Šolski center Celje

Srednja šola za kemijo, elektrotehniko in računalništvo

OPOMNIK

Raziskovalna naloga

Avtorja: David Zupan Fenko, 4. letnik

Jure Tot, 4. letnik

Mentor: prof. mag. Boštjan Resinovič

Pot na Lavo 22 , 3000 Celje

Celje, 2013

Kazalo

1. POVZETEK 4
   1.1. Ključne besede 4
2. SUMMARY 5
   2.1. Key words 5
3. UVOD 6
   3.1. Opis raziskovalnega problema 6
   3.2. Opis raziskovalnih metod 6
   3.3. Hipoteza in cilji 6
4. Rezultati ankete 7
5. OPOMNIK 12
   5.1. Visual Studio 12
   5.2. Izdelava Windows Phone aplikacije 12
      5.2.1. Namen aplikacije 12
      5.2.2. Zgradba uporabniškega vmesnika po straneh 12
      5.2.3. Uporaba aplikacije 13
      5.2.4. Življenski cikel aplikacije 14
      5.2.5. Navigacija v aplikaciji 15
      5.2.6. Sestava uporabniškega vmesnika 15
      5.2.7. Uporaba komponent za navigacijo 16
      5.2.8. Hranjenje podatkov v aplikaciji 17
      5.2.9. Opis strani v aplikaciji in njihova uporaba 23
   5.3. Izdelava Windows form aplikacije 25
      5.3.1. C# 25
      5.3.2. .NET Framework 25
      5.3.3. Web service 25
      5.3.4. Windows Form 26
      5.3.5. Spletna storitev 28
6. ZAKLJUČEK 31
7. VIRI 32
8. Zahvala 33

KAZALO SLIK

Slika 1: Prva stran v aplikaciji Windows Phone 13
Slika 2: Stran Settings 13
Slika 3: Application bar ter gumbi na njej 16
Slika 4: Stran categories 24
Slika 5: Glavno okno Windows form aplikacije 26
Slika 6: Okno za dodajanje opomnika 27
Slika 7: Podatki, ki jih vrne metoda spletne storitve 29

1. POVZETEK

V raziskovalni nalogi vam bova predstavila opomnik. To je orodje, s pomočjo katerega si lahko nastavimo opozorilo, da nas opomni na prihajajoči sestanek, rojstni dan, rok za oddajo projekta, itd. Ker živimo v času, ko imamo doma že skoraj vsi računalnik in pametni telefon, sva se odločila, da bova opomnik naredila na dveh različnih platformah. Windows phone in Windows.Obe platformi se lahko povežeta na spletno storitev, na kateri se nahaja podatkovna baza in v njo prenašata svoje opomnike.

Na obeh platformah je prav tako omogočeno lokalno hranjenje podatkov, v primeru, ko uporabnik ne potrebuje funkcije po sinhronizaciji čez več platform. Uporaba spletne storitve pa omogoča razširitev na druge platforme kot so Android in iOS.

1.1. Ključne besede

Razvoj večplatformskih aplikacij
Windows Phone
Windows Form
spletna storitev
podatkovna baza

2. SUMMARY

In this research project, we will demonstrate a reminder. This is a tool, with which we can set an alarm, to remind us about an upcoming meeting, birthday, project due date etc. Because we live in a time, when almost everyone owns a computer and a smartphone, we decided to create a reminder application on two different platforms. Windows Phone and Windows. Both can connect to a web service, where a database is located, and transfer their data to it.

Also, both platforms enable local data storage, just in case the user does not require the synchronization function over multiple platforms. The use of the web service enables expansion onto other platforms such as Android and iOS.

2.1. Key words

Multi-platform development
Windows Phone
Windows Form
Web Service
Database

3. UVOD

Za najino raziskovalno nalogo, sva se odločila, da bova naredila opomnik oz. aplikacijo za pomoč pri organizaciji. Na voljo je veliko takšnih aplikacij, vendar so prezahtevne za uporabo in slabo zasnovane, kar pomeni, da uporabniku tratijo čas. Najin cilj je izdelati opomnik, ki bo prijaznejši uporabniku. Naredila bova opomnik, ki bo enostaven za uporabo, pregleden, urejanje v njem pa enostavno.

Deloval bo na mobilnem operacijskem sistemu Windows Phone in operacijskem sistemu Windows kot okenska aplikacija (*Windows Form*). Obe aplikaciji bosta omogočali lokalno hranjenje vnešenih nalog oz. opomnikov. Podatki se bodo hranili v podatkovno bazo na trdem disku mobilnika in osebnega računalnika.

Vendar, ker ima dandanes veliko ljudi pametne mobilnike in vedno več uporabljajo osebne računalnike, bosta aplikaciji omogočali posodabljanje med sabo. To bova dosegla z uporabo spletne storitve oz. *WebService.*

3.1. Opis raziskovalnega problema

Težava pri razvoju aplikacij, za katere želimo, da delujejo na več platformah, nastane pri kompatibilnosti tako med programskimi jeziki kot med komponentami, ki so na voljo na eni platformi, na drugi pa ne. Primer tega je npr. uporaba Microsoft SQL Serverja in lokalnih SQL podatkovnih baz na operacijskem sistemu Android. Prav tako se lahko pojavi težava pri povezavi iz op. sistema Windows Phone na op. sistem Windows, čeprav je osnova .NET Framework.

Na tržišču je na voljo veliko aplikacij za podporo organizaciji. Nekatere so preproste, druge pa zapletene za uporabo. Nekatere omogočajo funkcije, ki jih druge ne. Če bi želeli imeti »popolno« verzijo aplikacije, bi jih morali imeti nameščenih več ter v vsako vpisati enake opomnike, kar pa preprosto ni praktično.

3.2. Opis raziskovalnih metod

Najprej sva preizkusila nekaj že obstoječih opomnikov, ki sva jih našla na spletu, Google Play trgovini in na Windows Phone Marketplace. Nato sva svojim sošolcem posredovala anketo v kateri sva jih povprašala o uporabi opomnikov. Tako sva ugotovila katere funkcije bova vključila in kakšen izgled bo imel najin opomnik.

3.3. Hipoteza in cilji

Hipoteza raziskovalne naloge:

Tehnologija .NET omogoča razvoj strežniške aplikacije za opomnik na tak način, da se bo z njo mogoče povezati z odjemalci, napisanimi za različne platforme, npr. Windows Forms, ASP.NET, Windows Phone, Android in iOS.

Cilji raziskovalne naloge:

- ustvariti mobilni opomnik za Windows Phone
- ustvariti opomnik kot okensko aplikacijo za op. sistem Windows
- omogočiti povezovanje mobilne aplikacije z okensko aplikacijo
- preučiti možnosti za ustvarjanje opomnika na platgormah ASP.NET, Android in iOS

4. Rezultati ankete

Vprašanje 1

Q1	Ali ste že uporabljali aplikacijo za pomoč pri organizaciji?				
	Odgovori	Frekvenca	Odstotek	Veljavni	Kumulativa
	1 (Da)	7	41%	44%	44%
	2 (Ne)	9	53%	56%	100%
Veljavni	Skupaj	16	94%	100%	

Iz prvega vprašanja lahko vidimo, da je razmerje med številom oseb, ki so že uporabljali aplikacijo za pomoč pri organizaciji in tistimi, ki je še niso skoraj enako (41% - 53%).

Vprašanje 2

Q2	Kako pogosto ste jo / jo še uporabljate?				
	Odgovori	Frekvenca	Odstotek	Veljavni	Kumulativa
	1 (Redko)	4	24%	57%	57%
	2 (Občasno)	3	18%	43%	100%
	3 (Redno)	0	0%	0%	100%
Veljavni	Skupaj	7	41%	100%	

Vidimo lahko, da tisti, ki takšne aplikacije uporabljajo, tega ne počnejo redno, ampak zgolj občasno in po potrebi. Razlog za to, je morda lahko slaba zasnova aplikacije.

Vprašanje 3

Q3	Če bi bila na voljo preprosta, hitra in uporabniku prijazna aplikacija za pomoč pri organizaciji, bi jo uporabljali?				
	Odgovori	Frekvenca	Odstotek	Veljavni	Kumulativa
	1 (Da)	10	59%	59%	59%
	2 (Ne)	7	41%	41%	100%
Veljavni	Skupaj	17	100%	100%	

Večina anketirancev pa bi uporabljala dobro zasnovano aplikacijo za pomoč pri organizaciji, če bi ta bila dobro zasnovana in preprosta za uporabo.

Vprašanje 4

Q4	Imate na osebnem računalniku nameščeno aplikacijo za pomoč pri organizaciji (planer, opomnik...)?				
	Odgovori	Frekvenca	Odstotek	Veljavni	Kumulativa
	1 (Da)	3	18%	19%	19%
	2 (Ne)	13	76%	81%	100%
Veljavni	Skupaj	16	94%	100%	

Reultat tega vprašanja je bil pričakovan – večina anketirancev na osebnem računalniku nima nameščene aplikacije za pomoč pri organizaciji (76%).

Vprašanje 5

Q5	Ste zadovoljni z njo (preglednost, hitrost, zahtevnost za uporabo...)?				
	Odgovori	Frekvenca	Odstotek	Veljavni	Kumulativa
	1 (Da)	3	18%	100%	100%
	2 (Ne)	0	0%	0%	100%
Veljavni	Skupaj	3	18%	100%	

Tisti, ki pa imajo na osebnem računalniku nameščeno takšno aplikacijo, so zadovoljni z njo.

