SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

LABORATORIJ PROFILA

PROJEKT IZ PREDMETA RASPODIJELJENI SUSTAVI

Tema: Prisutnost na društvenim mrežama

Tehnička dokumentacija

Josip Ćavar, Nikola Martinec, Petar Mrazović, Vjekoslav Ranogajec, Mario Volarević

# *Mentor: Doc. dr. sc. Mario Kušek*



Zagreb, Prosinac, 2011.

**1. Uvod**

U sklopu laboratorija profila *Programsko inženjerstvo i informacijski sustavi* praktično obrađujemo temu „Prisutnost na društvenim mrežama“. Glavni cilj ovog projekta je razviti web-aplikaciju koja će provjeravati prisutnost korisnikovih prijatelja na različitim društvenim mrežama kao što su npr. Facebook, Google+, Twitter, LinkedIn ili Gowalla. Također, vrlo važno je da se mi, studenti, koji se bavimo ovim projektom, upoznamo s osnovnim principima razvoja web-aplikacija, te problemima koji on donosi.

Ova tehnička dokumentacija donosi detaljan opis programskog rješenja koji je rezultat našeg rada na projektu. Budući da smo razvili Facebookovu web-aplikaciju koja koristi usluge nekoliko različitih društvenih mreža, u nastavku donosimo opis implementacije i odgovarajuće UML dijagrame podijeljene po društvenim mrežama koje smo uključili u aplikaciju.

Facebookova aplikacija koja je rezultat našeg rada omogućava korisniku jednostavan uvid u prisutnost i statuse njegovih prijatelja na drugim društvenim mrežama. Svaki student koji je bio član našeg razvojnog tima bavio se jednom društvenom mrežom. Za svaku pojedinu društvenu mrežu bilo je potrebno detaljno proučiti njene karakteristike, dostupna sučelja za programiranje aplikacija, tj. API-je (eng. *Application Programming Interface*) i odgovarajuću dokumentaciju.

Programsko rješenje ostvareno je korištenjem ASP .NET radnog okvira u razvojnom okruženju Visual Studio 2010. ASP .NET je Microsoftovo okruženje u okviru .NET platforme koje se može koristit za razvoj web-mjesta, web-aplikacija i XML web-usluga. Budući da se kao i ostali dijelovi .NET platforme temelji na zajedničkom CLR-u (eng. *Common Language Runtime*), njegov kod se može pisat u bilo kojem .NET jeziku. Budući da svi članovi tima imaju najviše iskustva u programiranju u programskom jeziku C#, upravo smo njega odabrali za jezik implementacije.



**Slika 1.1.** ASP .NET okruženje

**2. Arhitektura programskog rješenja**

tu bi trebalo ukratko o rješenju, struktura aplikacije, arhitektura itd.. pa onda dijagram klasa i deployment dijagram

**3. Društvena mreža Facebook**

Najpoznatiju i najveću društvenu mrežu osnovao je 2004. godine Mark Zuckenberg. Facebook danas broji 800 milijuna korisnika, a ključna razlika između dotadašnjih društvenih mreža i Facebooka je što se prema Facebookovim uvjetima korištenja korisnici moraju registrirati sa stvarnim imenom i prezimenom. Time je povezivanje između ljudi olakšano jer korisnici prijatelje mogu pronaći prema imenu. Od 2007. Godine Facebook podržava integraciju korisničkih aplikacija s Facebookovom platformom. Aplikacije za koje koriste Facebookovu platformu mogu izrađivati isključivo korisnici Facebooka. Takve aplikacije se integriraju s Facebookom preko sučelja *Graph API*. *Graph API* aplikacijama pruža sučelje za pristup podacima o Facebookovim korisnicima i slanje podataka prema Facebooku. Uz to dozvoljava integraciju s ostalim web stranicama, nepokretnim i pokretnim uređajima i slično. Kako bi aplikacije mogle koristiti i pristupati korisničkim podacima Graph API omogućava autentikaciju aplikacije s korisničke strane čime korisnik dozvoljava aplikaciji da pristupa njegovim podacima u Facebookovoj bazi. Kako bi se aplikacija mogla izvoditi unutar facebook.com web stranice potrebno ju je stvoriti i povezati s korisničkim profilom na Facebooku na stranice developers.facebook.com. Facebook za svaku aplikaciju generira jedinstvene identifikator i tajni ključ koje je potrebno zapisati u izvršni kod aplikacije. Aplikacije na facebook.com stranici s platformom se integriraju korištenjem Iframe mogućnosti Facebookove platforme. Iframe na poseban dio stranice *Canvas* lijepi eksternu web stranicu koja tako postaje *CanvasPage* unutar aplikacije na Facebooku i prikazuje se kao dio konteksta facebook.com web stranice. Projektni zadatak ostvaren je korištenjem Facebook Graph Toolkit sučelja. Ovo sučelje namijenjeno je razvoju aplikacija za Facebook u razvojnom okruženju Microsoft .NET 4.0 i jeziku C#.



Slika 3.1. Facebook Developers

**3.1. Funkcionalni zahtjevi**

Funkcionalni zahtjevi na aplikaciju su izrada aplikacije na facebook.com web stranici koja će za trenutno prijavljenog korisnika ispisivati popis prijatelja koji su prijavljeni na facebook.com. Kako Facebook ne dopušta pristup stvarnom status prijave na stranici, za izradu aplikacije koristit će se status korisnikovih prijatelja na Facebook Chatu. Status na Facebooku otvara se pri pokretanju aplikacije te pri pritisku na sličicu Facebooka unutar aplikacije. Slijedi detaljni opis ta dva slučaja korištenja.

Korisnik pokreće aplikaciju za praćenje prisutnosti

1. Sustav utvrđuje da je korisnik autorizirao aplikaciju

2. Sustav dohvaća podatke o trenutno prijavljenom korisniku

3. Sustav sustav dohvaća listu prijatelja prijavljenih na chat

4. Sustav ispisuje listu prijatelja prijavljenih na chat

Ekstenzija:

1a. Sustav utvrđuje da korisnik nije autorizirao aplikaciju

1. Sustav zahtjeva autorizaciju od korisnik
2. Korisnik autorizira aplikaciju

Korisnik pregledava status prijatelja na chatu

1.Korisnik odabire opciju za prikaz status prijatelja na Facebook Chatu

2. Sustav dohvaća podatke o trenutno prijavljenom korisniku

3. Sustav sustav dohvaća listu prijatelja prijavljenih na chat

4. Sustav ispisuje listu prijatelja prijavljenih na chat

**3.2. Programsko rješenje**

Unutar projektnog rješenja implementirana je Default.aspx stranica. To je početna stranica aplikacije te služi za ispisivanje statusa korisnikovih prijatelja na Facebooku. Identifikator i tajni ključ aplikacije zapisani su u konfiguracijskoj datoteci aplikacije web.config. Stranica je povezana s aplikacijom na Facebooku kao Canvas Page. Iz tog razloga nasljeđuje klasu Facebook Graph Toolkita CanvasPage koja implementira polja i metode potrebne za pristup Facebook Platformi i omogućava stranici da bude dio Iframe koncepta platforme.

3.2.1. Autorizacija aplikacije

Autorizacija aplikacije ostvarena je u metodi Page\_PreInit. Unutar metode postavljaju se polja klase CanvasPage RequiredLogin i CheckForExtendedPermissions. Kada korisnik uključuje aplikaciju Facebook platforma učitava identifikator aplikacije i tajni broj te pokušava aplikaciju prikazati korisniku. Postavljanjem gornjih polja od platforme se zahtjeva da od korisnika traži autorizaciju aplikacije čime korisnik aplikaciji omogućava pristup osnovnim podacima o svom profilu. Traženjem dodatnih dozvola, to jest polje CheckForExtendedPermissions, se zahtjeva da korisnik odobri pristup dodatnim podacima o svom profilu. U kontekstu ove aplikacije potrebno je tražiti dozvole za pristup prisutnosti prijatelja na mreži. U kodu to izgleda ovako.

protected void Page\_PreInit(object sender, EventArgs e)

{

RequireLogin = true;

ExtendedPermissions = "friends\_online\_presence";

CheckExtendedPermissions = true;

}

Nakon što korisnik odobri aplikaciju platforma popunjava podatke iz CanavsPage klase. Ključni podatak je identifikator trenutno prijavljenog korisnika na Facebook te polje Api koje za pruža sučelje za pristup metodama Graph API komponente Facebookove platforme.

3.2.2. Pristup podacima s Facebookovog korisničkog profila

Unutar metode Page\_Load implementira se funkcionalnost koju je potrebno izvršiti prije nego se prikaže sama stranica. Implementirane su dvije klase. Te klase su FacebookClient i FacebookTable. Klasa FacebookClient pruža javno sučelje u obliku metode Connect. