UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Miha Starič

Organizacija poizvedovanja navkljub pomanjkljivostim v bazi slovenskih plovil

DIPLOMSKO DELO
NA VISOKOŠOLSKEM STROKOVNEM ŠTUDIJU

Mentor: prof. dr. Borut Robič

Ljubljana, 2011

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: 00549/2011
Datum: 05.09.2011

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

Kandidat: **MIHA STARIČ**

Naslov: **ORGANIZACIJA POIZVEDOVANJA NAVKLJUB POMANJKLJIVOSTIM V BAZI SLOVENSKIH PLOVIL**

ORGANISATION OF SEARCH IN SPITE OF INCONSISTENCIES IN THE SLOVENE BOAT DATABASE

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija

Tematika naloge:

Zasnujte poizvedovanje po bazi slovenskih plovil tako, da bo kljub njenim pomankljivostim možno čim boljše filtriranje podatkov.
Omogočite tudi izvoz podatkov v program Excel in njihovo tiskanje. Povežite podatke z drugimi bazami in s tem obogatite nabor informacij.
Te podatke standardizirajte, da bo možna njihova izmenjava med državami članicami Evropske unije.

Mentor:

prof. dr. Borut Robič

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:

prof. dr. Nikolaj Zimic

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani Miha Starič,

z vpisno številko 63000276,

sem avtor diplomskega dela z naslovom:

Organizacija poizvedovanja navkljub pomanjkljivostim v bazi slovenskih plovil

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom
  prof. dr. Boruta Robiča
- 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 19.9.2011 Podpis avtorja:

Zahvala

Zahvaljujem se dr. Rajku Mahkovicu za potrpežljivost in pomoč, ki mi jo je nudil pri izdelavi diplomskega dela.

Zahvaljujem se tudi svojim staršem za vztrajno spodbudo v času študija.

Nadi, Jožetu in Lidiji

Kazalo

Seznam uporabljenih kratic in simbolov

ASP.NET	ogrodje ASP.NET
ASPX	active server page extended
BE	business entity
BL	business logic
C#	c sharp
CSS	cascading style sheet
DAL	data access logic
DDV	davek na dodano vrednost
EU	Evropska unija
GUI	graphical user interface
IDIS	integrirani davčni informacijski sistem
HTML	hyper-text markup language
SQL	structured query language
VAT	value added tax
XML	extensible markup language
XSD	XML schema definition language
XSL	extensible stylesheet language

Povzetek

V diplomi obravnavamo bazo podatkov o čolnih, njihovih motorjih ter lastnikih. Podatki, ki so shranjeni v bazo, so zaradi nepravilne zasnove sistema in pomanjkanja določenih varnostnih mehanizmov slabe kvalitete. Tipi polj so nedosledni, manjkajo ključi in v nekaterih primerih celo za nas pomembni podatki. Za primarno uporabo je bil tak sistem verjetno zadovoljiv, pri naši uporabi teh podatkov in njihovi primerjavi s tistimi v drugih bazah pa utegne priti do težav. Tako je osrednja zahteva v nalogi izdelati aplikacijo, ki bi kljub vsem napakam v bazi omogočala kar se da uspešno poizvedovanje po podatkih. Le-te bi nato prikazali na zaslonu, jih izvozili v program Excel in po potrebi tudi omogočili tiskanje podatkov na tiskalnik.

Podatke bi radi tudi povezali z drugimi bazami, v katerih imamo na primer podatke o poštnih številkah. S tem bi pridobili možnost iskanja po imenu pošte, ne zgolj po poštni številki, pa tudi izpisani podatki bi bili na ta način dopolnjeni z več informacijami.

Za izvajanje Zakona o davku na vodna plovila (Uradni list RS št. 117/2006) potrebuje Davčna uprava Republike Slovenije bazo podatkov o plovilih, za kakršna je davek predpisan, in o njihovih lastnikih. Podatki, potrebni za izračun zakonsko določenega davka, zadevajo identifikacijo lastnika ali lastnikov, identifikacijo plovila in njegov opis ter vrsto in moč pogona. Vse to zahteva skrbno vzdrževano bazo podatkov in možnost povezave z drugimi relevantnimi viri podatkov.

Podatke iz poizvedb bi radi še standardizirali na način, s katerim bi bila morebitna izmenjava podatkov med državami članicami Evropske unije čim lažja.

Ključne besede:

davki, plovila, baza podatkov, register, .NET, C#, SQL

Abstract

In our thesis we examined a database of boats, their drives and their owners. The data stored in the database is of inferior quality due to flawed design and missing security features. Field types are inconsistent, they lack keys and sometimes even important information. Such system is adequate for its primary use but is not suited for our application and may cause problems in interconnectivity of data with other databases. So the main objective of the thesis is to design an application that would allow us to successfully browse through the data despite all the flaws. The data would then be presented on screen, exported to the Excel program or printed if needed.

We would also like to connect the data with other databases, such as the database of postcodes. That would enable us to search also by the post name instead of only by postcodes. The presented data would also be enriched with more information.

In order to enforce the boat tax law (Uradni list RS št. 117/2006) the tax office requires a database of boats and their owners. The data needed to calculate the lawfully prescribed taxes are required to identify the owner or owners of the boat and the boat itself as well as to define the boat's properties together with the type of the drive and power output. This requires a thorough maintenance of the database and its ability to connect with other relevant data sources.

We want to standardize the data output to enable easy exchange of information with other European Union member states.

Key words:

taxes, boats, database, registry, .NET, C#, SQL

Uvod

Za izvajanje Zakona o davku na vodna plovila (Uradni list RS št. 117/2006) potrebuje Davčna uprava Republike Slovenije bazo podatkov o plovilih, za kakršna je davek predpisan, in o njihovih lastnikih. Podatki, potrebni za izračun zakonsko določenega davka, zadevajo identifikacijo lastnika ali lastnikov, identifikacijo plovila in njegov opis ter vrsto in moč pogona. Vse to zahteva skrbno vzdrževano bazo podatkov in možnost povezave z drugimi relevantnimi viri podatkov.

V obstoječi bazi shranjeni podatki pa niso vedno brezhibni in vsebujejo napake, kar uporabnikom seveda otežuje kakovostno delo. Če želi uporabnik, na primer, iskati podatke za določeno osebo, ki ima davčno številko 12345678, utegne naleteti na problem. Ni namreč nujno, da je davčna številka v bazi res zapisana kot niz 12345678, prav mogoče je, da je zapisana s predpono - SI12345678 - ali pa kar brez zadnje števke - 1234567.

Pri realizaciji naloge sem se srečal z različnimi problemi. Moral sem čim bolje predstaviti podatke iz baze kljub njihovim pomanjkljivostim ter jih prikazati v več nivojih, uspešno beležiti dogodke, ki se vršijo nad aplikacijo, in omogočiti tiskanje izbranih podatkov ter njihov izvoz v program Excel.

Kot orodje za realizacijo sem uporabil Microsoftovo ogrodje .NET, programski jezik C# in strežnik za podatkovne baze SQL Server. Pri tem sem se držal standardnih pravil o strukturi kode [5].

Pri prevajanju strokovnih izrazov iz angleščine sem si pomagal tudi s terminološkim slovarjem informatike, dostopnim prek spleta [11].

Za reševanje problemov in realizacijo naloge sem porabil približno mesec dni.

Del I

Glavni del

Poglavje 1

Bazni del

Podatki o lastnikih, plovilih in motorjih so zapisani v podatkovni bazi *SQL*. Za potrebe diplomskega dela sem uporabil program *SQL server 2008*. Podatki so v bazi zapisani v več tabelah. Nekatere tabele so med seboj v razmerju ena na več (angl. *one-to-many*). To pomeni, da je za en zapis iz tabele A možnih več povezav na več zapisov iz tabele B, medtem ko je za en zapis iz tabele B možna povezava na zgolj en zapis iz tabele A. Druge pa so v razmerju več na več (angl. *many-to-many*), kar pomeni, da ima lahko en zapis iz tabele A povezave na več zapisov iz tabele B, in obratno, da ima lahko tudi en zapis iz tabele B povezave na več zapisov iz tabele A [6, str. 35].

Na primer, en lastnik ima lahko v lasti več čolnov. Pa tudi posamezen čoln lahko pripada več lastnikom. To je torej primer zapisa več na več, oziroma *many-to-many*. Po drugi strani pa ima lahko en čoln več motorjev, posamezen motor pa ne more pripadati več čolnom. To je primer zapisa ena na več, oziroma *one-to-many*.

1.1 Tabele

Podatki v bazi so shranjeni v tabelah. Glavni podatki naloge so shranjeni v petih glavnih tabelah, ki so naštete v tabeli 1.1.

1.1.1 Neskladja

Pri tem je treba povedati, da podane tabele nimajo privzetih mehanizmov, ki bi skrbeli za neokrnjenost podatkov. Ne uporabljajo namreč tako imenovanih primarnih ključev in omejitev po ključih. To pomeni, da sama podatkovna baza ne more zagotavljati pravilnosti razmerij med zapisi v tabelah, tako da lahko

Tabela 1.1: Tabela glavnih baznih tabel

tblLastnik	tabela lastnikov
tblColn	tabela čolnov
tblColnLastnikVmesna	vmesna tabela med čolni in lastniki
tblOpisColna	tabela z opisom čolna
tblMotor	tabela motorjev

pride do nekonsistentnosti med podatki. Pri implementaciji naloge moramo zaradi tega upoštevati vsa možna neskladja in jih tudi ustrezno obravnavati, da kar najbolj zmanjšamo njihov vpliv na uporabnika [10, str. 102].

Primeri neskladij:

- lastnik ima povezavo na neobstoječ čoln
- čoln ima povezavo na neobstoječega lastnika
- motor ima povezavo na neobstoječ čoln
- opis ima povezavo na neobstoječ čoln

Poleg neskladij so podatki včasih tudi nepravilno zapisani. Davčne številke, na primer, ki so sestavljene iz točno osmih števk, so v bazi zapisane v polju tipa `NVARCHAR(48)` [10, str. 72]. Namesto točno osmih števk je tako lahko v polju kar od enega do 48 znakov v unikodu. Pri tem je polje za davčno številko tudi neobvezno, tako da je lahko njegova vrednost ničelna (angl. *null*). Nekatere davčne številke imajo število števk večje ali manjše od osem, druge pa vsebujejo poleg števk tudi druge znake. Nekatere so v bazi zapisane s predpono '*SI*', kar nakazuje na slovensko davčno številko (po standardu ISO 3166), na primer *SI12345678*, ali pa s presledkom *SI 12345678*. Take davčne številke lahko kasneje enostavno prepoznamo in jih vključimo v poizvedbe.

1.1.2 Problem občutljivosti podatkov

Ker podatki v bazi vsebujejo osebne podatke (ime, priimek, naslov itd.), jih moram pred testiranjem pravilnosti programa ustrezno spremeniti. Pri tem moram upoštevati dejstvo, da morajo spremenjeni podatki odražati resnično stanje, čeprav ne smejo razkrivati podatkov resničnih oseb. Zaradi tega sem se odločil, da ne bom kar ustvaril naključnih podatkov.

Za realizacijo sem uporabil metodo premešavanja podatkov, kjer vzamem posamezna polja iz tabel, ki vsebujejo podatke resničnih oseb, na primer ime in priimek, ki skupaj določata osebo, in jih naključno premešam med seboj. Na ta način ohranim zadovoljivo mero resničnosti, hkrati pa onemogočim možnost, da bi se dalo kakšno osebo v tabeli prepoznati.

Primer premešanja podatkov je prikazan v tabelah 1.2 in 1.3. Za imena sem vzel najpogostejša imena in priimke s seznama prebivalstva Slovenije z dne 1. januarja 2010 [12].

Tabela 1.2: Originalna tabela imen

Mesto	Ime	Priimek
1	Franc	Novak
2	Marija	Horvat
3	Janez	Kovačič
4	Ana	Krajnc
5	Anton	Zupančič
6	Maja	Kovač

Tabela 1.3: Pomešana tabela imen

Mesto	Ime	Priimek
1	Ana	Kovačič
2	Marija	Kovač
3	Anton	Horvat
4	Janez	Zupančič
5	Maja	Novak
6	Franc	Krajnc

1.1.3 Algoritem za premešanje baze

Algoritem, ki sem ga naredil za premešavanje podatkov, temelji predvsem na ukazu `ORDER BY NEWID()`, s katerim določen izbor naključno premešamo. Sistemska funkcija `NEWID()` vrne naključno ustvarjeno vrednost, ki je edinstvena

za vsak posamezen zapis. Ukaz `ORDER BY` pa nato zloži zapise po tej naključni vrednosti [7, poglavje System Functions].