Vprašanje 6

Q7	Ali imate pametni mobilnik?				
	Odgovori	Frekvenca	Odstotek	Veljavni	Kumulativa
	1 (Da)	10	59%	67%	67%
	2 (Ne)	5	29%	33%	100%
Veljavni	Skupaj	15	88%	100%	

Večina anketirancev ima pametni mobilnik, kar je pričakovan rezultat.

Vprašanje 7

Q8	Imate morda več pametnih mobilnikov?				
	Odgovori	Frekvenca	Odstotek	Veljavni	Kumulativa
	1 (Enega)	9	53%	90%	90%
	2 (Dva)	1	6%	10%	100%
	3 (Tri ali več)	0	0%	0%	100%
Veljavni	Skupaj	10	59%	100%	

Prav tako sva pričakovala, da ima večina anketirancev le en pametni mobilnik.

Vprašanje 8

Q9	Kateri operacijski/e sistem/e imate nameščen/e na mobilniku/ih?						
	Podvprašanja	Odgovori		Veljavni	Št. enot	Navedbe	
		Frekvence	%			Frekvence	%
Q9a	Android	8	80%	10	17	8	73%
Q9b	iOS	2	20%	10	17	2	18%
Q9c	Windows Phone	0	0%	10	17	0	0%
Q9d	Drugo:	1	10%	10	17	1	9%
	SKUPAJ			10	17	11	100%

Rezultat vprašanja se ujema s trenutnim stanjem na trgu operacijskih sistemov za mobilne naprave – tj. večina pripada operacijskemu sistemu Android.

Vprašanje 9

Q10	Ali imate na mobilniku nameščeno aplikacijo za pomoč pri organizaciji (planer, opomnik...)?				
	Odgovori	Frekvenca	Odstotek	Veljavni	Kumulativa
	1 (Da)	7	41%	70%	70%
	2 (Ne)	3	18%	30%	100%
Veljavni	Skupaj	10	59%	100%	

Rezultat vprašanja glede razširjenosti aplikacij za pomoč pri organizaciji na mobilnih napravah je pričakovan, saj ima dandanes vsak mobilnik nameščen vsaj koledar, ki omogoča vnos opravil.

Vprašanje 10

Q11	Ste zadovoljni z njo (preglednost, hitrost, zahtevnost za uporabo...)?				
	Odgovori	Frekvenca	Odstotek	Veljavni	Kumulativa
	1 (Da)	5	29%	71%	71%
	2 (Ne)	2	12%	29%	100%
Veljavni	Skupaj	7	41%	100%	

Rezultat tega vprašanja (večina anketirancev je zadovoljna z aplikacijami na mobilniku) je verjetno posledica tega, da so uporabniki navajeni na uporabo privzetih aplikacij, ter nezavedanja možnosti, ki bi jih lahko ponujale.

Vprašanje 11

Q12	Kaj vas moti?						
	Podvprašanja	Odgovori		Veljavni	Št. enot	Navedbe	
		Frekvence	%			Frekvence	%
Q12a	Prepočasna	0	0%	2	17	0	0%
Q12b	Prezapletena za uporabo	1	50%	2	17	1	20%
Q12c	Nepregledna	2	100%	2	17	2	40%
Q12d	Prevelik obseg funkcij	1	50%	2	17	1	20%
Q12e	Drugo:	1	50%	2	17	1	20%
	SKUPAJ			2	17	5	100%

Tisti anketiranci, ki z aplikacijo na mobilniku niso zadovoljni, pa so izrazili mnenje, da je le-ta prezapletena za uporabo, nepregledna in omogoča preveč funkcij, ter ne deluje po pričakovanjih.

Vprašanje 12

Q12e_text	Q12 (Drugo:)				
	Odgovori	Frekvenca	Odstotek	Veljavni	Kumulativa
	nedela tak ko bi hotu pa da ga j****	1	6%	100%	100%
Veljavni	Skupaj	1	6%	100%	

Vprašanje 13

XSPOL	Spol:				
	Odgovori	Frekvenca	Odstotek	Veljavni	Kumulativa
	1 (Moški)	15	88%	100%	100%
	2 (Ženski)	0	0%	0%	100%
Veljavni	Skupaj	15	88%	100%	

Ker je bila anketa izvedena v okviru našega razreda, so vsi anketiranci moškega spola.

Vprašanje 14

XSTAR2a4	V katero starostno skupino spadate?				
	Odgovori	**Frekvenca**	**Odstotek**	**Veljavni**	**Kumulativa**
	1 (do 20 let)	15	88%	100%	100%
	2 (21 - 40 let)	0	0%	0%	100%
	3 (41 - 60 let)	0	0%	0%	100%
	4 (61 let ali več)	0	0%	0%	100%
Veljavni	Skupaj	15	88%	100%	

Vsi anketiranci prav tako spadajo v starostno skupino do 20 let.

Vprašanje 15

XDS2a4	Kakšen je vaš trenutni status?				
	Odgovori	**Frekvenca**	**Odstotek**	**Veljavni**	**Kumulativa**
	1 (Šolajoči)	15	88%	100%	100%
	2 (Aktivni)	0	0%	0%	100%
	3 (Neaktivni)	0	0%	0%	100%
	4 (Brezposelni)	0	0%	0%	100%
Veljavni	Skupaj	15	88%	100%	

Prav tako so se v času izvedbe ankete vsi anketiranci šolali.

5. OPOMNIK

5.1. Visual Studio

Za izdelavo izdelka sva uporabila programsko orodje Visual Studio, ki se uporablja za razvoj širokega spektra programskih rešitev, kot so konzolske aplikacije, Windows Forms aplikacije, spletne strani in storitve ipd.

Visual Studio prav tako omogoča razvoj mobilnih aplikacij za operacijski sistem Windows Phone. Vendar ker Visual Studio v osnovi tega ne omogoča, sva morala s spleta prenesti programski paket Windows Phone SDK, ki je v bistvu dodatek k Visual Studiu in vsebuje orodja potrebna za razvoj Windows Phone aplikacij.

5.2. Izdelava Windows Phone aplikacije

5.2.1. Namen aplikacije

Najina želja je uporabniku olajšati organiziranje vsakodnevnih opravkov. Glavni namen aplikacije je, da si uporabnik te opravke zapiše in po želji nastavi opomnik oz. alarm z namenom, da ga aplikacija pravočasno opomni na vpisan opravek.

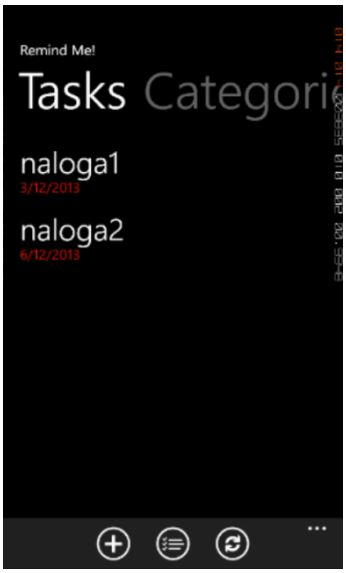
Ker ima dandanes veliko ljudi več kot le en pametni mobilnik in je računalnik že pravzaprav obvezno orodje v vsakem gospodinjstvu, najina aplikacija prav tako omogoča sinhronizacijo vpisanih nalog, opravkov... med več mobilniki ter sinhronizacijo med mobilnim telefonom ter osebnim računalnikom.

Ta funkcija aplikacije lahko predstavlja veliko korist tako »navadnim uporabnikom« kot tudi podjetjem. Če neko podjetje izvaja določen projekt, lahko delavci v svojo aplikacijo pišejo, kaj želijo opraviti, možnost sinhronizacije pa omogoča, da vsi udeleženci v projektu vedo, kaj počne nekdo drug in s tem pospeši delo in prepreči napake pri izvedbi.

5.2.2. Zgradba uporabniškega vmesnika po straneh

Če upoštevamo dejstvo, da je glavna funkcija aplikacije zapis opomnika oz. naloge, mora biti uporabniški vmesnik aplikacije preprost, uporabniku prijazen ter mora omogočati hitro delo pri dodajanju, brisanju in urejanju opomnikov.

S tem namenom je aplikacija sestavljena iz treh osnovnih strani: *Tasks, Categories* in *Settings.* Nadaljne strani, ki so izpeljane iz osnovnih so: *AddTask, AddCategory, CategorySelector, CategoryColorSelector, TaskOverview, Language, About, ConfigureBackground, ConfigureLiveTiles* ter *SyncSettings.*



Slika 1: Prva stran v aplikaciji Windows Phone

5.2.3. Uporaba aplikacije

Ko uporabnik zažene aplikacijo, se odpre stran *Tasks.* Na tej strani se izpišejo vse naloge, ki jih je uporabnik dodal. Na spisku lahko izberemo določeno nalogo, da se odpre stran s podrobnostmi. Na tej strani lahko prav tako dodajamo, brišemo ter urejamo naloge.

Iz strani *Tasks* lahko uporabnik preide na stran *Categories* na kateri se izpišejo imena vseh kategorij, ki jih je dodal. Kategorije so namenjene razvrščanju in organizaciji dodanih nalog. Da si uporabnik kategorije lažje zapomni, mu omogočimo, da vsaki kategoriji določi barvo. To pomeni, da se bodo na strani izpisala imena kategorij, vendar bo vsaka imela barvo besedila takšno, kot jo je nastavil uporabnik (če uporabnik ne izbere barve je privzeta barva bela).

Zadnja stran se imenuje *Settings,* na kateri lahko uporabnik prilagaja nastavitve aplikacije. Spreminja lahko jezik uporabniškega vmesnika, izve več o avtorju, spreminja ozadje in več.

