Mobilna aplikacija za dogovaranje sastanaka

Grubić, Bojan

Master's thesis / Diplomski rad

2016

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:405128

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-07-16**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
ELEKTROTEHNIČKI FAKULTET

Sveučilišni studij

MOBILNA APLIKACIJA ZA DOGOVARANJE SASTANAKA

Diplomski rad

Bojan Grubić

Osijek, 2016.

SADRŽAJ

1. UVOD 6
1.1. Zadatak završnog rada 6
2. TEHNOLOGIJE I PROGRAMSKI JEZICI KORIŠTENI ZA IZRADU RADA 7
2.1. Windows Phone aplikacija 7
2.2. REST API 7
2.2.1. GET 8
2.2.2. POST 8
2.2.3. PUT 8
2.3. C# 8
2.3.1. .NET 9
2.4. XAML 9
2.5. T-SQL 12
2.5.1. Varijable 12
2.5.2. Kontrola toka izvršavanja 12
3. MODEL PODATAKA I ARHITEKTURA SUSTAVA 14
3.1. Relacijski dijagram baze podataka 14
3.1.1. Entitet Event 15
3.1.2. Entiteti User i AspNetUser 15
3.1.3. Entitet RSVP 16
3.1.4. Entiteti UserRSVP i EventRSVP 17
3.2. Arhitektura sustava 17
4. REST API i Backend Business Layer 19
4.1. Sloj „EventData“ 20
4.1.1. Rad s podacima u bazi podataka 20

4.1.2. Metode sloja „EventData“ 22
4.2. Objekti za prijenos podataka (DTO) 23
4.3. Kontroler „EventsController“ 24
4.4. Push servis 26
5. MOBILNA APLIKACIJA 28
5.1. Arhitektura aplikacije 28
5.1.1. MVVM 29
5.2. Autorizacija i sigurnost 35
5.2.1. Bearer token 35
5.3. Zaštita od rušenja aplikacije 35
5.4. Push notifikacije 36
5.5. Testiranje 36
5.6. Upute za korištenje aplikacije 36
5.7. Moguće nadogradnje 42
6. ZAKLJUČAK 43
7. Literatura 44
8. Sažetak 45
„Mobile event planer“ 46
9. Životopis 47
10. Prilozi 48
10.1. Prilog P10.1. – Skripta baze podataka 48
10.2. Prilog P10.2. – Kompletan programski kod REST API-a 48
10.3. Prilog P10.3. – Kompletan programski kod 48

Popis oznaka i kratica

XAML - Extensible Application Markup Language

REST - Representational State Transfer

API – Application programming interface

URI - Uniform Resource Identifier

CLR - Common Language Runtime

IL - Intermediate Language

FCL - Framework Class Library

SQL - Structured Query Language

ER diagram - entity relationship diagram

DTO – Dana Transfer Object

JSON - JavaScript Object Notation

MVVM – Model-View-ViewModel

Popis slika

Sl. 2.1. Ključna riječ GET 8
Sl. 2.2. XAML kod za korisničko sučelje 10
Sl. 2.3. C# kod za korisničko sučelje 11
Sl. 2.4. Izgled aplikacije definiran C# i XAML kodom 11
Sl. 2.5. T-SQL varijable 12
Sl. 2.6. Kontrola toka programa IF ELSE petljom 13
Sl. 2.7. Kontrola toka programa ključnim riječima BEGIN i END 13
Sl. 3.1. Relacijski model baze podataka 14
Sl. 3.2. Atributi entiteta Event 15
Sl. 3.3. Entitet User 16
Sl. 3.4. Entitet RSVP 16
Sl. 3.5. Spojne tablice 17
Sl. 3.6. Arhitektura sustava 18
Sl. 4.1. Arhitektura Busines layera 19
Sl. 4.2.Klasa „Events“ 20
Sl. 4.3. Storana procedura za kreiranje/uređivanje događaja 21
Sl. 4.4. Metode sloja „EventData“ 22
Sl. 4.5. Metoda „GetUserCreatedEventsByUserId“ 23
Sl. 4.6. Objekti za prijenos podataka 24
Sl. 4.7. Sve metode kontrolera „EventsConntroler“ 25
Sl. 4.8. Metoda kontrolera „GetUserCreatedEventsByUserId“ 26
Sl. 4.9. Metoda kontrolera „ConfirmEventRSVP“ 26
Sl. 4.10. Slanje push notifikacije 26
Sl. 4.11. Životni ciklus push notifikacije 27
Sl. 5.1. Arhitektura mobilne aplikacije 28

Sl. 5.2. MVVM arhitektura 30
Sl. 5.3. Primjer modela 31
Sl. 5.4. Primjer View-a 32
Sl. 5.5. Primjer View Modela 34
Sl. 5.6. Tijek protokola OAuth 2.0 35
Sl. 5.7. Forma za Login 37
Sl. 5.8. Forma za registraciju 37
Sl. 5.9. Attending events 37
Sl. 5.10. Servis nije dostupan ili nije moguće uspostaviti vezu 37
Sl. 5.11. Prikaz događaja na karti 38
Sl. 5.12. Prikaz detalja o događaju 39
Sl. 5.13. Prikaz korisnika koji su potvrdili dolazak na događaj 39
Sl. 5.14. Lokacija događaja 39
Sl. 5.15. Navigacija do događaja (1) 40
Sl. 5.16. Navigacija do događaja (2) 40
Sl. 5.17. About (1) 40
Sl. 5.18. About (2) 40
Sl. 5.19. Cancel push 41
Sl. 5.20. Confirm push 41
Sl. 5.21. Uspješna prijava 41
Sl. 5.22. Uspješna odjava 41
Sl. 5.23. Neuspješna prijava dolaska na događaj 42
Sl. 5.24. Neuspješno otkazivanje dolaska na događaj 42

1. UVOD

Cilj ovog rada je napraviti Windows phone aplikaciju koja će omogućiti prikaz, kreiranje i potvrdu/otkazivanje dolaska na neki event(događaj). Uz mobilnu aplikaciju također izradit će se pripadajući REST API kao i baza podataka.

Rad je podijeljen u sedam poglavlja. U prvom poglavlju definira se zadatak završnog rada. Sljedeće poglavlje opisuje tehnologije i programske jezike korištene prilikom izrade rada. Programski jezik korišten za izradu mobilne aplikacije kao i REST API-a je C#, dok se za izradu korisničkog sučelja mobilne aplikacije koristio XAML, a za izradu baze podataka T-SQL.

U trećem poglavlju opisan je model podataka cijelog sustava te prikazane relacije između entiteta. Četvrto i peto poglavlje bave se izradom i komponentama REST API-a i mobilne aplikacije.

U šestom poglavlju opisano je kako se mobilna aplikacije koristi te su nabrojane njene funkcionalnosti. U posljednjem poglavlju dan je osvrt na ciljeve postavljene u zadatku diplomskog rada i postignute rezultate.

U posljednjem poglavlju dan je osvrt na ciljeve postavljene u zadatku završnog rada i postignute rezultate.

1.1. Zadatak završnog rada

Zadatak diplomskog rada je izraditi mobilnu aplikaciju, relacijsku bazu podataka te REST API koji će omogućiti komunikaciju mobilne aplikacije s bazom podataka.

Cijeli sustav mora omogućiti korisniku kreiranje novih događaja, te pozivanje drugih korisnika na kreirane događaje, kao i pregled događaja na koje je korisnik pozvan od strane drugih korisnika.

2. TEHNOLOGIJE I PROGRAMSKI JEZICI KORIŠTENI ZA IZRADU RADA

U ovom poglavlju bit će opisane tehnologije i programski jezici korišteni za izradu mobilne aplikacije kao i kompletnog sustava.

2.1. Windows Phone aplikacija

Tijekom projekta razvijana je Windows Phone aplikacija s funkcionalnostima opisanim u sljedećim poglavljima.

Aplikacija je razvijana u Visual studio 2013 razvojnom okruženju s instaliranim WP8.1 SDK. Aplikacija je razvijana u programskom jeziku C# i XAML jeziku za prikaz dokumenata kojim se definira korisničko sučelje aplikacije. Aplikaciju je moguće instalirati na mobilne uređaje s operacijskim sustavom Windows Phone 8.1, te Windows mobile (Windows Phone 10).

Za korištenje aplikacije potrebno je imati pristup internetu, te uključen lokacijski servis. Dozvola za korištenje *push* notifikacija dodjeljuje se pri prvom pokretanju aplikacije.

2.2. REST API

REST (*Representational State Transfer*) možemo definirati kao arhitekturni stil koji se nalazi povrh niza principa. Učestalo korištenje REST-a zadnjih godina povezano je s dizajnom API-a web aplikacija koje često pružaju dodatne mogućnosti pomoću REST-a.

Ključna stvar REST-a su resursi - „stvari“ kojima želimo upravljati koristeći API. Resurs može biti objava na blogu, dokument, kupac, općenito sve što želimo otkriti prema van. Resurs možemo zatražiti putem URI-a, a dobit ćemo resurs u određenom formatu. Format dogovaraju server i klijent i može biti bilo koji, od najčešćih XML-a i JSON-a, do HTML-a, PNG-a, CSV-a ili bilo kojeg drugog binarnog formata. S dobivenim resursom klijent može upravljati njegovim stanjem te vršiti operacije na njemu, ovisno o pravima koja su mu dodijeljena. *Statelessness* (eng. bez čuvanja stanja) je temeljni princip REST-a. Server nikada ne bi trebao spremati informacije o klijentu. To znači da, kada na server dođe zahtjev, server iz baze učitava tražene podatke te ih šalje klijentu u zadanom obliku. Ako se zbog novog zahtjeva, sekundu kasnije, stanje resursa u bazi promjeni, klijent nema potrebu znati za to. Ovaj pristup također znači da server nikada ne bi trebao koristit sesije ili druge mehanizme za pohranu informacija o klijentu, te da niti jedan zahtjev nije povezan s

prethodnim niti budućim. Na ovaj način klijent i server nisu povezani, te server može promijeniti lokaciju resursa bez da klijent prestane raditi.

Operacije s resursima temelje se na HTTP ključnim riječima u kombinaciji s URI-em.

2.2.1. GET

Ključna riječ GET koristi se za dohvat resursa.

```
GET /posts HTTP/1.1
Accept: application/json
```

Sl. 2.1. Ključna riječ GET

Na slici 2.1. zahtjev sa ključnom riječi GET koristi se za dohvat resursa pod imenom „posts“ u JSON formatu. Ovi zahtjevi smatraju se sigurnima i ne mijenjaju stanje resursa. Ako zahtjev prođe uredu, server odgovara sa HTTP statusom 200 „OK“, ako URI pokazuje na nepostojeći resurs server vraća status 404 „Not found“, a ako zahtijev nije dobro složen, server vraća status 400 „Bad request“.

2.2.2. POST

Kada se zahtjev sa ključnom riječi POST koristi za kreiranje resursa, podaci o resursu šalju se na server kao dio tijela(eng. *body*) zahtjeva. Ako sve prođe uredu, server odgovara sa statusom 201 „CREATED“.

2.2.3. PUT

PUT se koristi prilikom modificiranja resursa. URI definira resurs koji želimo mijenjati, a tijelo zahtjeva sadrži nove vrijednosti resursa. Odgovori koje server vraća klijentu trebali bi biti 200 „OK“ ili 204 „No content“ u slučaju da odgovor ne sadrži izmijenjeni resurs. Rezultat uspješnog zahtjeva s ključnom riječi PUT trebao bi biti neovisan o broju izvršavanja tog zahtjeva. Mora biti moguće poslati dva identična zahtjeva na server i server ne bi smio vratiti grešku; drugi zahtjev samo ponovi radnju prvoga iako to ne radi nikakve promjene na resursu.

2.3. C#

C# je objektno orijentirani jezik, sveopće namjene, koji se koristi za programiranje bazirano na komponentama. Pošto je jezik sveopće namjene, postoje nebrojeni načini njegove primjene kako bi se odradili određeni zadaci. Korištenjem ASP.NET-a moguće je

raditi web aplikacije, uporabom *Windows Presentation Foundation* (WPF) moguće je razvijati desktop aplikacije, također moguće je razvijati i mobilne aplikacije kako za Windows Phone tako i za druge mobilne operativne sustave. Druge primjene C#-a uključuju kod koji se pokreće u oblaku pomoću Windows Azure-a, iOS, Android i Windows Phone aplikacije razvijane pomoću platforme Xamarin. Naravno, postoje stvari za koje je bolje koristiti C ili C++, kao što je primjerice komunikacija s hardverom ili u sustavima stvarnog vremena.

Gledajući globalno, korištenjem programskog jezika C# moguće je napraviti jako mnogo različitih stvari u raznim primjenama.

2.3.1. .NET

.NET je platforma koja uključuj jezike, *runtime*, te razvojni okvir sa mnoštvom biblioteka koji razvojnim inženjerima omogućuju razvoj velikog broja različitih aplikacija. Uz C#, .NET također sadrži Visual Basic, F#,C++ i dr. *Runtime* se službeno naziva *Common Language Runtime* (CLR). Programski jezici koje CLR koristi kompiliraju se u asemblerski jezik (eng. Intermediate Language). CLR sam po sebi je zapravo virtualna mašina na kojoj se izvršava asemblerski kod i omogućuje pristup raznim servisima, kao što je upravljanje memorijom, skupljanje otpada u kodu, upravljanje pogreškama, sigurnosni servis i dr.

Biblioteka klasa razvojnog okvira (eng. *Framework Class Library*) je skup višestruko upotrebljivog programskog koda koji omogućuje standardne mogućnosti, kao i neke posebne mogućnosti, ovisno o platformi na kojoj se primjenjuje. Standardne mogućnosti uključuju rad s kolekcijama, kriptografiju, rad s mrežom i još mnogo toga. Uz osnovne mogućnosti FCL također omogućuje rad i s nekim specifičnostima vezanim uz platformu za koju se razvija aplikacija, kao što je ASP.NET, WPF, web servisi i dr. FCL omogućuje da različite aplikacije imaju identične, dijeljene dijelove koda, koji se koriste kao komponente i tako omogućuju višestruku primjenu istog koda što višestruko štedi vrijeme i novac.