Na začetku za vsako polje, ki ga želim premešati, ustvarim posebno začasno tabelo s poljem `ID` in poljem, ki ustreza polju v prvotni tabeli, v tem primeru sem si izbral polji `Ime` in `Priimek`. Začasne tabele nato napolnim s podatki iz prvotne tabele tako, da jih naključno premešam s pomočjo ukaza `ORDER BY NEWID()`. Ko sta obe začasni tabeli premešani, vnesem nove premešane vrednosti v prvotno tabelo glede na novi ID. Pri tem uporabim funkcijo `ROW_NUMBER()`, ki vrne sekvenčno številko zapisa v naboru [7, poglavje Ranking Functions].

Na koncu uporabim še ukaz `DROP TABLE`, s katerim pobrišem začasne tabele [7, poglavje DROP Statements].

```
CREATE TABLE #Imena (
     ID int IDENTITY(1,1)
    ,Ime nvarchar(100)
    )

INSERT INTO #Priimki
SELECT Priimek
FROM dbo.tblLastnik
ORDER BY NEWID();

CREATE TABLE #Priimki (
     ID int IDENTITY(1,1)
    ,Priimek nvarchar(100)
    )

INSERT INTO #Imena
SELECT Ime
FROM dbo.tblLastnik
ORDER BY NEWID();

WITH [Lastnik sortiran]
AS (SELECT ROW_NUMBER() OVER (ORDER BY NEWID(); ) AS ROWID, Priimek
    FROM dbo.tblLastnik)
UPDATE [Lastnik sortiran]
SET Priimek = (SELECT Priimek FROM #Priimki WHERE ROWID = ID);

WITH [Lastnik sortiran]
AS (SELECT ROW_NUMBER() OVER (ORDER BY NEWID(); ) AS ROWID, Ime
    FROM dbo.tblLastnik)
UPDATE [Lastnik sortiran]
SET Ime = (SELECT Ime FROM #Imena WHERE ROWID = ID);

DROP TABLE #Imena;
DROP TABLE #Priimki;
```

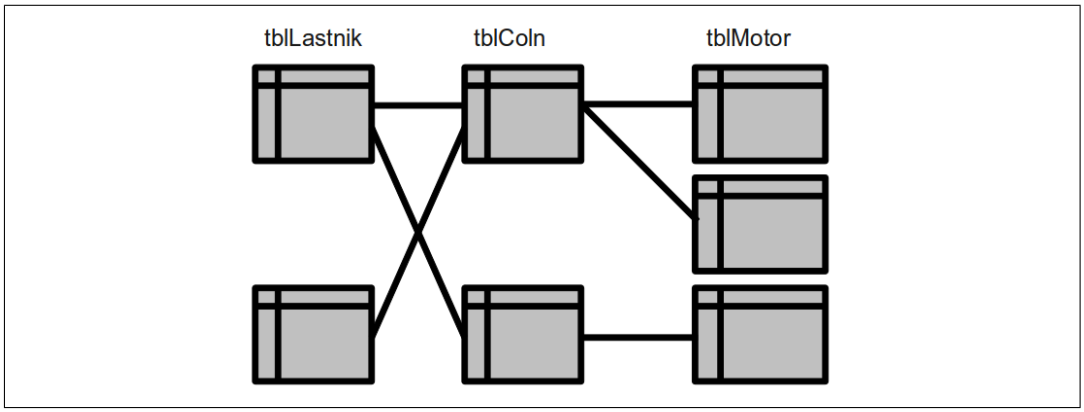
Koda 1.1: Primer algoritma za premešanje polj tabele `tblLastnik`

Pri premešavanju tabel sem poleg polj `Ime` in `Priimek` premešal še polja `DavcnaSevilka` [sic], `Ulica` in `EMSO`.

1.1.4 Glavne tabele

Vsebinska razmerja med glavnimi tabelami so taka, da ima vsak lastnik lahko več čolnov. Prav tako pa velja, da ima en čoln lahko tudi več lastnikov.

Te povezave so shranjene v vmesni tabeli med čolni in lastniki (`tblVmesna`). Vsakemu čolnu pripada po en zapis iz tabele z opisom čolna. Vsak čoln ima lahko tudi več motorjev, ki pa ne morejo hkrati pripadati več čolnom. Logična razmerja entitet natančneje opisuje slika 1.1.



Slika 1.1: Prikaz primera razmerij med glavnimi tabelami v bazi

Tabela lastnikov

Tabela lastnikov (`tblLastnik`) vsebuje podatke o lastnikih čolnov, na primer ime, priimek, emšo, naslov itd. Vsebuje tudi polje `ID_lastnika`, ki naj bi služilo nedvoumnemu določanju zapisov iz tabele. Z njegovo in pomočjo vmesne tabele lahko logično povežemo lastnike in čolne. V idealnem primeru bi se polje `ID_lastnika` označilo kot primarni ključ. Na ta način bi podatkovna baza z vdelanimi mehanizmi poskrbela za to, da se vrednosti polja `ID_lastnika` ne bi ponavljale.

Tabela čolnov

Tabela čolnov (`tblColn`) vsebuje podatke o čolnih, kot so registrska številka, model, datum vpisa, barva itd. Poleg tega ima, podobno kot tabela lastnikov, polje `ID_colna`, ki naj bi služilo v vlogi primarnega ključa tabele. Tudi to polje namreč ni definirano kot primarni ključ.

Vmesna tabela

Tabela `tblColnLastnikVmesna` skrbi za povezovanje med lastniki in čolni. Vsebuje polje `Priimek` (ki ustreza polju `ID_lastnika` v tabeli lastnikov) in polje `ID_colna`. Poleg tega vsebuje tudi nekatera polja, ki opisujejo povezavo

lastnika in čolna, na primer polje `delez`, ki označuje lastniški delež lastnika pri tem čolnu.

Tabela žal ne uporablja tujih ključev (angl. *foreign key*), ki bi skrbeli, da imajo določena polja lahko le vrednosti, ki ustrezajo točno določenim poljem iz drugih tabel. V tem primeru bi poskrbeli, da bi polji `ID_colna` in `Priimek` vsebovali zgolj obstoječe vrednosti iz tabele `tblColn` (polje `ID_colna`) in `tblLastnik` (polje `ID_lastnika`). Zaradi tega imamo v tabeli določene napake, kjer se eno ali drugo polje sklicuje na neobstoječ zapis [10, str. 104].

Tabela z opisi čolnov

Tabela `tblOpisColna` vsebuje dodatne podatke o čolnih, kot so datum gradnje, vrsta pogona ipd. Na tabelo čolnov se navezuje preko polja `Id_Colna`, ki prav tako kot v primeru vmesne tabele nima definirane omejitve tujega ključa, tako da tudi to polje včasih navaja neobstoječ zapis.

Tabela motorjev

Tabela `tblMotor` vsebuje podatke o motorjih, na primer moč, znamko, datum izdelave itd. Na tabelo čolnov se veže prek polja `idColna`. Tudi to polje nima tujega ključa in včasih navaja neobstoječ zapis.

1.1.5 Pomožne tabele

Aplikacija za delo potrebuje še podatke iz dveh tabel, ki sta že v sistemu. To sta `ddv_AVT_UPORABNIKI` in `POSTA`. Prva vsebuje seznam uporabnikov v sistemu. Ta seznam potrebujemo zaradi tega, da lahko pri sledenju dogodkov točno določimo uporabnika, ki je sprožil poizvedbo, tiskanje, izvoz v Excel ipd. Druga tabela pa vsebuje poštne številke in nazive pošt. To nam pride prav pri poizvedovanju po kraju, ker nam s pomočjo te tabele ni treba iskati samo po poštni številki kraja, ampak lahko iščemo tudi po njegovem imenu.

1.1.6 Dodatne tabele

Za potrebe sledenja dogodkov - uporabniških poizvedb, tiskanja dokumentov, izvažanja v Excel - sem kreiral tri dodatne tabele: `plv_EVID_DOGODEK`, `plv_SIFR_DOGODEK` in `plv_TiskaniDokumentiInfo`.

Tabela dogodkov (plv_EVID_DOGODEK)

Tabela ima shranjene podatke o dogodkih, ki so se zgodili v aplikaciji. Vsebuje polja o času dogodka (CAS_DOGODKA), uporabniku, ki je sprožil dogodek (UP_KODA), vrsti dogodka, ki referencira šifrant dogodkov (VRSTA_DOGODKA) in pa zahteve dogodka (ZAHTEVE_DOGODKA). Prav tako vsebuje še sistemska polja s podatki o spremembah. Ima pa tudi primarni ključ ID_EVID_DOGODKA.

Prožilec

V tej tabeli uporabljam tudi poseben mehanizem, ki se mu reče prožilec (angl. *trigger*). Ta se sproži ob določeni operaciji nad tabelo, v našem primeru pri vnosu v tabelo in pri spremembi tabele (*insert* in *update*). S pomočjo prožilca se tako v tabelo avtomatsko zapišejo tudi podatki uporabnika, ki je zadnji spreminjal podatke. Prožilec je namreč realiziran tako, da pokliče proceduro, ki mu vrne uporabnikove podatke, le-te pa prožilec nato zapiše v tabelino sistemsko polje SysInfoZadSpr. Prav tako v polje SysDatZadSpr shrani trenutni čas, da vemo, kdaj točno so se podatki nazadnje spremenili. To nam poleg varnosti omogoči tudi zelo dober pregled nad logiko [10, str. 367].

```
CREATE TRIGGER [dbo].[trg_plv_EVID_DOGODEK] ON [dbo].[plv_EVID_DOGODEK]FOR INSERT, UPDATE
AS
  DECLARE @sysinfo NVARCHAR(100)
  EXEC SP_sys_SysInfoZadSpr @sysinfo OUT
  UPDATE [dbo].[plv_EVID_DOGODEK]
  SET [SysInfoZadSpr] = @sysinfo
     ,[SysDatZadSpr]  = getdate()
  FROM INSERTED AS I
  WHERE I.ID_EVID_DOGODKA = dbo.plv_EVID_DOGODEK.ID_EVID_DOGODKA
GO
```

Koda 1.2: Izsek iz tabele plv_EVID_DOGODEK

Tabela šifranta dogodkov (plv_SIFR_DOGODEK)

Ta tabela služi kot šifrant različnih dogodkov. Vsebuje primarni ključ ID_DOGODEK, ki je preko tujega ključa referenciran iz tabele plv_EVID_DOGODEK, in polje z opisom dogodka OPIS. Prednost take uporabe šifranta je v tem, da imamo tipe dogodkov točno določene na enem mestu, kar preprečuje morebitne napake pri zapisih tipov dogodkov v tabeli dogodkov, hkrati pa nam omogoči veliko lažje spreminjanje samih opisov dogodkov.

Možne vrednosti dogodkov v tabeli so *Poizvedba*, *Tiskanje*, *Ogled*, *Ponatis* in *Poizvedba_podrobno*. Poizvedba označuje poizvedbo po podatkih, Tiskanje

označuje tiskanje teh podatkov, Ogled označuje ogled že natisnjenega dokumenta s podatki, Ponatis označuje ponovno tiskanje že natisnjenega dokumenta, Poizvedba_podrobno pa označuje detajlno poizvedbo po podatkih.

Tabela tiskanih dokumentov (`plv_TiskaniDokumentiInfo`)

Ta tabela vsebuje podatke o tiskanih dokumentih za posameznega lastnika čolna. Vsebuje polji `ID_VSEBINE`, ki je primarni ključ tabele, in `IDLastnika`, ki označuje lastnika iz tabele `tblLastnik`.

1.2 Pogledi

Za lažje poizvedovanje po podatkih sem v bazi ustvaril poglede (angl. *views*), ki podatke iz ene ali več tabel predstavijo na drug način[10, str. 280]. V njih sem uporabil nekatere funkcije, preimenovanja polj in druge prijeme, ki so mi podatke pomagali bolje prirediti za dano nalogo. Pogledi v bazi ne obstajajo fizično, uporabljajo pa se na enak način kot tabele.

1.2.1 Pogled za poštne številke (`vi_plv_POSTA`)

Pogled `vi_plv_POSTA` sem naredil tako, da prikaže samo slovenske kraje in poštne številke. To sem naredil s pomočjo pogoja *where*. Ta nam pomaga omejiti zapise iz tabele `POSTA` zgolj na tiste, ki ustrezajo pogoju. V tem primeru je pogoj `DRZ_ID = '705'`, kar pomeni, da vrne samo poštne številke in kraje, ki so v Sloveniji (koda 705).

```
CREATE VIEW [dbo].[vi_plv_POSTA]
AS
SELECT  PTT_ID  PostnaStevilka
       ,NAZIV   Kraj
FROM  dbo.POSTA
WHERE DRZ_ID = '705'
GO
```

Koda 1.3: Izsek iz pogleda `vi_plv_POSTA`

1.2.2 Pogled za čolne (`vi_plv_tblColn`)

Pri pogledu `vi_plv_tblColn` sem uporabil funkcijo *isnull*, ki v primeru, da je podana vrednost ničelna (angl. *null*), vrne drugo določeno vrednost. Pri tem pogledu sem uporabil tudi desni stik (angl. *right join*) na vmesno tabelo. Desni stik vrne vse zapise iz desne tabele (v tem primeru vmesne tabele), četudi ti nimajo ujemajočih se zapisov iz leve tabele (v tem primeru tabele čolnov).

