UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Marko Kastelec

PRIMERJAVA NAČINOV DOSTOPA DO PODATKOVNE BAZE V PROGRAMSKEM JEZIKU C#

DIPLOMSKO DELO
VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE
RAČUNALNIŠTVO IN INFORMATIKA

Mentor: viš. pred. dr. Igor Rožanc

Ljubljana, 2013

Rezultati diplomskega dela so intelektualna lastnina Fakultete za računalništvo in informatiko Univerze v Ljubljani. Za objavljanje ali izkoriščanje rezultatov diplomskega dela je potrebno pisno soglasje Fakultete za računalništvo in informatiko ter mentorja.

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: 00523/2013
Datum: 02.09.2013

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

Kandidat: **MARKO KASTELEC**

Naslov: **PRIMERJAVA NAČINOV DOSTOPA DO PODATKOVNE BAZE V PROGRAMSKEM JEZIKU C#**

A COMPARISON OF DATABASE ACCESS METHODS IN C# PROGRAMMING LANGUAGE

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

V diplomski nalogi sistematično primerjajte tri različne načine dostopa do podatkovne baze v programskem jeziku C#: dostop s sprotno povezavo ob vsaki operaciji, dostop brez sprotne povezave in Entity Framework dostop, kjer je interakcija omogočena s LINQ poizvedbami. Objektivnost analize zagotovite z uporabo večjega števila kriterijev, utemeljenih ocen in smiselno uteženе končne vsote ocen.

Mentor:

viš. pred. dr. Igor Rožanc

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:

prof. dr. Nikolaj Zimic

IZJAVA O AVTORSTVU

DIPLOMSKEGA DELA

Spodaj podpisani Marko Kastelec,

z vpisno številko 63060104,

sem avtor diplomskega dela z naslovom:

Primerjava načinov dostopa do podatkovne baze v programskem jeziku C#

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom viš. pred. dr. Igorja Rožanca
- 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 Podpis avtorja:

ZAHVALA

Zahvaljujem se mentorju viš. pred. dr. Igorju Rožancu za njegovo doslednost, strokovnost in potrpežljivost.

Zahvala gre tudi mojim staršem, ki so mi ves čas študija stali ob strani in me moralno ter finančno podpirali. Križem rok pa ni sedela niti moja punca, ki je tiho, pa vendar vztrajno trkala na mojo vest in me tako stalno spodbujala k izdelavi diplomskega dela...

KAZALO

SEZNAM UPORABLJENIH KRATIC IN SIMBOLOV

ADO - ActiveX Data Objects: nabor COM objektov za dostop do podatkovnih virov.

ADO.NET - Active Data Objects for .NET: nabor objektov, ki jih lahko programer uporabi za dostop do podatkov.

CLR - Common Language Runtime: skupno izvajalno jedro.

CLS - Common Language Specification: dokument, ki določa, kako naj se računalniški programi prevajajo v prenosno kodo (ang. bytecode). Ker se različni jeziki prevajajo v enako prenosno kodo, so lahko različni deli programa sprogramirani v različnih jezikih.

COM - Component Object Model: predhodnik .NET tehnologije. Omogočal je IPC, ustvarjanje dinamičnih objektov in programiranje v več programskih jezikih.

CTS - Common Type Language: standard, ki določa, kako so definicije tipov in njihove vrednosti predstavljene v pomnilniku.

DOM - Document Object Model: dogovor za predstavljanje in delo z objekti v HTML, xHTML in XML dokumentih.

EDMX - Entity Data Model XML: XML datoteka, ki definira konceptualni in pomnilniški model ter preslikave med tema modeloma.

EF - Entity Framework: nabor tehnologij v ADO.NET-u, ki podpirajo razvoj podatkovno usmerjenih programskih aplikacij.

HTML - Hyper Text Markup Language: označevalni jezik za izdelavo spletnih strani.

IPC - Inter-Process Communication: nabor metod, ki omogočajo izmenjavo podatkov med več nitmi znotraj enega ali več procesov.

JSON - JavaScript Object Notation: preprost format za prenos podatkov, ki temelji na podmnožici programskega jezika JavaScript.

LINQ - Language Integrated Query: komponenta ogrodja .NET, ki omogoča izvajanje poizvedb v jezikih ogrodja .NET.

ODbL - Open Database License: licenca, ki omogoča kopiranje, razširjanje in spreminjanje podatkov, če navajamo vir podatkov.

ORM - Object-Relational Mapping: tehnologija za prevajanje podatkov med nezdružljivima sistemoma v objektno naravnanem programskem jeziku.

SOA - Service-Oriented Arhitecture: programska arhitektura, ki temelji na zbirki programskih modulov, imenovanih servisi, ki skrbijo za funkcionalnost ogromnih sistemov.

SQL - Structured Query Language: strukturirani povpraševalni jezik za delo s podatkovnimi bazami.

SSMS - SQL Server Management Studio: orodje za delo z Microsoftovim podatkovnim strežnikom SQL.

UDF - User Defined Functions: funkcija v jeziku SQL s poljubnim številom vhodnih parametrov, ki vrača skalarno vrednost.

URL - Uniform Resource Locator: naslov spletnih strani v svetovnem spletu.

WCF - Windows Communication Foundation: jedro in množica vmesnikov za izdelavo SOA aplikacij v ogrodju .NET

xHTML - označevalni jezik, ki ima enak namen kot HTML, vendar je usklajen s sintakso XML.

XML - Extensible Markup Language: označevalni jezik, ki določa format za opisovanje strukturiranih podatkov.

POVZETEK

Namen diplomskega dela je primerjava različnih načinov dostopa do podatkovne baze v programskem jeziku C#. Na podlagi naših ugotovitev bo razvijalec lažje izbral zanj najustreznejšo tehnologijo dostopa do podatkovne baze. Primerjali smo tri načine dostopa: povezan dostop, nepovezan dostop in *Entity Framework*. Pri povezanem načinu odjemalec vzpostavi povezavo s podatkovno bazo, izvede operacijo in povezavo prekine. Če odjemalec želi spreminjati podatke, mora povezavo ponovno vzpostaviti, izvesti operacijo in povezavo zapreti. Nepovezani način za razliko od povezanega načina omogoča manipulacijo s podatki tudi takrat, ko povezava s podatkovno bazo ni vzpostavljena. Objekt `DataAdapter` poskrbi za avtomatsko vzpostavitev in prekinitev seje ter za posodabljanje podatkov. *Entity Framework* uporablja ORM pristop - podatke vrača v obliki strogo tipiziranih objektov. Interakcija z bazo je razvijalcu omogočena z uporabo LINQ poizvedb.

Načine dostopa smo primerjali na podlagi sledečih kriterijev: učne krivulje, dokumentiranosti, časa realizacije, hitrosti delovanja, prilagodljivosti in objektnosti. Vsak kriterij smo pri vsakem načinu ocenili z oceno od 1 do 5. Končna ocena vsakega izmed načinov dostopa je uteženo povprečje vseh kriterijev. Kriterije smo utežili po lastni presoji.

S primerjavo smo ugotovili, da se načini dostopa med seboj razlikujejo tako po težavnosti, potrebni količini programske kode, kakor tudi po hitrosti delovanja. Slabši oceni navkljub smo programerju z izkušnjami priporočili uporabo EF, programerju na začetku svoje razvijalske poti pa smo uporabo te tehnologije odsvetovali.

Ključne besede:

načini dostopa do podatkovne baze, povezan dostop, nepovezan dostop, Entity Framework

ABSTRACT

The purpose of this study is comparison of different database access methods in C# programming language. Based on our findings the developer will easier choose the most appropriate database access technology, best suited for his needs. We compared three different ways of access: connected, disconnected and through the Entity Framework. When using connected layer, client establishes connection, performs action and closes the connection. When client wants to update the data the connection must be reestablished. Unlike the connected layer the disconnected layer allows data manipulation also when connection is not established. Object `DataAdapter` automatically establishes and closes the session. It also provides automatic update of data. Entity Framework is ORM framework - data returned through it is returned as strongly typed objects. The developer can interact with database using LINQ queries.

Different manners of access were compared on the basis of following criteria: learning curve, documentation and support, implementation time, speed, flexibility and level of object orientation. Each of criteria was assessed with score on scale from 1 to 5. Final score of each of access manners based on weighted average. Each of criteria was weighted by common sense.

During the comparison we found out that all three access manners differ by difficulty, the amount of code needed and by performance. We recommended usage of Entity Framework to more expirienced developer even if it's score is not the best. But we advised against using this technology to developers on beginning of their programming path.

Key words:

database access manners, connected layer, disconnected layer, Entity Framework

1. UVOD

Pri programiranju spletnih ali namiznih rešitev razvijalec programske opreme slej ko prej pride do odločitve, kje in kako bo hranil podatke ter kako bo do njih dostopal. Običajno je to zelo pomembna odločitev, ki mora temeljiti na upoštevanju več dejavnikov: hitrosti, skalabilnosti, varnosti, stopnje zahtevnosti administracije sistemov, količine podatkov v podatkovni bazi, tipa podatkov, ipd. Prav tako je v primeru, ko nam programski jezik to dopušča, pomembno, katero tehnologijo dostopa do podatkov bomo izbrali.

V tem diplomskem delu bomo primerjali tri načine dostopa in ovrednotili, kateri način je za povprečnega razvijalca najugodnejši.

Namen diplomskega dela je:

- predstaviti tri načine dostopa do podatkov v programskem jeziku C#,
- predstaviti ustrezen nabor kriterijev za primerjavo tehnologij,
- prikazati pripravo in zajem primernih podatkov za primerjavo,
- primerjati tri načine dostopa do podatkov na podlagi predstavljenih kriterijev,
- analizirati rezultate, pridobljene s primerjavo, in priti do uporabnega zaključka.

Cilj tega diplomskega dela:

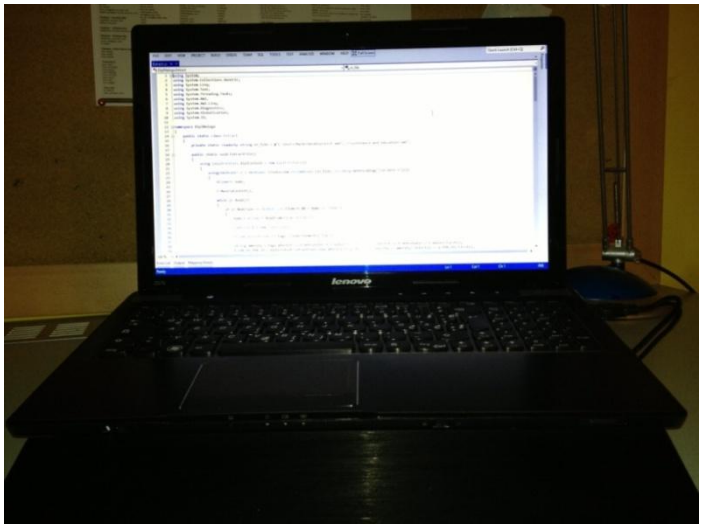
- bralcu prikazati uporabo treh različnih možnih načinov dostopanja do podatkov v programskem jeziku C#,
- bralcu predstaviti pozitivne in negativne lastnosti vsakega od načinov,
- na podlagi analize podatkov podati uporabno mnenje, s pomočjo katerega se bo bralec lažje odločil za izbiro načina dostopa do podatkov v svojem primeru.

2. STROJNA IN PROGRAMSKA OPREMA

2.1. STROJNA OPREMA

Za namen izdelave našega diplomskega dela smo uporabljali dva različna računalnika. Večino programiranja smo opravili na prenosnem računalniku Lenovo Ideapad Z570 s sledečimi specifikacijami:

- procesor Intel Core i7 2670QM 2.20 GHz,
- 8 GB Dual-Channel DDR3, 1333 MHz,
- trdi disk SATA-III Samsung SSD 830 Series, 119 GB,
- operacijski sistem Microsoft Windows 8 x64.



Slika 1: Lenovo Ideapad Z570.

Za dolgotrajne naloge, kot je bilo denimo razčlenjevanje XML datoteke in sprotno vstavljanje zapisov v podatkovno bazo, smo uporabili namizni računalnik, ki nam je služil tudi kot strežnik.

- procesor AMD FX-6300 Black Edition BOX 3.50 GHz,
- 16 GB DDR3, 1600 MHz,
- trdi disk Westeren Digital Scorpion Blue, 640 GB, 5400 RPM,
- računalnik je virtualiziran z ESXi Hypervisorjem - uporabljali smo virtualni računalnik, na katerem teče operacijski sistem Windows Server 2012 z dvema dodeljenima jedroma in 8 GB RAM-a.