2.4. XAML

Extensible Application Markup Language ili XAML, je jezik za označavanje dokumenata koji se temelji na XML-u, a razvio ga je Microsoft. XAML je jezik koji se koristi za vizualnu prezentaciju aplikacije, odnosno izradu korisničkog sučelja, isto kao što se HTML koristi pri izradi web aplikacija. Korištenjem XAML-a definiraju se vidljivi elementi korisničkog sučelja koji se nalaze u .xaml dokumentima dok se kod, odnosno logika izvršavanja, nalazi

u dokumentima pozadinskog koda (eng. *code-behind*). Na taj način postiže se odvojenost programske logike aplikacije od njenog izgleda, odnosno dizajna. Ta dva tipa dokumenata povezani su definiranjem parcijalnih klasa. XAML kod predstavlja izravnu instancu objekta pomoću skupa pomoćnih tipova definiranih u *assembly*-u. Po ovome se XAML razlikuje od ostalih sličnih jezika, koji su obično samo interpretacija jezika, bez veze 1 na 1 između tagova vizualnih elemenata i stvarnih klasa u kodu. Što znači da, za svaki vizualni element koji se može definirati u XAML kodu, mora postojati klasa koja definira taj element, a to svakako razvojnim inženjerima olakšava izradu specifičnih elemenata po potrebi. Odvojenost izrade vizualnog sučelja aplikacije i logike aplikacije omogućuje da na istom dijelu aplikacije istovremeno rade dizajner i razvojni inženjer, ne smetajući jedan drugome, čak mogu koristit i različite razvojne alate. Kada se gledaju kao tekst, XAML dokumenti su zapravo XML dokumenti koji imaju ekstenziju .xaml, mogu biti enkodirani korištenjem XML ili UTF-8 standarda.

```
<Window x:Class = "XAMLVsCode.MainWindow"
   xmlns = "http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x = "http://schemas.microsoft.com/winfx/2006/xaml" Title = "MainWindow" Height = "350" Width = "525">

    <StackPanel>
        <TextBlock Text = "Dobro došli na XAML tutorijal :-)" Height = "20" Width = "200" Margin = "5"/>
        <Button Content = "Ok" Height = "20" Width = "60" Margin = "5"/>
    </StackPanel>

</Window>
```

Sl. 2.2. XAML kod za korisničko sučelje

Na slici 2.2. prikazan je XAML kod kojim je definiran izgleda aplikacije, dok je na slici 2.3. prikazan C# kod potreban kako bi se izradilo identično korisničko sučelje. Očito je da je korištenjem XAML-a mnogo lakše izrađivati korisničko sučelje, što zbog same preglednosti koda, što zbog činjenice da nije potrebno veliko poznavanje programiranja i C#-a za izradu sučelja. Također, mogu se koristit alati za rad s XAML-om poput Microsoft Blend-a koji dizajnerima pruža poznato sučelje.

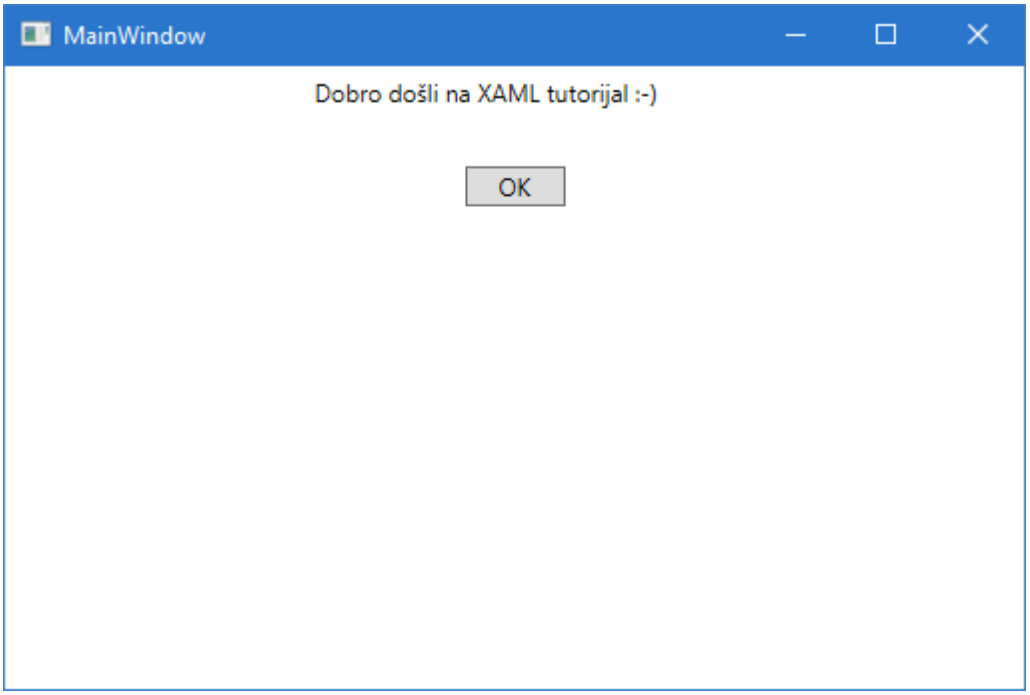
Na slici 2.4. prikazan je rezultat nakon izvršavanja obje verzije koda, koji je identičan.

```
public partial class MainWindow : Window
{
    0 references
    public MainWindow()
    {
        InitializeComponent();
        // Kreiranje StackPanela
        StackPanel stackPanel = new StackPanel();
        this.Content = stackPanel;

        // Kreiranje TextBlocka
        TextBlock textBlock = new TextBlock();
        textBlock.Text = "Dobro došli na XAML tutorijal :-)";
        textBlock.Height = 20;
        textBlock.Width = 200;
        textBlock.Margin = new Thickness(5);
        stackPanel.Children.Add(textBlock);

        // Kreiranje Buttona
        Button button = new Button();
        button.Content = "OK";
        button.Height = 20;
        button.Width = 50;
        button.Margin = new Thickness(20);
        stackPanel.Children.Add(button);
    }
}
```

Sl. 2.3. C# kod za korisničko sučelje



Sl. 2.4. Izgled aplikacije definiran C# i XAML kodom

2.5. T-SQL

Transact-SQL (T-SQL) je proizvod Microsofta i tvrtke Sybase, a predstavlja proširenje SQL-a koji se koristio za rad s relacijskim bazama podataka. T-SQL proširuje SQL standard uključenjem proceduralnog programiranja, lokalnih varijabli, također omogućuje funkcije za obradu stringova, datuma, razne matematičke funkcije i slično, te donosi promjene za DELETE i UPDATE naredbe.

Transact-SQL koristi Microsoft SQL Server. Sve aplikacije koje komuniciraju sa instancom SQL servera, čine to slanjem T-SQL naredbi serveru, neovisno o korisničkom sučelju aplikacije.

2.5.1. Varijable

Deklaracija varijabli vrši se pomoću ključnih riječi DECALRE, SET i SELECT, kao što je i prikazano na primjeru sa slike 2.5.

```
DECLARE @var1 NVARCHAR(30)
SET @var1 = 'Some Name'
SELECT @var1 = Name
FROM Sales.Store
WHERE CustomerID = 100
```

Sl. 2.5. T-SQL varijable

2.5.2. Kontrola toka izvršavanja

Ključne riječi za kontrolu toka su BEGIN i END, BREAK, CONINUE, GOTO, IF i ELSE, RETURN, WAITFOR i WHILE.

Na slikama 2.6. i 2.7. prikazan je primjer uporabe BEGIN i END, te IF i ELSE ključnih riječi za kontrolu toka programa.

IF – ELSE petlja omogućava uvjetno izvođenje programa. Kao što vidimo na slici 2.6., ako je zadovoljen prvi uvjet[1] ispisat će se tekst „Vikend je“, a u slučaju da taj uvjet nije zadovoljen ispisat će se tekst „Radni je dan.“

Ako moramo, nakon ispunjenja određenog uvjeta, odraditi više stvari ili pokrenuti blok naredbi, za to nam je potrebno korištenje ključnih riječi BEGIN END unutar kojih možemo staviti više naredbi čije je izvršavanje potrebno u slučaju da je uvjet zadovoljen, kao što je i prikazano na slici 2.7.

```
IF DATEPART(dw, GETDATE()) = 7 OR DATEPART(dw, GETDATE()) = 1
   PRINT 'Vikend je.'
ELSE
   PRINT 'Radni je dan.'
```

Sl. 2.6. Kontrola toka programa IF ELSE petljom

```
IF DATEPART(dw, GETDATE()) = 7 OR DATEPART(dw, GETDATE()) = 1
BEGIN
   PRINT 'Vikend je.'
   PRINT 'Vikendom je potrebno odmoriti se!'
END
ELSE
BEGIN
   PRINT 'Radni je dan.'
   PRINT 'Radnim danom se ide na posao!'
END
```

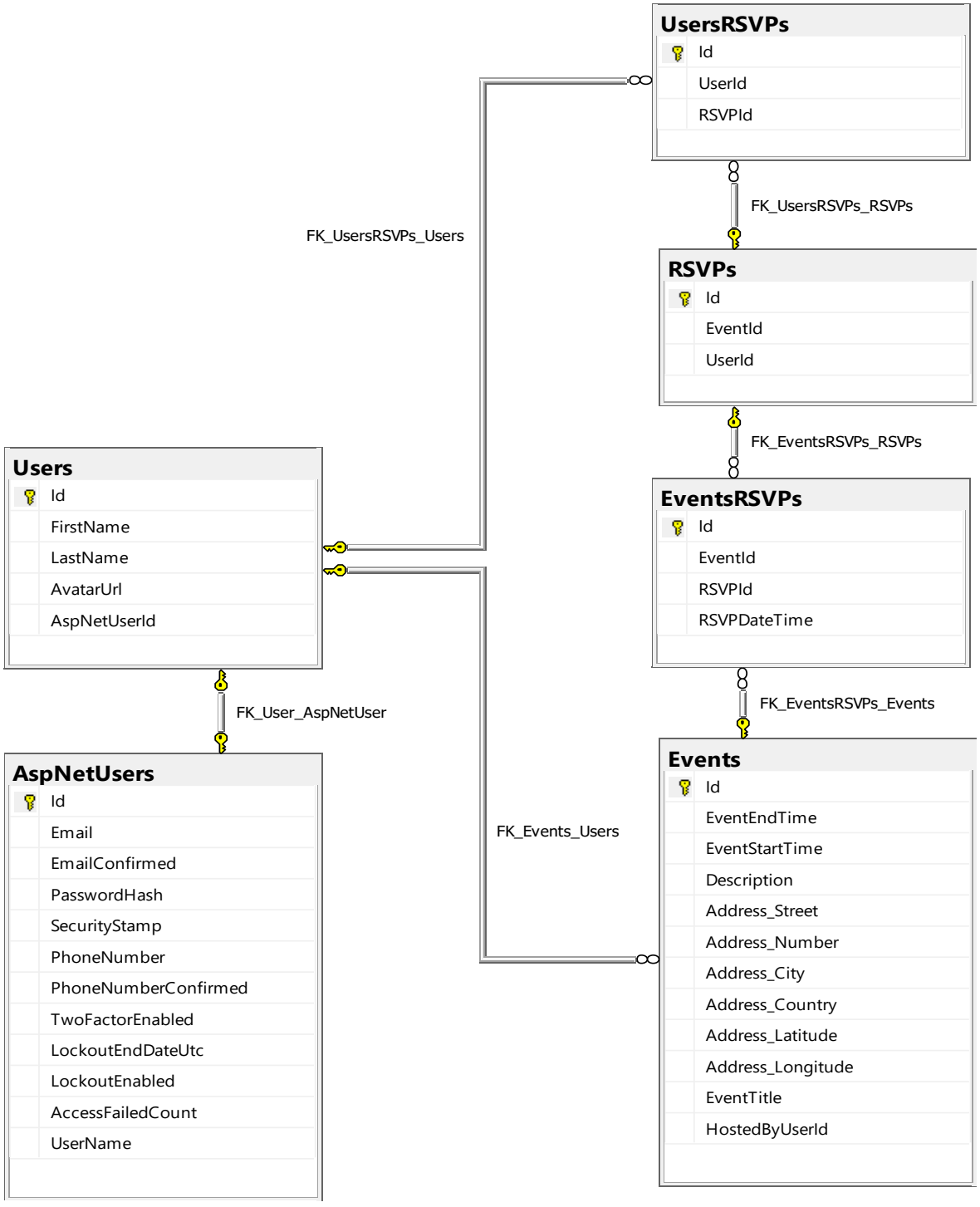
Sl. 2.7. Kontrola toka programa ključnim riječima BEGIN i END

[1] Dan u tjednu je nedjelja ili subota, uz napomenu da je prvi dan u tjednu nedjelja, a subota zadnji.

3. MODEL PODATAKA I ARHITEKTURA SUSTAVA

Ovo poglavlje sadrži pregled modela podataka i opis arhitekture cijelog sustava. Bit će opisan relacijski model baza podataka, te veze između glavnih komponenata sustava.