To sem naredil zato, da bo pogled prikazal tudi ID vrednosti čolnov, ki ne obstajajo v tabeli čolnov, so pa kljub temu vpisani v vmesni tabeli (neskladja iz poglavja 1.1.1), in posledično se bo dalo te vrednosti dobiti prek poizvedb.

```
CREATE VIEW [dbo].[vi_plv_tblColn]
AS
SELECT IDColna               = ISNULL(COLN.ID_colna, VMESNA.IDColna)
      ,Zastavna              = ISNULL(COLN.zastavna, NULL)
      ,Prepoved              = ISNULL(COLN.prepoved, NULL)
      ,Obremenitev           = ISNULL(COLN.obremenitev, NULL)
      ,RegistrskaStevilka    = ISNULL(COLN.registrskaStevilka, NULL)
      ,ImeColna              = ISNULL(COLN.ImeColna, NULL)
      ,Izpostava             = ISNULL(COLN.Izpostava, NULL)
      ,DatumVpisa            = ISNULL(COLN.DatumVpisa, NULL)
      ,StevilkaVpisa         = ISNULL(COLN.oznacba_ev_ime, NULL)
      ,DatumIzdaje           = ISNULL(COLN.DatumIzdaje, NULL)
      ,VrstaColna            = ISNULL(COLN.vrsta_colna, NULL)
      ,Oznacba               = ISNULL(COLN.Oznacba, NULL)
FROM  dbo.tblColn COLN
RIGHT JOIN dbo.vi_plv_tblVmesna VMESNA
    ON VMESNA.IDColna = COLN.ID_colna
GO
```

Koda 1.4: Izsek iz pogleda `vi_plv_tblColn`

1.2.3 Pogled za lastnike (`vi_plv_tblLastnik`)

Pogled `vi_plv_tblLastnik` se nanaša na tabelo lastnikov. V njem uporabim funkcijo *convert*, ki spremeni tip polja. V tem primeru sem polje `Posta` spremenil iz tipa *FLOAT* v tip *NVARCHAR(15)*, kar bo olajšalo primerjavo polj pri poizvedbah, saj je polje `PTT_ID` v tabeli `POSTA` tipa *VARCHAR(15)*. Funkcija *isnull* pa pomaga v primeru, ko ima polje `POSTA` ničelno vrednost, in vrne prazen niz - ”.

```
CREATE VIEW [dbo].[vi_plv_tblLastnik]
AS
SELECT ID_lastnika           IDLastnika
      ,statusFizicnaPravna   StatusFizicnaPravna
      ,DavcnaSevilka         Davcna
      ,EMSO                  EMSO
      ,Ime                   Ime
      ,Priimek               PriimekNaziv
      ,Ulica                 Naslov
      ,CONVERT(NVARCHAR(15),ISNULL(Posta,''))
                             PostnaStevilka
      ,stevilka              Stevilka
FROM   dbo.tblLastnik
GO
```

Koda 1.5: Izsek iz pogleda `vi_plv_tblLastnik`

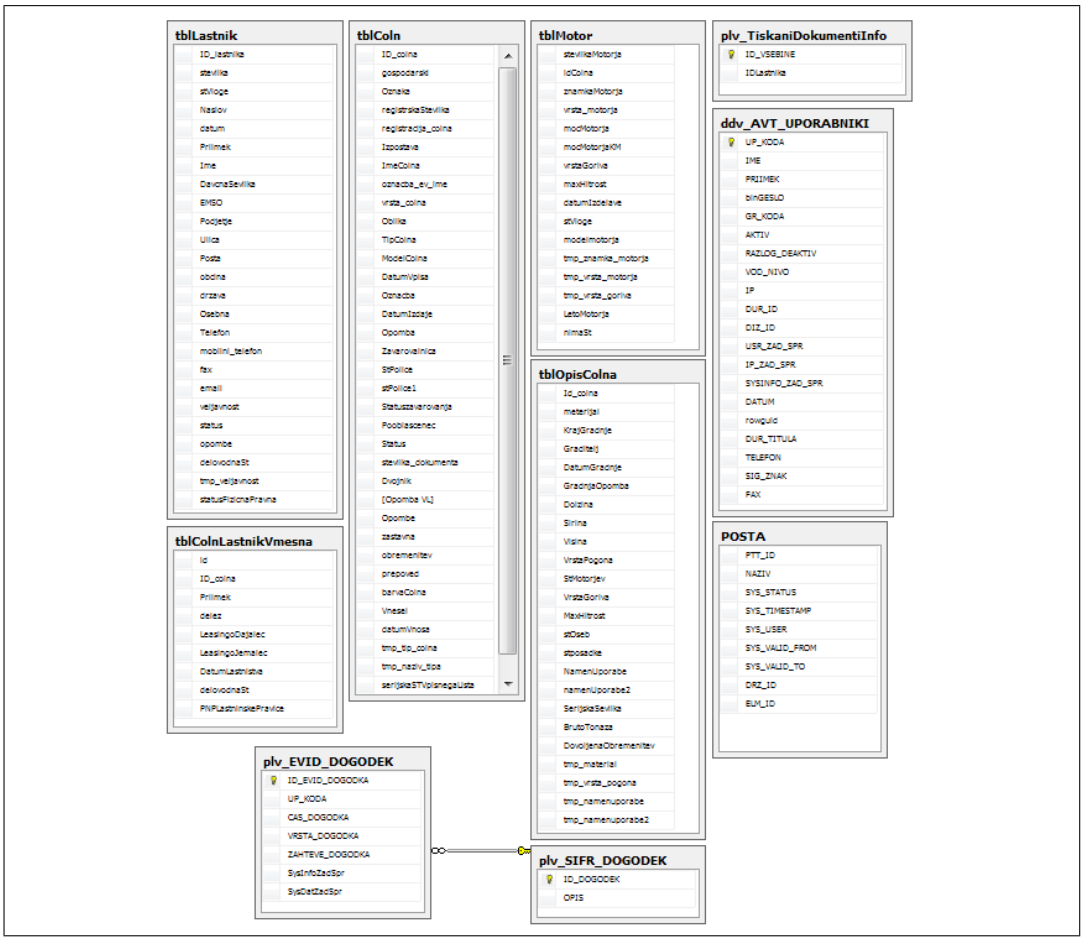
1.2.4 Pogled za vmesno tabelo (`vi_plv_tblVmesna`)

Vmesno tabelo prikažem prek pogleda `vi_plv_tblVmesna`. V njem sem uporabil zgolj preimenovanje imen polj, da so bolj skladna. Na primer polje `Priimek` sem preimenoval v `IDLastnika`, kar je dosti bolj skladno z uporabo.

Diagram baze z vsemi tabelami je prikazan na sliki 1.2.

```
CREATE VIEW [dbo].[vi_plv_tblVmesna]
AS
SELECT    ID_colna          IDColna
         ,Priimek           IDLastnika
         ,LeasingoDajalec   LeasingoDajalec
         ,LeasingoJemalec   LeasingoJemalec
         ,delez             Delez
FROM  dbo.tblColnLastnikVmesna
GO
```

Koda 1.6: Izsek iz pogleda `vi_plv_tblVmesna`



Slika 1.2: Diagram baze

1.3 Bazne procedure

Same podatke iz tabel in pogledov pridobivamo s pomočjo baznih procedur (angl. *stored procedures*). To so posebni paketi baznih ukazov, napisanih v jeziku *Transact-SQL*, ki so shranjeni na samem strežniku, da izboljšajo zmogljivosti in usklajenosti pri ponavljajočih se opravilih. Preden se shranijo v bazo, se predhodno prevedejo. S tem strežnik prihrani veliko časa, saj mu teh procedur ni treba sproti prevajati, ko jih pokličemo [10, str. 222]. Baznim proceduram lahko tudi podamo parametre, ki se uporabijo pri izvajanju. S pomočjo procedur lažje nadzorujemo dostop uporabnikov do podatkov, omogočajo pa nam tudi nadzorno sled nad dostopi. Vsaki proceduri določimo pravice uporabe za posamezne uporabnike s pomočjo ukazov *GRANT* in *REVOKE*.

1.3.1 Bazna procedura za poizvedbo (`plv_sp_tblVmesnaPoizvedba`)

Glavna procedura za poizvedbe po bazi je `plv_sp_tblVmesnaPoizvedba`. Ima šest vhodnih parametrov: `@StatusFizicnaPravna`, `@Davcna`, `@EMSO`, `@PriimekNaziv`, `@Naslov` in `@Kraj`. Vsak od parametrov ima pri poizvedbi svoj pomen.

Parameter `@StatusFizicnaPravna` določa, ali poizvedujemo po fizičnih ali pa pravnih osebah. Za ločevanje med različnimi vrednostmi tega parametra v proceduri uporabimo pogojni stavek *IF*. Odvisno od vrednosti parametra `@StatusFizicnaPravna` se izvede prvi ali drugi del procedure.

```
-- Fizicna oseba ali null
IF (@StatusFizicnaPravna = '1' OR @StatusFizicnaPravna IS NULL)
BEGIN
...
END

-- Pravna oseba
ELSE
IF (@StatusFizicnaPravna = '2')
BEGIN
...
END
```

Koda 1.7: Izsek iz procedure `plv_sp_tblVmesnaPoizvedba`

Parametri `@Davcna`, `@PriimekNaziv`, `@Naslov` in `@Kraj` se uporabijo pri obeh tipih poizvedbe, `@EMSO` pa le pri poizvedbi za fizične osebe. Da bi dobili čim večjo fleksibilnost glede poizvedovanja po bazi, uporabimo operator *LIKE* namesto enačaja. Ta nam omogoči, da iščemo vrednosti po vzorcih. Na ta način lahko iščemo, na primer, vse davčne številke, ki se začnejo na vzorec '123', ali pa priimke, ki se začnejo na 'A', in tako naprej. Če bi uporabili zgolj enačaj, bi lahko dobili samo točno določene davčne številke ali priimke. Tak način poizvedovanja pa je tudi bolj intuitiven in naraven [10, str. 139].

```
WHERE ((StatusFizicnaPravna = '1') OR (StatusFizicnaPravna IS NULL))
      AND (UPPER(Davcna) LIKE UPPER(@Davcna) + '%' OR @Davcna='')
      AND (EMSO LIKE @EMSO + '%' OR @EMSO='')
      AND (UPPER(PriimekNaziv) LIKE UPPER(@PriimekNaziv) + '%' OR @PriimekNaziv='')
      AND (UPPER(Naslov) LIKE UPPER(@Naslov) + '%' OR @Naslov='')
      AND (UPPER(POSTA.Kraj) LIKE UPPER(@Kraj) + '%' OR @Kraj='')
```

Koda 1.8: Izsek iz procedure `plv_sp_tblVmesnaPoizvedba`

Procedura podatke vrne v obliki XML (angl. *extensible markup language*, kar dosežemo z ukazom `FOR XML EXPLICIT`. Način EXPLICIT nam omogoči večji nadzor nad samo hierarhijo dokumenta XML, ki ga ustvarimo iz tabele. Podatki so predstavljeni v univerzalni tabeli in so particionirani po skupinah. Vsaka od teh skupin nato postane posamezen element v drevesu XML [10, str. 665].

```
...
FROM #TmpResult WITH (NOLOCK)
GROUP BY IDLastnika, Davcna, PriimekNaziv, Naslov, PostnaStevilka, Kraj,
         IDColna, Delez, RegistrskaStevilka, ImeColna, Izpostava,
         DatumVpisa, StevilkaVpisa, DatumIzdaje, VrstaColna, StevilkaTrupa,
         Graditelj, LetoGradnje, Dolzina, Sirina, Oznacba, Stevilka,
         Zastavna, Prepoved, Obremenitev, LeasingoDajalec, LeasingoJemalec,
         StevilkaMotorja, ZnamkaMotorja, ModelMotorja, VrstaMotorja,
         MocMotorjakW, DatumIzdelave
ORDER BY
      [Lastnik!2!PriimekNaziv],
      [Lastnik!2!Davcna],
      [Coln!3!IDColna],
      [Motor!4!StevilkaMotorja]
FOR XML EXPLICIT
```

Koda 1.9: Izsek iz procedure `plv_sp_tblVmesnaPoizvedba`

1.3.2 Bazna procedura za tiskanje (`plv_sp_izvlecekZaTiskanje`)

Pri tiskanju podatkov uporabljamo proceduro `plv_sp_izvlecekZaTiskanje`, ki vrne podatke za tiskanje glede na vhodna parametra `@IDLastnika` in `@IDColna`. Ta procedura nam ravno tako vrne podatke v obliki XML s pomočjo ukaza `FOR XML EXPLICIT`. V tej proceduri uporabim tudi namig NOLOCK. Z njim lahko preberemo podatke iz tabel, ne da bi pri tem naredili deljeno zaseženje (angl. *shared lock*), kar pomeni, da lahko hkrati podatke berejo tudi druge procedure. Ima pa ta način pomanjkljivost v primeru, ko se nad tabelami vrši kakšna transakcija. Z namigom NOLOCK preberemo trenutno stanje v tabeli, kar pomeni, da bi v takem primeru prebrali podatke, ki jih je spremenila transakcija, le-ta pa bi nato izvedla razveljavitev (angl. *rollback*) in naši prebrani podatki bi s tem postali napačni [14, str 407].

Pri tiskanju uporabljamo tudi proceduro `plv_sp_izvlecekZaTiskanjeBrezColna`, ki nam vrne le podatke lastnika. Uporablja se v primeru, ko podatkov o čolnih za določenega lastnika ni na voljo.

```
FROM  dbo.[vi_plv_tblLastnik] LASTNIK WITH (NOLOCK)
INNER JOIN dbo.[vi_plv_tblVmesna] VMESNA WITH (NOLOCK)
  ON LASTNIK.IDLastnika = VMESNA.IDLastnika
```

Koda 1.10: Izsek iz procedure `plv_sp_izvlecekZaTiskanje`

1.4 Povezave z drugimi bazami (poštne številke, zavezanci...)

V bazi plovil imamo tabelo lastnikov, v kateri so navedene poštne številke, ne pa tudi nazivi pošt. Zaradi lažjega poizvedovanja bi radi imeli možnost iskanja po imenu pošte in ne zgolj po poštni številki. To lahko omogočimo s povezavo z drugo bazo, ki vsebuje poštne številke in imena pošt. V shranjeni proceduri za poizvedbo levo staknemo (angl. *left join*) tabelo lastnikov s tabelo pošt po ključu poštne številke in na ta način pridobimo novo polje (ime pošte), po katerem lahko poizvedujemo.

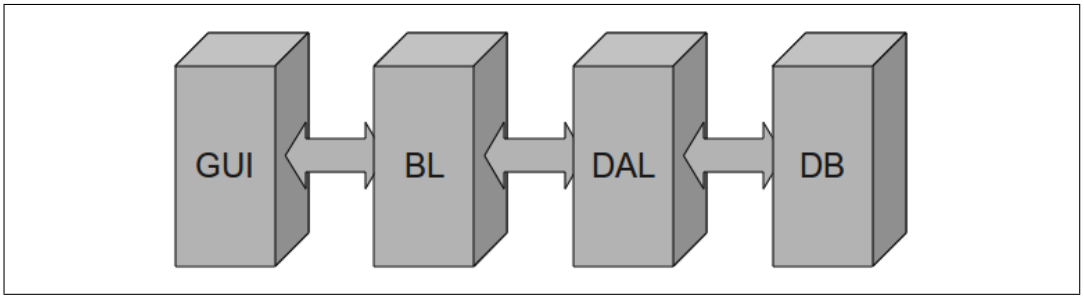
```
SELECT TOP 100
         LASTNIK.IDLastnika
        ,LASTNIK.Davcna
        ,LASTNIK.EMSO
        ,LASTNIK.Ime
        ,LASTNIK.PriimekNaziv
        ,LASTNIK.Naslov
        ,LASTNIK.PostnaStevilka
        ,LASTNIK.Stevilka
        ,POSTA.Kraj
FROM dbo.[vi_plv_tblLastnik] LASTNIK WITH (NOLOCK)
LEFT JOIN dbo.[vi_plv_POSTA] POSTA WITH (NOLOCK)
   ON LASTNIK.PostnaStevilka COLLATE SLOVENIAN_CS_AS = POSTA.PostnaStevilka COLLATE SLOVENIAN_CS_AS
```

Koda 1.11: Izsek iz procedure `plv_sp_tblVmesnaPoizvedba`

Poglavje 2

Logični model

Logični model z nivoji je prikazan na sliki 2.1.



Slika 2.1: Diagram logičnega modela [3, str. 26]

2.1 GUI

Grafični vmesnik ali GUI (angl. *graphical user interface*) je nivo, ki uporabniku predstavi podatke v vizualni obliki na zaslon. To stori s pomočjo strani ASPX (angl. *active server page extended*), ki jih odpremo v oknu brskalnika. Na ta način lahko uporabniku prikažemo podatke v obliki HTML (angl. *hyper-text markup language*), prav tako pa lahko uporabnik preko iste tehnologije sproži poizvedbo po podatkih ali pa sproži izvoz podatkov.

2.1.1 ASPX

Strani grafičnega vmesnika so v formatu ASPX. Ogrodje ASP.NET s pomočjo kode ASPX dinamično kreira kodo HTML, ki nam jo prikaže v brskalniku [16,

str. 1526]. Vsaka stran ASPX pa ima poleg svoje medvrstične kode (angl. *in-line code*) tudi posebno datoteko s kodo v ozadju (angl. *code-behind*). Te datoteke imajo končnico .aspx.cs, podrobneje so opisane v poglavju s poslovno logiko (angl. *business logic*).

Master page

GUI uporablja tako imenovano glavno stran (angl. *master page*), ki omogoča, da več ločenih strani uporablja določen del iste vsebine. V našem primeru gre za meni na vrhu strani, ki je prikazan tako na strani za preglede kot tudi na strani za sledenje. Na ta način je vzdrževanje strani dosti lažje, kot bi bilo, če bi morala vsaka stran imeti svojo kopijo menija, olajša pa tudi razširjanje in spreminjanje [16, str. 237-239].

Pregled.aspx

Stran za preglede (Pregled.aspx) vsebuje kontrolnike (angl. *controls*), ki nam pomagajo pri vnosu parametrov za poizvedbe in prikazovanju rezultatov le-teh. Pri izbiri tipa zavezanca imamo tako spustni seznam (angl. *drop-down list*). Ta nam omogoča enostavno izbiro le enega elementa iz vnaprej napolnjenega seznama elementov. Pri ostalih parametrih, torej davčni številki, emšu, primku, ulici in kraju, pa uporabljamo besedilno polje (angl. *textbox*). Vanj lahko s pomočjo tipkovnice vnesemo poljuben niz. Poleg naštetih kontrolnikov uporabljamo tudi potrditvena polja (angl. *checkbox*), s katerimi vključimo ali pa izključimo parameter iz poizvedbe. Poizvedbo sprožimo s klikom na gumb (angl. *button*). Prav tako lahko s klikom na gumb za izvoz rezultate izvozimo v obliki datoteke za Excel program. Kontrolnik literal (angl. *literal*) nam pomaga pri izpisu rezultatov poizvedbe, ki jih s pomočjo transformacije spremenimo v obliko HTML in prikažemo na mestu kontrolnika. Za prikaz sporočila v primeru, da poizvedba ni dobila rezultatov, pa uporabimo kontrolnik panel (angl. *panel*). Da se privzeto ne pojavi, ga skrijemo tako, da parameter 'visible' nastavimo na vrednost napačno (angl. *false*).

Stran Pregled.aspx vsebuje tudi kratko kodo v jeziku javascript; ta pobriše vrednosti `__EVENTTARGET` in `__EVENTARGUMENT`, ki ostaneta nastavljeni pri izvozu v Excel. Koda se pokliče pri kliku na gumb za izvoz v Excel s pomočjo parametra `OnClientClick`.

```
<script type="text/javascript">
  function clearRequest()
  {
    theForm.__EVENTTARGET.value = null;
    theForm.__EVENTARGUMENT.value = null;
    return true;
  }
</script>
...
<asp:Button runat="server" ID="_btnExportExcel"
            CssClass="buttonTitanik" Text="Izvoz v excel"
            OnClick="btnExportExcel_Click"
            OnClientClick="return clearRequest()"/>
```

Koda 2.1: Izsek iz datoteke `Pregled.aspx`

SledenjeDogodkov.aspx

Stran za pregled sledenja dogodkov (SledenjeDogodkov.aspx) prav tako uporablja kontrolnike, spustni seznam, besedilno in potrditveno polje, gumbe, panele in literal.

2.1.2 CSS

Pri izdelavi strani sem uporabil tudi prekrivne sloge ali CSS (angl. *cascading style sheet*). Ti nam pomagajo standardizirati izgled strani in podstrani ter zelo olajšajo spreminjanje izgleda, saj imamo osnovne lastnosti shranjene samo na enem mestu, torej v datoteki CSS. Kodo iz datoteke vključimo v stran s pomočjo parametra `CssClass` [16, str. 270].

Za primer sem vzel stran Pregled.aspx, kjer ima oznaka (angl. *label*) z naslovom strani vrednost parametra `CssClass` nastavljeno na `_lblTitle`.

```
<asp:Label runat="server" ID="_lblTitle"
           CssClass="_lblTitle"
           Text="Pregled lastnikov in plovil">
</asp:Label>
```

Koda 2.2: Izsek iz datoteke `Pregled.aspx`

Vrednosti `_lblTitle` so definirane v datoteki master.css in se avtomatsko prenesejo na oznako, ki jih referencira.

```
._lblTitle
{
  font-size: 18pt;
  font-weight: bold;
  color: #d3d3d3;
  margin-left: 72px;
}
```

Koda 2.3: Izsek iz datoteke `master.css`

2.2 BL

Poslovna logika ali BL (angl. *business logic*) je nivo v ozadju, ki sprejema zahteve iz grafičnega vmesnika in upravlja z logičnimi operacijami nad podatki: skrbi za dostop do baze, kreira zahteve za poizvedbe, spreminja podatke iz ene oblike v drugo (na primer med poslovnimi entitetami (angl. *business entity*) in XML) in jih vrača grafičnemu vmesniku. Poslovna logika poteka na samem strežniku in uporabnik do nje nima direktnega dostopa [3, str. 13-18].

Vsaka od dinamičnih strani ima svojo poslovno logiko, ki je shranjena v posebni datoteki s kratico .cs; ta datoteka se po angleško imenuje *code-behind* [1, str. 33].

2.2.1 Pregled.aspx.cs

Datoteka Pregled.aspx.cs vsebuje logiko za glavne poizvedbe po podatkih v bazi. Omogoča poizvedbe, tiskanje dokumentov in izvoz dokumentov v Excel. Poizvedovanje izvede tako, da prebere podatke o poizvedbi iz grafičnega vmesnika in jih preda naprej v metodo DAL. Nazaj dobi podatke v obliki dokumenta XML, ki ga potem s pomočjo transformacije XSL spremeni v kodo HTML, le-to pa nato prikaže v kontrolniku literal [13].

Za tiskanje dokumentov logika najprej zapiše dokument v bazo in nato pošlje ID zapisa iz baze strani `doc_Print.asp`. Za izvoz v Excel pa zapiše dokument XML s podatki v sejo (angl. *session*) in pokliče stran `ExcelExport.aspx`.

Poleg tega skrbi logika v tej datoteki tudi za sledenje dogodkom. Ob vsaki poizvedbi, kliku na podrobnosti, izvozu v Excel ali tiskanju se v bazo shrani zapis s tipom dogodka, parametri poizvedbe, kodo uporabnika, ki je poizvedbo pognal, in datumom dogodka.

2.2.2 ExcelExport.aspx.cs

Ta datoteka vsebuje logiko, ki skrbi za izvoz dokumentov v program Excel. Dokument v obliki XML prebere iz seje. To naredi s pomočjo ukaza `Request.QueryString["exportSessionVariable"]`. Z njim iz seje prebere spremenljivko, ki vsebuje dokument. Nato ustvari ime datoteke, ki bo vsebovala podatke iz dokumenta. To naredi tako, da najprej poskusi iz seje prebrati spremenljivko z imenom datoteke. Če spremenljivka ne obstaja, potem koda sama ustvari ime glede na trenutni datum in uro. Na koncu doda dokumentu še glavo (angl. *header*) in nogo (angl. *footer*) ter vrne dokument z ukazom `Response.Write`.

```
public partial class ExcelExport : System.Web.UI.Page
{
  protected void Page_Load(object sender, EventArgs e)
  {
    string sessionVariable = Request.QueryString["exportSessionVariable"];

    if (sessionVariable == null && Session[Variables.SessionExcelExportData] == null)
    {
      return;
    }

    string filename = Request.QueryString["filename"];
    if (string.IsNullOrEmpty(filename))
    {
       filename = "pregled_" + DateTime.Now.ToString("dd.MM.yyyy_HH:mm");
    }

    Response.Clear();
    Response.Buffer = false;
    Response.Charset = "utf-8";
    Response.ContentType = "application/vnd.ms-excel";
    Response.AddHeader("Content-Disposition", String.Format("attachment;␣filename={0}.xls", filename));

    if (string.IsNullOrEmpty(sessionVariable) == false)
    {
      Response.Write(ExportHtmlHeader() + Session[sessionVariable].ToString() + ExportHtmlFooter());
      Session[sessionVariable] = null;
    }
    else
    {
      Response.Write(ExportHtmlHeader() + Session[Variables.SessionExcelExportData].ToString()
         + ExportHtmlFooter());
      Session[Variables.SessionExcelExportData] = null;
    }
  }
```

Koda 2.4: Izsek iz datoteke `ExcelExport.aspx.cs`

2.2.3 SledenjeDogodkov.aspx.cs

Datoteka SledenjeDogodkov.aspx.cs vsebuje logiko za poizvedovanje po shranjenih podatkih pri sledenju dogodkov. Iz grafičnega vmesnika prejme podatke o poizvedbi, le-to izvede s pomočjo logike DAL in rezultate prikaže v obliki HTML v kontrolniku literal na grafičnem vmesniku.

2.2.4 SledenjeTiskanihDokumentov.aspx.cs

Logika iz te datoteke vsebuje metode za poizvedovanje po natisnjenih dokumentih ter omogoča njihov ogled in ponatis. Poleg tega tudi shranjuje podatke o dogodkih.

2.3 DAL

Logika DAL (angl. *data access logiclayer*) omogoča enostavnejši dostop do podatkov iz baze. Doda sloj, ki skrbi za dostop do baze. Na eni strani jemlje podatke iz baze ali pa jih piše vanjo, na drugi strani pa te podatke pretvori

v obliko, ki jo sloj poslovne logike lažje interpretira, ali pa jih pretvori nazaj v obliko, v kateri se lahko zapišejo v podatkovno bazo. S tem se glavni sloj poslovne logike osvobodi upravljanja z bazo in kliče zgolj metode sloja DAL.

Logiko DAL sem shranil v več datotekah v skupni mapi z imenom DAL. Datoteke vsebujejo kodo, ki je ločena glede na namen. Del sloja, ki skrbi za tiskanje, sem tako na primer shranil v datoteko Tiskanje.cs, del, ki skrbi za poizvedovanje, pa v Poizvedba.cs in tako naprej.

Vse metode, ki se uporabljajo v sloju DAL, kličejo metode za dostop do baze iz datoteke DataAccessHelper.cs. Le-te potem dejansko dostopajo do baze in kličejo bazne procedure s parametri, ki so podani v sloju DAL.

2.3.1 Tiskanje.cs

Datoteka Tiskanje.cs vsebuje dve metodi, ki se uporabljata za tiskanje podatkov. Prva metoda z imenom `Poizvedba` iz baze vrne podatke, potrebne za tiskanje. Druga, `GetIDLastnika`, pa vrne ID lastnika glede na vhodne parametre.

2.3.2 Sledenje.cs

V datoteki Sledenje.cs so shranjene metode, ki skrbijo za sledenje dostopa do podatkov in poizvedbe po shranjenih podatkih o dostopih. Metoda `Sledenje` kliče bazno proceduro, ki zapiše podatke o dostopu v bazo. Metoda `GetUporabniki` vrne seznam obstoječih uporabnikov iz baze, `Poizvedba` pa naredi poizvedbo po podatkih dostopov in vrne XmlDocument s podatki.

2.3.3 Poizvedba.cs

Poizvedba.cs skrbi za poizvedovanje po podatkih in vsebuje le eno metodo `Poizvedba`. Podatke pa vrne v formatu XmlDocument.

2.3.4 Motor.cs

Ta datoteka vsebuje metodo za poizvedbo po tabeli motorjev in vrne podatke v obliki DataSet.

2.3.5 Lastnik.cs

Metoda LastnikSearch poizveduje po tabeli lastnikov in vrne DataSet s podatki. V tej datoteki je tudi metoda PostnaStevilkaSearch, ki vrne najdene

poštne številke in imena krajev glede na podano ime kraja. Ni nujno, da je to ime popolno, saj lahko vsebuje le začetni del imena. Metoda uporablja t.i. izhodni parameter (`out String najdeniKraji`). Izhodni parametri (angl. *output parameters*) so parametri, pri katerih se ne ustvari nova lokacija za shranjevanje vrednosti, ampak se referencira lokacija podane spremenljivke. To pomeni, da se pri spremembi vrednosti izhodnega parametra spremeni kar vrednost prvotne spremenljivke [7, poglavje 10.5.1.3 Output parameters].

2.3.6 Dokument.cs

Datoteka Dokument.cs vsebuje metode, ki so namenjene shranjevanju natisnjenih dokumentov v bazo in poizvedovanju po natisnjenih dokumentih.

2.4 Poslovne entitete (BE)

V nalogi sem za prikaz podatkov uporabil tako imenovane poslovne entitete (angl. *business entities*). To so entitete, ki jim določimo pripadajoče lastnosti (angl. *properties*), tako da predstavljajo neki poslovni logični objekt. V našem primeru je to, na primer, entiteta LastnikType, ki v lastnostih vsebuje podatke, povezane z lastnikom nekega čolna, torej ime, priimek, naslov itd. Ali pa entiteta DokumentType, ki vsebuje podatke o dokumentu. Z uporabo poslovnih entitet naredimo logiko enostavnejšo in bolj skladno z realnostjo, saj imamo lastnosti določenega objekta združene na enem mestu, torej v eni entiteti. V nasprotnem primeru bi morali vsako lastnost posebej prenašati med metodami.

2.4.1 Lastnik.cs

Datoteka Lastnik.cs vsebuje definicijo entitete LastnikType, ki vsebuje podatke o lastniku čolna, natančneje tip zavezanca, davčno številko, emšo, ime, priimek oz. naziv, naslov, poštno številko in kraj. Ta tip entitete uporabimo za podatke o lastniku čolna, ki se uporabljajo pri poizvedovanju in tiskanju dokumentov.

2.4.2 Dokument.cs

V datoteki Dokument.cs imamo definicijo entitete DokumentType. Ta vsebuje podatke o dokumentu, med drugimi tudi sam dokument v obliki XML. Dokument je predstavljen v formatu *string*.

```
public class LastnikType
{
  private string _tipZavezanca;
  public string TipZavezanca
  {
    get { return _tipZavezanca; }
    set { _tipZavezanca = value; }
  }

  private string _davcnaStevilka;
  public string DavcnaStevilka
  {
    get { return _davcnaStevilka; }
    set { _davcnaStevilka = value; }
  }

  private string _emso;
  public string EMSO
  {
    get { return _emso; }
    set { _emso = value; }
  }
...
```

Koda 2.5: Izsek iz datoteke `App_CodeBELastnik.cs`

2.4.3 Dogodek.cs

Ta datoteka vsebuje definicijo entitete DogodekType, v kateri so določeni podatki o beleženih dogodkih. Nekaj tipov v tej entiteti je neobveznih (angl. *nullable*), kar pomeni, da so lahko brez vrednosti oziroma da so v stanju *null*. Takšne tipe lahko definiramo z besedo *nullable* in tipom v koničastih oklepajih (`Nullable<DateTime>`) ali pa na kratko z vprašajem za definicijo (`DateTime?`). Uporaba neobveznih tipov nam omogoči, da entiteto DogodekType bolj logično pravilno uporabimo tako za shranjevanje parametrov poizvedbe po dogodkih kot tudi za shranjevanje dokumentov samih. Določenih tipov, ki jih na primer potrebujemo pri poizvedbi (npr. `DatumDogodkaDo`), pri shranjevanju dokumenta preprosto ne določimo, namesto da bi jim morali nastaviti neko nepomembno vrednost [8, str. 271-272].

```
private DateTime? _datumDogodkaDo;
public DateTime? DatumDogodkaDo
{
   get { return _datumDogodkaDo; }
   set { _datumDogodkaDo = value; }
}
```

Koda 2.6: Izsek iz datoteke `App_CodeBEDogodek.cs`

Poglavje 3

Uporabljene tehnologije

3.1 ASP.NET

ASP.NET je ogrodje za spletne aplikacije in je del Microsoftovega ogrodja .NET (angl. *dot-net*), ki teče na sistemu Windows. Omogoča pisanje dinamičnih spletnih strani, aplikacij in storitev (angl. *services*) v več programskih jezikih, ki so podprti v ogrodju .NET. Večji del sistema IDIS (integrirani davčni informacijski sistem), v katerega sem dodal aplikacijo, teče na tem ogrodju.

Spletne strani v ogrodju .NET se imenujejo spletni obrazci (angl. *web forms*). Vsaka stran lahko vsebuje del z dinamično kodo, le-ta pa je lahko shranjena tudi v posebni datoteki. Takemu modelu, ko je dinamična koda spravljena v posebni datoteki, se po angleško reče *code behind*. Koda je lahko napisana v programskem jeziku Visual Basic ali pa C# [2, poglavje 1.2].

3.2 C#

Ogrodje .NET podpira med ostalimi tudi programski jezik C# (oziroma angl. *c sharp*). Za realizacijo sem ga izbral zaradi tega, ker imam z njim že dobra štiri leta izkušenj. Poleg tega pa je C# tudi prevladujoči programski jezik v sistemu IDIS. Za pisanje kode sem uporabil program Visual Studio 2008, v katerem sem ustvaril projekt tipa *web form* z vsemi datotekami z izvorno kodo [2, poglavje 1.1].

3.3 SQL

Podatkovna baza, na kateri so shranjene tabele s podatki iz naloge, teče na strežniku SQL Server 2005. SQL (angl. *structured query language*) je strukturiran povpraševalni jezik za delo s podatkovnimi bazami. Razvili so ga pri IBM v sedemdesetih letih prejšnjega stoletja, leta 1986 pa je bil sprejet kot standard pri ANSI (*American National Standards Institute*). Ena najbolj pogostih operacij v SQL je poizvedba (angl. *query*), s katero se iz baze vrne iskane podatke. Kot urejevalnik baznih poizvedb in skriptov sem uporabil SQL Server Management Studio.

Poglavje 4

Predlog standarda za izmenjavo podatkov med državami

Pri izdelavi diplomske naloge se je porodila zamisel, da bi funkcionalnost aplikacije še razširil. V primeru, če bi kakšna od direktiv Evropske unije namreč zahtevala izmenjavo tovrstnih podatkov, bi lahko aplikacija dobljene podatke o lastništvu nad vodnimi plovili posredovala drugim državam članicam. Pri ideji sem upošteval podobne že narejene rešitve, primere standardov in podatkov, ki so v takem primeru potrebni.

4.1 EU direktive za področje izmenjave podatkov med državami

Kot primer že realizirane izmenjave podatkov med državami sem vzel vračilo davka na dodano vrednost (DDV), oziroma projekt VAT Refund. Davčni zavezanci smejo preko davčne uprave Republike Slovenije zahtevati vračilo DDV za nabave blaga ali storitev, ki so jih nabavili za opravljanje svoje poslovne dejavnosti. Podatki o vloženih zahtevah za vračilo DDV se po določenem protokolu pošljejo preko lokalne davčne uprave na davčno upravo države, kjer je bil plačan DDV. Natančna pravila postopka za vračilo določa direktiva 2008/9/EC z dne 12. februarja 2008 [9].

4.2 Format XML

Iz tehnične dokumentacije projekta VAT Refund je razvidno, da si države članice EU izmenjavajo sporočila o zahtevkih za vračilo DDV v formatu XML.

Podanih je tudi več shem XSD (angl. *XML schema definition language*), ki določajo pravilne formate teh sporočil. Zaradi tega bi bilo smotrno, da bi tudi izmenjava podatkov o lastništvu plovil potekala na enak način, torej s sporočili v formatu XML z določenimi shemami [15].

Format XML je skupek pravil, ki definirajo semantične značke, s katerimi se dokument razdeli na določene dele. Shema XSD pa opisuje neki točno določen dokument, v katerem so značke vnaprej določene, skupaj s pravili glede tipov elementov, števila njihovih pojavitev, zaporedij in tako naprej [4, poglavje 1].

```
<?xml version="1.0"?>
<SEZONA>
  <LETO>2011</LETO>
  <LIGA>
    <LIGA_IME>Prva liga</LIGA_IME>
  </LIGA>
  <LIGA>
    <LIGA_IME>Druga liga</LIGA_IME>
  </LIGA>
</SEZONA>
```

Koda 4.1: Primer dokumenta XML

4.3 Minimalni podatki, ki so potrebni

Pri ustvarjanju dokumenta XML, ki ga pošiljamo, moramo določiti, katere podatke dejansko potrebujemo, da bomo z njimi točno določili davčnega zavezanca in plovilo. V primeru diplomske naloge imamo pri lastnikih na voljo podatke o davčni številki, emšu, imenu in priimku ali nazivu ter naslovu.

Glede na to, da posameznega davčnega zavezanca enolično določa davčna številka, bi to lahko bil eden od nujnih podatkov. Vendar imamo v našem primeru bazo podatkov, ki v veliko primerih ne vsebuje davčnih številk lastnikov, ali pa so le-te nepravilnih oblik. Pri tem je treba tudi povedati, da bi bila slovenska davčna številka za tujo državo, ki praviloma nima registra slovenskih davčnih zavezancev, brez pravega pomena, če ne bi podali tudi imena in priimka fizične osebe, oziroma v primeru pravne osebe naziva. Naslov z ulico, poštno številko in krajem je ravno tako pomemben podatek, ki nam pomaga povezati morebitne podatke tuje države s podatki o zavezancu.

Za zavezance so tako potrebni naslednji minimalni podatki:

- ime in priimek ali naziv
- naslov

Dodatno lahko v primeru, ko gre za fizične osebe, podamo še emšo, iz katerega je razviden rojstni datum. Pri pravnih osebah pa tega podatka seveda nimamo.

V pregledu plovil imamo na voljo podatke o registrski številki, imenu čolna, letu gradnje, dolžini in širini in tako naprej. Pri tem sta ključna podatka registrska številka plovila in datum izdaje, saj skupaj točno določata plovilo v času. Različna plovila imajo lahko namreč isto registrsko številko, vendar ne ob istem času. Ostali podatki niso ključnega pomena za enolično določitev plovila. Vendar se tudi tukaj soočamo s težavo, ko so podatki v bazi manjkajoči ali pa nepravilni, tako da nimajo vsi čolni registrske številke in datuma izdaje. Veliko jih sploh nima nikakršnih podatkov, razen samega zapisa v tabeli, da obstajajo. Take primere lahko ignoriramo in jih ne pošiljamo naprej, saj je bistvo ideje pošiljati podatke o lastništvu nad čolni, ne pa praznih sporočil.

Za plovila so tako potrebni naslednji minimalni podatki:

- registrska številka
- datum izdaje

Prav tako imamo tudi pri plovilih vrsto drugih podatkov, ki jih lahko podamo kot neobvezne, na primer dolžino in širino, številko trupa, naziv graditelja, vrsto čolna, podatke o motorjih in tako naprej.

Za realizacijo rešitve si lahko pomagamo s shemo XSD, ki določa pravila, katerih se mora držati dokument XML s podatki. Elementom, ki jih nujno potrebujemo, lahko določimo atribut, ki se imenuje *minoccurs*. Le-ta določa, najmanj koliko teh elementov mora dokument XML vsebovati. Če vrednost nastavimo na ena (1), se mora element pojaviti vsaj enkrat. Podobno lahko omejimo tudi maksimalno število pojavitev elementa, s pomočjo atributa *maxoccurs*. V našem primeru imamo podatke predstavljene tudi kot same atribute elementov. Element lastnik ima na primer atribute Ime, PriimekNaziv in tako naprej. Atributom, ki so tudi minimalni potrebni podatki, lahko tako v shemi XSD določimo atribut use in mu nastavimo vrednost na "required". S tem bo shema zahtevala, da se atribut v dokumentu XML nujno pojavi.

4.4 Izboljšava baze za prilagoditev standardu

V podatkovni bazi smo ugotovili kar nekaj neskladij s standardi in dobro prakso. Glavna neskladja, ki sem jih že prej opisal, so:

- lastnik ima povezavo na neobstoječ čoln
- čoln ima povezavo na neobstoječega lastnika

- motor ima povezavo na neobstoječ čoln
- opis ima povezavo na neobstoječ čoln

Poleg tega v bazi manjka vrsta mehanizmov, ki omogočajo skladje podatkov med seboj in po logiki. Manjkajo razne omejitve, od primarnih in tujih ključev, ki bi onemogočili, da katera od tabel referencira neobstoječ zapis ali pa večkrat referencira istega. Manjka tudi bolj rigorozno določanje tipa polj; za davčno številko je na primer definiran tip polja `nvarchar(24)`, kar omogoča, da se v to polje vpiše niz simbolov, dolg 24 znakov, namesto da bi se velikost vsaj omejila na osem znakov (davčna številka je sestavljena iz osmih števk), lahko pa bi se uporabil kar tip polja `int` (angl. *integer*), kar bi izbor simbolov omejilo na števke. Tudi nazivi polj so povsem nedosledni, saj so napisani včasih z veliko (`[Priimek]`), včasih z malo začetnico (`[datum]`), včasih s podčrtaji med besedami (`[mobilni_telefon]`), včasih pa brez (`[stVloge]`) ali pa s presledkom (`[Opomba VL]`), in vsebujejo celo tipkarske napake (`[DavcnaSevilka]`).

Omenjene neskladnosti in napake sem poskusil deloma odpraviti z uporabo pogledov, ki tako rekoč 'filtrirajo' neskladja pri imenih polj, vendar bi si bilo po mojem pametno vzeti čas za temeljito spremembo samih tabel, torej spremeniti tipe in imena polj ter dodati vse potrebne omejitve na ključe. Potrebno bi bilo tudi popraviti napačne zapise, torej zapise, ki se ne referencirajo na noben obstoječ zapis. Pri tem bi morali sicer preveriti pravilnost teh podatkov, saj ne vemo, ali so bivši referenčni podatki sploh obstajali in ali so bili naknadno brisani. Tako imamo na primer čoln, katerega ID se ne referencira na nobenega lastnika. To bi lahko pomenilo, da je bil lastnik pred tem dejansko v bazi in se je čoln referenciral nanj, nato pa je bil lastnik izbrisan iz baze, čoln pa ne in je ostal zapisan.

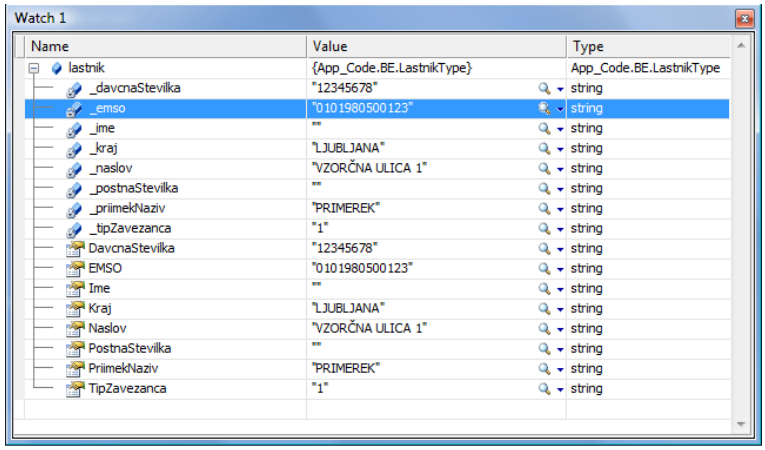
Poglavje 5

Vizualizacija podatkovnih struktur na primeru

Slika 5.1: Primer grafičnega vmesnika za poizvedbo po lastnikih

Na strani grafičnega vmesnika (slika 5.1) je podatek prikazan v obliki spustnega seznama, potrditvenih polj in vnosnih besedilnih polj. V spustnem seznamu imamo tipa zavezanca. Izbiramo lahko med pravno ali fizično osebo. Temu polju ni dodano potrditveno polje, kar pomeni, da se izbrana vrednost iz spustnega seznama upošteva pri vseh poizvedbah. Drugim, besedilnim poljem je dodano lastno potrditveno polje, ki označuje možnost vključitve v poizvedbo. Če je potrditveno polje pred besedilnim poljem označeno s ključkico, se vnešeni podatek upošteva pri poizvedbi, sicer pa se ne upošteva. S tem uporabnikom olajšamo poizvedovanje, saj lažje izločijo posamezno polje iz poizvedbe, ne da bi jim bilo treba brisati njegovo vsebino.

Podatki iz polj se ob kliku na gumb 'Iskanje' ali pa gumb 'Izvoz v excel' shranijo v obliko poslovne entitete (slika 5.2), ki smo jo ustvarili posebej za poizvedbo in jo poimenovali 'LastnikType'. Vsebuje polja tipa niz (angl.



Slika 5.2: Prikaz podatkov v poslovni entiteti LastnikType

string), v katera se prepišejo podatki iz besedilnih polj. Ta način nam omogoča lažji nadzor nad podatki in njihovo posredovanje med metodami. Tip s podatki se pošlje naprej v metodo DAL, kjer se ti podatki prenesejo v argumente SQL v tip spremenljivke `SqlParameter`.

```
public static XmlDocument Poizvedba(LastnikType lastnik)
{
  const string spName = "dbo.plv_sp_tblVmesnaPoizvedba";
  SqlParameter[] sqlParam = new SqlParameter[7];

  sqlParam[0] = new SqlParameter("@StatusFizicnaPravna",  SqlDbType.VarChar, 20);
  sqlParam[0].Value = lastnik.TipZavezanca;
  sqlParam[0].Direction = ParameterDirection.Input;

  sqlParam[1] = new SqlParameter("@Davcna", SqlDbType.VarChar, 24);
  sqlParam[1].Value = lastnik.DavcnaStevilka;
  sqlParam[1].Direction = ParameterDirection.Input;

  sqlParam[2] = new SqlParameter("@EMSO", SqlDbType.VarChar, 30);
  sqlParam[2].Value = lastnik.EMSO;
  sqlParam[2].Direction = ParameterDirection.Input;
...
```

Koda 5.1: Izsek iz datoteke `App_Code/DAL/Lastnik.cs`

Metoda DAL se nato poveže na bazo SQL s pomočjo niza za povezavo (angl. *connection string*), ki je v datoteki `web.config`. Nivo DAL tako pokliče bazno proceduro `dbo.plv_sp_tblVmesnaPoizvedba` in ji pošlje parametre iz poizvedbe. Ta procedura zatem naredi poizvedbo po bazi in vrne podatke v nivo DAL v obliki dokumenta XML.

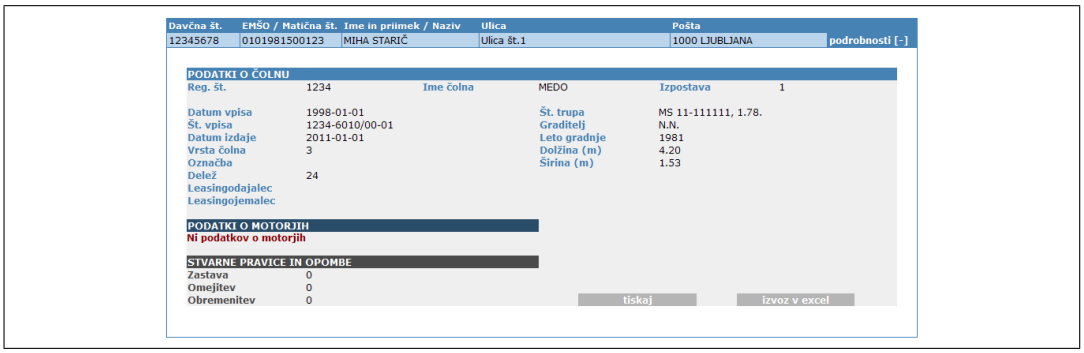
Ti podatki se nato v poslovni logiki povežejo s transformacijo XSL, ki je shranjena v datoteki xsl/Pregled.xsl in vsebuje podatke o tem, kje in kako naj se podatki iz dokumenta XML prikažejo, da se ustvari koda HTML. To vpišem v kontrolnik literal na dinamični strani.

Pregled s podrobnostmi posameznega zapisa je prikazan na sliki 5.3, prehod

```
<Lastniki>
  <Lastnik IDLastnika="123" Davcna="12345678" EMSO="0101981500100"
           Ime="Miha" PriimekNaziv="Staric" Naslov="Ulica␣st.␣1"
           PostnaStevilka="1000" Kraj="Ljubljana">
    <Coln IDColna="15" Delez="24" RegistrskaStevilka="1234"
          ImeColna="Barcica" Izpostava="1" DatumVpisa="2000-01-01"
          StevilkaVpisa="1234-6000/00-01" DatumIzdaje="2011-01-01"
          VrstaColna="3" StevilkaTrupa="MS␣01-010101,␣6.78."
          Graditelj="Plovba" LetoGradnje="1987" Dolzina="4.20"
          Sirina="1.53" Stevilka="13000" Zastavna="0" Prepoved="0"
          Obremenitev="0">
      <Motor />
    </Coln>
  </Lastnik>
</Lastniki>
```

Koda 5.2: Primer dokumenta XML iz poizvedbe

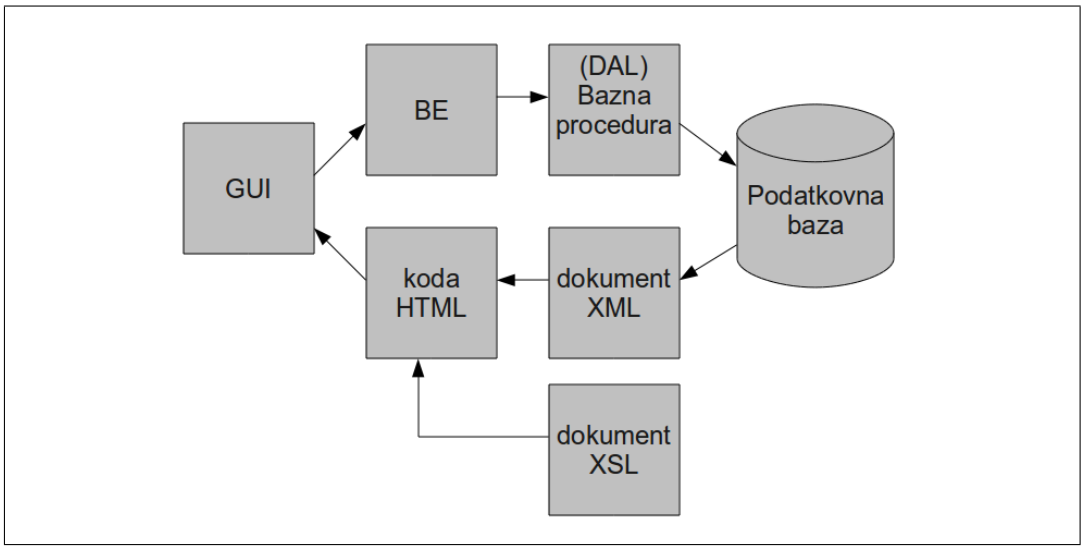
podatkov med različnimi nivoji pa lepo prikazuje slika 5.4.



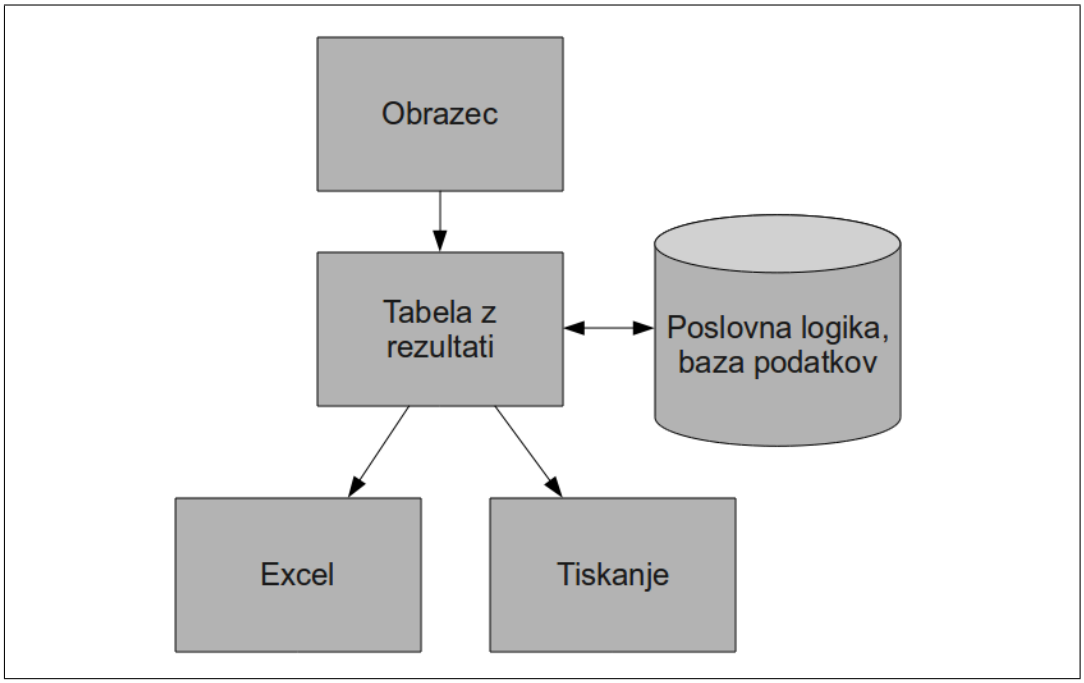
Slika 5.3: Primer prikaza podatkov s podrobnostmi

Slika 5.5 predstavlja uporabniško drevo, torej kako uporabnik dojema aplikacijo. Začne z vnosom parametrov v spletni obrazec, potem pa se preko poslovne logike in baze podatkov uporabniku vrne tabela z rezultati na grafični vmesnik. Te rezultate lahko potem izvozi v program Excel ali pa natisne na tiskalnik.

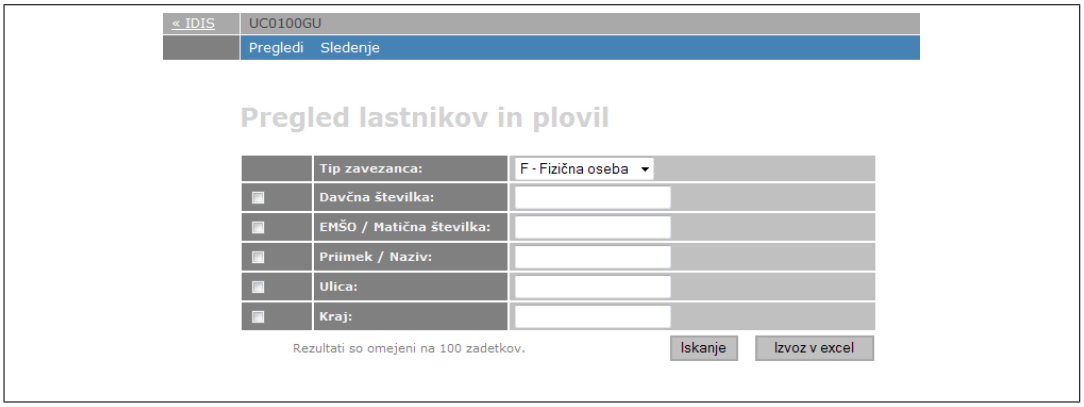
Spodaj so še štiri ekranske slike 5.6, 5.7, 5.8 in 5.9, ki predstavljajo uporabniški vidik aplikacije od vnosa parametrov v obrazec do prikaza podatkov v tabeli, s podrobnostmi, in izvoz dobljenih podatkov v program Excel.



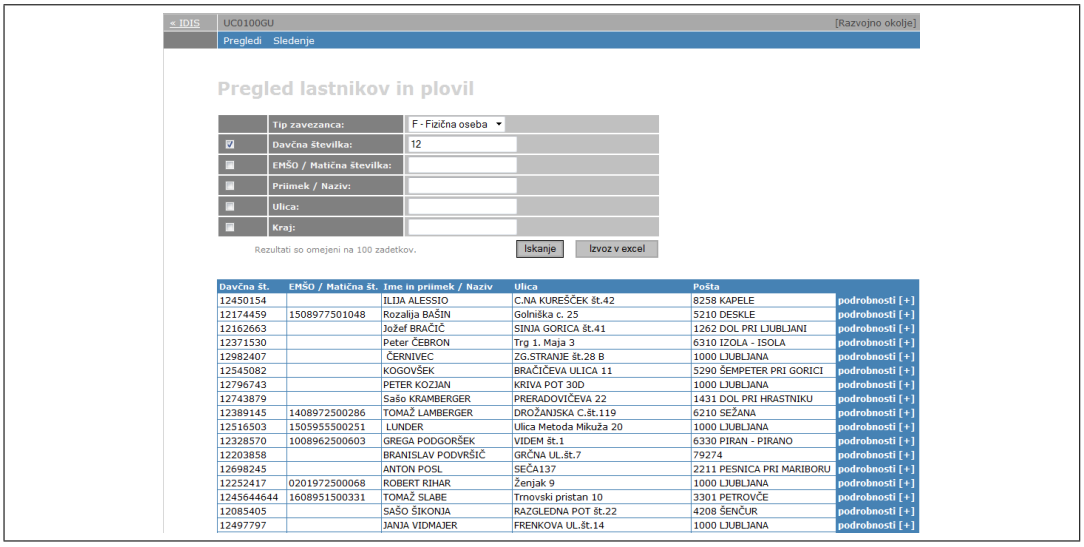
Slika 5.4: Diagram prenosa podatkov med nivoji



Slika 5.5: Uporabniško drevo

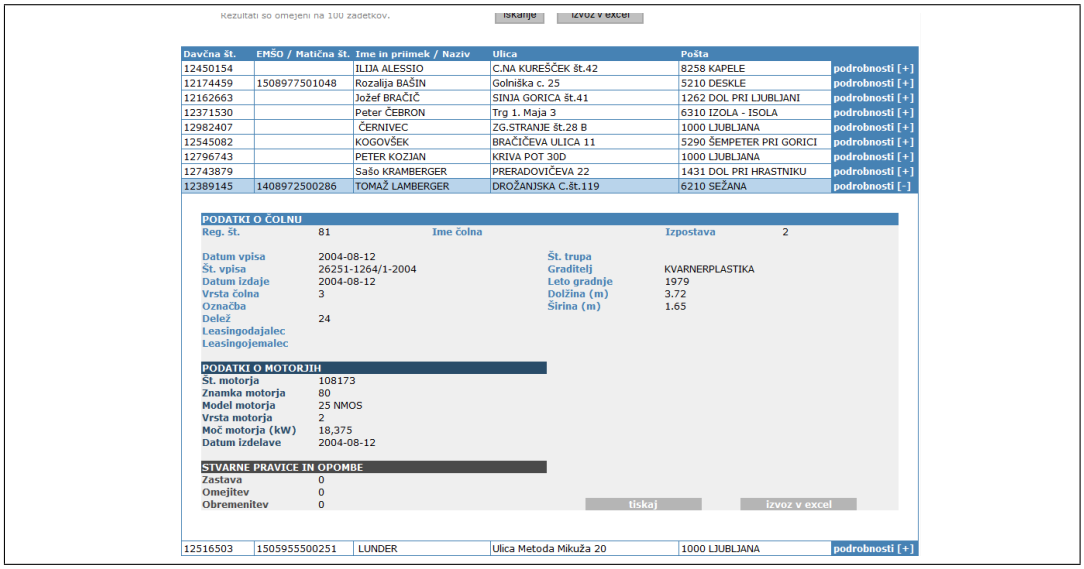


Slika 5.6: Obrazec



Davčna št.	EMŠO / Matična št.	Ime in priimek / Naziv	Ulica	Pošta	
12450154		ILIJA ALESSIO	C.NA KURESČEK št.42	8258 KAPELE	podrobnosti [+]
12174459	1508977501048	Rozalija BAŠIN	Golniška c. 25	5210 DESKLE	podrobnosti [+]
12162663		Jožef BRAČIČ	SINJA GORICA št.41	1262 DOL PRI LJUBLJANI	podrobnosti [+]
12371530		Peter ČEBRON	Trg 1. Maja 3	6310 IZOLA - ISOLA	podrobnosti [+]
12982407		ČERNIVEC	ZG.STRANJE št.28 B	1000 LJUBLJANA	podrobnosti [+]
12545082		KOGOVŠEK	BRAČIČEVA ULICA 11	5290 ŠEMPETER PRI GORICI	podrobnosti [+]
12796743		PETER KOZJAN	KRIVA POT 30D	1000 LJUBLJANA	podrobnosti [+]
12743879		Sašo KRAMBERGER	PRERADOVIČEVA 22	1431 DOL PRI HRASTNIKU	podrobnosti [+]
12389145	1408972500286	TOMAŽ LAMBERGER	DROŽANJSKA C.št.119	6210 SEŽANA	podrobnosti [+]
12516503	1505955500251	LUNDER	Ulica Metoda Mikuža 20	1000 LJUBLJANA	podrobnosti [+]
12328570	1008962500603	GREGA PODGORŠEK	VIDEM št.1	6330 PIRAN - PIRANO	podrobnosti [+]
12203858		BRANISLAV PODVRŠIČ	GRČNA UL.št.7	79274	podrobnosti [+]
12698245		ANTON POSL	SEČA137	2211 PESNICA PRI MARIBORU	podrobnosti [+]
12252417	0201972500068	ROBERT RIHAR	Ženjak 9	1000 LJUBLJANA	podrobnosti [+]
1245644644	1608951500331	TOMAŽ SLABE	Trnovski pristan 10	3301 PETROVČE	podrobnosti [+]
12085405		SAŠO ŠIKONJA	RAZGLEDNA POT št.22	4208 ŠENČUR	podrobnosti [+]
12497797		JANJA VIDMAJER	FRENKOVA UL.št.14	1000 LJUBLJANA	podrobnosti [+]

Slika 5.7: Tabela z rezultati



Rezultati so omejeni na 100 zadetkov.