Slika 2: Stran Settings

Podrobnejši opis vseh strani sledi v nadaljevanju, saj se moramo pred tem seznaniti še z nekaj postopki, ki so nujno potrebni za delovanje vseh navedenih strani.

5.2.4. Življenski cikel aplikacije

Življenjski cikel aplikacije (ang. application lifecycle) je izraz, ki predstavlja ves čas od zagona aplikacije do trenutka, ko se aplikacija popolnoma zapre. Okenske aplikacije na osebnih računalnikih imajo v glavnem časovno dobo od zagona do zaprtja. Okenska aplikacija se lahko zapre predčasno (npr. izguba napajanja) ali se zapre zaradi napake. Enako je pri mobilnih aplikacijah, vendar je tu veliko več možnosti, da se prekine življenjski cikel aplikacije. Nekaj dogodkov, ki lahko prekinejo normalno delovanje aplikacije: uporabnik prejme klic, dobi sporočilo, odpre drugo aplikacijo, mobilniku se izprazni baterija ipd.

Če pojasnim na primeru: uporabnik v aplikaciji vpisuje dolgo besedilo. Med tem prejme klic. Ko opravi s pogovorom, želi nadaljevati s pisanjem, vendar ko se vrne nazaj v aplikacijo, je vse, kar je do trenutka, ko je dobil klic, napisal, izginilo. Razlog za to je preprost ter če poskrbimo zanj, lahko občutno izboljšamo uporabnikovo izkušnjo.

5.2.4.1. Stanja aplikacije

Mobilna aplikacija lahko vstopi v različna stanja. V operacijskem sistemu Windows Phone obstajajo štiri različna stanja. Aktivno, mirujoče, *tombstoned*, prekinjeno. Ta stanja so omogočena s pomočjo *Fast Application Switching (FAS)* – hitro preklapljanje med aplikacijami.

V vsa stanja aplikacija prehaja po sistemu »zadnji noter, prvi ven«. Npr. če je na telefonu odprta samo naša aplikacija, je na prvem mestu. Uporabnik lahko nato odpre še eno aplikacijo, to pa pošlje našo aplikacijo na drugo mesto. Vsa nadaljna odprtja katerihkoli drugih aplikacij premaknejo našo po spisku navzdol.

Po uspešnem zagonu vstopi v aktivno stanje. V tem stanju aplikacija deluje normalno tj. vse spremembe, ki jih želimo opraviti (npr. če želimo shraniti podatke ali opraviti kakšno matematično operacijo) bodo potekle od začetka do konca. Vendar, če je npr. med pisanjem besedila ali izvedbo matematične operacije aplikacija prekinjena, lahko izgubimo vse, kar se je do takrat izvajalo.

Ko je aplikacija iz kakršnegakoli razloga prekinjena, vstopi v mirujoče stanje. V tem stanju je aplikacija še vedno naložena v pomnilniku z vsemi podatki in je pripravljena za takojšnje nadaljevanje.

Tombstoned je stanje, v katerega aplikacija preide iz mirujočega. Npr.: če je aplikacija v mirujočem stanju, uporabnik pa se nato odloči, da bo zagnal kakšno igro, ki zahteva več od komponent mobilnika, bo naša aplikacija prešla v stanje tombstoned. Kakršnekoli spremembe, ki jih je naredil uporabnik, se bodo zbrisale, razen če eksplicitno ne poskrbimo za to, da si spremembe zapomnimo.

Zadnje stanje je prekinjeno. V to stanje aplikacija preide iz stanja tombstoned. To stanje je v bistvu popolno zaprtje aplikacije, ko v pomnilniku ne ostane noben podatek več.

5.2.5. Navigacija v aplikaciji

Za navigacijo v aplikaciji uporabljamo storitev imenovano *Navigation service.* Ta storitev v svojem prostoru v pomnilniku hrani zapisnik vseh navigacij, ki so se zgodile od zagona aplikacije. Pri uporabi storitve moramo izvesti metodo *Navigate()*, pri kateri navedemo pot do strani, ki je lahko relativna ali absolutna, metoda pa nas potem pošlje na navedeno stran.

Pot do strani je navedena v obliki povezave imenovane URI (Uniform Resource Identifier). URI je niz znakov, v katerem je navedeno ime strani, dodamo pa lahko tudi poizvedbe, ki se prenesejo na stran do katere želimo priti. Primer poizvedbe je npr. ime naloge. Ob prihodu na novo stran lahko s pomočjo storitve *NavigationContext* iz URI naslova razberemo prenesene poizvedbe.

Če želimo priti nazaj na stran iz katere smo prišli, lahko prav tako uporabimo storitev *Navigation service* in uporabimo metodo *GoBack()* ali pa ponovno uporabimo metodo *Navigate()* ter vpišemo relativni / absolutni naslov prejšnje strani.

Ob navigaciji nazaj moramo biti pozorni, da uporaba metode *Navigate()* ustvari nov primerek strani, metoda *GoBack()* pa v bistvu povrne stran iz katere smo odšli v aktivno stanje. Ko se odločamo, katero metodo bomo uporabili za navigacijo nazaj, moramo premisliti, ali želimo nazaj prenesti kakšne poizvedbe oz. podatke, v obliki URI naslova, saj metoda *GoBack()* ne omogoča navedbe poti ampak iz zapisnika o navigaciji le prebere ime zadnje strani.

Zadnji zapis o navigaciji v zapisniku o navigaciji lahko pobrišemo z uporabo metode *RemoveBackEntry()*. Če želimo izbrisati več zapisov iz zapisnika, moramo to metodo poklicati večkrat.

5.2.6. Sestava uporabniškega vmesnika

Uporabniški vmesnik je v osnovi sestavljen iz treh komponent: *Pivot, Portrait page* in *Application bar.*

5.2.6.1. Pivot

Pivot je komponenta sestavljena iz več povezanih strani združenih v eno samo, ki sega »izven okvira mobilnika«. Navigacija po njej je zelo preprosta, uporabnik mora s prstom le podrsati levo oz. desno da se pomakne na naslednjo stran. Vendar pri tej komponenti ni pametno pretiravati z veliko elementi.

5.2.6.2. Portrait page

Portrait page je komponenta zelo podobna Pivotu, vendar ne omogoča »seganja čez okvir mobilnika«. Je ena sama stran, ustrezna pa je v primerih, ko nimamo potrebe po navigaciji, ki jo omogoča Pivot in ko želimo, da je uporabniku jasno, da je vse kar potrebuje le na tisti strani.

5.2.6.3. Application bar

Application bar je komponenta, ki se nahaja na dnu strani in ne zavzame veliko prostora (10% celotne strani). Primerna je za hranjenje že tako omejenega prostora na zaslonu mobilnika, kjer lahko namesto navigacijskih elementov prikazujemo pomembne podatke. Z uporabo komponente tudi dosežemo preglednost, jasnost ter uporabniku omogočimo hitrejše delo. Nahaja se lahko na vseh straneh ne glede na tip strani (pivot, portrait page...).

Slika 3: Application bar ter gumbi na njej

5.2.6.4. *Ozadje aplikacije*

Pri mobilnih aplikacijah je pomembna pravilna uporaba barv npr. barva črk in barva ozadja. Ko določamo barve črkam in ozadju, je potrebno oceniti, katere barve bi bile primerne glede na vsebino aplikacije ali samo trenutne strani. Prav tako ni primerno vnaprej točno določiti barve, ampak je primerneje nastaviti barvo na trenutno barvno shemo mobilnika. Tako zagotovimo, da bo vsebina berljiva (izognemo se npr. belim črkam na belem ozadju).

5.2.6.4.1. Dinamično ozadje

Če pa kljub temu želimo popestriti izgled aplikacije, lahko damo uporabniku možnost dinamičnega ozadja. S tem je mišljena spreminjajoča se slika na ozadju. To lahko dosežemo s pomočjo hranjenja slik v aplikaciji sami ter nastavimo, da se na določeno obdobje spremeni, lahko pa s spletnega vira prenesemo sliko ter jo nastavimo za ozadje.
Spletni brskalnik Bing vsak dan posodobi svojo domačo stran s sliko z odlično kvaliteto. Zaradi velikega povpraševanja po teh slikah, je Bing omogočil prenos le-teh (storitev Bing Homepage Image of the day).
Ta možnost ima dve slabosti: odvisna je od storitve Bing Homepage Image of the day, ter zahteva povezavo s spletom (če uporabnik nima na voljo brezžične povezave s spletom, mora uporabiti mobilni internet).

5.2.7. Uporaba komponent za navigacijo

Navigacija po aplikaciji je sestavljena iz več vidikov. Z uporabo komponent za navigacijo je mišljena komponenta *Application bar.* Po aplikaciji se lahko premikamo samo s podrsavanjem po *Pivot* komponenti, vendar naša aplikacija potrebuje še dodatne strani, do katerih lahko pridemo s pomočjo A*pplication bar* komponente. Na *Application bar* dodamo gumbe, ki služijo navigaciji do drugih strani v aplikaciji, ki se NE nahajajo na trenutni *Pivot* komponenti.

Na komponenti *Application bar* se nahajajo trije gumbi: *dodaj*, *izberi* in *sinhroniziraj*.

Naloga gumba *dodaj* je navigacija na novo stran. Odvisno od tega na katerem delu komponente *Pivot* smo, se bo gumb *dodaj* odzval temu primerno. Če smo na strani *Tasks,* bo *Navigation service* ob kliku na gumb obprl stran *AddTask,* če smo na strani *Categories* se bo odprla stran *AddCategory ,* na strani S*ettings* pa so vsi gumbi onemogočeni.

Razlog, da so vsi gumbi na *Application bar* onemogočeni na strani *Settings,* je dejstvo, da je vsebina te strani sestavljena iz druge vrste gumbov, katerih namen je prav tako navigacija na druge strani, na katerih lahko prilagajamo nastavitve aplikacije. Stran *Settings* si lahko predstavljamo kot odskočno desko na druge strani.

5.2.7.1. Gumb Nazaj

Gumb Nazaj oz. *Back key* se nahaja na ohišju mobilnika. Pritisk nanj izvede metodo *GoBack().* Presoditi moramo kdaj in če bomo gumb onemogočili, saj lahko z njegovo uporabo uporabnik nehote pobriše spremembe, ki jih je opravil. Gumb Nazaj lahko onemogočimo z prekritjem metode *OnBackKeyPress().*

5.2.7.2. Dogajanje v ozadju

Vsaka sprememba strani v aplikaciji sproži (ob mnogih drugih) dva dogodka: *OnNavigatedFrom()* ter *OnNavigatedTo()*.

5.2.7.2.1. OnNavigatedFrom()

Dobesedni prevod pomeni »ob navigaciji od«. Zgodi se, ko uporabnik npr. pritisne na gumb *dodaj*. Tik preden se skrije stran, iz katere uporabnik odhaja, se sproži dogodek, programer pa lahko določi, kaj se takrat zgodi. V tem času je primerno (po potrebi) shraniti morebitne spremembe, ki jih je uporabnik na strani opravil (npr. vpisal besedilo).

5.2.7.2.2. OnNavigatedTo()

Dobesedni prevod je obraten od prejšnjega in pomeni »ob navigaciji na«. Zgodi se, ko uporabnik na kakršenkoli način pride na katerokoli stran v aplikaciji (ali po izvedbi metode *GoBack()* ali *Navigate()*). Ta čas pa je primeren za npr. nalaganje podatkov iz podatkovne baze, ali za povezovanje na spletni vir.

5.2.8. Hranjenje podatkov v aplikaciji

Aplikacije za namizne računalnike lahko uporabljajo različne načine hranjenja podatkov. Najpogosteje hranimo podatke v podatkovno bazo. Ta se lahko nahaja na računalniku samem, ali pa na oddaljenem strežniku. Hranjenje podatkov v mobilni aplikaciji je lahko prav tako lokalno ali pa oddaljeno na strežniku.

Podatki v naši aplikaciji se hranijo v podatkovno bazo, ta pa se nahaja v prostoru imenovanem *Isolated storage,* ki se nahaja na trdem disku mobilnika.

5.2.8.1. Ustvarjanje podatkovne baze

Če želimo ustvariti podatkovno bazo na mobilniku, moramo v Visual Studiu ustvariti nov razred. Poimenujemo ga *MyDataContext*. Ta bo dedoval elemente razreda *DataContext*. *DataContext* je objekt, ki predstavlja podatkovno bazo in tabele v njej ter omogoča dostop do njih.

V razredu ustvarimo konstruktor v katerem določimo pot do podatkovne baze. Ta pot se imenuje *ConnectionString.* To je ime podatkovne baze z navedeno potjo do nje.

ConnectionString izgleda tako: `isostore:/ReminderDatabase.sdf`

Nato v razredu napišemo lastnosti, ki omogočajo dostop do posameznih tabel v podatkovni bazi. V *get* delu lastnosti uporabimo bazni razred *DataContext* in metodo *GetTable()*, ki kot rezultat vrne tabelo s podatki.

Primer lastnosti za novo tabelo:

```
public Table<ime_tabele> ime_lastnosti
        {
            get { return this.GetTable<ime_tabele>(); }
        }
```

Lastnost poimenujemo enako kot je ime tabele ki jo vrne metoda *GetTable()* saj nam takšno poimenovanje olajša kasnejše delo.

Čeprav v metodi *GetTable()* ne navedemo nobenih parametrov, moramo navesti tip, ki ga bo metoda vrnila med znakoma < in >. To prav tako storimo v definiciji lastnosti.

Ko ustvarimo razred *MyDataContext* imamo omogočen dostop do tabel ter podatkov v njih, vendar se samo ustvarjanje podatkovne baze opravi v drugih metodah, najbolje pa je, da se izvede takoj ob zagonu aplikacije, v namen temu uporabimo razred *App.xaml.cs.*

5.2.8.1.1. App.xaml.cs

App.xaml.cs je razred, katerega samodejno generira Visual Studio ko ustvarimo novo rešitev (ang. Solution). Vsebuje razne metode, ki se izvajajo samodejno ob morebitnih napakah, vsebuje pa tudi določene metode, ki so posebej namenjene trenutkom, ko je aplikacija na novo zagnana, prekinjena ali ko se zapira.

Metoda *ApplicationLaunching()* se izvede ob prvem zagonu aplikacije. Če je aplikacija prekinjena in poklicana nazaj v aktivno stanje, se metoda ne bo izvedla.

Ta metoda je primerna za opravljanje nalog, ki lahko trajajo nekaj sekund, niso pomembne za uporabnika, so pa obvezne za pravilno delovanje aplikacije. V času izvajanja te metode uporabnik nima vpliva na aplikacijo.

To metodo v našem primeru uporabimo za preverjanje stanja podatkovne baze. Uporabimo prej ustvarjen razred *MyDataContext* in metodo *DatabaseExists()*.

```
using (MyDataContext context = new MyDataContext(connString))
            {
                if (!context.DatabaseExists())
                {
                    context.CreateDatabase();
                }
```

Metoda *DatabaseExists()* tipa *bool,* vrne vrednost *true,* če podatkovna baza obstaja, ter *false,* če podatkovne baze ni na navedeni poti v *ConnectionString*-u.

V primeru, da podatkovna baza ne obstaja (ponavadi takrat, ko uporabnik prvič zažene aplikacijo), uporabimo metodo *CreateDatabase().* Ta metoda na mestu, ki ga navedemo v *ConnectionString*-u ustvari podatkovno bazo ter tabele v njej.

5.2.8.2. Vpis podatkov v podatkovno bazo

Ko uspešno ustvarimo podatkovno bazo, lahko vanjo začnemo vpisovati podatke. Pred tem pa se moramo vprašati, kaj sploh želimo hraniti v podatkovni bazi.

Če je namen aplikacije hranjenje nalog, moramo vanjo hraniti vse podatke povezane z nalogo, ki jo želi uporabnik hraniti. Aplikacija prav tako omogoča ustvarjanje kategorij, torej si moramo prav tako

zapomniti podatke o njih. Uporabnik lahko spreminja nastavitve v aplikaciji, to pa pomeni novo tabelo, v katero zapišemo vse nastavitve. Vsaki vpisani nalogi lahko uporabnik določi alarm, podatkov o alarmih pa ne smemo izgubiti, zato prav tako potrebujemo tabelo.

Kot lahko vidimo, bomo potrebovali štiri tabele. Te so: *Tasks, Reminders, Categories* in *Settings.*

5.2.8.2.1. Ustvarjanje tabele

Tako kot pri ustvarjanju podatkovne baze, ponovno ustvarimo nov razred, tokrat z imenom *Tables.* Znotraj tega razreda bomo definirali prej navedene tabele. Tabele v podatkovni bazi so sestavljene iz stolpcev in vrstic. Vsak stolpec v tabeli mora imeti ime in podatkovni tip. Vsaka vrstica pa predstavlja zapis (ang. record) tj. npr. vsi podatki o eni nalogi.

Primer ustvarjanja tabele in stolpca v njej:

```
[Table]
       class ime_tabele
       {
        [Column(Name = "ime_stolpca", IsPrimaryKey = true)]
        public int ime_stolpca
        { get; set; }
       }
```

Stolpcu lahko razen imena določimo še veliko drugih lastnosti, npr. določimo primarni ključ, določimo ali stolpec mora imeti vrednost ali ne...Pri ustvarjanju stolpcev se moramo prav tako vprašati, kaj točno želimo hraniti v določeni tabeli.

5.2.8.2.1.1. Tabela Tasks

V tabeli *Tasks* bomo hranili podatke o nalogah. Vsaka naloga mora imeti šifro, saj se lahko uporabnik odloči vpisati več nalog z istim nazivom, mi pa kot programerji potrebujemo način prepoznave določene naloge. Uporabnik bo vsaki nalogi lahko torej določil naziv, uvrstil jo bo lahko v kategorijo, nastavil datum in uro, si naredil zapiske, prav tako pa bo nalogi lahko nastavil opomnik oz. alarm ter določil, ali se bo alarm ponavljal.

Rezultat vseh teh možnosti so sledeči stolpci v tabeli: *Sifra, Naziv, Kategorija, Datum, Ura, Opomnik, Opomnik_se_ponavlja, Zapiski.*

Stolpec *Sifra* je število, katerega določimo sami, glede na to, koliko nalog že imamo v bazi. Npr.: če imamo shranjenih pet nalog, bo šifra nove naloge število šest. Stolpec *Naziv* je niz znakov, uporabnik pa sam vpiše željen naziv. Stolpec *Kategorija* je prav tako niz znakov, uporabnik pa na podlagi kategorij, ki jih je dodal, eno izbere, ta pa se zapiše v bazo. Stolpec *Zapiski* je niz znakov, uporabnik pa sam vpiše željene zapiske.

Stolpca *Datum* in *Ura* sta tipa *DateTime?* ter predstavljata datum in čas. Stolpca *Opomnik* in *Opomnik_se_ponavlja* sta tipa bool, ki omogoča vrednosti *true* in *false.* Podrobnejši opis vrednosti, ki bodo hranjene v teh stolpcih, sledi pozneje.

5.2.8.2.1.2. Tabela Categories

Tabela *Categories* ni tako obsežna, saj je vse, kar si moramo o kategorijah zapomniti ime ter barva. To pomeni, da bosta v tabeli stolpca *Ime_kategorije* in *Barva_kategorije,* oba pa sta niz znakov.

5.2.8.2.1.3. Tabela Settings

Tabela *Settings* hrani vse podatke o nastavitvah aplikacije. Uporabniku omogočamo spreminjanje jezika, nastavljanje ozadja, konfiguriranje ikon na start meniju (več o tem pozneje) ter nastavljanje sinhronizacije.

5.2.8.2.1.4. Tabela Reminders

Če se uporabnik odloči, da bo nalogi dodal alarm oz. opomnik, ima možnost izbrati datum in čas opomnika, ter določi, ali se bo opomnik ponavljal. Če se bo opomnik ponavljal, ima uporabnik prav tako možnost določiti, kolikokrat se bo ponovil oz. na kakšne časovne intervale. Ko uporabnik doda nalogo in ji je določil opomnik, se v tabelo shrani sifra naloge, ki ji opomnik pripada.

To pomeni, da imamo v tabeli *Reminders* stolpce *Sifra* in *FK_naloga_sifra* , oba sta niz znakov, ki jih določi programer (uporabnik na ta dva stolpca nima vpliva). Stolpca *Datum_zacetek* in *Ura_zacetek* sta tipa *DateTime?.* Stolpec *Tip_ponovitev* je niz znakov, uporabnik pa določi, kakšen tip ponovitev bo opomnik imel.

5.2.8.2.2. Vpis podatkov

Ko so vse željene tabele določene in ustvarjene, lahko v podatkovno bazo vnesemo podatke.

Za vnos podatkov se ponovno skličemo na prej ustvarjen razred *MyDataContext*. Z namenom, da naredimo našo kodo bolj pregledno in urejeno, ustvarimo nov razred in ga poimenujemo DML. Vanj bomo napisali več metod za vstavljanje, brisanje ter poizvedovanje po podatkih.

Primer metode za vpis:

```
public static void ime_metode(tip_stolpca _ime_parametra1, tip_stolpca _ime_parametra2...)
{
   using (MyCategoriesDataContext context = new CategoriesDataContext(connString))
   {
      ime_tabele ime_spremenljivke = new ime_tabele();
      ime_spremenljivke.ime_parametra1 = _ime_parametra1;
      ime_spremenljivke.ime_parametra2 = _ime_parametra2;
                              .
                              .
                              .
      context.ime_tabele.InsertOnSubmit(ime_spremenljivke);
      context.SubmitChanges();
   }
}
```

Vse metode za vpis podatkov so narejene na enak način. Vse so javne, statične ter ne vračajo ničesar. Z uporabo zank lahko takšne metode pokličemo večkrat, če želimo vnesti več podatkov naenkrat.

`ime_metode` – metodo je najbolje poimenovati *Add<ime elementa ki ga želimo dodati>.* Npr.: če napišemo metodo za dodajanje kategorije, jo poimenujemo *AddCategory.* To nam olajša poznejšo uporabo vseh metod za vpis podatkov.

`tip_stolpca` – tip parametra metode mora biti enak tipu stolpca za katerega vpisujemo podatke. Npr.: če želimo v tabelo vpisati šifro, ki je tipa *int,* mora biti tip parametra prav tako *int.*

`_ime_parametra1` – zaradi lažje prepoznave elementa, ki ga želimo vpisati, je pametno parametre poimenovati enako kot stolpce, v katere želimo vrednost parametra vpisati.

`ime_tabele` – enako imenu tabele v katero želimo vpisati podatke

`ime_spremenljivke` – poimenujemo jo po želji, čeprav jo je najbolje poimenovati enako kot je ime tabele

Primer metode za vpis kategorije:

```
private const string connString = @"isostore:/ReminderDatabase.sdf";

public static void AddCategories(string _imeKategorije, string _barvaKategorije)
{
   using (MyDataContext context = new MyDataContext (connString))
   {
       Categories category = new Categories();
       category.IME_KATEGORIJE = _imeKategorije;
category.BARVA_KATEGORIJE = _barvaKategorije;
context.Categories.InsertOnSubmit(category);
       context.SubmitChanges();
   }
}
```

Ker je metoda namenjena vpisu ene ali več kategorij , jo poimenujemo *AddCategories().* Število parametrov metode, njihova imena in tipi so enaki številu stolpcev v tabeli, njihovim nazivom in tipom. Tabela *Categories* ima dva stolpca. To sta *Ime_kategorije* in *Barva_kategorije.* Oba sta tipa *string.* Zato ima metoda dva parametra tipa *string* z imeni *_imeKategorije* in *_barvaKategorije.*

V stavku *using* ustvarimo nov primerek razreda *MyDataContext* in navedemo pot do podatkovne baze (določena je na začetku – *connString).* Nato v telesu *using* stavka ustvarimo nov primerek razreda *Categories,* ki predstavlja nov zapis, ki ga želimo vpisati v tabelo *Categories.*

Vsak stolpec tabele se v kodi predstavlja kot lastnost, to pomeni, da vrednost lastnosti nastavimo na vrednost pripadajočega parametra. Ta stavek se ponovi za vsak stolpec in parameter. Če nekemu stolpcu ne določimo vrednosti, bo imel vrednost *null.* Pri tem moramo biti pozorni, če smo tistemu stolpcu pri definiciji lastnost *CanBeNull* nastavili na *false.* V takšnem primeru dobimo napako.

Ko vsem lastnostim določimo vrednosti, uporabimo spremenljivko tipa *MyDataContext (*ki smo jo ustvarili na začetku v stavku *using)* ter izberemo ime tabele, ki se prav tako kaže kot lastnost (v našem primeru *Categories*) in izvedemo medoto *InsertOnSubmit,* v katero kot parameter prenesemo poprej ustvarjen primerek razreda *Categories.* Na koncu moramo poklicati še metodo *SubmitChanges(),* da se podatki zapišejo v bazo.

5.2.8.3. Poizvedovanja po podatkih

Structured Query Language – SQL, je jezik namenjen pridobivanju in ažuriranju podatkov v relacijski podatkovni bazi. Poznamo dve razdelitvi tega jezika: *Data Definition Language – DLL* in *Data Manipulation Language – DML.* Nas posebej zanima *DML,* saj omogoča vstavljanje, brisanje in posodabljanje podatkov prav tako pa omogoča pridobivanje podatkov iz podatkovnih baz.

Ker je naša aplikacija napisana v programskem jeziku C#, ki ne omogoča neposredne uporabe sintaks jezika *SQL,* moramo uporabiti nekakšen vmesnik med obema, to pa je *Language Integrated Query – LINQ* oz. *LINQ to SQL.* To je jezik, s katerim lahko ustvarjamo poizvedbe znotaj jezika C#.

LINQ smo v naši aplikaciji že uporabili in sicer pri ustvarjanju tabel.

```
[Table]
       class ime_tabele
       {
        [Column(Name = "ime_stolpca", IsPrimaryKey = true)]
        public int ime_stolpca
        { get; set; }
       }
```

Elementa `[Table]` in `[Column]` sta del jezika *LINQ.* Zgornji primer je ustvarjanje tabele. Primer poizvedovanja po podatkih pa izgleda takole:

```
public static IList<Tasks> GetTasks()
{
  IList<Tasks> taskList = null;
  using (MyDataContext context = new MyDataContext(connString))
  {
      IQueryable<Tasks> query = from c in context.Tasks select c;
      taskList = query.ToList();
  }
  return taskList;
}
```

V razredu *DML* ustvarimo novo javno, statično metodo tipa *IList. IList* je tip, ki predstavlja spisek elementov, do katerih lahko dostopamo preko indeksov. Tip elementov navedemo med znakoma < in >. Enako kot pri metodi za dodajanje podatkov, je pametno metodo za poizvedbe poimenovati tako: *Get<ime elementov ki jih iščemo v poizvedbi>,* v našem primeru so to elementi tabele *Tasks.*

IQueryable() je vmesnik jezika *LINQ* in omogoča izvajanje poizvedb v nekem viru podatkov – v našem primeru je to podatkovna baza, v kateri hranimo podatke o nalogah, kategorijah, opomnikih in nastavitvah.

Med znaka < in > navedemo ime tabele, iz katere želimo dobiti podatke. Nato navedemo ime spremenljivke tipa *IQueryable* (v našem primeru *query)* ter jo enačimo s poizvedbo napisano na desni strani enačaja. V poizvedbi določimo, v kateri tabeli so iskani podatki, nato podatke shranimo v spremenljivko *c,* ta pa se shrani v spremenljivko *query.*

Na koncu nad spremenljivko *query* izvršimo metodo *ToList(),* ki spremeni rezultat poizvedbe v spisek (v našem primeru je to spisek opravil), tega pa metoda tudi vrne kot rezultat.

5.2.8.4. *Brisanje podatkov*

Za brisanje podatkov prav tako uporabimo *LINQ.*

Primer metode za brisanje naloge:

```
public static void DeleteTask(int _sifra)
{
  using (MyDataContext context = new MyDataContext(connString))
  {
     IQueryable<Tasks> taskQuery = from t in context.Tasks where t.SIFRA == _sifra select t;
     Tasks taskToDelete = taskQuery.FirstOrDefault();
     context.Tasks.DeleteOnSubmit(taskToDelete);
     context.SubmitChanges();
  }
}
```

V razredu *DML* ustvarimo novo javno statično metodo, ki ne vrača ničesar. Poimenujemo jo po istem vzorcu kot prejšnje metode za dodajanje in poizvedovanje: *Delete<element ki ga želimo izbrisati>,* v navedenem primeru je to opravilo, torej je ime metode *DeleteTask.* Metoda ima parameter, katerega tip se mora ujemati z identifikatorjem elementa, ki ga želimo izbrisati (npr. tip *int*, če želimo nalogo iskati po šifri).

Ponovno ustvarimo spremenljivko tipa *IQueryable,* s pomočjo katere bomo izbrisali naveden podatek. Na desni strani enačaja napišemo poizvedbo, ki v določeni tabeli poišče nalogo, katere šifra se ujema s tisto, ki smo jo v metodo prenesli po parametru.

Ko poizvedba najde rezultat, se zapiše v novo spremenljivko, ki je enakega tipa kot tabela, v kateri se je poizvedba izvedla (v našem primeru spremenljivka *taskToDelete* tipa *Tasks)*.

Na koncu uporabimo metodi *DeleteOnSubmit()* (v katero kot parameter vnesemo spremenljivko *taskToDelete)* in *SubmitChanges(),* ki izbriše izbrani element iz podatkovne baze.

5.2.9. Opis strani v aplikaciji in njihova uporaba

Sedaj, ko bolje razumemo, kako delujejo glavne funkcije v aplikaciji, lahko ponovno pregledamo izgled in uporabo strani v aplikaciji.

Če ponovimo, kako je sestavljena prva stran naše aplikacije: sestavljena je iz komponente *Pivot,* ki vsebuje strani *Tasks, Categories* in *Settings.* Na dnu strani se nahaja komponenta *Application bar,* ki služi navigaciji, na njej pa se nahajajo trije gumbi. Ti so: *Dodaj, Izberi* in *Sinhroniziraj.*

Na straneh *Tasks* in *Categories* imamo prav tako komponento *ListBox,* ki je v bistvu spisek, v katerem se prikažejo vpisane naloge in kategorije. Če v podatkovni bazi nimamo vpisanih nalog ali kategorij, se v *ListBox* izpiše »*No tasks! Tap new to add more!*« na strani *Tasks,* ter »*No categories! Tap new to add more!*« na strani *Categories.*

Slika 4: Stran categories

5.2.9.1. Dodajanje, brisanje in urejanje podatkov

Ob pritisku na gumb *Dodaj,* se odvisno od tega, na kateri strani komponente *Pivot* smo, odpre stran *AddTask,* če smo na strani *Tasks* in *AddCategory,* če smo na strani *Categories.* Obe strani ki se odpreta sta namenjeni dodajanju nalog in kategorij.

Na obeh se nahajajo polja za vnos podatkov, ki smo jih prej definirali v tabelah v podatkovni bazi, prav tako pa imata komponento *Application bar,* ki ima le gumba *Confirm (*uporabnik shrani nalogo oz. kategorijo*)* in *Cancel (*uporabnik prekliče spremembe).

Ko uporabnik pritisne na gumb *Confirm,* se izvede metoda *AddTask* oz. *AddCategory* iz razreda *DML.* Če pa uporabnik pritisne na gumb *Cancel* se izvede metoda *GoBack().*

Ob pritisku na gumb *Izberi* se uporabniku na straneh *Tasks* oz. *Categories* pokaže možnost izbire vpisanih podatkov, na *Application bar* pa se pokažejo novi gumbi. Ti so: *Delete, Edit* in *Cancel.*

Če se uporabnik odloči izbrisati nalogo, jo na spisku izbere in pritisne na gumb *Delete.* Ob pritisku na gumb se izvede metoda *DeleteTask* iz razreda *DML.* Na ekranu se izpiše sporočilo, ki obvesti uporabnika, če se je naloga izbrisala ali če je prišlo do napake. Enako je pri brisanju kategorij, le da se izvede metoda *DeleteCategory.*

Če pa se uporabnik odloči urejati podatke o nalogi, se odpre nova stran *EditTask,* ki je vizualno identična strani *AddTask,* vendar so ob odprtju strani tokrat polja za vpis že polna – v njih lahko vidimo podatke o nalogi, ki jo želi uporabnik urejati. Stran *EditTask* prav tako vsebuje *Application bar* in gumba *Confirm* in *Cancel.* Ko uporabnik pritisne na gumb *Confirm,* se izvede metoda *DeleteTask().* Iz podatkovne baze se izbriše prejšnja verzija naloge (ob navigaciji na stran smo prenesli šifro naloge, ki jo želi uporabnik urediti) nato pa se izvede metoda *AddTask(),* ki v podatkovno bazo doda nalogo z

novimi podatki. Enako velja za urejanje kategorij, le da se izvedeta metodi *DeleteCategory()* in *AddCategory().*

Uporabnik lahko povrne komponento *Application bar* v prvotno stanje (tj. ponovno se pokažejo gumbi *Dodaj, Izberi* in *Sinhroniziraj)* tako, da pritisne na gumb *Cancel* ali pa podrsa na naslednjo stran komponente *Pivot.*

5.3. Izdelava Windows form aplikacije

5.3.1. C#

C#(C sharp) je objektno orientiran jezik, ki je bil oblikovan za delo z Microsoftovo .NET platformo(.NET Framework). Je nekakšen naslednik jezika C++, ki pa vsebuje tudi številne sestavine katere so značilne za Javo. C# poenostavi programiranje skozi uporabo »Extensible Markup Language«(XML) in »Simple object access protcol« (SOAP), ki omogočajo dostop do programiranega objekta ali metode brez potrebe, da programer piše dodatno kodo za vsak korak. Ker programerji lahko gradijo že na obstoječi kodi, kot pa da jo vsakič znova napišejo, se pričakuje da bodo na trg prišli produkti hitreje in mnogo ceneje.

5.3.2. .NET Framework

.NET Framework (dot net) je programsko ogrodje, ki ga je razvil Microsoft in deluje predvsem na Microsoft Windows. Programi napisani za .NET Framework se zaženejo v programskem okolju, *Common Language Runtime* (CLR), aplikacijska navidezna naprava, ki omogoča operacije kot so varnost, upravljanje pomnilnika in krmilnik izjem.

.NET Framework-ova knjižnica osnovnega razreda priskrbi uporabniški vmesnik, dostop do podatkov, povezanost podatkovnih baz, kriptografijo, razvoj spletne aplikacije, numerične algoritme in omrežno komunikacijo. Programerji naredijo svojo programsko opremo s tem da združijo svojo izvorno kodo s .NET Framework in drugimi knjižnicam. .NET Framework naj bi se uporabljal za večino novih aplikacij ustvarjenih za Windows platformo. Microsoft prav tako proizvaja integrirano razvojono okolje predvsem za .NET programsko opremo, imenovano Visual Studio.

5.3.3. Web service

Je aplikacija, ki je namenjena za komunikacijo z drugo aplikacijo preko svetovnega spleta. Na splošno se uporablja za izmenjavo podatkov. Spletna storitev je jezikovno neodvisna. To pomeni, da se lahko preko spletne storitve narejeno z .NET poveže npr. Java aplikacija. Spletna storitev spremeni sporočilo v XML format, ki ga potem pošlje preko HTTP protokola.
Dva objekta komunicirata eden z drugim preko spleta z uporabo standardnega spletnega protokola in obliko podatkov, ti pa so: HTTP, XML, SOAP.
HTTP (Hypertext transfer protocol)- je protokol, ki skrbi za ustrezen prenos informacij preko spleta. Je protokol za komunikacijo med odjemalci in strežniki. HTTP odjemalec je ponavadi spletni brkalnik.
XML (Extensible markup language) – je preprost računalniški jezik podoben HTML-ju, ki nam omogoča format za opisovanje strukturiranih podatkov ali arhitektura za prenos podatkov in njihovo izmenjavo med več omrežji.
SOAP (Simple object access protocol) – je specifikacija protokola za izmenjavo strukturiranih

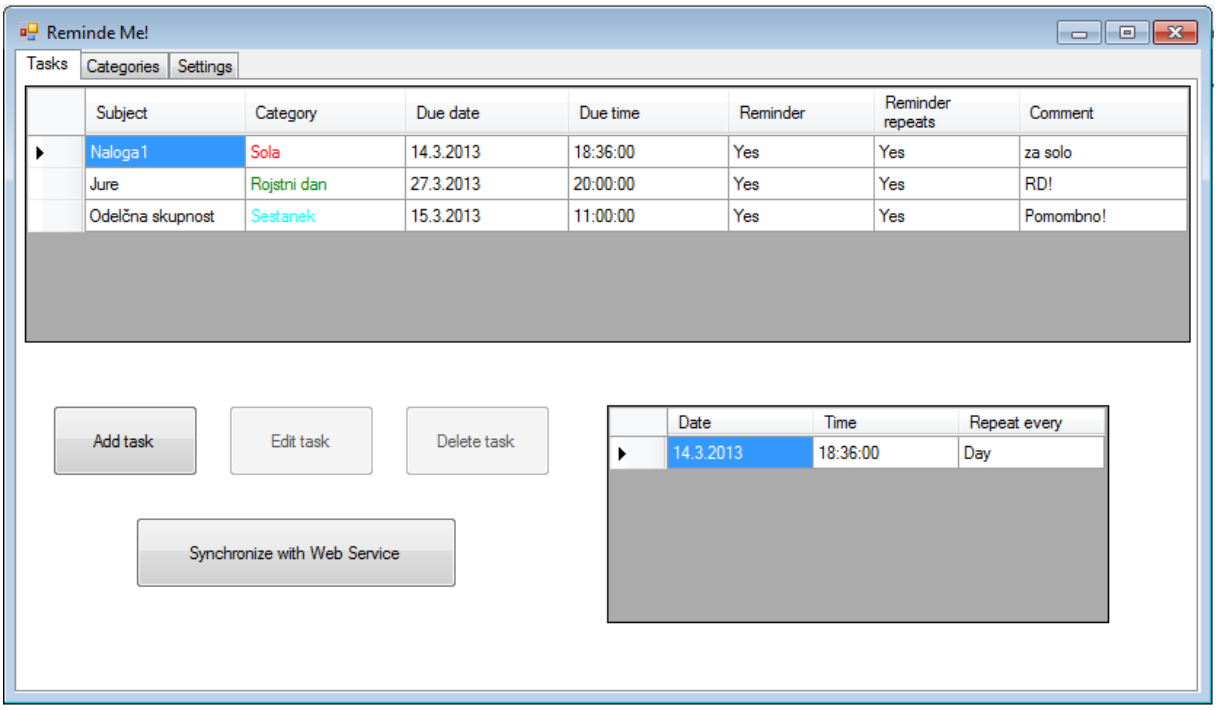
podatkov pri izvajanju spletnih storitev v računalniških omrežjih. Opira se na XML za njeno obliko sporočila. Za prenos sporočil pa običajno temelji na drugih protokolih, predvsem na HTTP.

V najini aplikaciji sva uporabila spletno storitev z namenom, da se lahko podatki iz mobilne aplikacije na Windows Phone posodobijo s tistimi, ki jih uporabnik vnese v okenski aplikaciji na osebnem računalniku.

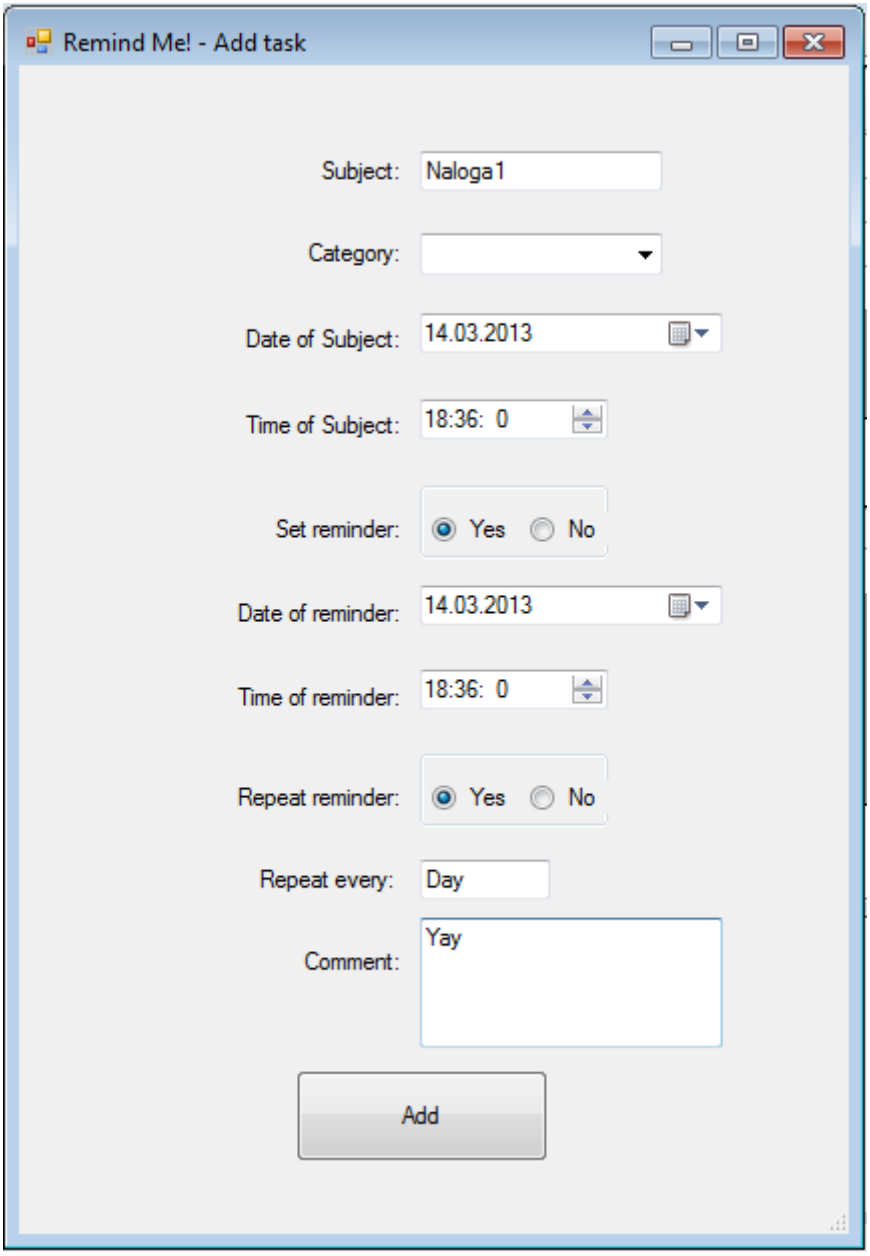
5.3.4. Windows Form

Windows Form je ime za grafični programski vmesnik, ki je del tehnologije Microsoft .NET Framework in zagotavlja dostop do domačih vmesnikovih elementov Microsoft Windows z ovojem Windows API v upravljalno kodo.

Ob zagonu aplikacije se nam pokaže glavno okno. Na njem imamo tri zavihke (*Tasks, Categories in Settings*) . V zavitku *Tasks* se nahajata dve tabeli in gumbi *Add task, Edit task, Delete task* in *Synchronize with Web Service*. V sami tabeli, ki je povezana s podatkovno bazo, so prikazani vsi opomniki, ki smo jih dodali. Za dodajanje opomnika v bazo moramo samo klikniti dodaj opomnik in prikaže se nam novo okno v katerega vpišemo vse potrebne podatke (ime, datum...). Ko so vsi podatki vpisani, z gumbom potrdimo in podatki se prenesejo v podatkovno bazo, poslednično se prikažejo v tabeli na glavnem oknu.



Slika 5: Glavno okno Windows form aplikacije



Slika 6: Okno za dodajanje opomnika

Da lahko podatke iz okna za dodajanje, prenesemo v podatkovno bazo moramo ob pristisku gumba dodaj najprej vzpostaviti povezavo z bazo. To naredimo tako:

```
string conString = "Data source=BazaOpomnik.sdf"; //Določimo pot, kjer se nahaja baza.
con = new SqlCeConnection(conString); // Ustvarimo novo povezavo.
```

Določiti moramo tudi ukaze, s pomočjo katerih bomo vnašali podatke v podatkovno bazo.

```
SqlCeCommand dbCmdTasks = new SqlCeCommand();// Ustvarimo nov ukaz za vstavljanje v
tabelo »TableTasks«.
SqlCeCommand dbCmdReminder = new SqlCeCommand();// Ustvarimo nov ukaz za vstavljanje v
tabelo »TableReminder«.
```

Nato pa pride na vrsto še samo vstavljanje v bazo, s pomočjo poizvedbe. (Spodaj prikazan samo del kode).

```
string sqlAddNewTasks = "INSERT INTO TableTasks Values(@ID, @subject, @category,
@dateSub, @timeSub, @reminder, @reminderRepeats, @comment)"; // Napišemo poizvedbo.
dbCmdTasks = new SqlCeCommand(sqlAddNewTasks, con); // Dodamo poizvedbo v ukaz
»dbCmdTasks« in dodamo še povezavo do baze (»con«).

dbCmdTasks.Parameters.AddWithValue("@ID", SqlDbType.Int).Value = ID; // Dodajanje
spremenljivk v poizvedbo.
dbCmdTasks.Parameters.AddWithValue("@subject", addForm.textBoxSubject.Text);

string sqlAddNewReminder = "INSERT INTO TableReminder (FK, DATE, TIME, REPEAT_EVERY)
Values(@fk, @dateRem, @timeRem, @repeatEvery)";
dbCmdReminder = new SqlCeCommand(sqlAddNewReminder, con);

dbCmdReminder.Parameters.AddWithValue("@fk", SqlDbType.Int).Value = ID;
dbCmdReminder.Parameters.AddWithValue("@dateRem",
addForm.dateTimePickerDateOfRem.Value.ToShortDateString());
```

Za vpis v bazo pa moramo še samo izvesti obe poizvedbi. Pred tem pa moramo še odpreti povezavo, ki smo jo naredili na začetku.

```
con.Open();// Odpre povezavo.
dbCmdTasks.ExecuteNonQuery();// Izvede poizvedbo za tabelo »TableTasks«
dbCmdReminder.ExecuteNonQuery();// Izvede poizvedbo za tabelo »TableReminder«
con.Close();// Zapre povezavo.
```

5.3.5. Spletna storitev

Ob pritisku na gumb *Synch with Web Service* prejmemo od spletne storitve vse naloge, ki so trenutno shranjene na strežniku, preverimo, ali imamo v lokalni podatkovni bazi morda shranjeno kakšno več oz. kakšno manj (če smo zbrisali kakšen vnos) ter posodobljeno verzijo pošljemo nazaj na strežnik.

Primer definicije metode v spletni storitvi:

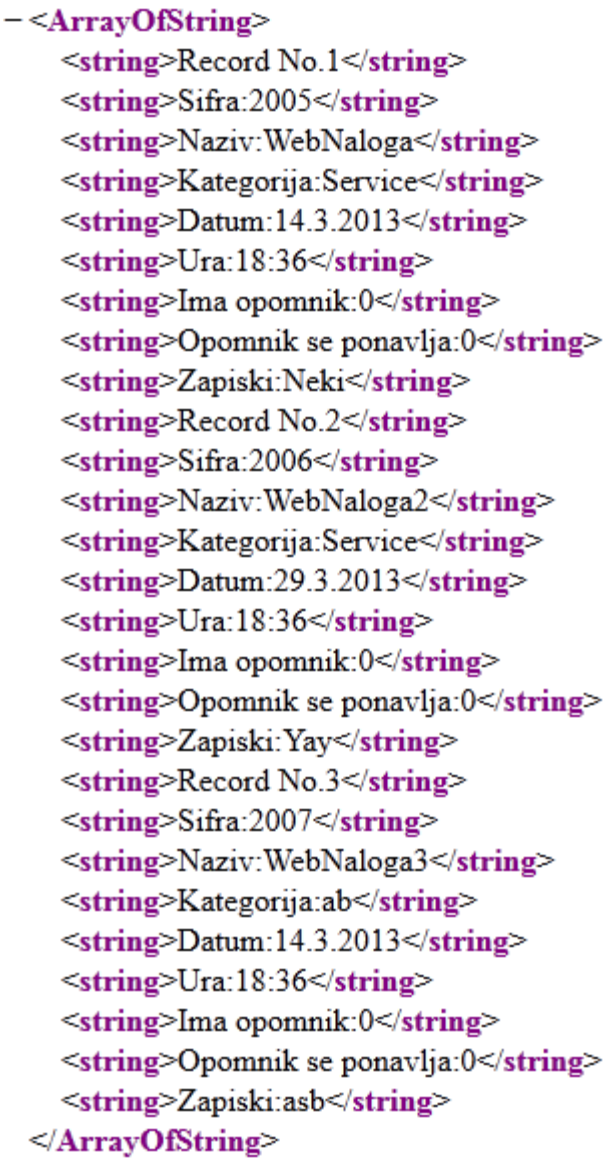
```
[WebMethod]
public void AddCategory(string _categoryName, string _categoryColor)
{
    SqlCeCommand command = new SqlCeCommand();
    command.Parameters.AddWithValue("@categoryName", _categoryName);
    command.Parameters.AddWithValue("@categoryColor", _categoryColor);
    command.Connection = conn;
    command.CommandText = "INSERT INTO Categories

VALUES(@categoryName,@categoryColor)";
    conn.Open();
    command.ExecuteNonQuery();
    conn.Close();
}
```

V metodi najprej določimo ukaz in mu dodamo parametre. V zgornjem primeru je to ukaz za dodajanje kategorije v podatkovno bazo, parametri pa so podatki o kategoriji, ki jih je vnesel uporabnik. Nato ukazu določimo povezavo z podatkovno bazo in besedilo ukaza, ki je niz, predstavlja pa stavek v SQL jeziku.

Ko pravilno določimo te podatke, odpremo povezavo z bazo, izvedemo poizvedbo ter povezavo zapremo.

Primer uporabe spletne storitve v programu:

```
ServiceReference1.ReminderServiceSoapClient service = new
ServiceReference1.ReminderServiceSoapClient();// Ustvarimo novo spremenljivko, s
katero lahko kličemo metode iz spletne storitve.
service.AddTask(ID, addForm.textBoxSubject.Text, addForm.comboBoxCategory.Text,
addForm.dateTimePickerDateOfSub.Value, addForm.dateTimePickerTimeOfSub.Value,
columnSetReminderint, columnRepeatReminderint, addForm.textBoxComment.Text); // Metoda
AddTask doda vse pridobljene atribute v podatkovno bazo na spletni storitvi.
```



Slika 7: Podatki, ki jih vrne metoda spletne storitve

Primer vpisa posodobljenih podatkov:

Pridobljene podatke moramo najprej »razbiti«, da ostanejo samo še podatki, ki jih potrebujemo za vpis v podatkovno bazo.

```
ServiceReference1.ReminderServiceSoapClient service = new
ServiceReference1.ReminderServiceSoapClient();// Ustvarimo novo spremenljivko, s
katero lahko kličemo metode iz spletne storitve.

List<string> tasks = service.GetTasks();// Ustvarimo spremenljivko tasks in notri se
zapiše kar nam spletni strežnik vrne.
StreamWriter sw = new StreamWriter(@".\temp.txt");// StreamWriter ustvari datoteko
temp.txt

foreach (string task in tasks)
{
sw.WriteLine(task); // V temp.txt se zapiše kar nam je spletni strežnik vrnil.
}
sw.Close();

// Ustvarimo prazne spremenljivke v katere bomo vnašali pridobljene podatke.
string recNo = string.Empty;
string id = string.Empty;
string subject = string.Empty;
string category = string.Empty;
string dateSub = string.Empty;
string timeSub = string.Empty;
string reminder = string.Empty;
string reminderRepeats = string.Empty;
string comment = string.Empty;
string temp = string.Empty;

StreamReader sr = new StreamReader(@".\temp.txt"); // S StreamReader-jem bomo prebrali
kar smo zapisali v datoteko temp.txt.

using (sr)
{
       while (!sr.EndOfStream)
       {
       temp = sr.ReadLine();// Preberemo vrstico.
       if (temp.StartsWith("Sifra:")) //Ce se string začne s »Sifra:« gre v if.
              {
                     id = temp.ToString();//V spremenljivko id dobimo string »Sifra:1«.
                     id = id.Substring(temp.IndexOf("Sifra:") + 6); //Odstranimo
              »Sifra:«, da nam ostane samo še 1.
              }
       }
}
```

Ko smo pridobljene podatke spravili v prave spremenljivke, ponovno odpremo povezavo do podatkovne baze in na enak načino, kot smo dodajali podatke v bazo v sami aplikaciji, dodamo podatke v lokalno podatkovno bazo.

V zavitku *Categories* imamo možnost dodati nove kategorije, ki jih lahko izberemo v okno za dodajanje opomnika. V zavitku *Settings* pa imamo možnost spremeniti jezik v slovenščino ali angleščino in podatke o programu.

6. ZAKLJUČEK

Za konec želiva ponoviti najino hipotezo in cilje:

Hipoteza raziskovalne naloge:

Tehnologija .NET omogoča razvoj strežniške aplikacije za opomnik na tak način, da se bo z njo mogoče povezati z odjemalci, napisanimi za različne platforme, npr. Windows Forms, ASP.NET, Windows Phone, Android in iOS.

Cilji raziskovalne naloge:

- ustvariti mobilni opomnik za Windows Phone
- ustvariti opomnik kot okensko aplikacijo za op. sistem Windows
- omogočiti povezovanje mobilne aplikacije z okensko aplikacijo
- preučiti možnosti za ustvarjanje opomnika na platformah ASP.NET, Android in iOS

Hipotezo sva potrdila delno, saj sva ugotovila, da je možno povezati aplikacijo narejeno za platformo Windows Phone z aplikacijo narejeno za platformo Windows, z uporabo spletne storitve oz. Web Service. Za popolno potrditev hipoteze bi bila potrebna še raziskava in preizkus možnosti povezovanja aplikacij narejenih za mobilni op. sistem Android ter iOS. Če upoštevava, da spletna storitev za komunikacijo uporablja jezike kot sta XML ter SOAP, je to zelo verjetno.

V Visual Studiu sva ustvarila okensko in mobilno aplikacijo, vsako z lokalno podatkovno bazo, v katero uporabnik hrani podatke. Obe aplikaciji sva povezala s pomočjo spletne storitve, prav tako narejene v Visual Studiu. S tem sva omogočila možnost sinhronizacije med aplikacijama.

Koncept takšne aplikacije bi bil koristen v podjetjih, pri izvajanju projektov. Uporaba aplikacije lahko zmanjša možnost napak pri izvedbi in pohitri delo. Za nadaljni razvoj aplikacije v tej smeri bi bila potrebna raziskava potreb po takšni aplikaciji v podjetjih.

7. VIRI

Uporaba delete stavka v SQL (http://www.w3schools.com/sql/sql_delete.asp, 14.3.2013)

Uporaba insert stavka v SQL (http://www.w3schools.com/sql/sql_insert.asp, 14.3.2013)

O visual studiu (http://en.wikipedia.org/wiki/Microsoft_Visual_Studio, 14.3.2013)

Windows phone development (http://www.microsoftvirtualacademy.com/training-courses/windows-phone-8-apps-development-jumpstart , 14.3.2013)

O C# (http://en.wikipedia.org/wiki/C_Sharp_%28programming_language%29, 15.3.2013)

O .NET Frameworku (http://en.wikipedia.org/wiki/.NET_Framework, 15.3.2013)

Ustvarjanje Web service (http://www.codeproject.com/Articles/863/Your-first-C-Web-Service 15.3.2013)

O spletni storitvi (http://en.wikipedia.org/wiki/Web_service, 15.3.2013)

Stran uporabljena za izdelavo ankete(https://www.1ka.si/, 15.3.2013)

8. Zahvala

Zahvalila bi se rada najinemu mentorju, Boštjanu Resinoviču, za pomoč pri izbiri teme ter nasvete in usmerjanje pri izdelavi raziskovalne naloge.