.com/2011/03/26/application-layer-application-architecture/.

[12] (2012) World Wide Web. Dostopno na: http://en.wikipedia.org/wiki/World_Wide_Web.

[13] (2012) Tourist farm Klemenšek Judita Klemenšek. Dostopno na: http://www.slovenia.info/en/homestead/Tourist-farm-Klemen%C5%A1ek-Judita-Klemen%C5%A1ek.htm?homestead=787&lng=2