2.2. PROGRAMSKA OPREMA

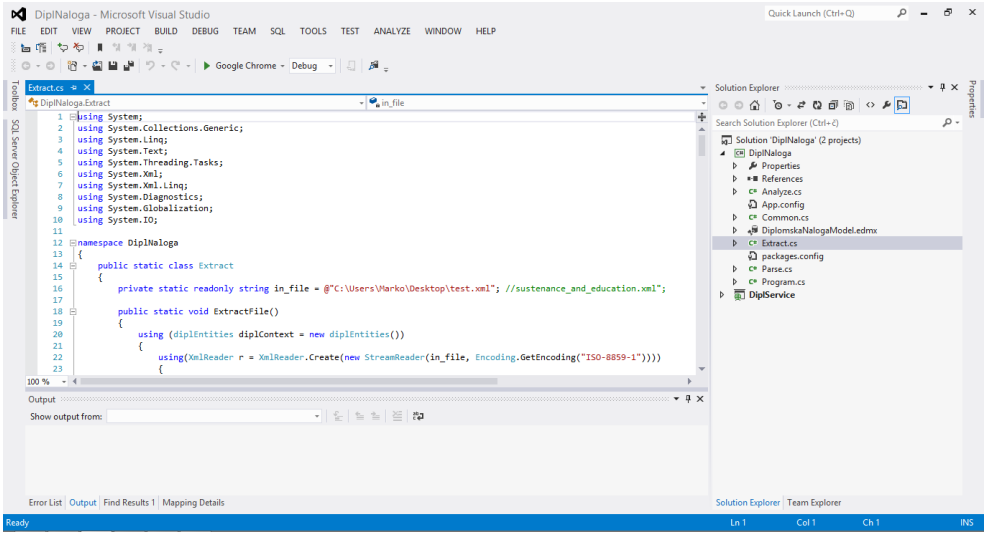
2.2.1. OKOLJE VISUAL STUDIO 2012 PROFESSIONAL

Ker smo programerski del našega diplomskega dela programirali v jeziku C#, se nam je odločitev, da za razvoj uporabimo okolje Visual Studio, zdela sama po sebi umevna. Visual Studio [1] je namreč izredno močno in z orodji ter dodatki bogato razvojno okolje (slika 2), ki razvijalcu omogoča hiter razvoj rešitev. Omogoča razvoj različnih aplikacij za različne platforme v različnih programskih jezikih. Slednje je velika prednost, saj programerju v primeru uporabe različnih tehnologij v eni rešitvi ni potrebno vlagati energije v raziskovanje novega razvojnega okolja, temveč se lahko osredotoči na samo izvedbo.

Visual Studio vsebuje orodja za grafični razvoj uporabniških vmesnikov, vsebuje podporo v obliki delčkov kode, ki jih po potrebi ponuja, orodja za delo s podatkovnimi bazami, pripomočke za pregledovanje projektov in objektov ter vgrajen sistem za pomoč. V to razvojno okolje so poleg navedenega vgrajeni tudi mnogi dodatki, kot na primer:

- orodje za pregledovanje in načrtovanje XML,
- podpora za razvoj aplikacij za mobilne naprave,
- podpora za razvoj v okviru paketa Microsoft Office,
- vgrajena podpora za spreminjanje kode (ang. code refactoring),
- orodja za načrtovanje objektov, itd.

To seveda niso vsi dodatki, ki jih Visual Studio ponuja. S tem kratkim seznamom smo želeli zgolj nakazati na obsežnost okolja [2].



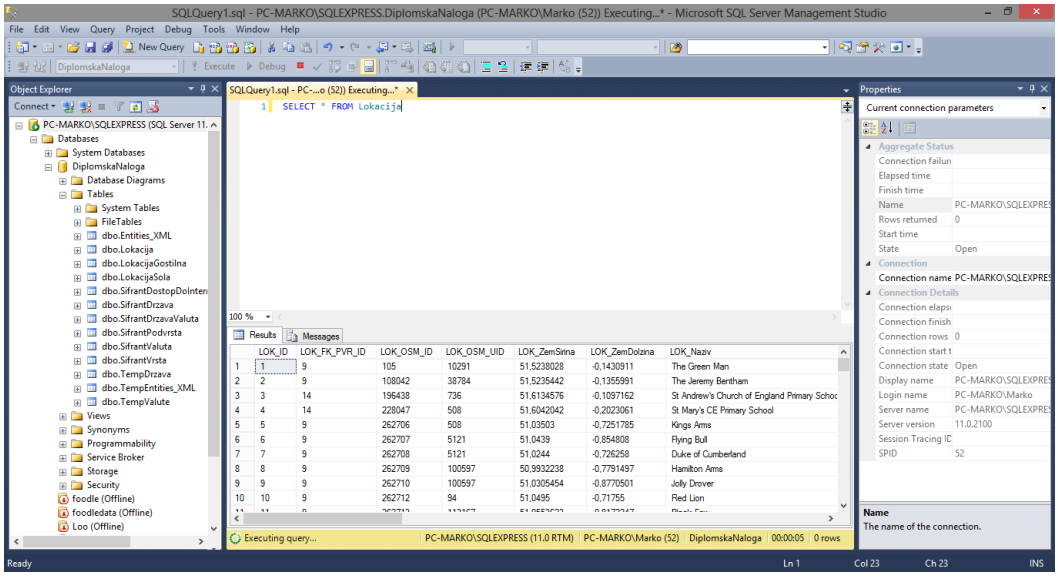
Slika 2: Okolje Visual Studio 2012 Professional.

2.2.2. *Orodje Microsoft SQL Server Management Studio*

SQL Server Management Studio (SSMS) [3] je orodje za prilagajanje, upravljanje in administriraje vseh komponent, ki so del Microsoftovega strežnika SQL. Orodje vključuje urejevalnik skript in skupek različnih grafičnih orodij. Osrednja funkcija orodja je raziskovalec objetkov, ki uporabniku omogoča pregledovanje objektov v okviru strežnika [4, 5].

SSMS nam omogoča urejanje tabel in pogledov v grafičnem urejevalniku, pisanje in izvajanje poizvedb, shranjenih poizvedb ter funkcij nad katerimi imamo ličen pregled v obliki drevesne strukture v okviru raziskovalca objektov. Omogoča nam tudi lep tabelaričen pregled podatkov, ki nam jih vrne naša poizvedba (slika 3).

Orodje smo uporabili za pisanje in izvajanje poizvedb, funkcij in shranjenih poizvedb. Uporaba grafičnih vmesnikov nam je čas načrtovanja stukture podatkovne baze bistveno skrajšala.



Slika 3: Orodje SQL Server Management Studio.

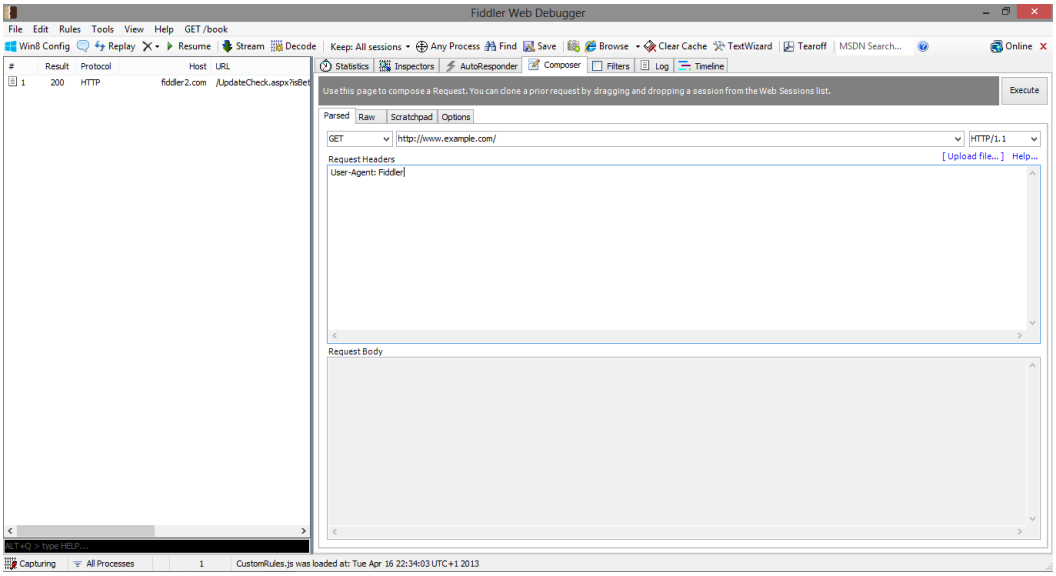
2.2.3. *Razhroščevalnik Fiddler*

Fiddler [6] (slika 4) je zastonjska aplikacija za razhroščevanje. Z njim lahko prožimo klice in preučujemo vrnjene podatke v XML ali JSON obliki. V primeru napake lahko le to hitreje odkrijemo in jo odpravimo.

Glavne funkcije Fiddlerja so:

- zajem vsega HTTP/HTTPS prometa,
- pregled vsebine sej,
- filtriranje zajetega prometa in
- arhiviranje in ponoven pregled zajetega prometa [7].

Z uporabo Fiddlerja smo prožili klice `PUT` in `POST` metod. V urejevalnik smo vnesli URL naslov servisa in objekte v obliki JSON zapisa.

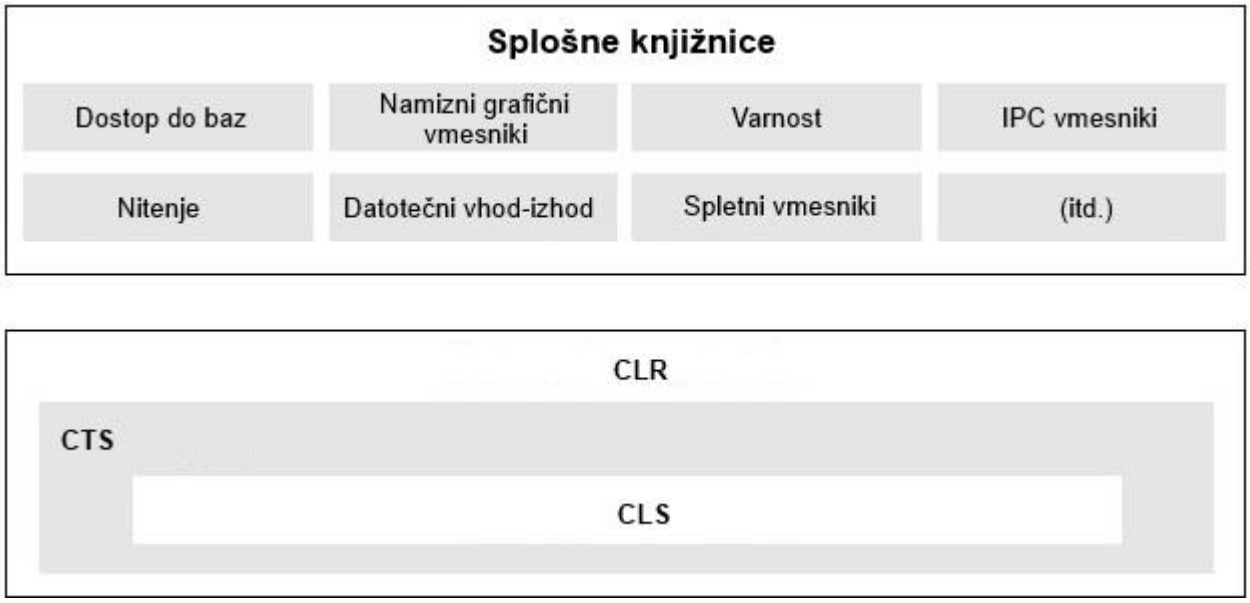


Slika 4: Razhroščevalnik Fiddler.

3. PREDSTAVITEV JEZIKA C# IN PLATFORME .NET

Ker programski jezik C# dobro poznamo, smo se odločili za .NET tehnologijo v kombinaciji z relacijsko podatkovno bazo Microsoft SQL. C# je namreč zelo eleganten objektno usmerjen programski jezik, ki temelji na platformi .NET Framework [8]. Oba sta bila prvič predstavljena leta 2002 z namenom ponuditi močnejši, fleksibilnejši in enostavnejši model za programiranje kot njun predhodnik model COM. Platforma .NET ima mnogo dobrih lastnosti:

- *Kompatibilnost z obstoječo programsko kodo*: nova platforma .NET lahko deluje v povezavi s starim COM programerskim modelom in obratno.
- *Podpora več programskim jezikom*: razvijalec lahko sam izbira med celo paleto programskih jezikov (C#, Visual Basic, F#,...).
- *CLR je skupen vsem jezikom .NET platforme*: tipi so zelo dobro razdelani in definirani, tako da jih razume vsak izmed jezikov platforme .NET.
- *Integracija več jezikov*: zaradi skupnega izvajalnega jedra .NET lahko podpira dedovanje in lovljenje izjem med različnimi jeziki. Tako lahko objekt definiramo v enem jeziku, razširimo pa ga v drugem.
- *Zelo domišljena splošna knjižnica*: omogoča tako nizkonivojske klice kakor tudi objektni model za vse jezike okolja .NET.
- *Enostavna objava*: za razliko od COM se knjižnice okolja .NET ne nahajajo v sistemskem registru. Platforma .NET omogoča več verzij iste `*.dll` knjižnice na istem računalniku [9].



Slika 5: Relacija med CLR, CTS, CLS in splošno knjižnico.

Glavna naloga CLR je lociranje, nalaganje in upravljanje z .NET objekti. Skrbi tudi za različne nizkonivojske operacije: upravljanje s pomnilnikom, koordinacija niti,... CLR je skupen vsem platforme .NET zavedajočim se jezikom (slika 5).

CTS opisuje vse podatkovne tipe in objekte, ki so podprti med izvajanjem. Natančno določa, na kakšne načine lahko te entitete komunicirajo med seboj ter kako so popisane v metapodatkih (slika 5).