Davčna št.	EMŠO / Matična št.	Ime in priimek / Naziv	Ulica	Pošta	
12450154		ILIJA ALESSIO	C.NA KUREŠČEK št.42	8258 KAPELE	podrobnosti [+]
12174459	1508977501048	Rozalija BAŠIN	Golniška c. 25	5210 DESKLE	podrobnosti [+]
12162663		Jožef BRAČIČ	SINJA GORICA št.41	1262 DOL PRI LJUBLJANI	podrobnosti [+]
12371530		Peter ČEBRON	Trg 1. Maja 3	6310 IZOLA - ISOLA	podrobnosti [+]
12982407		ČERNIVEC	ZG.STRANJE št.28 B	1000 LJUBLJANA	podrobnosti [+]
12545082		KOGOVŠEK	BRAČIČEVA ULICA 11	5290 ŠEMPETER PRI GORICI	podrobnosti [+]
12796743		PETER KOZJAN	KRIVA POT 30D	1000 LJUBLJANA	podrobnosti [+]
12743879		Sašo KRAMBERGER	PRERADOVIČEVA 22	1431 DOL PRI HRASTNIKU	podrobnosti [+]
12389145	1408972500286	TOMAŽ LAMBERGER	DROŽANJSKA C.št.119	6210 SEŽANA	podrobnosti [-]

PODATKI O ČOLNU

Reg. št.	81	Ime čolna		Izpostava	2
Datum vpisa	2004-08-12		Št. trupa		
Št. vpisa	26251-1264/1-2004		Graditelj	KVARNERPLASTIKA	
Datum izdaje	2004-08-12		Leto gradnje	1979	
Vrsta čolna	3		Dolžina (m)	3.72	
Označba			Širina (m)	1.65	
Delež	24				
Leasingodajalec					
Leasingojemalec					

PODATKI O MOTORJIH

Št. motorja	108173
Znamka motorja	80
Model motorja	25 NMOS
Vrsta motorja	2
Moč motorja (kW)	18,375
Datum izdelave	2004-08-12

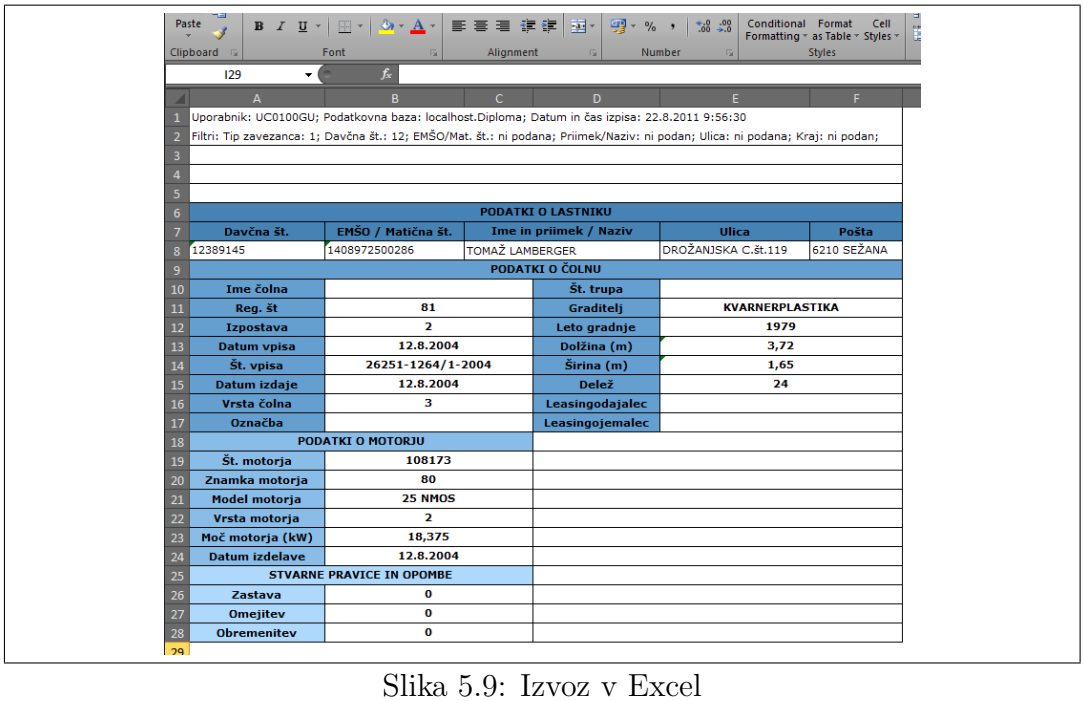
STVARNE PRAVICE IN OPOMBE

Zastava	0
Omejitev	0
Obremenitev	0

tiskaj izvoz v excel

12516503	1505955500251	LUNDER	Ulica Metoda Mikuža 20	1000 LJUBLJANA	podrobnosti [+]

Slika 5.8: Tabela z rezultati (podrobnosti)



Paste Clipboard | Font | Alignment | Number | Conditional Formatting | Format as Table | Cell Styles | Styles

I29

	A	B	C	D	E	F
1	Uporabnik: UC0100GU; Podatkovna baza: localhost.Diploma; Datum in čas izpisa: 22.8.2011 9:56:30					
2	Filtri: Tip zavezanca: 1; Davčna št.: 12; EMŠO/Mat. št.: ni podana; Priimek/Naziv: ni podan; Ulica: ni podana; Kraj: ni podan;					
3						
4						
5						
6	**PODATKI O LASTNIKU**					
7	**Davčna št.**	**EMŠO / Matična št.**	**Ime in priimek / Naziv**		**Ulica**	**Pošta**
8	12389145	1408972500286	TOMAŽ LAMBERGER		DROŽANJSKA C.št.119	6210 SEŽANA
9	**PODATKI O ČOLNU**					
10	**Ime čolna**			**Št. trupa**		
11	**Reg. št**	**81**		**Graditelj**	**KVARNERPLASTIKA**	
12	**Izpostava**	**2**		**Leto gradnje**	**1979**	
13	**Datum vpisa**	**12.8.2004**		**Dolžina (m)**	**3,72**	
14	**Št. vpisa**	**26251-1264/1-2004**		**Širina (m)**	**1,65**	
15	**Datum izdaje**	**12.8.2004**		**Delež**	**24**	
16	**Vrsta čolna**	**3**		**Leasingodajalec**		
17	**Označba**			**Leasingojemalec**		
18	**PODATKI O MOTORJU**					
19	**Št. motorja**	**108173**				
20	**Znamka motorja**	**80**				
21	**Model motorja**	**25 NMOS**				
22	**Vrsta motorja**	**2**				
23	**Moč motorja (kW)**	**18,375**				
24	**Datum izdelave**	**12.8.2004**				
25	**STVARNE PRAVICE IN OPOMBE**					
26	**Zastava**	**0**				
27	**Omejitev**	**0**				
28	**Obremenitev**	**0**				
29						

Slika 5.9: Izvoz v Excel

Del II

Sklepne ugotovitve

Pri realizaciji naloge sem na več mestih dobil ideje, kako bi lahko kaj še optimiziral, da bi sistem bolje in hitreje deloval.

Lahko bi si pomagal s tabelo o poizvedbah po bazi tako, da bi glede na pogostost iskanja po določenih parametrih tabelam dodal indekse (angl. *index*). To so dodatne podatkovne strukture, ki omogočajo hitrejši dostop do podatkov v tabeli. Poizvedbe z njihovo pomočjo hitreje dobijo podatke iz baze, saj so ti podatki v indeksu fizično sortirani po stolpcu, ki ga določimo za vsak indeks posebej [10, poglavje 10, Indices].

Pri odločanju, za katere stolpce bi bilo smotrno določiti indekse, bi si lahko pomagal s tabelo dogodkov, v katero se zdaj shranjujejo tudi parametri poizvedb. Dodal bi lahko dodatno polje, ki beleži čas, ki ga je procedura potrebovala za poizvedbo. Na ta način bi lažje identificiral ozka grla poizvedb, saj bi natančno videl, pri katerih parametrih je čas poizvedbe najdaljši. Ustvaril bi ustrezne indekse, vendar bi moral pri tem paziti na njihovo število. Sama baza sicer nima omejitev za število indeksov v tabeli, vendar vsak indeks porabi določen del diskovnega prostora, prav tako pa vpliva tudi na vnašanje in spreminjanje zapisov v tabeli – za vsak nov ali spremenjen zapis je treba tudi reorganizirati indekse, kar pa je lahko zamudno. Da bi se izognili tem težavam, je potrebno pri vsakem dodanem indeksu znova natančno preveriti novo časovno potratnost procedur.

Samih podatkov iz tabele dogodkov pa ni nujno pridobivati in analizirati samo s pomočjo logičnih poizvedb na bazi. Lahko bi vse hkrati izvozil v tekstovni obliki, nato pa jih analiziral s programom za leksikalno analizo (angl. *lexical analysis*). S tem bi razbremenil strežnik, saj mu ne bi bilo treba odžirati časa s poizvedbami, hkrati pa bi lahko podatke analiziral hitreje in bolj natančno. Za realizacijo bi uporabil program *flex*, ki omogoča analizo niza znakov s pomočjo regularnih izrazov (angl. *regular expression*). Tako bi lahko podatke natančno analiziral in dobil seznam najbolj potratnih poizvedb ter parametre, ki so bili pri tem uporabljeni.

Poleg tega bi lahko z uporabo jezika javascript omogočil dinamično odpiranje in zapiranje več podrobnosti hkrati, saj sedaj uporabljam samo po eno odprto vrstico.

Slike

Tabele

Koda

Literatura

[1] E. Ahmed, J. Chandler, B. Hatfield, R. Lassan, P. MacIntyre, M. Parihar, D. Wanta, ASP.NET Bible, Wiley Publishing, 2001, ISBN: 0764548166

[2] M. de Champlain, B. G. Patrick, C# 2.0: Practical Guide for Programmers (The Practical Guides), Morgan Kaufmann, 2005, ISBN: 0121674517

[3] C. Darie, K. Watson, Beginning ASP.NET 2.0 E-Commerce in C# 2005: From Novice to Professional, 2005, Apress, ISBN: 1590594681

[4] E. R. Harold, XML Bible, Wiley Publishing, 1999, ISBN: 0764532367

[5] M. Horner, Pro .NET 2.0 Code and Design Standards in C#, Apress, 2006, ISBN: 1590595602

[6] E. Johnson, J. Jones, A Developer's Guide to Data Modeling for SQL Server: Covering SQL Server 2005 and 2008, Addison-Wesley Professional, 2008, ISBN: 9780321497642

[7] (2010) Microsoft Developer Network Library. Dostopno 17.9.2010 na http://msdn.microsoft.com/

[8] T. Nash, Accelerated C# 2005, Apress, 2006, ISBN: 1590597176

[9] Official Journal of the European Union , COUNCIL DIRECTIVE 2008/9/EC . Dostopno 28.5.2011 na http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:L:2008:044:0023:0028:EN:PDF

[10] D. Petković, Microsoft SQL Server 2008 A Beginner-s Guide, 2008, McGraw- Hill Osborne Media, ISBN: 0071546383

[11] (2010) Slovar informatike. Dostopno 19.9.2010 na http://www.islovar.org/

[12] (2011) Statistični urad Republike Slovenije. Dostopno 2.3.2011 na http://www.stat.si/imena.asp

[13] D. Tidwell, XSLT: Mastering XML Transformations, O'Reilly Media, 2001, ISBN: 0596000537

[14] P. Turley, D. Wood, Beginning T-SQL with Microsoft SQL Server 2005 and 2008 (Wrox Programmer to Programmer), Wiley Publishing, 2009, ISBN: 9780470257036

[15] VAT Refund - Technical Specifications, 2009

[16] S. Walther, ASP.NET 3.5 Unleashed, Sams, 2008, ISBN: 0672330113