3.1. Relacijski dijagram baze podataka

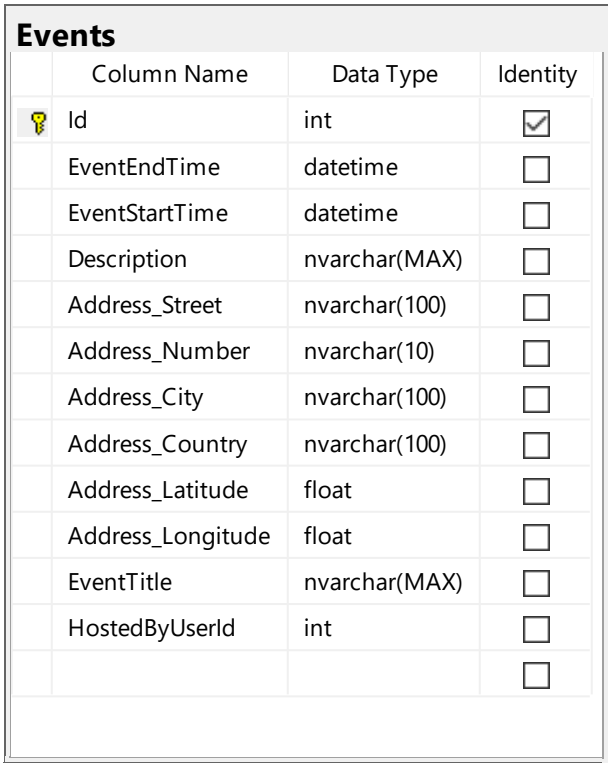


Sl. 3.1. Relacijski model baze podataka

Na slici 3.1. prikazan je relacijski dijagram baze podataka s vezama između entiteta te sa svim atributima pojedinog entiteta. Iz ER dijagrama vidljivo je kako se model podataka sastoji od ukupno šest entiteta. Četiri entiteta predstavljaju podatke o korisnicima, događajima i odazivom na događaje, dok dva entiteta zapravo predstavljaju relaciju *many to many* između entiteta.

3.1.1. Entitet Event

Entitet Event predstavlja događaje koje korisnici mogu kreirati te je opisan atributima prikazanima na slici 3.2.



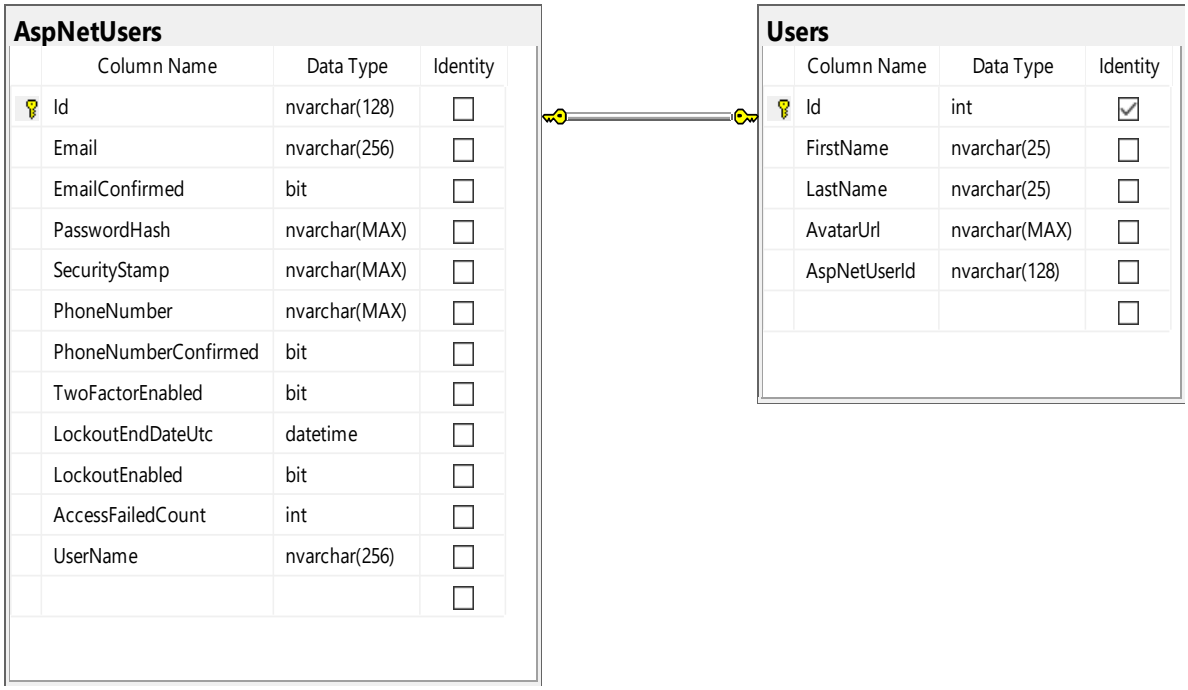
Sl. 3.2. Atributi entiteta Event

Sa slike 3.2. vidljivo je kako entitet Event ima atribute kao što su vrijeme i datum početka, vrijeme i datum svršetka, opis, naziv događaja, adresu događaja.

Primarni ključ ovog entiteta je „Id“, dok je strani ključ atribut „HostedByUserId“.

3.1.2. Entiteti User i AspNetUser

Entiteti User i AspNetUser zajedno definiraju podatke o korisniku, odnosno njegovom korisničkom računu. Entitet AspNetUser sadrži atribute koji opisuju korisničke podatke koji su vezani za njegov korisnički račun, dok entitet User sadrži atribute koji opisuju osobne podatke korisnika.



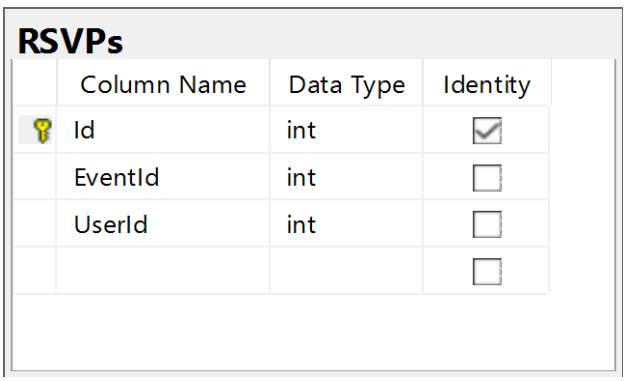
Sl. 3.3. Entitet User

Na slici 3.3. prikazani su atributi entiteta AspNetUser i User. Veza između ova dva entiteta je „jedan na jedan“ što znači da za svaki unos u AspNetUsers tablicu mora postojati i unos u Users tablicu. Entitet User sadrži strani ključ pod nazivom „AspNetUserId“ koji je zapravo veza na atribut „Id“ entiteta AspNetUser.

3.1.3. Entitet RSVP

Ovaj entitet sadrži atribute „EventId“ i „UserId“ koji preko pomoćnih entiteta UserRSVP i EventRSVP predstavljaju veze sa entitetima User i Event .

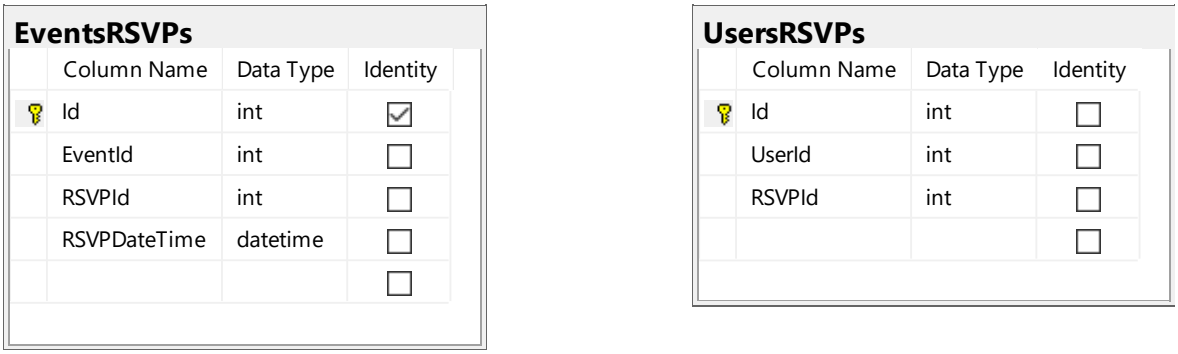
Na slici 3.4. prikazani su atributi entiteta RSVP



Sl. 3.4. Entitet RSVP

3.1.4. Entiteti UserRSVP i EventRSVP

Ova dva entiteta u bazi, zapravo, predstavljaju pomoćne tablice koje pomažu pri povezivanju entiteta User i RSVP, te povezivanju entiteta Event i RSVP. Ovakva praksa spojnih tablica između entiteta koji su vezani vezom *many to many* primjenjuje se kako bi se očuvala čistoća i konzistentnost podataka.



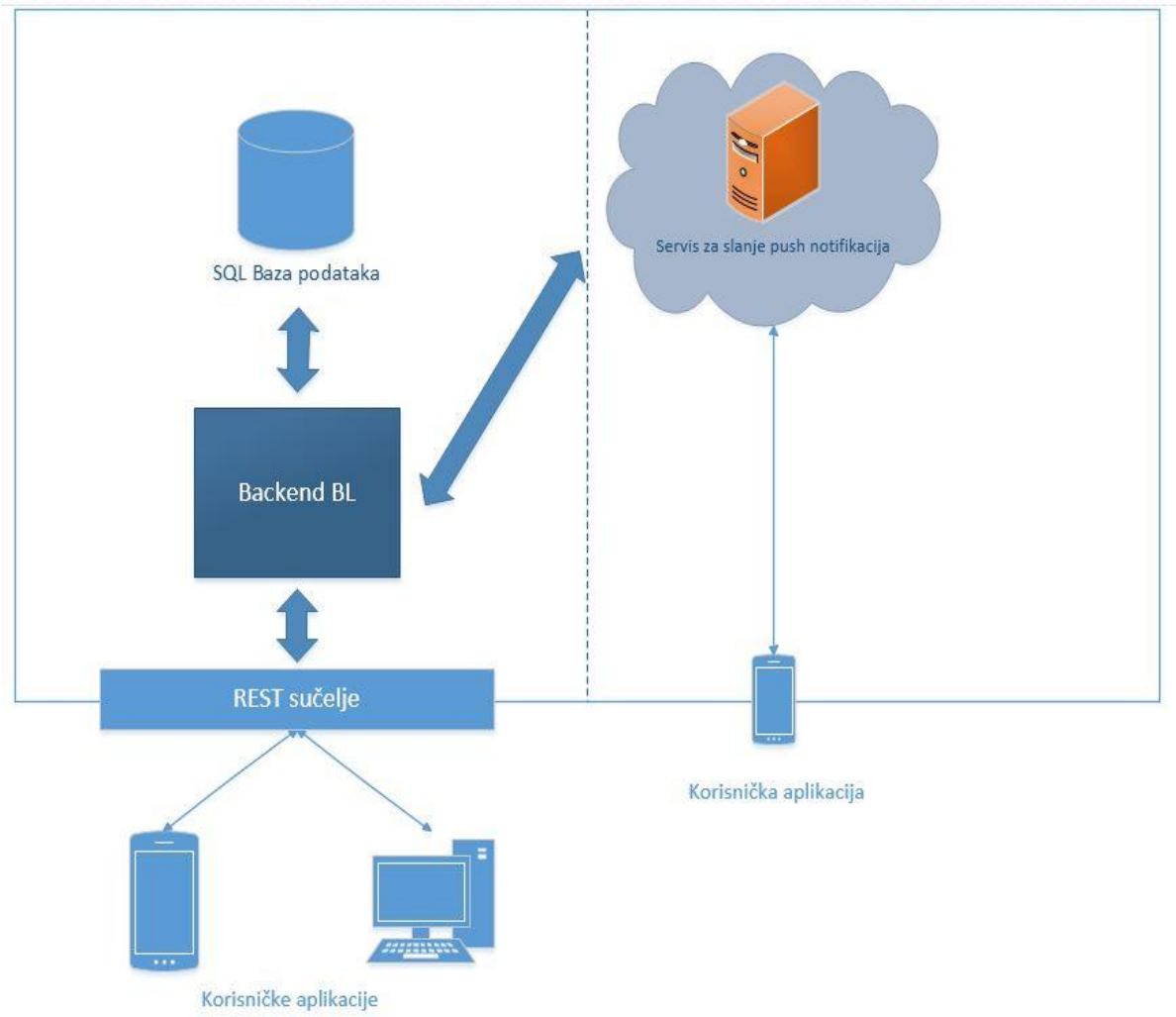
Sl. 3.5. Spojne tablice

3.2. Arhitektura sustava

Osnovne komponente sustava su baza podataka, REST API i mobilna aplikacija koja preko REST API-a komunicira s bazom podataka.

Na slici 3.6. prikazana je arhitektura sustava. Vidljivo je kako mobilna aplikacija ne pristupa bazi podataka direktno, već koristi HTTP metode koje su joj dostupne preko sučelja REST API-a. Ova komunikacija je dvosmjerna. Mobilna aplikacija šalje zahtjev na REST API i od istoga dobije odgovor. REST API podatke šalje u JSON formatu.