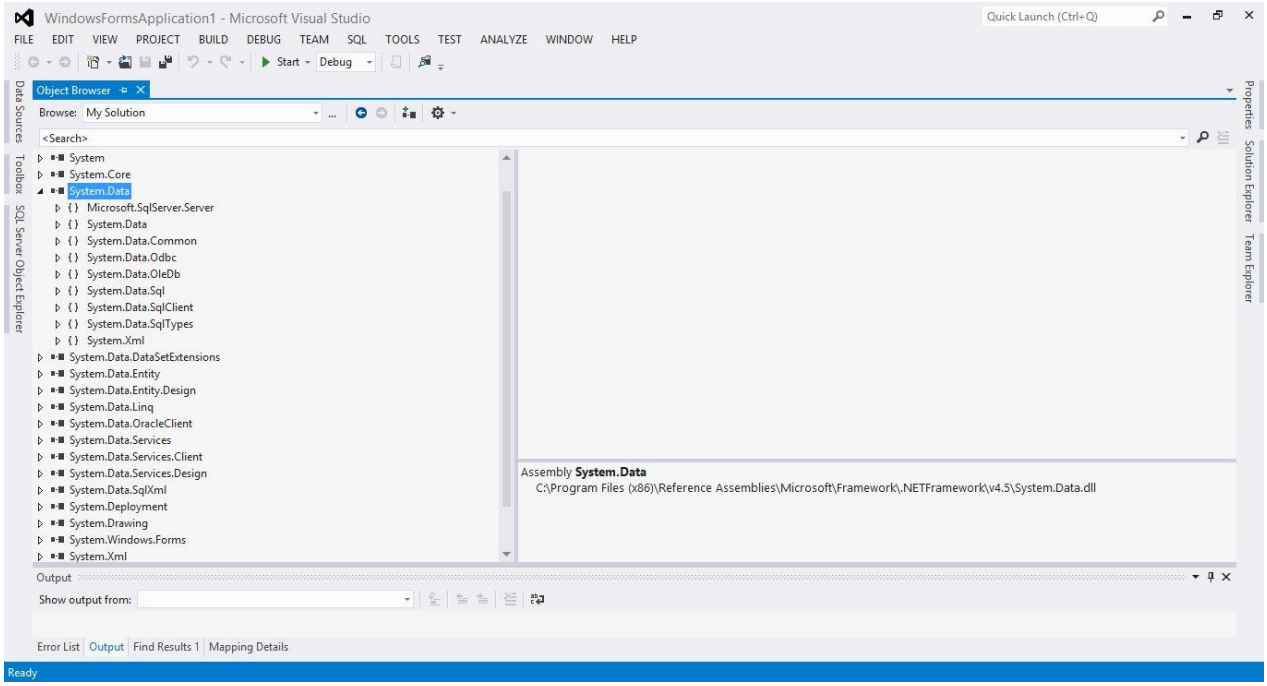
CLS je specifikacija, ki definira podmnožico tipov in objektov, ki so skupni vsem .NET programskim jezikom. Če programer upošteva meje specifikacij CLS, lahko svojo knjižnico uporabi v vseh .NET jezikih (slika 5).

Splošna knjižnica je skupna vsem .NET jezikom. Vsebuje različne primitivne elemente (niti, I/O, interakcijo z zunanjo strojno opremo,...) in je temelj za vse tipe aplikacij [10] (slika 5).

3.1. PREDSTAVITEV ADO.NET

Platforma .NET definira množico naslovnih prostorov, ki omogočajo povezljivost z različnimi relacijskimi podatkovnimi sistemi (*Microsoft SQL Server*, *Oracle*, *MySQL*). Skupek teh naslovnih prostorov je znan pod imenom ADO.NET. ADO.NET je naslednik modela ADO, ki je del programerskega modela COM. Temu navkljub imata razen imena bolj malo skupnega.

Za razliko od klasičnega ADO, ki je bil v prvi vrsti narejen za uporabo v tesno povezanih odjemalec/strežnik sistemih, je bil ADO.NET načrtovan v duhu zgolj občasne povezave med odjemalcem in strežnikom. Z uporabo tipa `DataSet` omogoča nalaganje podatkov v objekte tipa `DataTable`, kjer so shranjeni v tabelarični obliki (vrstica/stolpec). Tako lahko odjemalec podatke uporablja in jih spreminja tudi tedaj, ko ni povezan z virom podatkov, spremembe pa na podatkovnem nivoju izvede kasneje. Jedro ADO.NET predstavlja knjižnica `System.Data.dll` (slika 6).



Slika 6: `System.Data.dll` je jedro ADO.NET-a.

ADO.NET knjižnice lahko uporabljamo na tri konceptualno različne načine:

- povezano,
- nepovezano in
- z uporabo EF.

Te tri načine povezovanja bomo obravnavali tudi v nadaljevanju diplomske naloge. Na podlagi vnaprej izbranih kriterijev bomo poskušali izmed treh načinov izbrati tistega, ki za razvijalca pomeni največjo pridobitev s programerskega stališča in s stališča učinkovitosti - sprogramirati čim več v čim krajšem času in s čim manj napora.

Tip objekta	Korenski razred	Pomembni vmesniki	Pomen
`Connection`	`DbConnection`	`IDbConnection`	Skrbi za možnost vzpostavitve in prekinitve seje s podatkovno bazo. Skrbijo tudi za povezavo s transakcijskim objektom.
`Command`	`DbCommand`	`IDbCommand`	Objekt predstavlja poizvedbo SQL ali shranjeno proceduro. Prav tako skrbi za povezavo z objektom za branje.
`DataReader`	`DbDataReader`	`IDataReader`, `IDataRecord`	Skrbi za zaporedni bralni dostop do podatkov z uporabo kurzorja na strani strežnika.
`DataAdapter`	`DbDataAdapter`	`IDataAdapter`, `IDbDataAdapter`	Skrbi za podatke v `DataSetu`. Vsebuje povezavo in nabor štirih objektov za `select`, `update`, `insert` in `delete`.
`Parameter`	`DbParameter`	`IDataParameter`, `IDbDataParameter`	Predstavlja parameter v parametrizirani poizvedbi.
`Transaction`	`DbTransaction`	`IDbTransaction`	Enkapsulira transakcijo.

Tabela 1: Glavni objekti ADO.NET skrbnika povezav (ang. data provider).

ADO.NET ne zagotavlja skupnega nabora objektov za komunikacijo z različnimi tipi sistemov za upravljanje podatkovnih baz, temveč podpira nabor objektov za vsak podprt sistem. S takim pristopom ima programer odprte roke, saj lahko po potrebi razvije dodatne funkcionalnosti. Dodatna prednost tovrstnega pristopa je tudi direktna povezava brez vmesnika za prevajanje med objekti. Čeprav se nekatera imena razredov med skrbniki povezav (ang. data provider) razlikujejo (`SqlConnection`, `OracleConnection`, `OdbcConnection`, `MySqlConnection`) vsi izhajajo iz istega splošnega razreda (v tem primeru `DBConnection`). Tako se lahko programer, ki obvlada enega izmed objektov, zelo hitro nauči uporabe ostalih objektov. Tabela 1 vsebuje osnovne gradnike ADO.NET skrbnika povezav.

3.1.1. ADO.NET - POVEZANO

Pri uporabi tega načina odjemalec vzpostavi sejo s podatkovno bazo, izvede željene operacije in sejo prekine. Ob uporabi tega načina komunikacija z bazo običajno poteka z uporabo objektov za povezavo (`connection objects`), ukaze se pošilja z uporabo objektov za ukaze (`command objects`), vrnjena množica podatkov pa se prebira z objekti za branje (`reader objects`).

3.1.2. ADO.NET - NEPOVEZANO

Nepovezan način omogoča manipulacijo z množico objektov tipa `DataTable`, ki so del objekta `DataSet`. Celotna množica objektov se obnaša kot kopija podatkov na odjemalcu, ki se sicer nahajajo v podatkovni bazi. Podatke v `DataSet` pridobivamo z uporabo objekta `DataAdapter`, ki poskrbi za avtomatsko vzpostavitev in prekinitev seje. S takim pristopom pripomoremo k manjšemu številu povezav na bazo in s tem k večji skalabilnosti sistema.

Ko odjemalec podatke pridobi v obliki `DataSeta`, jih lahko uporablja ali spreminja. Ko želi, da se spremembe aplicirajo na bazi, za to ponovno poskrbi `DataAdapter` - ponovno vzpostavi sejo, posodobi podatke in nato povezavo takoj prekine.

3.1.3. ADO.NET - EF

EF uporablja ORM pristop. To pomeni, da podatke iz baze vrne v obliki strogo tipiziranih objektov, ki jih razvijalec lahko uporabi v svoji aplikaciji. EF model programerju omogoča interakcijo s podatkovno bazo z uporabo strogo tipiziranih LINQ poizvedb (z uporabo sintakse *LINQ to Entities*) [11].

3.2. OCENJEVANJE, KRITERIJI PRIMERJAVE IN UTEŽITEV KRITERIJEV

Po tehtnem premisleku smo se odločili, da bomo tri načine dostopa ocenjevali po principu Likertove skale [12] z ocenami od 1 do 5:

- 1 = zelo slabo,
- 2 = slabo,
- 3 = povprečno,
- 4 = dobro in
- 5 = zelo dobro.

Likertovo skalo bomo še nekoliko razširili: ocenjevali bomo na eno decimalko natančno.

Končno oceno za vsakega od načinov dostopa bomo računali po principu uteženega povprečja [13], rezultat vsote pa bomo prav tako matematično zaokrožili na eno decimalko natančno:

$$\sum \frac{\omega_1 \times o_1 + \omega_2 \times o_2 + \ \ldots}{\omega_1 + \omega_2 + \cdots}$$

Izbira ustreznih kriterijev primerjave treh opisanih načinov je ključnega pomena. V primeru izbora neprimernih kriterijev lahko namreč zelo hitro pride do popačenja dejanske slike in s tem do prednosti tehnologije, ki si tega dejansko ne zasluži.

Prav tako zelo pomembno vlogo igra ustrezna utežitev kriterijev. Kriterije bomo utežili z vidika programerja. Poudariti moramo, da so razvijalcu lahko pomembnejše druge stvari, kot uporabniku. Uporabniku bo verjetno pomembno, da bo neka rešitev hitro delovala, razvijalec pa si bo verjetno želel, da bo rešitev čim hitreje in čim bolj enostavno realiziral.

Primerjavo bomo delali na podlagi naslednjih šestih kriterijev:

- učna krivulja,
- dokumentiranost,
- čas realizacije,
- hitrost delovanja,
- prilagodljivost in
- objektnost.

Učna krivulja ima najmanjšo utež, ker je vložek v učenje tehnologije enkraten dogodek. Ko tehnologijo dokaj dobro poznamo, nadgrajevanje znanja z novimi vedenji ni več tako časovno potratno kot pred tem. **Utež: 5**.

Dokumentiranost je za programerja precej pomembna. Programer pri delu namreč pogosto naleti na problem, ki mu je nov. Če se je pred njim s tem problemom soočil že nekdo drug, našel ustrezno rešitev in problem ustrezno dokumentiral na spletu ali na kakem drugem mediju, to pomeni prihranek časa pri delu. **Utež: 20**.

Čas realizacije je verjetno pomemben tako izvajalcu kakor tudi uporabniku. Uporabnik običajno rešitev potrebuje v čim krajšem času, razvijalec pa želi projekt čim prej zaključiti. Zato ima čas realizacije najvišjo utež. **Utež: 25**.

Hitrost delovanja je običajno kompromis med enostavnostjo izvedbe in med sprejemljivostjo delovanja. V primeru, da neka rešitev deluje sprejemljivo hitro, programer ne izgublja časa z optimizacijo delovanja. **Utež: 15**.

Prilagodljivost je v primeru nadgradenj in vzdrževanja zelo pomemben faktor. Če je neka rešitev zasnovana premišljeno z mislijo na morebitne nadgradnje in spremembe, je za izvajalca v primeru le-teh vložek majhen. V nasprotnem primeru se lahko vložek drastično poveča. **Utež: 20.**

Objektnost se je izkazala za pomembno v primerjavi načinov dostopa do podatkovne baze. V primeru, da vrnjeni podatki niso strogo tipizirani, to za programerja pomeni več vrstic kode, saj se mora ukvarjati s pretvarjanjem med tipi. S tem se seveda podaljšuje čas realizacije. **Utež: 15.**

4. PRIPRAVA PODATKOV

4.1. PRIDOBIVANJE PODATKOV

Prvi korak je priprava ustreznih podatkov, nad katerimi izvajamo poizvedbe. Če bi naključno zgenerirali poljubno mnogo podatkov, nad takimi podatki ne bi mogli izvajati smiselnih poizvedb. Zato smo se odločili, da bomo različne načine dostopa do podatkovne baze primerjali na podlagi pravih podatkov in uporabili prosto dostopne podatke OpenStreetMap [14] (v nadaljevanju OSM). OSM je projekt, ki ustvarja in razširja bazo geografskih podatkov sveta. Ustvarjen je bil v duhu deljenja podatkov brez omejitev uporabe (uporaba proti plačilu, uporaba s posebnimi pogoji in podobno). Podatki so zaščiteni z ODbL licenco in jih lahko prosto uporabljamo, razširjamo in spreminjamo, če OSM navajamo kot vir [15].

Podatki so dostopni na spletu v obliki stisnjene XML datoteke [16]. Stisnjena je z metodo *BZip2* [17], ki z uporabo več zaporednih tehnik stiskanja doseže znatno zmanjšanje velikosti datoteke. Stisnjena datoteka je bila velika 25 GB, razpakirana datoteka pa 370 GB. Datoteko smo razpakirali s pomočjo programa *7zip* [18].

Ker je celotna baza OSM enostavno prevelika za naše potrebe, smo se odločili, da bomo uporabili samo del podatkov. V lokalno podatkovno bazo smo shranili le gostinske in učne ustanove [19] (tabela 2):

Gostinski objekti	
Bar	Prostor, kjer se prodaja alkoholne pijače.
Bbq	Javen prostor z žarom za pečenje mesa in/ali zelenjave. Običajno se nahaja zunaj.
Biergarten	Zunanji prostor, kjer strežejo pivo in lokalno hrano. Strežejo lahko tudi druge pijače.
Cafe	Prostor, kjer prodajajo tople napitke in prigrizke.
Drinking_water	Vir pitne vode.
Fast_food	Restavracije s hitro hrano.
Food_court	Območje večih restavracij s skupnim prostorom za goste. Taki prostori se običajno nahajajo v nakupovalnih centrih, letališčih...
Ice_cream	Prodajalne sladoleda.
Pub	Prostor, kjer prodajajo pivo in ostale pijače. Ponekod ponujajo tudi hrano in prenočišča (Velika Britanija).
Restaurant	Restavracije.
Učne ustanove	
College	Študentsko naselje ali stavbe.
Kindergarten	Otroški vrtec.
Library	Knjižnica.
School	Šola.
University	Univerza.

Tabela 2: Vrste učnih ustanov in gostinskih objektov.

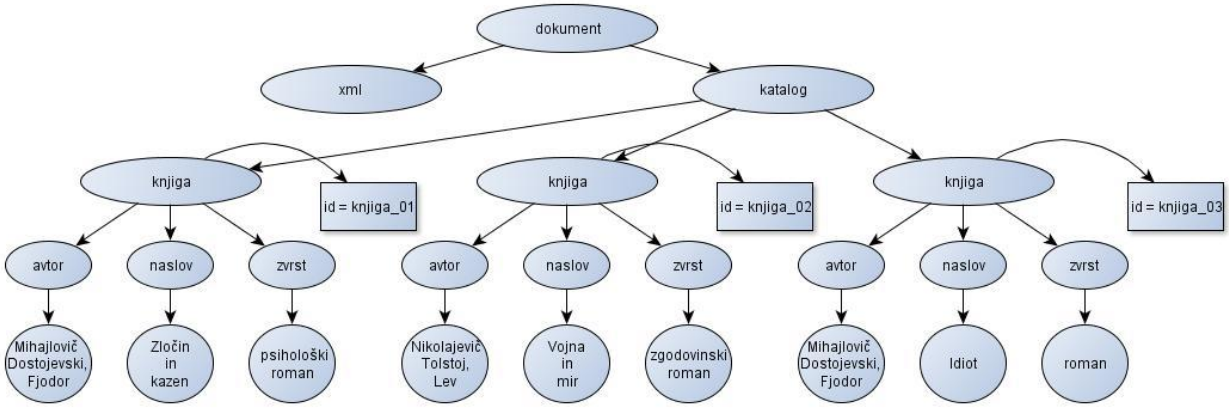
4.2. RAZČLENJEVANJE XML DATOTEKE

Sledilo je razčlenjevanje XML datoteke ter pridobivanje želenih podatkov. Najprej smo poskusili z uporabo razreda `XmlDocument` [20].

`XmlDocument` je razred, ki izvaja W3C DOM [21]. Glavna funkcija DOM-a je možnost spreminjanja dokumenta. DOM podatke v pomnilniku predstavi kot drevesno strukturo (slika 7) kljub temu, da so XML podatki v datoteki shranjeni v linearnem načinu. Kakšna je razlika med linearnim in strukturiranim načinom si lahko pogledamo na primeru izvorne kode 1.

```xml
<?xml version="1.0"?>
<katalog>
   <knjiga id="knjiga_01">
      <avtor>Mihajlovič Dostojevski, Fjodor</avtor>
      <naslov>Zločin in kazen</naslov>
      <zvrst>psiholoski roman</zvrst>
   </knjiga>
   <knjiga id="knjiga_02">
      <avtor>Nikolajevič Tolstoj, Lev</avtor>
      <naslov>Vojna in mir</naslov>
      <zvrst>zgodovinski roman</zvrst>
   </knjiga>
   <knjiga id="knjiga_03">
      <avtor>Mihajlovič Dostojevski, Fjodor</avtor>
      <naslov>Idiot</naslov>
      <zvrst>roman</zvrst>
   </knjiga>
</katalog>
```

Izvorna koda 1: XML datoteka v linearnem načinu.



Slika 7: Vizualna podoba drevesne strukture zgornje XML datoteke.

Ker razred `XmlDocument` podpira le izvajanje nad dokumentom kot celoto (nalaganje podatkov v pomnilnik ali shranjevanje podatkov v dokument), se je tak pristop izkazal za neuporabnega, saj 370 GB velike datoteke ne moremo kar naložiti v pomnilnik.

Nato smo v dokumentaciji našli razred `XmlReader` [22], ki je hiter in dokument obdela v obliki toka (ang. stream) ter ga lahko v pomnilnik nalaga po fragmentih. V primerjavi z razredom `XmlDocument` ima sicer to slabo lastnost, da omogoča le bralni zaporedni dostop do dokumenta, vendar to glede na naše potrebe ni ovira.

Kot eden od učinkovitejših načinov uporabe razreda `XmlReader` za branje `XElement` objektov velja način z uporabo tako imenovane središčne metode (ang. axis method). Središčna metoda vrača zbirko `IEnumerable` objektov tipa `XElement`. Ta metoda mora vračati zbirko `IEnumerable`, če želimo pri vračanju uporabiti ključno besedo `yield`, z uporabo katere središčno metodo označimo kot iterator. Poleg tega so pogoji za uporabo ključne besede `yield` tudi:

- prepoved uporabe ključnih besed `ref` in `out` pri vhodnih parametrih,
- središčna metoda ne sme biti anonimna (ang. anonymous method [23]) in
- središčna metoda ne sme vsebovati blokov s ključno besedo `unsafe` [24].

Z uporabo razreda `XmlReader` v kombinaciji z razredom `XmlWriter` smo iz podatkov OSM izluščili zgolj tiste podatke, ki nas zanimajo in jih zapisali v ločeno XML datoteko (izvorna koda 2).

```
static void Main(string[] args)
{
	XmlWriterSettings xws = new XmlWriterSettings();
	xws.OmitXmlDeclaration = true;
	xws.Indent = true;
	using (XmlWriter xw = XmlWriter.Create(dest_file_path, xws))
	{
		xw.WriteStartElement("Root");
		foreach (XElement node in GetNode().Select(g => g))
		{
			node.WriteTo(xw);
		}
		xw.WriteEndElement();
	}
}

private static IEnumerable<XElement> GetNode()
{
	string[] amenities = new string[]
	{
		"bar", "bbq", "biergarten", "cafe", "drinking_water", "fast_food",
		"food_court", "ice_cream", "pub", "restaurant", "college",
		"kindergarten", "library", "school",  "university"
	};

	using (XmlReader reader = XmlReader.Create(file_path))
	{
		XElement node = null;
		reader.MoveToContent();

		while (reader.Read())
		{
			if (reader.NodeType == XmlNodeType.Element && reader.Name == "node")
			{
				node = XElement.ReadFrom(reader) as XElement;

				foreach (XElement child in node.Elements())
				{
					if (amenities.Contains(child.Attribute("v").Value)
					    && child.Attribute("k").Value == "amenity")
					{
						yield return node;
					}
				}
			}
		}
	}
}
```

Izvorna koda 2: Pridobivanje podatkov iz izvorne XML datoteke in shranjevanje le-teh v ločeno XML datoteko.

4.3. ANALIZA PODATKOV

Ko so bili želeni podatki ločeno shranjeni v svojem XML dokumentu, smo analizirali podatke. Najprej nas je zanimalo število zapisov različnih lokacij. S ponovnim sprehodom čez XML smo ugotovili, da smo s prej opisanim načinom pridobili 1023635 različnih entitet, kar je zelo lepa množica podatkov za primerjanje značilnosti različnih dostopov do podatkovne baze.

Nato smo se lotili analize lastnosti posameznih entitet. Zanimalo nas je koliko oziroma kakšne lastnosti nastopajo v XML dokumentu. Ponovo smo se sprehodili čez XML in v zbirko `IEnumerable<string>` zbrali vse različne lastnosti. Vmesnik `IEnumerable<T>` je generičen vmesnik, ki omogoča enostavne iteracije čez objekte v zbirki. Omogoča tudi uporabo razširitvenih metod iz knjižnice `System.Linq`. Na vrnjeni zbirki smo nato s pomočjo tehnologije *LINQ to Object* naredili enostavno poizvedbo (izvorna koda 3).

```
var output = getUniqueAttirbutes().GroupBy(a => a).OrderByDescending(a =>
              a.Count()).Select(a => new
              {
                     value = a.Key,
                     count = a.Count()
             });
```

Izvorna koda 3: Analiza entitet - pridobivanje vseh različnih atributov.

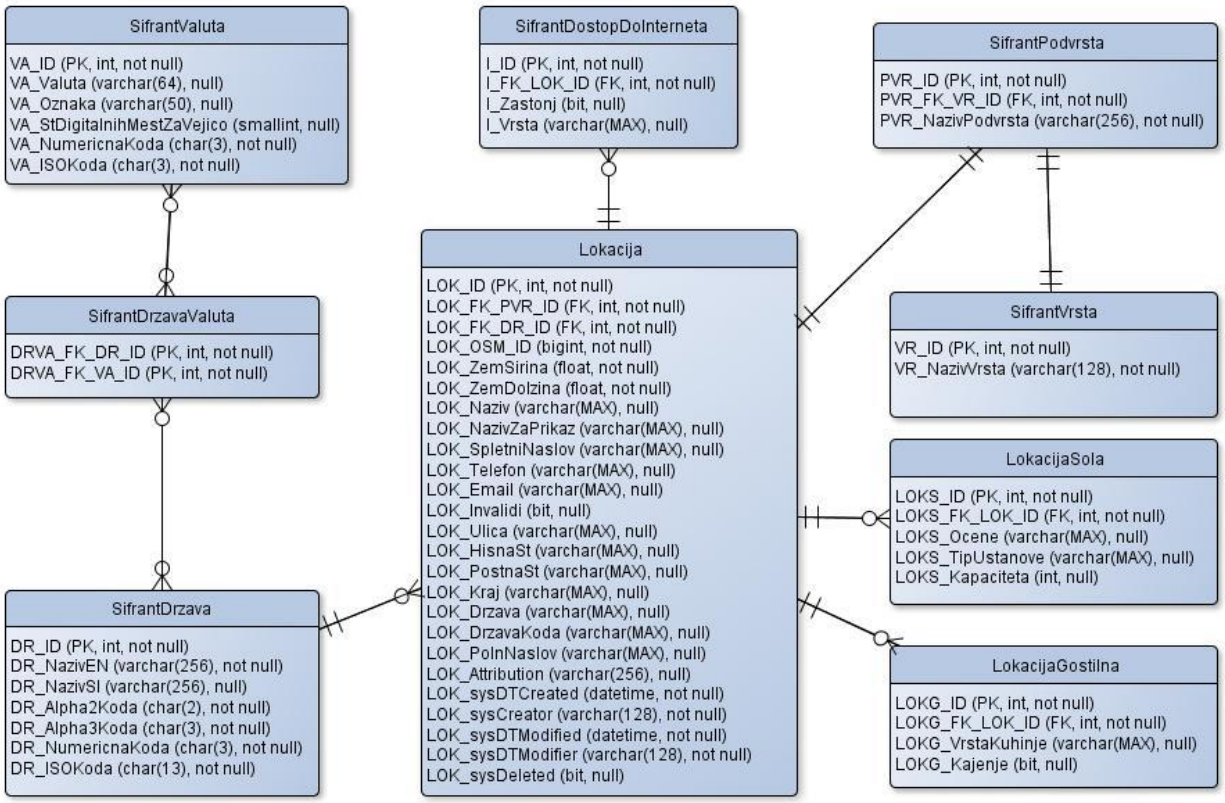
Dobljene vrednosti smo grupirali in jih padajoče uredili glede na seštevek atributov po posamezni skupini. Vse skupaj smo morali shraniti v spremenljivko implicitnega tipa `var`, saj je rezultat poizvedbe zbirka anonimnih tipov. Ime vrnjenega tipa je dostopno le prevajalniku samemu. Na ta način smo pridobili vseh 3237 atributov, ki se pojavljajo v naših podatkih. Velik delež se jih ponovi zgolj nekajkrat, zato smo kar takoj izločili vse, ki se ponovijo manj kot stokrat. Tako smo v ožji izbor pridobili 435 atributov, ki smo jih pregledali ročno. Nad pridobljenimi podatki smo bili razočarani, saj so resnično neurejeni. Obsežni OSM dokumentaciji navkljub ljudje dodajajo zapise povsem po svoje. Iz tabele 3 je razvidno, kateri atributi vse označujejo naziv lokacije.

Atribut	Število ponovitev
Name	882350
name:en	26649
addr:housename	7776
alt_name	3565
int_name	2155
old_name	1669
official_name	391
short_name	110

Tabela 3: Primer atributov, ki označujejo naziv lokacije.

4.4. LOGIČNI PODATKOVNI MODEL

Na podlagi podatkov, ki smo jih pridobili iz XML datoteke, smo pripravili podatkovni model na sliki 8.



Slika 8: Logični podatkovni model.

V podatkovnem modelu nastopajo sledeče entitete:

- `Lokacija`: v tej tabeli se hranijo glavni podatki o lokacijah, ki so skupni tako učnim kot gostinskim ustanovam. Tu hranimo pozicijo posamezne lokacije (zemljepisna širina in dolžina), naslov in ostale osnovne podatke. Hranimo tudi ključ iz OSM, ki bi ga lahko uporabili v primeru, če bi želeli podatek posodobiti s pomočjo orodja Nominatim, ki omogoča reverzno geokodiranje [25]. Šifrant države smo z lokacijami povezali na podlagi kod držav, ki jih je bilo na žalost bolj malo. Če bi želeli med seboj povezati še več držav in lokacij, bi prav tako morali izvajati klice na servis Nominatim. Zaradi velike količine klicev v kratkem času bi nam verjetno prepovedali uporabo servisa, zato tega nismo storili.
- `LokacijaSola`: tu se hrani nekaj podatkov, ki so specifični za učne ustanove. Hranimo načine ocenjevanja, tip ustanove in kapaciteto učencev/dijakov/študentov, ki jih določena ustanova lahko sprejme.
- `LokacijaGostilna`: v tej tabeli so shranjeni podatki skupni gostinskim objektom. Hranimo vrsto kuhinje (če neka gostilna poleg pijače ponuja tudi hrano) in podobno, če je kajenje dovoljeno.
- `SifrantDostopDoInterneta`: tabela hrani podatek o tem, ali je na neki lokaciji razpoložljiva internetna povezava, za kakšen tip povezave gre (eternet, brezžična povezava) in ali se na neki dostopni točki lahko na splet povežemo zastonj.
- `SifrantValuta`: tabela je namenjena hranjenju podatkov o valutah. Hranijo se sledeči podatki: valuta (npr. kanadski dolar) z oznako (npr. $), število decimalnih mest za decimalno vejico, numerična koda, ki je po ISO 4217 standardu in ISO koda. Slednja je sestavljena iz dvomestne kode države (npr. CA), tretji znak pa je inicialka valute (torej D, skupaj CAD). Podatke smo pridobili na wikipediji (standard ISO 4217 [26] in seznam valut [27]).
- `SifrantDrzavaValuta`: je povezovalna tabela med tabelama `SifrantValuta` in `SifrantDrzava`.
- `SifrantDrzava`: šifrant držav hrani podatke o državah. Hrani angleški in slovenski naziv. Tabela hrani tudi podatke po standardu ISO 3166-1 [28]:
  - dvomestno oznako posamezne države, ki je izmed kod tudi najbolj uporabljana,
  - trimestno oznako in
  - numerično kodo, ki je trimestna numerična oznaka.

  Tabeli `SifrantDrzava` in `SifrantValuta` smo povezovali na podlagi kod. Nekaj primerkov smo morali povezati tudi ročno.

4.5. VSTAVLJANJE PODATKOV V PODATKOVNO BAZO

Vstavljanja podatkov v podatkovno bazo smo se lotili z uporabo razreda `XmlReader` in tehnologije *Linq To Entities*. Iz naše XML datoteke s podatki o lokacijah smo zaporedno prebirali zapis za zapisom in jih vstavljali v našo podatkovno bazo. Vendar smo z uporabo te metode naleteli na problem - časovna zahtevnost vstavljanja enega elementa je začela naraščati. Tako je čas vnosa iz manj kot ene sekunde na zapis zelo hitro poskočil na tri sekunde na zapis. S tem se seveda celoten čas zapisovanja v bazo trikrat poveča, kar pomeni, da bi s tem načinom namesto malo manj kot 12 dni zapisovanja porabili praktično 36 dni! Seveda nista ne prvi ne drugi scenarij dobra za vsakodnevno rabo. Bi se pa za razliko od druge variante s prvo lažje zadovoljili, saj gre za enkratni dogodek.

Zaradi prevelike časovne zahtevnosti vstavljanja podatkov na zgoraj opisani način smo poskušali najti boljši (hitrejši) način za vnos podatkov v podatkovno bazo. Naleteli smo na operacijo `bulk insert` [29], ki omogoča množični uvoz podatkov iz dateteke tipa XML. V bazi smo pripravili tabelo, ki nam je služila kot vmesna tabela (tabela 4).

Entities_XML		
Stolpec	**Tip**	**Opis**
XML_ID	bigint, not null	Enolični identifikator zapisa.
XML_OSM_ID	bigint, not null	Identifikator zapisa iz OSM.
XML_Lat	float, null	Zemljepisna širina lokacije.
XML_Lon	float, null	Zemljepisna dolžina lokacije.
XML_Tag_K	varchar(MAX), null	Naziv podatka.
XML_Tag_V	varchar(MAX), null	Vrednost podatka.

Tabela 4: Struktura tabele Entities_XML.

```
DECLARE @xml XML = (SELECT CONVERT(xml, BulkColumn, 2)
                                    FROM OPENROWSET(Bulk 'C:\Users\Marko\Desktop\test.xml',
SINGLE_BLOB) data)

INSERT INTO Entities_XML (XML_OSM_ID, XML_LAT, XML_LON, XML_Tag_K, XML_Tag_V)
SELECT  [XML_OSM_ID] = myParsedNode.NodeEntity.value('@id', 'bigint'),
              [XML_LAT] = myParsedNode.NodeEntity.value('@lat', 'float'),
              [XML_LON] = myParsedNode.NodeEntity.value('@lon', 'float'),
              [XML_Tag_K] = myParsedTag.TagEntity.value('@k', 'varchar(max)'),
              [XML_Tag_V] = myParsedTag.TagEntity.value('@v', 'varchar(max)')
FROM    @xml.nodes('//root') AS myParsedXml(Entity)
                     CROSS APPLY Entity.nodes('node') AS myParsedNode(NodeEntity)
                     CROSS APPLY NodeEntity.nodes('tag') AS myParsedTag(TagEntity)
```

Izvorna koda 4: Vstavljanje atributov iz izvorne XML datoteke v vmesno tabelo `Entities_XML`.

Nato smo vmesno tabelo napolnili s podatki iz fizične XML datoteke (izvorna koda 4).

Vsebino XML dokumenta smo naprej z operacijo `bulk insert` shranili v spremenljivko tipa `XML`. Na tem mestu velja opomniti, da smo z uporabo tipa `XML` omejeni na $2^{31} - 1$ bajtov prostora kar nanese nekaj več kot 2 gigabajta [30].

Ker atributi še zdaleč niso enotni, smo se odločili za vrstičen vnos, kar pomeni, da vsak podatek lokacije dobi eno vrstico in ne en stolpec. Tako smo morali pripraviti še nekaj procedur, ki so poskrbele za pravilen vnos podatkov iz tabele `Entities_XML` v ostale tabele.

Tako smo uspeli iz ogromne množice le deloma urejenih podatkov pridobiti željeno podmnožico podatkov, ki smo jih uspeli sistematično urediti in jih shraniti v strukturirani obliki. Ob urejanju podatkov smo spoznali več različnih tehnologij. Kljub temu, da smo nekajkrat zašli v slepo ulico, ker so se določene tehnologije za naše potrebe izkazale za neuporabne, smo nalogo uspešno izvedli.

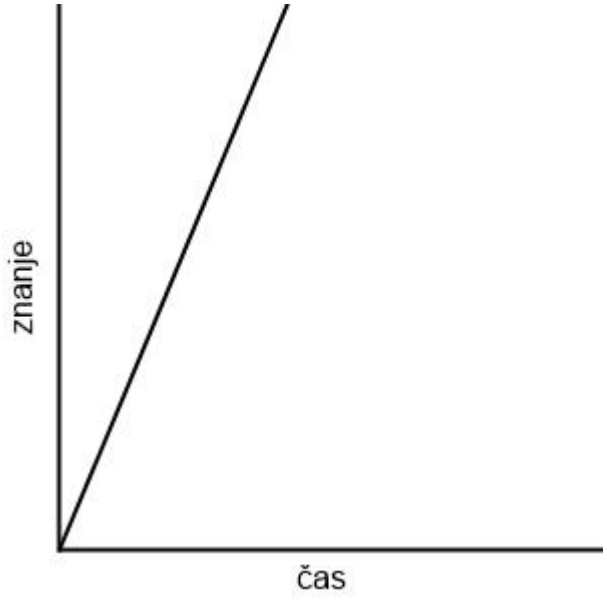
5. PRIMERJAVA NAČINOV DOSTOPA PO KRITERIJIH

5.1. KRIVULJA UČENJA

Krivulja učenja [31] je grafična predstavitev naraščanja znanja (y os) v odvisnosti od vloženega časa (x os). V primeru večkratnih ponovitev je povprečje ponovitev ponavadi lepa krivulja, ki jo lahko običajno opišemo z matematično funkcijo. Krivulja učenja je pomemben faktor pri izbiri tehnologij, saj so običajno mejniki na projektih zelo na gosto postavljeni, zato v učenje neznanih zapletenih tehnologij v takih situacijah pogosto ne moremo vlagati preveč časa.

5.1.1. ADO.NET - POVEZANO

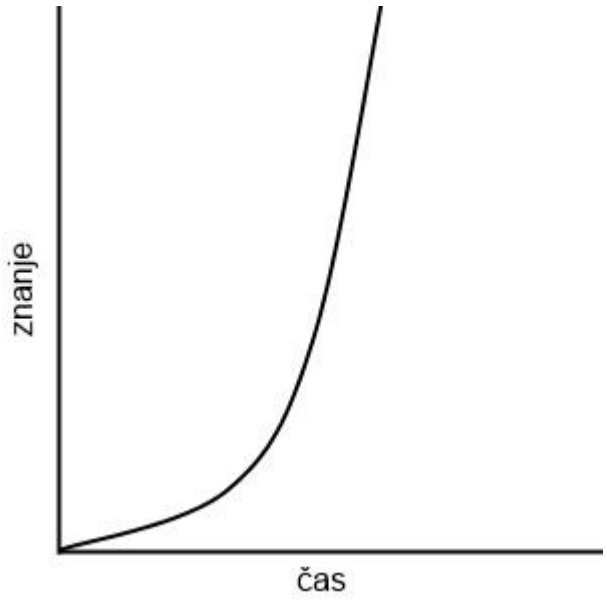
Povezan način dostopa do podatkovne baze je sila preprost. Že po nekajminutnem prebiranju spletne dokumentacije ter po pregledu nekaj primerov [32, 33] smo se na ta način lahko povezali na podatkovno bazo in vračali podatke. Seveda je znanje iz pisanja SQL poizvedb predpogoj za uporabo tega načina dostopa. Učna krivulja tega načina je razvidna iz slike 9. **Število točk: 4,5**.



Slika 9: Krivulja učenja ADO.NET - povezano.

5.1.2. ADO.NET - *NEPOVEZANO*

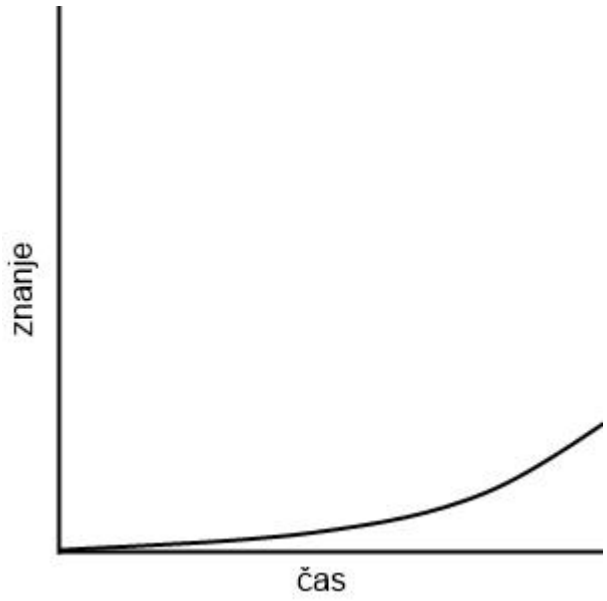
V učenje nepovezanega načina smo morali vložiti nekoliko več truda kot v učenje povezanega načina. Znanje smo zopet črpali iz spletne dokumentacije [34]. Poglobiti se je bilo treba v sam koncept, na podlagi katerega delujejo in so povezani objekti tipov `DataSet`, `DataTable` in `DataAdapter`. Naučiti smo se morali delati tudi z grafičnim vmesnikom, ki nam omogoča, da povezavo praktično poklikamo, v kodi pa z uporabo zgolj nekaj vrstic kasneje delamo poizvedbe. Za učenje smo porabili približno šest ur. Znanje iz pisanja SQL poizvedb je tudi v tem primeru predpogoj. Učna krivulja tega načina je razvidna iz slike 10. **Število točk: 3,5**.



Slika 10: Krivulja učenja ADO.NET - nepovezano.

5.1.3. *ENTITY FRAMEWORK*

V učenje tehnologij EF smo vložili daleč največ časa. Za spoznavanje z uporabniškim vmesnikom za delo z entitetnim modelom in za spoznavanje s sintakso komponente *LINQ To Entities* [35, 36], ki omogoča pisanje poizvedb na entitetni model z uporabo jezika C#. Zavedati se moramo, da sta prej navedeni možnosti v nepravični prednosti, ker je za njuno uporabo potrebeno znanje jezika SQL. Ker ga vsak razvijalec načeloma pozna, potrebuje za učenje manj časa kot za učenje EF. Za spoznavanje osnov smo potrebovali približno dva dni. Učna krivulja tega načina je razvidna iz slike 11. **Število točk: 2.**



Slika 11: Krivulja učenja EF.

5.2. DOKUMENTIRANOST

Količina in kvaliteta dostopne dokumentacije, količina različnih primerov uporabe na forumih in portalih ter število knjig na določeno temo je za programerja ključnega pomena. Hitrost programiranja v neki tehnologiji je namreč zelo odvisna tudi od virov različnih znanj, ki so programerju dostopna.

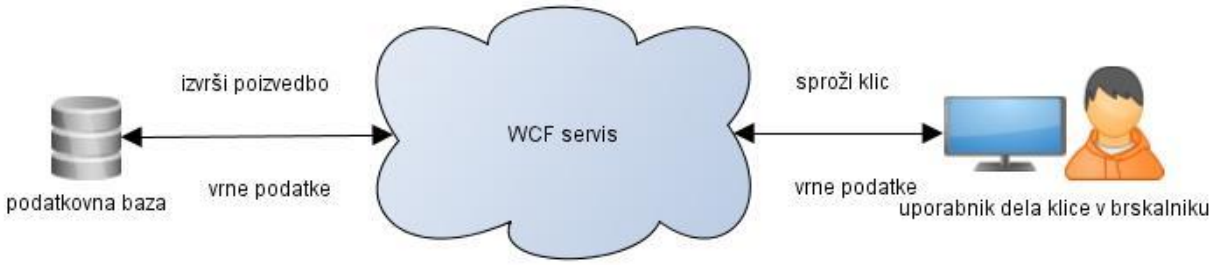
Vse tri tehnologije so dobro popisane v Microsoftovi spletni dokumentaciji. Za vse tri vrste se na spletu najde veliko primerov. V splošnem izstopata portala Stack Overflow [37] in Code Project [38]. Malenkostno se mogoče pozna, da je EF s komponento *LINQ to Entities* najmlajša tehnologija od navedenih (v .NET Framework je bil dodan s servisnim paketom (ang. Service Pack) in Visual Studio 2008 servisnim paketom 1 v avgustu 2008 [39]). Prav tako je na voljo tudi kar nekaj kvalitetne literature, ki opisuje tudi temo dostopa do baz. Povezan način: **5 točk**, nepovezan način: **4,5 točke** in EF: **4 točke**.

5.3. ČAS REALIZACIJE

Čas realizacije je čas od začetka do konca izvedbe neke rešitve. Za primerjavo vseh treh načinov dostopa smo si izbrali poizvedbo, ki smo jo poskušali izvesti v vseh načinih. Poizvedba v opisni obliki se glasi:

Vrni vse lokacije, ki so tipa 'gostinski objekt', ki ponuja zastonj brezžični dostop do interneta in je od Ljubljane (46.055556° N, 14.508333° E) oddaljen več kot 100 kilometrov in hkrati manj kot 1000 kilometrov. Zadetki naj bodo razvrščeni od najbližjega proti najbolj oddaljenemu.

Za končni produkt smo si zamislili WCF servis s tremi klici - za vsak tip dostopa svoj klic. Kot končni cilj smo si zastavili prikaz podatkov v brskalniku. Diagram poteka ob proženju klicev je viden na sliki 12.



Slika 12: Diagram poteka ob proženju klicev.

Računanja geografske razdalje smo se lotili s Havershinovo formulo [40, 41], ki smo jo na SQL strežniku spisali v obliki skalarne funkcije, ki se kliče iz poizvedbe. Za preučitev formule smo porabili pol ure.

5.3.1. ADO.NET - POVEZANO

V tem primeru smo se najprej lotili pisanja skalarne funkcije (izvorna koda 5) in SQL poizvedbe (izvorna koda 6), ki je vračala pravilne rezultate. Oboje skupaj nam je vzelo pol ure časa. Ko smo imeli poizvedbo dokončano, smo se lotili še pisanja logike na WCF servisu (izvorna koda 7), kar je trajalo še dodatne pol ure. **Število točk: 4**.

```
DECLARE @radius FLOAT = 6371;
DECLARE @lat_lj FLOAT = RADIANS(46.055556);
	DECLARE @lon_lj FLOAT = RADIANS(14.508333);

	SET @lat = RADIANS(@lat);
	SET @lon = RADIANS(@lon);

	DECLARE @dlat FLOAT = @lat_lj - @lat, @dlon FLOAT = @lon_lj - @lon;

	RETURN 2 * @radius * ASIN(SQRT(POWER(SIN(@dlat / 2), 2) + COS(@lat_lj) * COS(@lat)
		* POWER(SIN(@dlon / 2), 2)));
```

Izvorna koda 5: Skalarna funkcija za izračun geografske razdalje med dvema točkama na podlagi geografskih širin in dolžin.

```
SELECT LOK_Naziv, LOK_OSM_ID,
	ROUND(dbo.HavershineFormula(LOK_ZemSirina, LOK_ZemDolzina), 2) AS distance
FROM Lokacija INNER JOIN LokacijaGostilna ON LOK_ID = LOKG_FK_LOK_ID
		INNER JOIN SifrantDostopDoInterneta ON LOK_ID = I_FK_LOK_ID
WHERE I_Zastonj = 1 AND I_Vrsta = 'wireless'
	AND dbo.HavershineFormula(LOK_ZemSirina, LOK_ZemDolzina) > 100
	AND dbo.HavershineFormula(LOK_ZemSirina, LOK_ZemDolzina) < 1000
ORDER BY dbo.HavershineFormula(LOK_ZemSirina, LOK_ZemDolzina) ASC
```

Izvorna koda 6: Poizvedba SQL, ki smo jo združili v servis.

```
using (SqlConnection conn = new SqlConnection(connectionstring)) {
	conn.Open();
	using (SqlCommand command = new SqlCommand(sb.ToString(), conn)
		{ CommandType = CommandType.Text })
	{
		using (SqlDataReader reader = command.ExecuteReader())
		{
			while (reader.Read())
			{
				lLokacija.Add(new Lokacija
				{
					name = Convert.ToString(reader["LOK_Naziv"]),
					OSM_ID = Convert.ToInt64(reader["LOK_OSM_ID"]),
					distance = Convert.ToDouble(reader["distance"])
				});
			}
		}
	}
	conn.Close();
}
```

Izvorna koda 7: Fragment kode, ki je potrebna za pridobitev podatkov.

5.3.2. *ADO.NET - NEPOVEZANO*

Ker tudi ta način dostopa temelji na SQL poizvedbi, smo uporabili že kar pripravljeno poizvedbo. Nato smo v projekt dodali `DataSet` in v grafičnem vmesniku poklikali `TableAdapter`, na podlagi katerega se nam je zgenerirala tudi `DataTable`, ki hrani podatke. Programske kode je v primerjavi s prvo varianto občutno manj; to je vidno tudi iz spodaj priložene izvorne kode 8. Programiranje na servisu nam je vzelo 15 minut. **Število točk: 4,5**.

```
public DiplomskaNaloga.LokacijaTableDataTable SelectGostilneDisconnected()
{
	DiplomskaNaloga.LokacijaTableDataTable dt;
	dt = new DiplomskaNaloga.LokacijaTableDataTable();
	LokacijaTableAdapter ta = new LokacijaTableAdapter();

	ta.Fill(dt);

	return dt;
}
```

Izvorna koda 8: Količina kode je v tem primeru bistveno manjša kot v prejšnjem.

5.3.3. *ENTITY FRAMEWORK*

Pri uporabi tehnologij EF smo naleteli na največ težav. Prvi problem se je pojavil pri kreiranju EDMX sheme. Čeprav je skalarno funkcijo za izračun geografske razdalje v model dodalo, je nismo mogli uvoziti, ker je v spustnem seznamu v oknu za uvoz funkcij in shranjenih procedur sploh ni bilo. Na spletu smo nato naleteli na prispevek na forumu MSDN [42], kjer je predstavljen enak problem. V odgovorih smo zasledili, da UDF v EF zaenkrat še niso podprte. Nato smo na portalu Stack Overflow naleteli na prispevek, kjer je popisano, kako lahko ročno popravimo EDMX datoteko [43]. Žal tudi ta rešitev ni bila zadovoljiva. V primeru enkratnega klica funkcije je le-ta vrnila rezultat in ga shranila v spremenljivko. V primeru klica funkcije v *LINQ to Entities* poizvedbi pa je prišlo do izjeme. Pojavila pa se je še težava, da EDMX sheme preko grafičnega vmesnika nismo mogli več posodabljati, tako da smo navsezadnje morali odstraniti ročno narejene spremembe. Poskusili smo narediti tudi klic skalarne funkcije iz shranjene procedure, vendar slednje ne moremo klicati iz `where` pogoja (prevajalnik sicer ne javi napake, pride pa do napake ob sproženem klicu). Z EF smo se ukvarjali več kot dve uri, do celovite rešitve pa v tem času nismo prišli (izvorna koda 9). **Število točk: 1**.

```
public List<Lokacija> SelectGostilneEF()
{
	using (DiplomskaNalogaEntities dnContext = new DiplomskaNalogaEntities())
	{
		return dnContext.Lokacijas.Join(dnContext.LokacijaGostilnas, l => l.LOK_ID,
			lg => lg.LOKG_FK_LOK_ID, (l, lg) => new { l, lg })
			.Join(dnContext.SifrantDostopDoInternetas, lgl => lgl.l.LOK_ID,
			i => i.I_FK_LOK_ID, (lgl, i) => new { lgl, i })
			.Where(lgi => lgi.i.I_Zastonj == true && lgi.i.I_Vrsta == "wireless")
			.Select(l => new Lokacija
			{
				name = l.lgl.l.LOK_Naziv,
				OSM_ID = l.lgl.l.LOK_OSM_ID
			})
			.OrderBy(l => l.distance)
			.ToList<Lokacija>();
	}
}
```

Izvorna koda 9: LINQ to Entities brez pogojev razdalje.

5.4. HITROST

Hitrost smo merili od trenutka, ko je servis vzpostavil povezavo z bazo do trenutka, ko je vrnil objekt, pripravljen na serializacijo v JSON. Ker povezan način dostopa ne vrača strogo tipiziranih objektov, smo podatke iz `DataReaderja` shranili v objekt, ki smo ga zasnovali sami. Podobno smo storili tudi pri nepovezanem načinu dostopa, saj objekta `DataTable` ne moremo vrniti v obliki JSON zapisa. Da je meritev relevantna, smo tudi pri uporabi *LINQ to Entities* vračali naš objekt, zasnovan po meri. Meritve smo izvajali z uporabo sledečih ključnih besed, ki so v splošnem med bolj uporabljanimi:

- `select`
- `select` in `join`
- `select` in `distinct`
- `select` s ključno besedo `like` v `where` pogoju
- `select` in `order by`
- `insert`
- `update`

Poizvedbo za vsako od kombinacij smo zagnali petkrat zaporedoma, upoštevali smo povprečen čas izvajanja. Čas smo merili v sekundah na tri decimalna mesta natančno.

Kot je razvidno iz izbora, smo se osredotočili na tri glavne sklope poizvedb: `select`, `insert` in `update`. Ker je poizvedba `select` v splošnem daleč najbolj uporabljana, smo jo izvedli petkrat v kombinaciji z različnimi ključnimi besedami. Vsako izmed petih kombinacij smo glede na hitrost ocenili, nato smo izračunali povprečje vseh ocen ter pridobljen rezultat matematično zaokrožili. Najhitrejši način dobi 5 točk, drugi najhitrejši način dobi 3 točke, zadnji pa 1 točko. Upoštevali smo tudi možnost napake $\pm 0{,}01$ sekunde. V primeru, da je razlika med dvema meritvama manjša od napake, se štejeta kot enako hitra. Tako smo pridobili povprečno oceno poizvedbe `select`. Podobno smo ocenili tudi poizvedbi `insert` in `update`. Ker smo se odločili, da bomo na vrhnjem nivoju naše primerjave poizvedbe `select`, `insert` in `update` enakovredno upoštevali, smo izračunali povprečje ocen za vsak način dostopa ter ga prav tako matematično zaokrožili.

- `SELECT`

Najprej smo izvedli klasičen `select` stavek (tabela 5). Izbrali smo vse zapise iz tabele `Lokacija`, kjer se nahaja 1023635 zapisov. Presenečeni smo bili, da se je poizvedba *LINQ to Entities* v povprečju izvedla najhitreje.

SELECT			
	Povezan	*Nepovezan*	*EF*
Meritev 1	7,812	31,235	5,191
Meritev 2	7,842	31,087	5,183
Meritev 3	7,855	31,664	5,126
Meritev 4	7,916	33,138	5,186
Meritev 5	7,859	31,613	5,227
Povprečje	**7,857**	**31,748**	**5,183**

Tabela 5: Klasičen `select`.

Ocene:

- povezan način: **3 točke**,
- nepovezan način: **1 točka** in
- EF: **5 točk**.

- `SELECT` in `JOIN`

`Join` je ena izmed najbolj uporabljanih ključnih besed v SQL poizvedbah. Uporablja se za združevanje podatkov med različnimi tabelami. Tu se je zopet za najhitrejšega izkazal *LINQ to Entities*. Izbrali smo vse lokacije, kjer so učne ustanove. Vrnjenih je bilo 4818 zadetkov. Rezultati so prikazani v tabeli 6.

SELECT in JOIN			
	Povezan	*Nepovezan*	*EF*
Meritev 1	0,059	0,142	0,042
Meritev 2	0,055	0,125	0,041
Meritev 3	0,060	0,129	0,037
Meritev 4	0,058	0,128	0,039
Meritev 5	0,059	0,128	0,041
Povprečje	**0,058**	**0,130**	**0,040**

Tabela 6: `Select` v kombinaciji s ključno besedo `join`.

Ocene:

- povezan način: **5 točk**,
- nepovezan način: **1 točka** in
- EF: **5 točk**.

- `SELECT` in `DISTINCT`

Pri pridobivanju vseh različnih krajev lokacij se je tehntnica prevesila na stran povezanega dostopa. Vrnjenih je bilo 9792 različnih imen krajev. Rezultati so prikazani v tabeli 7.

SELECT in DISTINCT			
	Povezan	*Nepovezan*	*EF*
Meritev 1	0,622	0,614	0,614
Meritev 2	0,599	0,604	0,617
Meritev 3	0,594	0,599	0,604
Meritev 4	0,596	0,615	0,596
Meritev 5	0,613	0,607	0,607
Povprečje	**0,605**	**0,608**	**0,607**

Tabela 7: `Select` v kombinaciji s ključno besedo `distinct`.

Ocene:

- povezan način: **5 točk**,
- nepovezan način: **5 točk** in
- EF: **5 točk**.

- `SELECT` s ključno besedo `LIKE` v `WHERE` pogoju

Ob uporabi ključne besede `like` se EF zopet izkaže za hitrejšega. Poizvedba je vračala vse lokacije, pri katerih ime kraja vsebuje niz "ing"; vrnila je 2294 zadetkov. Rezultati so prikazani v tabeli 8.

SELECT s ključno besedo LIKE v WHERE pogoju			
	Povezan	*Nepovezan*	*EF*
Meritev 1	0,358	0,383	0,354
Meritev 2	0,352	0,368	0,355
Meritev 3	0,355	0,367	0,357
Meritev 4	0,354	0,367	0,351
Meritev 5	0,362	0,369	0,355
Povprečje	**0,356**	**0,371**	**0,354**

Tabela 8: `Select` s ključno besedo `like` v `where` pogoju.

Ocene:

- povezan: **5 točk**,
- nepovezan: **1 točka** in
- EF: **5 točk**.

- `SELECT` in `ORDER BY`

Pri vračanju urejenih podatkov se je najslabše odrezal nepovezan dostop. Podatke smo razvrščali po nazivu naraščajoče. Vrnjeni so bili vsi zapisi iz tabele lokacij. Rezultati so prikazani v tabeli 9.

SELECT in ORDER BY			
	Povezan	*Nepovezan*	*EF*
Meritev 1	28,971	52,229	23,864
Meritev 2	27,437	55,629	23,882
Meritev 3	27,423	53,008	23,832
Meritev 4	27,717	54,028	23,749
Meritev 5	27,764	52,162	23,508
Povprečje	**27,862**	**53,411**	**23,767**

Tabela 9: `Select` v kombinaciji s ključno besedo `order by`.

Ocene:

- povezan: **3 točke**,
- nepovezan: **1 točka** in
- EF: **5 točk**.

- `INSERT`

Čas izvajanja *LINQ to Entities* poizvedbe je bil v primeru vstavljanja podatkov v tabelo zelo dolg (več kot 60 sekund). Nato smo na forumu Stack Overflow zasledili prispevek, ki je govoril ravno o tej težavi [44]. Število klicev metode `SaveChanges()` smo zmanjšali na en klic na 1000 vpisov. Ob posameznem klicu omenjene metode smo ponovno ustvarili tudi kontekst in s tem bistveno (za približno polovico) zmanjšali čas vnašanja. Nato smo zasledili še en članek [45] na to temo, kjer smo izvedeli, da lastnost `AutoDetectChangesEnabled` v primeru, da ima vrednost `false`, pomeni bistveno pohitritev. Ker v primeru vstavljanja podatkov avtomatskega zaznavanja sprememb ne potrebujemo, smo ga seveda nastavili po priporočilih in tako dosegli še dodatno pohitritev vnosov. Optimizaciji navkljub se je *LINQ to Entities* izkazal za najpočasnejšega. Vstavili smo 10000 zapisov, ki smo jih zgenerirali v obliki JSON zapisa, nato pa s Fiddlerjem z metodo `POST` v `REST` klicu poslali na servis. Rezultati so prikazani v tabeli 10.

INSERT			
	Povezan	*Nepovezan*	*EF*
Meritev 1	1,684	2,956	5,002
Meritev 2	1,685	2,910	4,994
Meritev 3	1,701	2,959	5,089
Meritev 4	1,699	2,968	5,321
Meritev 5	1,727	2,912	4,799
Povprečje	**1,699**	**2,941**	**5,041**

Tabela 10: Vstavljanje novih zapisov.

Ocene:

- povezan: **5 točk**,
- nepovezan: **3 točke** in
- EF: **1 točka**.

- `UPDATE`

Posodobili smo 10000 zapisov. Zapise smo podobno kot pri `insertu` zgenerirali v obliki JSON zapisa, ki smo ga s Fiddlerjem z metodo `PUT` v `REST` klicu poslali na servis. Prav tako kot pri vnosu novih zapisov v bazo se je tudi pri posodabljanju za najpočasnejšega izkazal EF. Rezultati so prikazani v tabeli 11.

UPDATE			
	Povezan	*Nepovezan*	*EF*
Meritev 1	1,054	2,329	5,746
Meritev 2	1,012	2,355	5,818
Meritev 3	1,019	2,287	5,632
Meritev 4	1,011	2,316	5,791
Meritev 5	1,086	2,329	5,319
Povprečje	**1,036**	**2,323**	**5,661**

Tabela 11: Posodabljanje zapisov.

Ocene:

- povezan: **5 točk**,
- nepovezan: **3 točke** in
- EF: **1 točka**.

Po ovrednotenju vseh izbranih kombinacij, smo najprej izračunali povprečno oceno poizvedb `select` za vsak način dostopa (tabela 12).

	Povezan	Nepovezan	EF
`Select`	3	1	5
`Select` in `join`	5	1	5
`Select` in `distinct`	5	5	5
`Select` in `like`	5	1	5
`Select` in `order by`	3	1	5
Povprečje	**4,2**	**1,8**	**5**

Tabela 12: Povprečna ocena poizvedb `select`.

Nato smo izračunali povprečje ocen vseh treh poizvedb (tabela 13).

	Povezan	Nepovezan	EF
`Select`	4,2	1,8	5
`Insert`	5	3	1
`Update`	5	3	1
Povprečje	**4,7**	**2,6**	**2,3**

Tabela 13: Povprečje ocen poizvedb `select`, `insert` in `update`.

Povezan način dostopa se je v povprečju izkazal za najbolj hitrega. Opazili pa smo, da ima slabo lastnost: za delovanje potrebuje največ vrstic tipkane kode. **Število točk: 5**.

Nepovezan dostop se je glede na hitrost izkazal za srednjo pot. Poleg tega smo prišli tudi do spoznanja, da je za uporabo v kombinaciji s tehnologijo WCF in zapisom JSON dokaj nepraktičen. S tem prisopom smo imeli namreč največ težav s serializacijo in deserializacijo podatkov. Deloma gre verjetno tudi temu zasluga za nižjo hitrost izvajanja in s tem tudi za slabšo oceno. **Število točk: 3**.

EF se je v okviru poizvedbe `select` izkazal za izredno hitrega, v okviru dodajanja in spreminjanja zapisov pa se je izkazal za najpočasnejšega. V primeru, da bi se odločili za utežitev glede na dejansko rabo določene poizvedbe v praksi, bi se zagotovo uvrstil na prvo mesto, saj je po naših izkušnjah poizvedba `select` daleč najbolj uporabljana. Ker pa smo se odločili za enakovredno primerjavo, sta mu ostali poizvedbi za vstavljanje/spreminjanje podatkov bistveno znižali oceno. **Število točk: 2**.

5.5. PRILAGODLJIVOST

Prilagodljivost je zmožnost sprememb sistema glede na morebitne potrebe. Za primerjavo načinov dostopa bomo privzeli sledeč scenarij: imamo WCF servis, ki pridobiva podatke v obliki JSON datoteke. Te podatke nato ustrezno deserializira in jih shrani v lokalno podatkovno bazo. Zanima nas, kako velik je vložek pri spremembi servisa ob uporabi posamezne tehnologije v

primeru, če JSON, ki ga pridobivamo, po novem nosi dva nova podatka, ki ju moramo tudi shraniti v bazo.

5.5.1. ADO.NET - POVEZANO

Potrebna je sprememba objekta, v katerega se deserializira prejeti JSON, dodati je potrebno tudi nekaj vrstic programske kode. Ustrezne spremembe je potrebno narediti tudi na podatkovni bazi. **Število točk: 4**.

5.5.2. ADO.NET - NEPOVEZANO

Serializacija in deserializacija objektov tipa `DataTable` brez uporabe zunanjih knjižnic ni enostavna. Če pa uporabimo neko zunanje orodje (kot je recimo Json.NET [46, 47]) pa serializacija in deserializacija postaneta precej enostavnejša. Slednje lahko vidimo na primeru izvorne kode 10.

```
string json = " [{\"clientID\":\"1788\",\"projectID\":\"19\"},
                {\"clientID\":\"1789\",\"projectID\":\"24\"},
                {\"clientID\":\"1790\",\"projectID\":\"24\"}]";

DSPrimer.dtPrimerDataTable primer = (DSPrimer.dtPrimerDataTable)
             JsonConvert.DeserializeObject(json, (typeof(DSPrimer.dtPrimerDataTable)));
```

Izvorna koda 10: Primer uporabe knjižnice Json.NET.

Seveda smo morali v grafičnem uporabniškem vmesniku orodja Visual Studio ustvariti `DataSet`, ki je vseboval strogo tipiziran objekt tipa `DataTable`. Ker sta brez uporabe zunanjih knjižnic serializacija in deserializacija objektov tipa `DataTable` precej zahtevna in ogrodje .NET nima vgrajenih knjižnic za enostavno prevajanje iz datoteke tipa JSON v objekt tipa `DataTable`, si kljub enostavnosti uporabe s knjižnico Json.NET zasluži nekoliko nižjo oceno. **Število točk: 2**.

5.5.3. *Entity Framework*

Tudi tu so potrebne zgolj manjše spremembe. Prav tako je potrebna sprememba objekta, v katerega se deserializira prejeti JSON, dodati je potrebno tudi samo nekaj vrstic programske kode. Kot pri ostalih dveh načinih je tudi tu potrebno urediti tudi spremembe na nivoju podatkovne baze. **Število točk: 4**.

5.6. Objektnost

Pod izrazom objektnost želimo popisati obliko oziroma tipiziranost vrnjenih podatkov. Zanima nas, ali so podatki strogo tipizirani ali ne. Za programerja je stroga tipizacija običajno boljša, saj mu omogoča enostavnejše delo z vrnjenimi podatki, zato bodo načini dostopa, ki vrnejo strogo tipizirane podatke, bolje ocenjeni.

5.6.1. *ADO.NET - povezano*

Podatki so v tem primeru vrnjeni v objektu tipa `SqlDataReader`, kar pomeni, da moramo vsak podatek pretvoriti v ustrezen tip. Proces kodiranja (izvorna koda 11) takega klica sicer ni miselno zahteven, je pa časovno potraten. **Število točk: 2**.

```
while (reader.Read())
{
       string LOK_ID = Convert.ToInt32(reader["LOK_ID"]);
       double LOK_ZemSirina = Convert.ToDouble(reader["LOK_ZemSirina"]);
       double LOK_ZemDolzina = Convert.ToDouble(reader["LOK_ZemDolzina"]),
       string LOK_Naziv = reader["LOK_Naziv"] != DBNull.Value ?
                          Convert.ToString(reader["LOK_Naziv"])
                          : null;
       break; //prekinemo zanko
}
```

Izvorna koda 11: Primer pretvarjanja vrnjenih podatkov v povezanem načinu.

Ob pretvarjanju moramo biti pozorni tudi na to, da ne poskušamo pretvarjati podatka, ki je brez vrednosti (`null`), saj pride do izjeme.

5.6.2. *ADO.NET - NEPOVEZANO*

Podatki se nahajajo v objektu tipa `DataTable`, ki je lahko strogo tipiziran ali pa tudi ne. V primeru, da se programer odloči za uporabo netipiziranega objekta tipa `DataTable`, lahko to stori na način, prikazan v izvorni kodi 12.

```
string sql = "SELECT TOP 10 * FROM Lokacija";
SqlConnection conn = new SqlConnection(@"Data Source=.\SQLEXPRESS;InitialCatalog=dbOSM;
Integrated Security=True");
SqlDataAdapter da = new SqlDataAdapter(sql, conn);

DataSet ds = new DataSet();

da.Fill(ds, "Lokacija");
```

Izvorna koda 12: Uporaba netipiziranega objekta `DataTable`.

Če želi na tak način dostopati do podatka o IDju lokacije prvega zapisa, to za razliko od stroge tipizacije, kjer se lahko neposredno sklicuje na podatek o IDju, stori tako, kot je prikazano v izvorni kodi 13. Ker smo mi uporabljali strogo tipizacijo, nam podatkov ni bilo potrebno pretvarjati. **Število točk: 3**.

```
long ID = ds.Tables["Lokacija"].Rows[0]["LOK_ID"];
```

Izvorna koda 13: Pridobitev podatka o IDju lokacije prvega zapisa.

5.6.3. *ENTITY FRAMEWORK*

EF že sam od sebe vrača podatke v obliki objektov. To so lahko entitetni tipi, lahko so sestavljeni tipi ali objekti, ki smo jih zasnovali sami.

Entitetni tipi so temeljni gradniki za opisovanje strukture z entitetnim podatkovnim modelom. V aplikaciji instanca entitetnega tipa predstavlja točno določen objekt, ki mora imeti unikaten ključ znotraj množice podatkov [48].

Sestavljeni tipi so lahko (podobno kot entitetni tipi) sestavljeni iz večih podatkov različnih primitivnih tipov ali drugih sestavljenih tipov. Nimajo pa unikatnih ključev in zato posledično ne morejo biti del povezav (ang. associations) [49].

Ker je Entity Framework kot ORM že sam po sebi naravnan k objektnosti, si seveda zasluži največje število točk. **Število točk: 5**.

6. ANALIZA REZULTATOV

KRITERIJ	ADO.NET - povezano	ADO.NET - nepovezano	Entity Framework
Krivulja učenja	4,5	3,5	2
Dokumentiranost	5	4,5	4
Čas realizacije	4	4,5	1
Hitrost	4,7	2,6	2,3
Prilagodljivost	4	2	4
Objektnost	2	3	5
Skupaj	**4,0**	**3,4**	**3,0**

Tabela 14: Ocene, izračunane na podlagi uteženega povprečja.

Glede na rezultate, predstavljene v tabeli 14, je zmagovalec povezani način. Sledi mu nepovezani način, na zadnjem mestu pa je pristal EF.

1. Krivulja učenja za vse tri tehnologije se je izkazala za točno tako, kot smo pričakovali. EF se je izkazal za tehnično najzahtevnejšo tehnologijo, najenostavnejši za uporabo pa je bil povezan način dostopa. Vsekakor se je potrebno zavedati, da je spoznavanje tehnologije enkraten vložek - ko se programer z neko tehnologijo dodobra spozna, nadgrajevanje že osvojenih znanj zanj pomeni bistveno manj vloženega časa kot pred tem. V skladu s tem smo ta kriterij utežili z manjšo utežjo kot ostale.

2. Razlika v količini dostopnih virov, povezanih z uporabo omenjenih tehnologij dostopa, je še vedno prisotna, vendar je bistveno manjša kot v preteklosti. Od objave prve verzije EF, ki je najmlajša izmed navednih tehnologij, se je na spletu pojavilo ogromno prispevkov na forumih in zapisov na blogih, ki so programerju v veliko pomoč. Glede na dosedanji trend pričakujemo, da se bo količina virov v prihodnosti še povečala.

3. Glede na krivuljo učenja in na nekoliko slabšo dokumentiranost smo predvidevali, da bo tu EF najslabši. Res pa je tudi, da smo si izbrali pristop k problemu, ki v EF zaenkrat še ni podprt, in ga s tem postavili v slabši položaj. Težavo z uvozom UDF v shemo bi lahko zaobšli z uporabo shranjene procedure, vendar bi s tem logiko prestavili na nivo podatkovne baze. S tem seveda naša meritev ne bi bila več verodostojna.

4. Najbolj od vseh kriterijev nas je presenetila hitrost delovanja. Predvidevali smo namreč, da se bo za najpočasnejšega v povprečju izkazal EF, pa se je zgodilo ravno nasprotno - v primeru poizvedb `select` se je izkazal kot najhitrejši. V primeru primerjave hitrosti bi lahko namesto enakovredne primerjave posamezen sklop poizvedb utežili, saj se glede na naše izkušnje ne uporabljajo enako pogosto. Največkrat se izvajajo stavki `select` (dodelili bi jim

utež 80), stavki `insert` (dodelili bi jim utež 15) in `update` (dodelili bi jim utež 5) so običajno v manjšini. Razmerje uporabe seveda lahko niha od rešitve do rešitve. V primeru drugačne utežitve bi bili rezultati bistveno drugačni.

5. Pri prilagodljivosti ni bilo presenečenj. Nepovezan način si je prislužil nekoliko slabšo oceno predvsem zaradi zahtevnejše serializacije in deserializacije.

6. Kriterij objektnosti v našem primeru ni bil neznan faktor, prej dejstvo. Glede na to, da je EF ORM smo pričakovali, da bo dobil najvišjo možno oceno. Podobno smo pričakovali tudi pri nepovezanem načinu, saj vrača podatke v strukturirani obliki, ki je lahko strogo tipizirana.

Programerju z izkušnjami bi slabši oceni navkljub priporočili rabo tehnologije *Entity Framework*. Prepričani smo namreč, da bodo v prihodnosti šibke točke EF odpravljene. Dokumentacija se bo dopolnjevala, z novimi verzijami platforme .NET pa se bo izboljševal tudi EF. V prid mu govori tudi dejstvo, da najhitreje obdela stavke `select`, ki so najpogostejši. Uporabe te tehnologije pa ne bi priporočili nekomu, ki je še bolj na začetku svoje programerske poti, saj bo za realizacijo zadanih ciljev zagotovo porabil bistveno več časa, kot bi ga porabil pri ostalih dveh tehnologijah.

7. ZAKLJUČEK

Cilj diplomskega dela sta bili primerjava in predstavitev treh različnih načinov dostopa do podatkovne baze za lažjo odločitev razvijalcu pri izbiranju zanj najbolj primerne tehnologije. Diplomsko delo je sestavljeno iz dveh logičnih sklopov. Prvi sklop zajema pridobivanje, popis razčlenitve XML datoteke, analizo pridobljenih podatkov ter načrtovanje in pripravo podatkovne baze. Drugi sklop se osredotoča na primerjavo načinov dostopa po kriterijih in analizo pridobljenih rezultatov.

Za pridobitev realnih podatkov smo se odločili, ker bi nad naključno zgenerirano množico poljubno velikega števila podatkov težko izvajali smiselne poizvedbe. Predstavili smo več zanimivih tehnologij za pridobivanje podatkov iz strukturirane datoteke. Kljub temu, da so se nekatere izmed njih za naše potrebe izkazale za neuporabne, smo podatke uspešno izluščili in jih uvozili v podatkovno bazo.

V drugem sklopu diplomskega dela smo ovrednotili posamezen način dostopa glede na izbrane kriterije. Ocenjevali smo na podlagi razširjene Likertove skale (ocene niso cela števila, ampak so števila natančna na eno decimalno mesto). Nabor ocen zaobsega zaprti interval od 1 do 5, kjer je 1 najslabša ocena in 5 najboljša. Končno oceno smo izračunali na podlagi uteženega povprečja. Omeniti velja, da smo kriterije poskušali smiselno utežiti s programerskega vidika.

Diplomsko delo bi se zagotovo dalo tudi izboljšati. Končna ocena bi se zagotovo spremenila, če bi razširili nabor kriterijev. Nekatere ocene so podane subjektivno, na podlagi naših izkušenj. Prav tako smo subjektivno utežili posamezne kriterije. Končna ocena posameznega načina dostopa v diplomskem delu je torej za razvijalca lahko referenca za lažjo odločitev, ne sme pa biti edini faktor pri odločanju.

IZVORNA KODA

SLIKE

TABELE

LITERATURA IN VIRI

[1] (2013) Uradna spletna stran okolja Visual Studio. Dostopno na: http://www.microsoft.com/visualstudio/eng/2013-preview

[2] Andrew Troelsen, *Pro C# 5.0 and the .NET 4.5 Framework, Sixth Edition*. New York: Apress Media LLC, 2012, str. 53.

[3] (2013) Orodje SQL Server Management Studio (spletna dokumentacija msdn). Dostopno na: http://msdn.microsoft.com/en-us/library/hh213248.aspx

[4] (2013) Orodje SQL Server Management Studio. Dostopno na: http://en.wikipedia.org/wiki/SQL_Server_Management_Studio

[5] (2013) Orodje SQL Server Management Studio (spletna dokumentacija msdn). Dostopno na: http://msdn.microsoft.com/en-us/library/ms174173(v=sql.90).aspx

[6] (2013) Uradna spletna stran razhroščevalnika Fiddler. Dostopno na: http://fiddler2.com/

[7] (2013) Glavne funkcije razhroščevalnika Fiddler. Dostopno na: http://fiddler2.com/Features/http-https-traffic-recording

[8] (2013) Uvod v jezik C# in .NET Framework (spletna dokumentacija msdn). Dostopno na: http://msdn.microsoft.com/en-us/library/z1zx9t92.aspx

[9] Andrew Troelsen, *Pro C# 5.0 and the .NET 4.5 Framework, Sixth Edition*. New York: Apress Media LLC, 2012, str. 4.

[10] Andrew Troelsen, *Pro C# 5.0 and the .NET 4.5 Framework, Sixth Edition*. New York: Apress Media LLC, 2012, str. 5.

[11] Andrew Troelsen, *Pro C# 5.0 and the .NET 4.5 Framework, Sixth Edition*. New York: Apress Media LLC, 2012, pogl. 21.

[12] (2013) Likertova skala. Dostopno na: https://en.wikipedia.org/wiki/Likert_scale

[13] (2013) Uteženo povprečje. Dostopno na: http://www.investopedia.com/terms/w/weightedaverage.asp

[14] (2013) Wiki stran OpenStreetMap. Dostopno na:
http://wiki.openstreetmap.org/wiki/Main_Page

[15] (2013) Avtorske pravice OpenStreetMap in licenca ODbL. Dostopno na:
http://www.openstreetmap.org/copyright

[16] (2013) Povezava do spletne strani, kjer je na voljo kopija OpenStreetMap baze v obliki arhivirane XML datoteke. Dostopno na:
http://planet.openstreetmap.org/

[17] (2013) bZip2 - zastonjski odprtokodni program za arhiviranje. Dostopno na:
http://en.wikipedia.org/wiki/Bzip2

[18] (2013) 7-Zip - program, ki smo ga uporabili za ekstrahiranje XML datoteke. Dostopno na:
http://www.7-zip.si/

[19] (2013) Razlaga ključa *amenity* in navedba vseh možnih vrednosti. Dostopno na:
http://wiki.openstreetmap.org/wiki/Key:amenity

[20] (2013) Razred *XmlDocument* (spletna dokumentacija msdn). Dostopno na:
http://msdn.microsoft.com/en-us/library/6kza7w4k.aspx

[21] (2013) Xml Document Object Model - DOM (spletna dokumentacija msdn). Dostopno na:
http://msdn.microsoft.com/en-us/library/hf9hbf87.aspx

[22] (2013) Branje XML datoteke po fragmentih z uporabo toka in razreda *XmlReader* (spletna dokumentacija msdn). Dostopno na:
http://msdn.microsoft.com/en-us/library/bb387035.aspx

[23] (2013) Anonimna metoda (spletna dokumentacija msdn). Dostopno na:
http://msdn.microsoft.com/en-us/library/vstudio/0yw3tz5k.aspx

[24] (2013) Pomen ključne besede *unsafe* (spletna dokumentacija msdn). Dostopno na:
http://msdn.microsoft.com/en-us/library/vstudio/chfa2zb8.aspx

[25] (2013) Wiki stran orodja Nominatim, ki omogoča iskanje po podatkih OSM in reverzno geokodiranje. Dostopno na:
http://wiki.openstreetmap.org/wiki/Nominatim

[26] (2013) Standard ISO 4217. Dostopno na:
http://en.wikipedia.org/wiki/ISO_4217

[27] (2013) Seznam valut v uporabi. Dostopno na: http://en.wikipedia.org/wiki/List_of_circulating_currencies_by_country

[28] (2013) Standard ISO 3166-1. Dostopno na: http://en.wikipedia.org/wiki/ISO_3166-1

[29] (2013) Uvoz velike količine podatkov (spletna dokumentacija msdn). Dostopno na: http://msdn.microsoft.com/en-us/library/ms175915.aspx

[30] (2013) Omejitve kapacitete tipov na MS SQL strežniku (spletna dokumentacija msdn). Dostopno na: http://msdn.microsoft.com/en-us/library/ms143432.aspx

[31] (2013) Krivulja učenja. Dostopno na: http://en.wikipedia.org/wiki/Learning_curve

[32] (2013) Primer uporabe razreda *SqlClient* (spletna dokumentacija msdn). Dostopno na: http://msdn.microsoft.com/en-us/library/dw70f090.aspx

[33] (2013) Primer enostavne Windows Forms aplikacije, ki se na bazo povezuje s povezanim načinom. Dostopno na: http://visualcsharptutorials.com/ado-net/querying-database-connected-approach

[34] (2013) Uporaba *DataSetov* (spletna dokumentacija msdn). Dostopno na: http://msdn.microsoft.com/en-us/library/ss7fbaez(v=vs.80).aspx

[35] (2013) LINQ to Entities. Dostopno na: http://www.codeproject.com/Articles/246861/LINQ-to-Entities-Basic-Concepts-and-Features

[36] (2013) LINQ to Entities (spletna dokumentacija msdn). Dostopno na: http://msdn.microsoft.com/en-us/library/bb386964.aspx

[37] (2013) Stack Overflow. Dostopno na: http://stackoverflow.com/

[38] (2013) Code Project. Dostopno na: http://www.codeproject.com/

[39] (2013) EF. Dostopno na: https://en.wikipedia.org/wiki/Entity_Framework

[40] (2013) Havershinova formula (formula številka 4). Dostopno na: http://ragrawal.wordpress.com/2009/07/11/calculating-geographic-distance/

[41] (2013) Havershinova formula. Dostopno na: http://en.wikipedia.org/wiki/Haversine_formula

[42] (2013) Težave pri uvozu skalarne funkcije (forum msdn). Dostopno na: http://social.msdn.microsoft.com/Forums/en-US/adodotnetentityframework/thread/756865e5-ff25-4f5f-aad8-fed9d741c05d/

[43] (2013) Ročna sprememba EDMX sheme. Dostopno na: http://stackoverflow.com/questions/12481868/how-to-use-scalar-valued-function-with-linq-to-entity

[44] (2013) Prispevek iz foruma Stack Overflow, ki govori kako zoptimizirati vstavljanje podatkov v bazo. Dostopno na: http://stackoverflow.com/questions/5940225/fastest-way-of-inserting-in-entity-framework

[45] (2013) Pohitritev vstavljanja v bazo. Dostopno na: http://stackoverflow.com/questions/5943394/why-is-inserting-entities-in-ef-4-1-so-slow-compared-to-objectcontext

[46] (2013) Stran knjižnice Json.NET na portalu CodePlex. Dostopno na: http://json.codeplex.com/

[47] (2013) Dokumentacija knjižnice Json.NET. Dostopno na: http://james.newtonking.com/projects/json/help/

[48] (2013) Entitetni tipi (spletna dokumentacija msdn). Dostopno na: http://msdn.microsoft.com/en-us/library/ee382837.aspx

[49] (2013) Kompleksni tipi (spletna dokumentacija msdn). Dostopno na: http://msdn.microsoft.com/en-us/library/ee382831.aspx