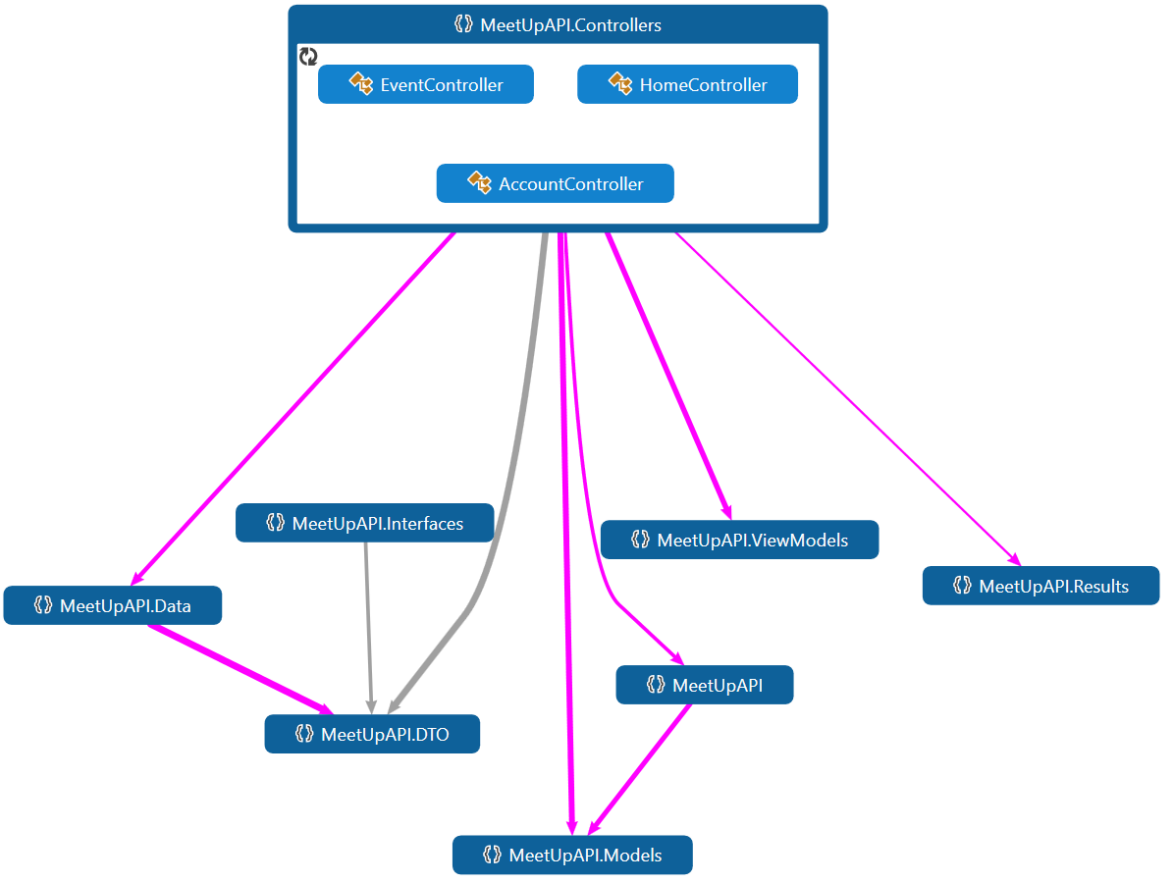
REST API za komunikaciju s bazom koristi pozadinsku logiku Web API-a (eng. Backend Business Layer) koji zapravo vrši sav rad na bazi podataka. On ubacuje nove podatke u bazu, kao što i, prema zahtjevu, vadi podatke iz baze te ih putem REST sučelja šalje klijentu, koji je, u ovom slučaju mobilna aplikacija.



Sl. 3.6. Arhitektura sustava

4. REST API i Backend Business Layer

U prethodnom poglavlju prikazana je arhitektura cijelog sustava sa svim slojevima, od pristupa bazi, do klijenta koji konzumira REST API. U ovom poglavlju bit će detaljnije opisani modeli, DT(eng. *Data transfer*) objekti, logika pozadinskog sloja i kontroleri REST sučelja.



Sl. 4.1. Arhitektura Busines layera

Na slici 4.1. prikazana je arhitektura pozadinskog sloja. Iz slike je vidljivo kako prvi sloj iznad baze predstavljaju modeli. Model je objekt koji predstavlja podatke aplikacije, odnosno klasu koja predstavlja entitet baze podataka sa svim njegovim atributima.

U sljedećim potpoglavljima objasnit će se rad s entitetom „Events“ od dohvata/spremanja u bazu preko sloja „Data“ do slanja klijentu putem kontrolera.

Isti princip korišten je za rad sa svim ostalim entitetima koji se nalaze u bazi podataka.

4.1. Sloj „EventData“

Na slici 4.2. prikazana je klasa „Events“ koja predstavlja entitet „Events“ baze podataka, dok svojstva klase predstavljaju atribute entiteta.

```
public class Events
{
    3 references | Bojan Grubić | 1 change
    public int Id { get; set; } // Id (Primary key)
    2 references | Bojan Grubić | 1 change
    public DateTime EventEndTime { get; set; } // EventEndTime
    2 references | Bojan Grubić | 1 change
    public DateTime EventStartTime { get; set; } // EventStartTime
    1 reference | Bojan Grubić | 1 change
    public string Description { get; set; } // Description
    2 references | Bojan Grubić | 1 change
    public string Address_Street { get; set; } // Address_Street
    2 references | Bojan Grubić | 1 change
    public string Address_Number { get; set; } // Address_Number
    2 references | Bojan Grubić | 1 change
    public string Address_City { get; set; } // Address_City
    2 references | Bojan Grubić | 1 change
    public string Address_Country { get; set; } // Address_Country
    2 references | Bojan Grubić | 1 change
    public double Address_Latitude { get; set; } // Address_Latitude
    2 references | Bojan Grubić | 1 change
    public double Address_Longitude { get; set; } // Address_Longitude
    1 reference | Bojan Grubić | 1 change
    public string EventTitle { get; set; } // EventTitle
    3 references | Bojan Grubić | 1 change
    public int HostedByUserId { get; set; } // HostedByUserId

    // Reverse navigation
    2 references | Bojan Grubić | 1 change
    public virtual ICollection<EventsRSVPs> EventsRSVPs { get; set; } // EventsRSVPs.FK_EventsRSVPs_Events

    // Foreign keys
    4 references | Bojan Grubić | 1 change
    public virtual Users Users { get; set; } // FK_Events_Users

    0 references | Bojan Grubić | 4 changes
    public Events()
    {
        EventsRSVPs = new List<EventsRSVPs>();
    }
}
```

Sl. 4.2. Klasa „Events“

Klasu „Events“ koristi sloj pod nazivom „EventData“. Ovaj sloj obavlja komunikaciju s bazom podataka pomoću storanih procedura. Kako bi se podaci iz baze podataka mogli konzumirati u „višim“ slojevima aplikacije, ovaj sloj obavlja mapiranje entiteta baze u C# klase.

4.1.1. Rad s podacima u bazi podataka

Sav rad s podacima i entitetima baze podataka obavlja se storanim procedurama. Storana procedura je zapravo skup SQL naredbi kojima je dodijeljeno zajedničko ime pod kojim su pohranjene u bazi podataka kako bi se mogle koristiti od strane više aplikacija istovremeno.

Primjer storane procedure koja se koristi za kreiranje novog, odnosno uređivanje postojećeg, prikazan je na slici 4.3.

```
ALTER PROCEDURE [dbo].[UpsertEvent]
    @EventId int
    ,@EventStartTime datetime
    ,@EventEndTime datetime
    ,@Description nvarchar(max)
    ,@Address_Street nvarchar(100)
    ,@Address_Number nvarchar(10)
    ,@Address_City nvarchar(100)
    ,@Address_Country nvarchar(100)
    ,@Address_Latitude float
    ,@Address_Longitude float
    ,@EventTitle nvarchar(max)
    ,@HostedByUserId int
AS
BEGIN
    SET NOCOUNT ON;
    IF(@EventId IS NULL OR @EventId = 0)
    BEGIN
            INSERT INTO dbo.Events(EventStartTime,EventEndTime,Description,Address_Street,Address_Number,
                                   Address_City,Address_Country,Address_Latitude,Address_Longitude,EventTitle,HostedByUserId)
            values
            (
                @EventStartTime
                ,@EventEndTime
                ,@Description
                ,@Address_Street
                ,@Address_Number
                ,@Address_City
                ,@Address_Country
                ,@Address_Latitude
                ,@Address_Longitude
                ,@EventTitle
                ,@HostedByUserId
            );
            --SELEKTIRANJE ZADNJEG UPDATEANOG REDA
            SELECT TOP (1) *
         from dbo.Events as E
         order By E.Id  DESC
    END
    ELSE
        BEGIN
            UPDATE dbo.Events
            SET EventStartTime = @EventStartTime
                ,EventEndTime = @EventEndTime
                ,Description = @Description
                ,Address_Street = @Address_Street
                ,Address_Number = @Address_Number
                ,Address_City = @Address_City
                ,Address_Country = @Address_Country
                ,Address_Latitude = @Address_Latitude
                ,Address_Longitude = @Address_Longitude
                ,EventTitle = @EventTitle
                ,HostedByUserId = @HostedByUserId
            WHERE @EventID = Id

        SELECT TOP (1) *
         from dbo.Events
         where Id = @EventId
    END
END
```

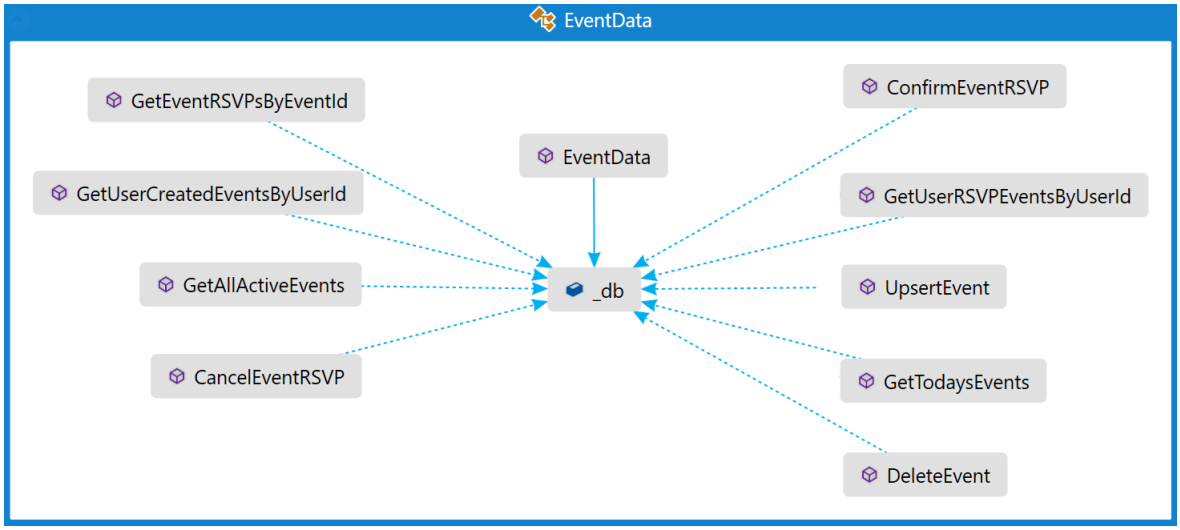
Sl. 4.3. Storana procedura za kreiranje/uređivanje događaja

Kako bi se izbjegla redudantnost, ista procedura koristi se za kreiranje novog kao i za uređivanje postojećeg događaja. To je izvedeno na sljedeći način. Kao što je vidljivo u SQL kodu na slici 4.3., u slučaju da se proceduri preda parametar „EventId“ s vrijednosti nula ili „NULL“, kreirat će se novi događaj sa vrijednostima atributa koji su predani kao parametri

storanoj proceduri, te će se to zabilježiti u tablicu „Events“. U slučaju da se proceduri preda parametar „EventId“ s vrijednošću koja već postoji u tablici „Events“, tada će ostalim atributima događaja sa tim Id-em, biti dodijeljene vrijednosti parametara koji su predani storanoj proceduri.

4.1.2. Metode sloja „EventData“

Na slici 4.4. prikazane su metode sloja „EventData“. Metode direktno pozivaju procedure koje su spremljene u bazi podataka te rade mapiranje entiteta baze u C# klase i obratno.



Sl. 4.4. Metode sloja „EventData“

4.1.2.1. Metoda za dohvat događaja koje je kreirao određeni korisnik

Na slici 4.5. prikazan je kod metode koja služi za dohvaćanje svih događaja koje je kreirao određeni korisnik. Korisnik je definiran svojim Id-em pa se prema tom Id-u radi upit na bazu. Metoda „GetUserCreatedEventsByUserId“ prima parametar „userId“ koji predaje storanoj proceduri koja prima isti taj parametar. U ovoj metodi može se primjetiti kako se rezultat storane procedure mapira u objekt listu objekata klase „EventDto“, to je zbog toga što „EventController“ prima listu tih objekata. O DTO objektima više u sljedećem potpoglavlju.

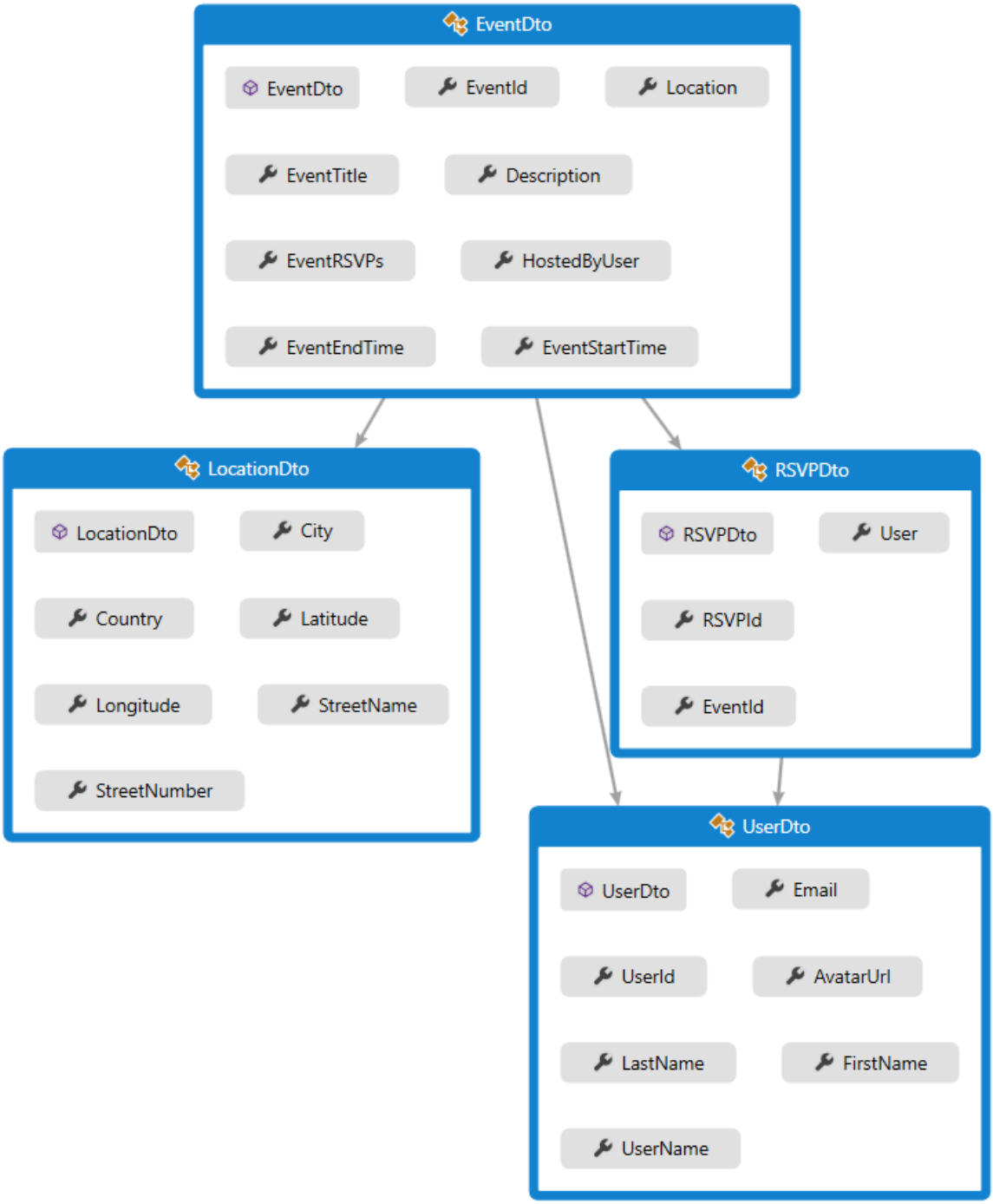
```
public async Task<IEnumerable<EventDto>> GetUserCreatedEventsByUserId(int userId)
{
    var userCreateEvents = this._db.GetUserCreatedEventsByUserId(userId);
    var res = new List<EventDto>
        (userCreateEvents.Select(t => new EventDto
                                    {
                                        Description = t.Description,
                                        HostedByUser = new UserDto
                                                    {
                                                        AvatarUrl = t.AvatarUrl,
                                                        UserName = t.UserName
                                                    },
                                        EventEndTime = t.EventEndTime,
                                        EventId = t.Id,
                                        EventRSVPs = new List<RSVPDto>(),
                                        EventStartTime = t.EventStartTime,
                                        EventTitle = t.EventTitle,
                                        Location = new LocationDto
                                                  {
                                                      City = t.Address_City,
                                                      Country = t.Address_Country,
                                                      Latitude = t.Address_Latitude,
                                                      Longitude = t.Address_Longitude,
                                                      StreetName = t.Address_Street,
                                                      StreetNumber = t.Address_Number
                                                  },
                                    }
                                    ));
    return Mapper.Map<IEnumerable<EventDto>>(userCreateEvents);
}
```

Sl. 4.5. Metoda „GetUserCreatedEventsByUserId“

4.2. Objekti za prijenos podataka (DTO)

DTO (eng. *Data Transfer Object*) ili objekt za prijenos podataka je objekt koji enkapsulira podatke te ih šalje od jednog podsistema ili sloja aplikacije do drugog. DTO-ovi najčešće se koriste u servisnom sloju prilikom prijenosa podataka u više slojeve koji su najčešće korisničko sučelje. U ovom slučaju, viši sloj predstavlja klijentska aplikacija koja dobiva podatke putem REST API HTTP zahtjeva, ti podaci su zapravo DTO-ovi serijalizirani te poslani u JSON formatu.

Na slici 4.6. vidimo sve DTO-ove sustava. Može se primijetiti kako je entitet baze „Events“ podijeljen na dva DTO-a. To su „LocationDTO“ i „EventDTO“, podjela je napravljena radi lakšeg upravljanja podacima i njihovom strukturom prilikom slanja putem REST API-a klijentskoj aplikaciji.



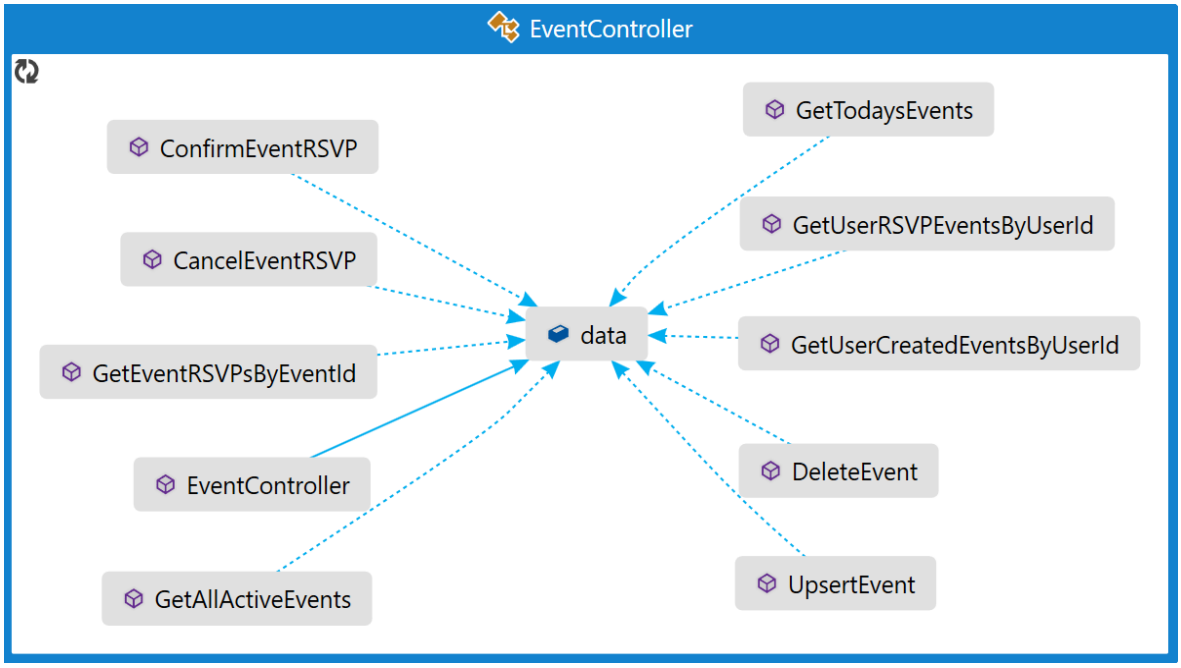
Sl. 4.6. Objekti za prijenos podataka

4.3. Kontroler „EventsController“

Kao i sa slojem za pristup podataka, REST API sloj odnosno princip Web API kontrolera bit će opisan na primjeru jednog kontrolera. Kao primjer uzet je „EventsConntroler“ koji služi za dohvat podataka o događajima iz servisnog sloja aplikacije koji komunicira s bazom. Te iste podatke kontroler serijalizira i pomoću HTTP zahtjeva prosljeđuje klijentu. Stvar

funkcionira i u drugom smjeru, tj. moguće je da kontroler primi podatke od klijentske aplikacije te ih potom proslijedi servisnom sloju koji ih pohranjuje u bazu.

Slika 4.7. prikazuje metode kontrolera zaduženog za rad s događajima.



Sl. 4.7. Sve metode kontrolera „EventsConntroler“

Na slici 4.8. možemo vidjeti kod metode „GetUserCreatedEventsByUserId“ do koje se iz klijentske aplikacije dolazi putem URI-a. Metoda vraća rezultat u JSON formatu. Metode kontrolera mogu imati atribute koji se postavljaju direktno iznad imena metode unutar četvrtastih zagrada. Ova metoda ima dva atributa, a to su: „Authorize“ i „HttpGet“. Atribut metode „Authorize“ koristi se kako bi se metodi „reklo“ da klijent mora biti autoriziran kako bi smio pristupiti traženim podacima. „HttpGet“ definira da je metoda dohvatljiva samo korištenjem ključne riječi „GET“ prilikom sastavljanja HTTP zahtjeva. Moguće je staviti i atribut „HttpPost“ ili neki drugi atribut koji definira kojom se ključnom riječi pristupa metodi putem HTTP zahtjeva. Autorizacija će biti objašnjena u poglavlju koje se bavi klijentskom aplikacijom.

```
[Authorize]
[HttpGet]
0 references | Bojan Grubić | 5 changes
public async Task<IEnumerable<EventDto>> GetUserCreatedEventsByUserId(int userId)
{
    var userCreatedEvents = await data.GetUserCreatedEventsByUserId(userId);

    return AutoMapper.Mapper.Map<IEnumerable<EventDto>>(userCreatedEvents);

}
```

Sl. 4.8. Metoda kontrolera „GetUserCreatedEventsByUserId“

```
[Authorize]
[HttpPost]
0 references | Bojan Grubić | 4 changes
public async Task<bool> ConfirmEventRSVP(int userId, [FromBody]int eventId)
{
    var res = await data.ConfirmEventRSVP(userId, eventId);
    return res;
}
```

Sl. 4.9. Metoda kontrolera „ConfirmEventRSVP“

4.4. Push servis

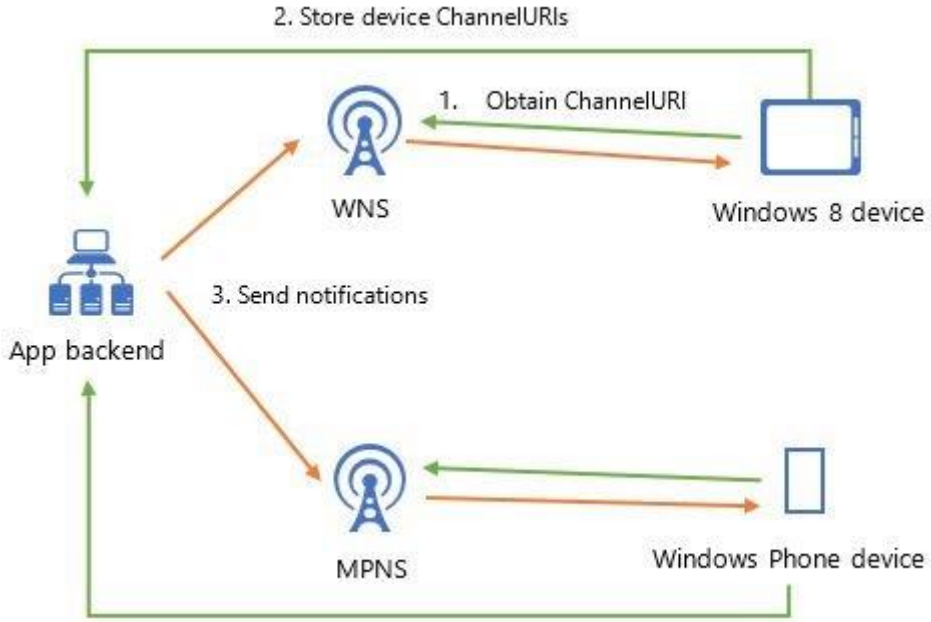
Push notifikacije su vitalni dio svake aplikacije koja se koristi za neki oblik interakcije i komunikacije između korisnika. U ovom projektu za slanje push notifikacija koristi se Windows Azure Push Notification Hub.

Nakon što se aplikacija registrira na Nofitication Hub, slanje push notifikacija kroz web API odvija se pomoću naredbe koja je pokazana na slici 4.10.

```
public void SendPush()
{
    var toast = @"<toast><visual><binding template=""ToastText01""><text id=""1"">Hello everybody!</text></binding></visual></toast>";
    await hub.SendWindowsNativeNotificationAsync(toast);
}
```

Sl. 4.10. Slanje push notifikacije

Na slici 4.11. prikazan je životni ciklus push notifikacije. Iz slike je vidljivo da je za svaku platformu koja je pretplaćena na notifikacije, odnosno za svaku platformu na koju se želi poslati push notifikaciju, potrebno koristit servis koji je specifičan za svaku od platformi. Tako primjerice, za Windows store aplikacije korisit se Windows Notification Service (WNS); Microsoft Push Notification Service (MPNS) za slanje push notifikacja na Windows phone; Apple Push Notification Service za iOS i Google Cloud Messaging (GCM) za Android.

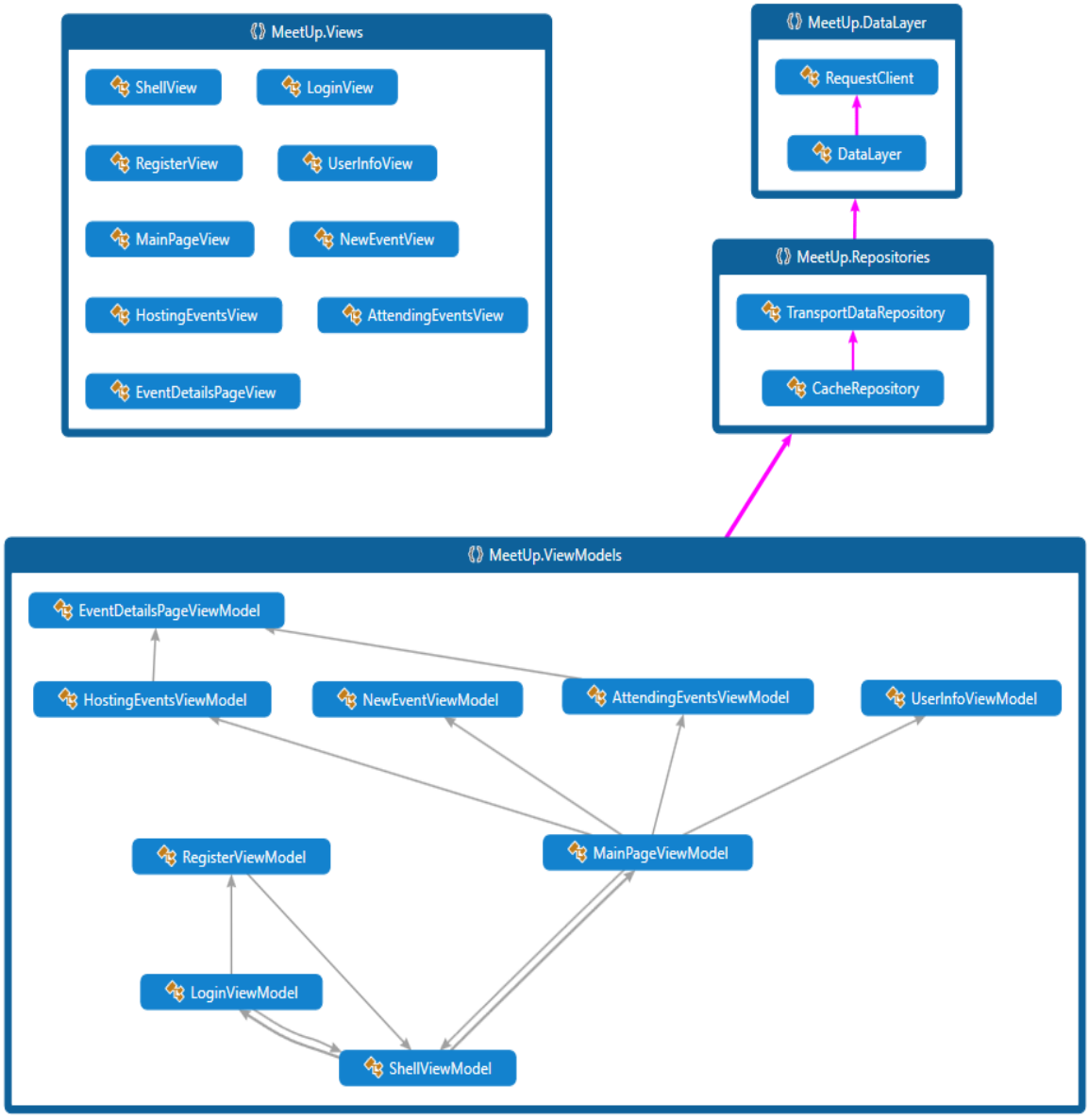


Sl. 4.11. Životni ciklus push notifikacije

5. MOBILNA APLIKACIJA

U zadnjem poglavlju opisana je klijentska aplikacija koju predstavlja Windows phone 8.1 aplikacija. Objašnjena je arhitektura i koncepti korišteni prilikom rada, te je opisan rad s aplikacijom.

5.1. Arhitektura aplikacije



Sl. 5.1. Arhitektura mobilne aplikacije

Kao što je vidljivo na sa slike 5.1., koja predstavlja arhitekturu aplikacije, mobilna aplikacija podijeljena je na slojeve. Za komunikaciju sa servisom aplikacija koristi sloj pod nazivom „DataLayer“. „DataLayer“ je sloj koji je zadužen za komunikaciju sa servisom. U

osnovi, sastoji se od klase „RequestClient“ koja predstavlja univerzalnu metodu za slanje, primanje i obradu svih vrsta HTTP zahtjeva prema REST API-u.

Za keširanje podataka i neke pomoćne radnje zadužen je sloj pod imenom „Repositories“. Ovaj sloj zadužen je za privremenu pohranu i rad s podacima. Podaci se pohranjuju samo za vrijeme dok je aplikacija pokrenuta kako bi u svako vrijeme određeni podaci, koji su nužni za rad aplikacije, bili dostupni, te kako bi se izbjegli nepotrebni pozivi prema API-u. Na ovaj način štedimo resurse, ali i ubrzavamo rad aplikacije, jer nije potrebno čekati podatke sa servera.

Sloj View modela zadužen je za svu pozadinsku logiku korisničkog sučelja. Ovaj sloj također brine se za pripremu podataka koji se prikazuju na korisničkom sučelju, kao i za obradu korisničkih akcija. Zapravo je on veza između korisničkog sučelja i sloja „Repositories“, odnosno, neizravno, samog REST API-a.

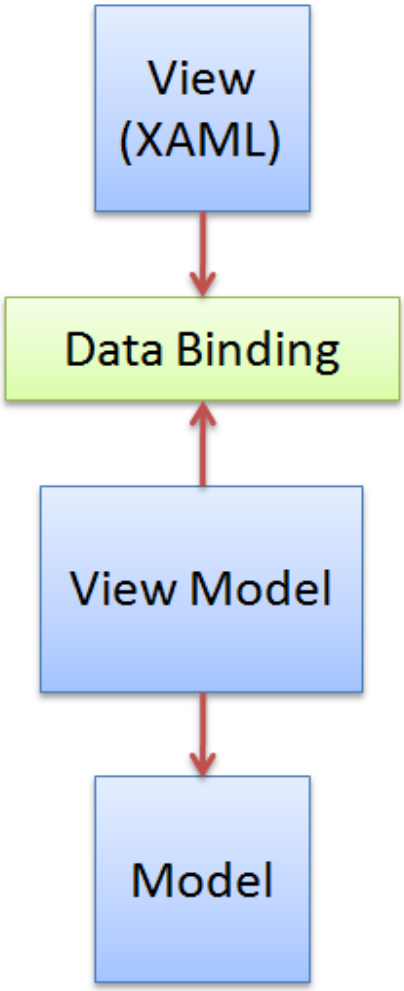
Može se primijetiti kako sloj korisničkog sučelja odnosno „Views“ nema nikakvu izravnu referencu niti na jedan od ostalih slojeva. To je stoga što je glavni koncept arhitekture aplikacije MVVM pristup.

5.1.1. MVVM

Model-View-ViewModel (MVVM) koncept je zapravo Windows phone inačica MVC-a (Model-View-Controller). MVVM predstavlja raslojavanje developmenta od grafičkog sučelja, odnosno poslovnu logiku aplikacije od dizajana. View model predstavlja pretvarač vrijednosti (eng. *value converter*). To znači da je View model odgovoran za konverziju objekata modela na način da ih je moguće jednostavno prikazati na korisničkom sučelju. View model je više model nego View i obrađuje sve pozadinske događaje View-a.

MVVM nije *framework* već koncept programiranje i može biti dio *frameworka*, ali je samo dio ukupnog rješenja aplikacije. Ne zanima ga, niti mu je bitno, što se događa na serveru, niti kako aplikacija komunicira sa serverom.

Na slici 5.2. prikazana je arhitektura MVVM koncepta, čije komponente su objašnjene u daljnjem tekstu.



Sl. 5.2. MVVM arhitektura

5.1.1.1. Model

Model predstavlja stvarne podatke i/ili informacije s kojima radimo. Primjer modela koji je prikazan na slici 5.3. predstavlja čest slučaj kada moramo definirati neki entitet koji predstavlja osobu ili kontakt sa svim njegovim atributima (ime, prezime, adresa, broj telefona i sl.).

Ključno za upamtiti je da model sadrži informacije, ali ne servise i metode za manipulaciju informacijama. Model nije odgovoran za formatiranje teksta kako bi izgledao ljepše na ekranu, niti je odgovoran za dohvat listi određenih objekata sa servera. Poslovna logika drži se odvojeno od modela i enkapsulira se u drugim klasama koje rade s modelom. Postoje iznimke kada model radi neku logiku, jedan od primjera je validacija podataka.

```
namespace MVVMExample
{
    public class ContactModel : INotifyPropertyChanged
    {
        private string _firstName;

        public string FirstName
        {
            get { return _firstName; }
            set
            {
                _firstName = value;
                RaisePropertyChanged("FirstName");
                RaisePropertyChanged("FullName");
            }
        }

        private string _lastName;

        public string LastName
        {
            get { return _lastName; }
            set
            {
                _lastName = value;
                RaisePropertyChanged("LastName");
                RaisePropertyChanged("FullName");
            }
        }

        public string FullName
        {
            get { return string.Format("{0} {1}", FirstName, LastName); }
        }

        private string _phoneNumber;

        public string PhoneNumber
        {
            get { return _phoneNumber; }
            set
            {
                _phoneNumber = value;
                RaisePropertyChanged("PhoneNumber");
            }
        }

        protected void RaisePropertyChanged(string propertyName)
        {
            PropertyChangedEventHandler handler = PropertyChanged;
            if (handler != null)
            {
                handler(this, new PropertyChangedEventArgs(propertyName));
            }
        }

        public event PropertyChangedEventHandler PropertyChanged;

        public override bool Equals(object obj)
        {
            return obj is ContactModel && ((ContactModel) obj).FullName.Equals(FullName);
        }

        public override int GetHashCode()
        {
            return FullName.GetHashCode();
        }
    }
}
```

Sl. 5.3. Primjer modela

5.1.1.2. View

View je nešto što nam je poznato i ono s čime korisnik zapravo ostvaruje interakciju prilikom korištenja aplikacije. View je zapravo prezentacija, odnosno prikaz podataka. View uljepšava prikaz atributa objekata koji su definirani modelom. View može korisniku omogućiti i unos podataka, kako bi mijenjao atribute modela.

U MVVM-u, View je aktivna komponenta, što znači da nije svjestan nekih događaja koje njegov View model mapira s raznim objektima, metodama i naredbama, ali postoje neki događaji i akcije za koje je odgovoran sam View.

Slika 5.4. predstavlja primjer View-a koji odgovara modelu prikazanom na slici 5.3.

```
<UserControl x:Class="MVVMExample.DetailView"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
    <Grid x:Name="LayoutRoot" Background="White"
                DataContext="{Binding CurrentContact}">
        <Grid.RowDefinitions>
            <RowDefinition/>
            <RowDefinition/>
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition/>
            <ColumnDefinition/>
        </Grid.ColumnDefinitions>
        <TextBlock Text="Name:" HorizontalAlignment="Right" Margin="5"/>
        <TextBlock Text="{Binding FullName}"
               HorizontalAlignment="Left" Margin="5" Grid.Column="1"/>
        <TextBlock Text="Phone:" HorizontalAlignment="Right"
               Margin="5" Grid.Row="1"/>
        <TextBlock Text="{Binding PhoneNumber}"
               HorizontalAlignment="Left" Margin="5"
               Grid.Row="1" Grid.Column="1"/>
    </Grid>
</UserControl>
```

Sl. 5.4. Primjer View-a

5.1.1.3. View model

View model je ključan dio ovog trija jer uvodi odvajanje prezentacijskog sloja (eng. *Presentation Separation*), ili koncept kojim se odvajaju radnje i prikaz view-a od modela. Umjesto da model čini svjesnim korisničkog sučelja, view samo prikazuje podatke formatirane za prikaz, view model čini sponu između view-a i modela. View model može

uzeti unos s korisničkog sučelja i postaviti novu vrijednost modela, ili može ostvariti komunikaciju s web servisom i dohvatiti model, mapirati tribute i postaviti ih na view.

View model također pruža i metode, komande i ostale funkcionalnosti koje pomažu u očuvanju stanja view-a, omogućuje manipulaciju modela kao rezultat korisničke akcije i dr.

View model prikazan na slici 5.5. je napravljen kako bi radio s listom kontakata. Također omogućuje naredbe za brisanje i zastavicu koja definira da li je brisanje trenutno omogućeno.

```
namespace MVVMExample
{
    public class ContactViewModel : BaseINPC
    {
        public ContactViewModel()
        {
            Contacts = new ObservableCollection<ContactModel>();
            Service = new Service();

            Service.GetContacts(_PopulateContacts);

            Delete = new DeleteCommand(
                Service,
                ()=>CanDelete,
                contact =>
                    {
                        CurrentContact = null;
                        Service.GetContacts(_PopulateContacts);
                    });
        }

        private void _PopulateContacts(IEnumerable>ContactModel> contacts)
        {
            Contacts.Clear();
            foreach(var contact in contacts)
            {
                Contacts.Add(contact);
            }
        }

        public IService Service { get; set; }

        public bool CanDelete
        {
            get { return _currentContact != null; }
        }

        public ObservableCollection<ContactModel> Contacts { get; set; }

        public DeleteCommand Delete { get; set; }

        private ContactModel _currentContact;

        public ContactModel CurrentContact
        {
            get { return _currentContact; }
            set
            {
                _currentContact = value;
                RaisePropertyChanged("CurrentContact");
                RaisePropertyChanged("CanDelete");
                Delete.RaiseCanExecuteChanged();
            }
        }
    }
}
```

Sl. 5.5. Primjer View Modela

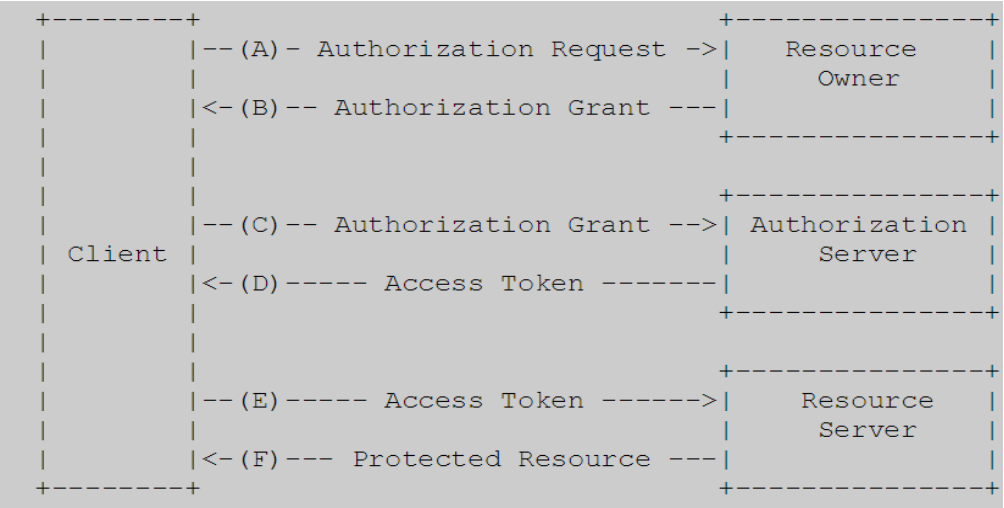
5.2. Autorizacija i sigurnost

Za dohvat podataka sa servisa koriste se HTTPS protokol i SSL veza. Autentikacija korisnika napravljena je pomoću OAuth 2.0 Authorization Framework-a.

OAuth klijentskoj aplikaciji omogućuje pristup zaštićenim resursima korištenjem pristupnog tokena. Token je *string* koji predstavlja „pristupnu dozvolu“ koja je izdana klijentskoj aplikaciji [3]. Tokene izdaje autorizacijski server uz dozvolu vlasnika resursa. Klijent koristi pristupni token kako bi pristupio zaštićenim resursima koje čuva server koji je vlasnik tih resursa.

5.2.1. Bearer token

„Bearer token“ predstavlja sigurnosni token. Njegovo svojstvo je to što bilo koji klijent koji posjeduje ovaj token može pristupiti resursima na koje je tom tokenu pravo pristupa dao „server vlasnik resursa“. Donosilac tokena ne mora dokazati da posjeduje kriptografski ključ za koji je token izdan.



Sl. 5.6. Tijek protokola OAuth 2.0

Slika 5.3. prikazuje interakciju između klijenta, vlasnika resursa autorizacijskog servera i servera koji drži resurse.

5.3. Zaštita od rušenja aplikacije

Kako bi se aplikacija zaštitila od rušenja, obrađene su neke iznimne situacije do kojih bi moglo doći tokom rada aplikacije.

Ako prilikom prvog pokretanja aplikacije nije omogućen pristup internetu ili je servis, sa kojeg se dohvaćaju podaci, nedostupan, sustav će obavijestiti korisnika o pogrešci. Korisnik može pritiskom na *refresh* dugme pokušati ponovno dohvaćanje podataka.

Ako servis postane nedostupan tokom korištenja aplikacije, ili se prekine veza s internetom, aplikacija će koristiti podatke koji su već dohvaćeni i pohranjeni *cache,* a pri pokušaju prijave ili odjave na event, sustav će obavijestiti korisnika da nije moguć pristup servisu te se prebaciti na početnu stranicu aplikacije, s koje je moguće pokušati osvježiti podatke ili nastaviti koristiti one koji su već pohranjeni u *cache*.

Prilikom pokušaja prijave/odjave s eventa sustav provjerava da li je korisnik već prijavljen/odjavljen, te mu sukladno tome pruža opciju prijave/odjave.

5.4. Push notifikacije

Aplikacija ima mogućnost primanja i slanja obavijesti putem *push* notifikacija. Korištenjem *pusha* primaju se obavijesti o aktivnostima drugih korisnika(prijava i odjava s eventa), te druge razne obavijesti sa servisa.

Za implementaciju *push* notifikacija korišten je Microsoft Azure Notification Hub.

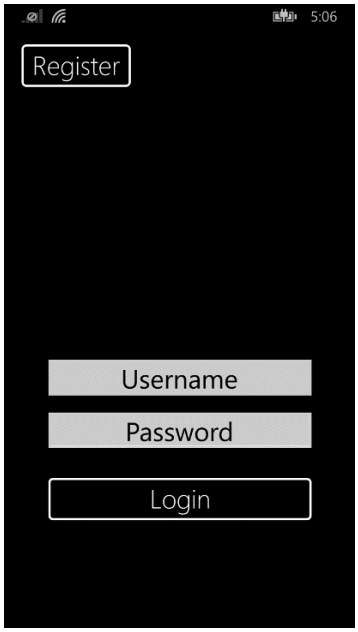
5.5. Testiranje

Aplikacija je testirana na uređaju HTC 8X koji koristi operativni sustav Windows Phone 8.1. Rezolucija ekrana uređaja je 1280x720

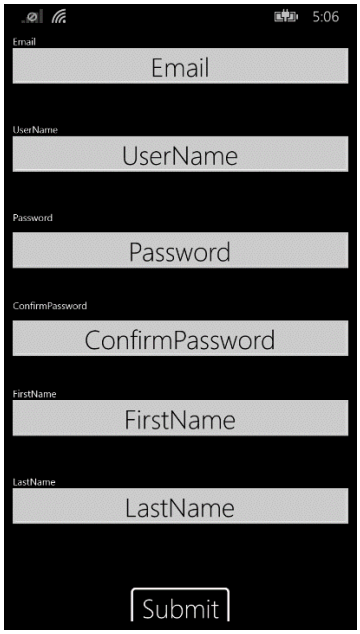
Također testirana je i na uređaju Nokia Lumia 630 koja ima operativni sustav Windows mobile (Windows 10). rezolucija ekrana je 800x480

5.6. Upute za korištenje aplikacije

I. Prilikom prvog pokretanja prikazuje se forma za prijavu, na kojoj postoji i dugme za registraciju (Sl. 5.7.), koje vodi do forme za registraciju (Sl. 5.8.), u slučaju da korisnik nije registriran.

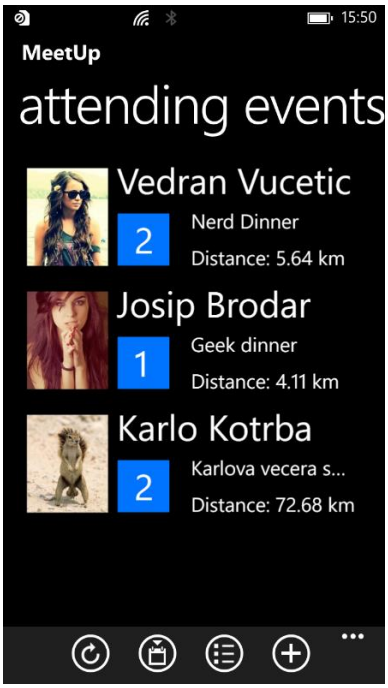


Sl. 5.7. Forma za Login

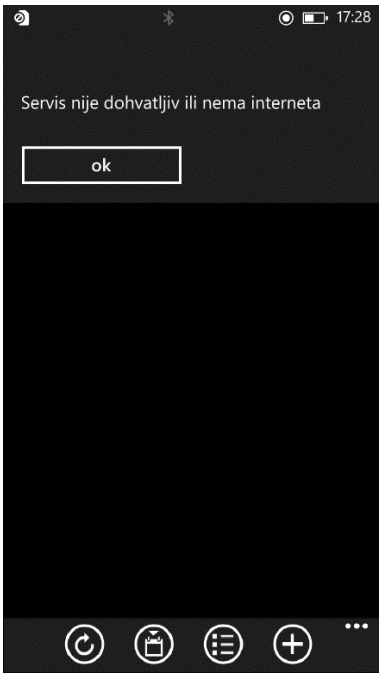


Sl. 5.8. Forma za registraciju

II. Nakon uspješne prijave u aplikaciju prikazuje se popis događaja koji se dohvati sa servera, a ako nije omogućen pristup internetu ili servis nije dohvatljiv, ispisuje se poruka o pogrešci.



Sl. 5.9. Attending events

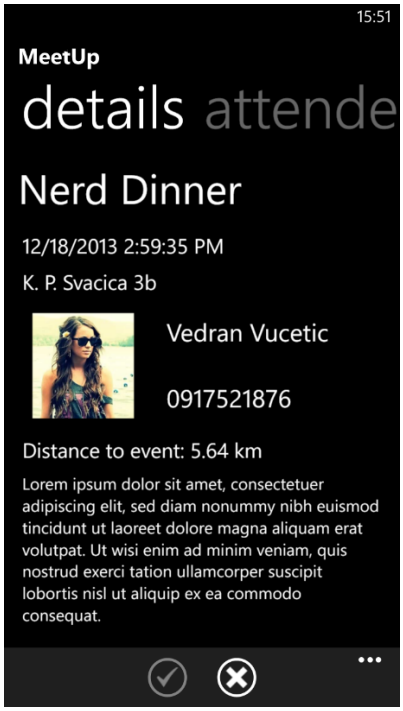


Sl. 5.10. Servis nije dostupan ili nije moguće uspostaviti vezu

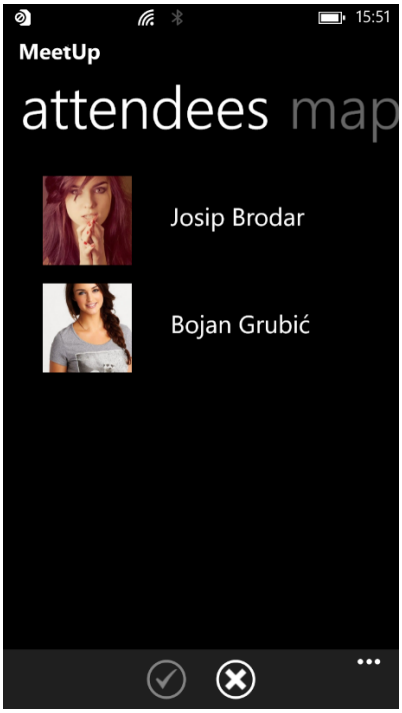
III. Za prikaz lokacija svih evenata potrebno odabrati tab „map“, te se dobije prikaz prikazan na slici 5.11.

Sl. 5.11. Prikaz događaja na karti

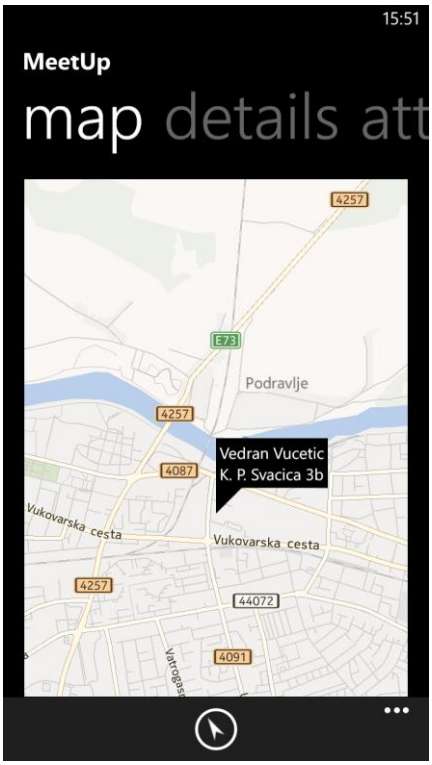
IV. Ako korisnik želi pregledati detalje za određeni događaj, potrebno je samo odabrati željeni sa liste, te je moguće pregledavati razne detalje koji su prikazani na slikama 5.12. – 5.14.



Sl. 5.12. Prikaz detalja o događaju

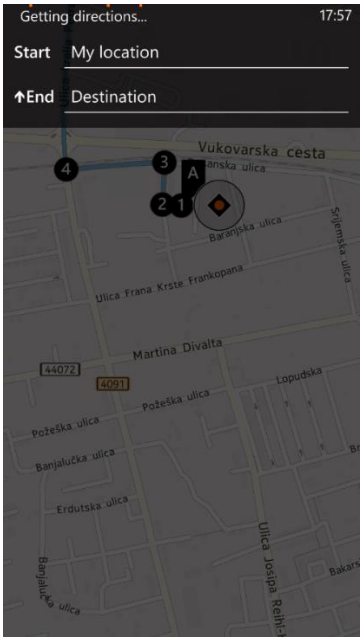


Sl. 5.13. Prikaz korisnika koji su potvrdili dolazak na događaj

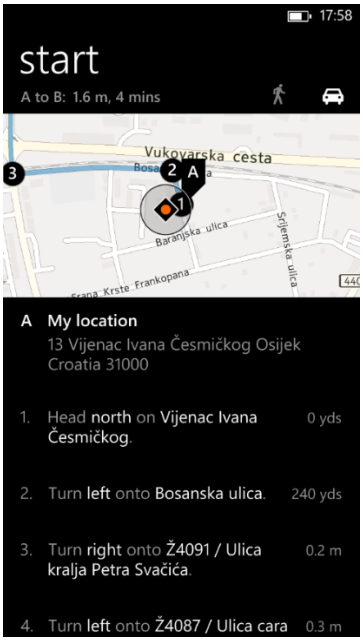


Sl. 5.14. Lokacija događaja

V. Za navigaciju od trenutne lokacije do lokacije događaja potrebno je pritisnuti dugme *navigate*. Nakon toga sustav pokreće navigaciju kao što je prikazano na slikama 5.15. i 5.16.

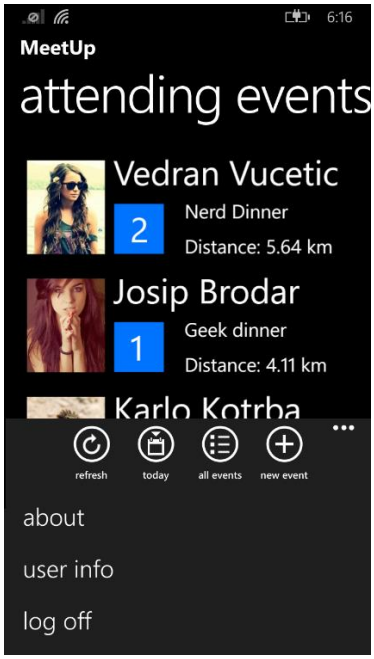


Sl. 5.15. Navigacija do događaja (1)



Sl. 5.16. Navigacija do događaja (2)

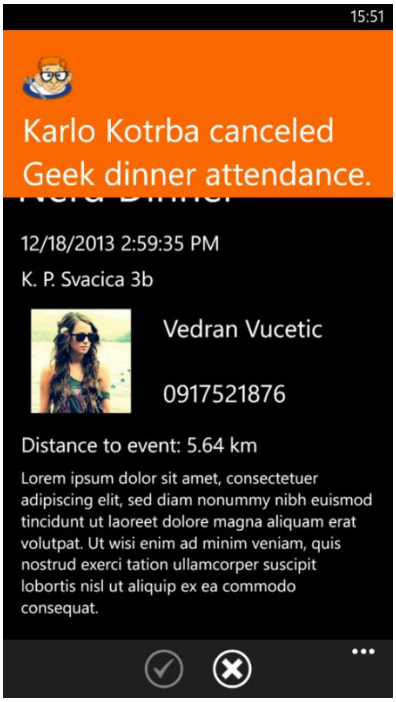
VI. Ako se s glavne stranice odaberu opcije i potom „about“ dobit ćemo prikaz „about“ stranice koja sadrži neke podatke o aplikaciji i proizvođaču.



Sl. 5.17. About (1)

Sl. 5.18. About (2)

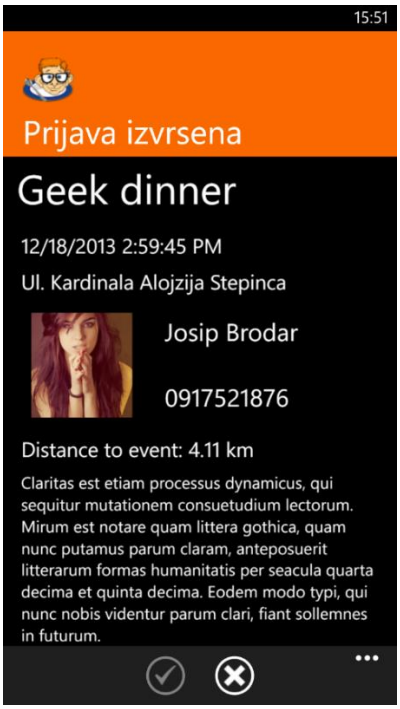
VII. U slučaju dolaska push notifikacije prikazat će se *toast* poruka koja nestane nakon nekoliko sekundi. Različite push notifikacije prikazane su na slikama 5.19. i 5.20.



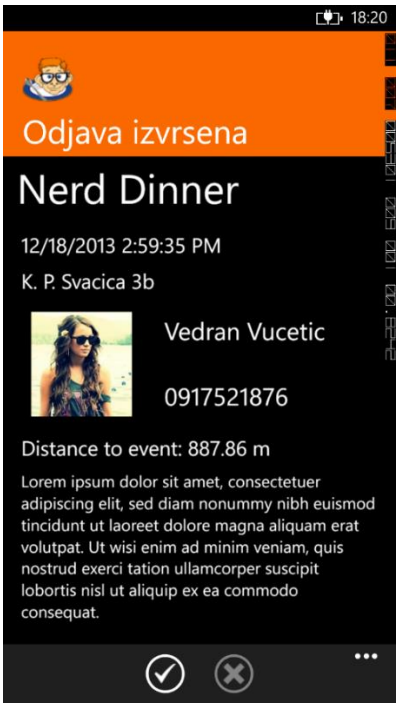
Sl. 5.19. Cancel push

Sl. 5.20. Confirm push

VIII. U slučaju uspješne prijave i odjave korisnik je obaviješten o radnji *toast* porukom

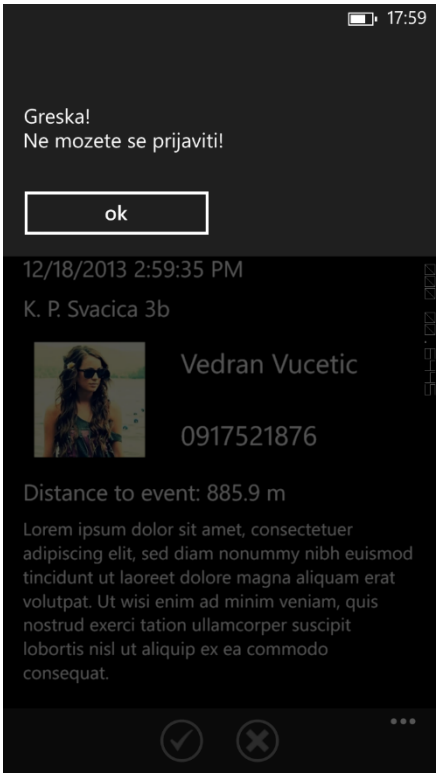


Sl. 5.21. Uspješna prijava

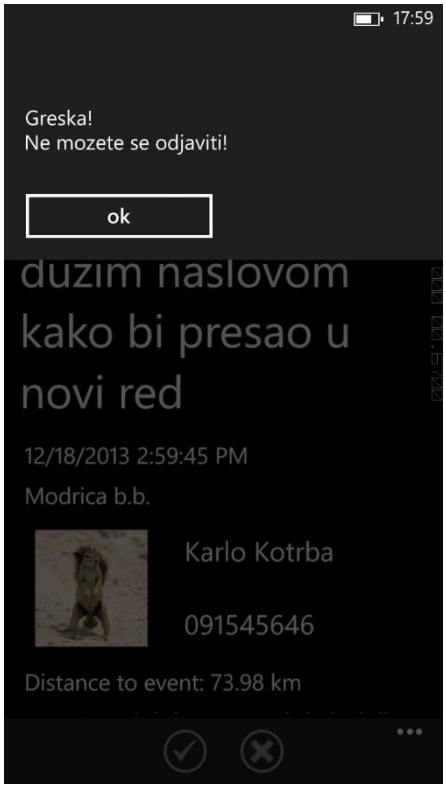


Sl. 5.22. Uspješna odjava

IX. U slučaju da nije moguće obaviti prijavu/odjavu korisnik dobija obavijesti kakve su prikazane na slikama 5.23. i 5.24.



Sl. 5.23. Neuspješna prijava dolaska na događaj



Sl. 5.24. Neuspješno otkazivanje dolaska na događaj

5.7. Moguće nadogradnje

Kako bi ova aplikacija imala širu uporabu potrebne su još neke modifikacije i dodaci. Prva stvar koju bi trebalo promijeniti je redizajn korisničkog sučelja i toka aplikacije kako bi se korisnicima pružilo što ugodnije iskustvo prilikom rada s aplikacijom.

Druga stvar koja bi poboljšala uporabljivost aplikacije je integracija „Friends“ modula. Taj modul bi korisnicima omogućavao da stvore listu prijatelja te da samo s njima dijele svoje događaje, osobne informacije, te da na taj imaju mogućnost i pozivanja samo određenog broja ljudi na svoje događaje.

6. ZAKLJUČAK

U radu se bavimo izradom Windows phone mobilne aplikacije i API-a pomoću kojeg aplikacija komunicira s bazom podataka i ostalim komponentama sustava.

Za izradu pojedinih komponenti sustava korištene su razne Microsoft tehnologije. Tako se za izradu baze koristio T-SQL i Microsoft SQL Server Management studio, dok se za izradu mobilne aplikacije i REST API-a koristio Visual studio 2013 i programski jezik C#, korisničko sučelje mobilne aplikacije izrađeno je u XAML-u.

Prvi i najteži zadatak bio je izrada modela podataka, entiteta, definiranje atributa svakog od entiteta te kreiranje veza između entiteta. Nakon što je taj temeljni sloj dovršen, prelazi se na izradu API-a.

Prilikom izrade rada dosta pažnje posvećeno je arhitekturi i raslojavanju komponenata cijelog sustava. Raslojavanje je bitno zbog modularnosti samog sustava. Ako je API dobro raslojen, prema potrebi, moguće je zamijeniti neku od komponenti, ili čak samu bazu podataka bez da sloj s kojim komuniciraju klijentske aplikacije primijeti ikakvu promjenu ili zahtijeva ikakve modifikacije.

Još jedna stvar na koju je potrebno obratit pažnju je sloj kontrolera API-a, jer s tim slojem putem HTTP metoda komuniciraju klijentske aplikacije. Bitno je da taj sloj služi samo za prezentaciju podataka u obliku koji mu je dostavljen od nižih slojeva, ne smije sadržavati nikakvu logiku. To je nužno kako bi ga mogle konzumirati razne aplikacije. U ovom konkretnom slučaju konzument je samo Windows phone aplikacija, ali bilo koja mobilna, desktop ili web aplikacija koja zna raditi s HTTP zahtjevima može konzumirati ovaj API.

Arhitektura mobilne aplikacije također je raslojena i to na tri glavna sloja. To je bitno kako bi se logika aplikacije odvojila od prezentacijskog sloja, koji je u ovom slučaju korisničko sučelje. S raslojavanjem aplikacije dobivena je mogućnost promjene korisničkog sučelja bez promjene poslovne logike aplikacije.

Prilikom izrade ovog rada stekao sam neka nova iskustva i vještine koje će mi koristit u daljnjem stručnom usavršavanju i radu. Smatram da je za širu primjenu, kako mobilne aplikacije tako i cijelog sustava, potrebno još dosta rada, ponajviše u dizajnu korisničkog sučelja mobilne aplikacije, ali i u dodavanju novih te proširenju postojećih funkcionalnosti.

7. Literatura

[1] https://msdn.microsoft.com/en-us/library/windows/apps/gg521153(v=vs.105).aspx

[2] http://www.codeproject.com/Articles/100175/Model-View-ViewModel-MVVM-Explained

[3] http://self-issued.info/docs/draft-ietf-oauth-v2-bearer.html#anchor1

[4] Garland, John : „Windows Store Apps Succinctly“, 2013

[5] Mayo, Joe : „C# Succinctly“, 2015

[6] Vaughan, Daniel : „Windows phone 8 unleashed“, 2013

[7] http://www.asp.net/web-api/overview/security/external-authentication-services#MICROSOFT

[8] http://www.benday.com/2014/02/25/walkthrough-asp-net-mvc-identity-with-microsoft-account-authentication/

[9] https://msdn.microsoft.com/en-us/library/ms130214.aspx

[10] http://aspnetboilerplate.com/Pages/Documents/NLayer-Architecture

[11] http://weblogs.asp.net/sukumarraju/creating-n-tier-web-api-application

[12] http://www.asp.net/web-api/overview/getting-started-with-aspnet-web-api/tutorial-your-first-web-api

[13] https://blogs.msdn.microsoft.com/martinkearn/2015/01/05/introduction-to-rest-and-net-web-api/

[14] https://docs.asp.net/en/latest/tutorials/first-web-api.html

[15] https://blogs.windows.com/buildingapps/2013/09/16/delivering-push-notifications-to-millions-of-devices-with-windows-azure-notification-hubs/#rhBlmZMdtMTw3jCo.97

8. Sažetak

U radu je obrađena problematika izrade Windows phone mobilne aplikacije, te API-a pomoću kojeg mobilna aplikacija komunicira s bazom podataka i ostalim komponentama sustava. Kroz mobilnu aplikaciju korisnik, nakon što se registrira, ima mogućnost kreiranja novih događaja, pozivanja prijatelja, potvrđivanje dolaska na događaje drugih korisnika i slično. Mobilna aplikacija komunicira s bazom podataka posredstvom REST API-a.

API je arhitekturalno raslojen kako bi se pojedinačno njegovi slojevi mogli mijenjati i adaptirati bez utjecaja na ostale slojeve. Arhitektura mobilne aplikacije također je podijeljena u slojeve, kako bi se odvojila izrada korisničkog sučelja od programske logike.

Mobilna aplikacija i API rade očekivano i bez problema.

Ključne riječi: Windows phone, REST API, MVVM, XAML, C#, Web API, planiranje događaja.

„Mobile event planer“

Summary

This paper explains how to create Windows phone application and Web API which is used for mobile application to communicate with database and other system components.

After registration, using mobile application, user can create new events, invite friends to that events, confirm or decline event invitation, etc.

Mobile application uses REST API for communication with database. API has layered architecture, so that his layers could be changed and edited without influencing other layers. Mobile application also has layered architecture, so the user interface layer could be independent of backend logic.

Keywords: Windows phone, REST API, MVVM, XAML, C#, Web API, Event planning

9. Životopis

Bojan Grubić rođen je u Našicama 11.6.1990. Osnovnu školu pohađao je u Đurđenovcu. Kako od malih nogu pokazuje interes za elektrotehniku i računarstvo, nakon završene osnovne škole upisuje srednju Tehničku školu u Požegi, smjer Tehničar za računarstvo. Godine 2009. upisuje preddiplomski studij, smjer Računarstvo, na Elektrotehničkom fakultetu u Osijeku. Nakon završenog preddiplomskog studija, 2012. upisuje diplomski studij Procesnog računarstva na istom fakultetu.

Godine 2013. još kao redovan student zapošljava se u osječkoj podružnici zagrebačke tvrtke Span d.o.o. gdje i danas radi u ulozi razvojni inženjer/voditelja projekata. U Spanu, prvo kao mlađi razvojni inženjer, a potom i kao razvojni inženjer i voditelj projekata, sudjeluje u planiranju, izradi i implementaciji mnogobrojnih komercijalnih projekata. Glavne tehnologije s kojima radi su: .Net aplikacije (Windows phone, Windows store, desktop i ASP.Net aplikacije), SQL Server, SQL Server Reporting Services, mFiles i dr.

10. Prilozi

10.1. Prilog P10.1. – Skripta baze podataka

- Nalazi se na CD-u

10.2. Prilog P10.2. – Kompletan programski kod REST API-a

- Nalazi se na CD-u

10.3. Prilog P10.3. – Kompletan programski kod

- Nalazi se na CD-u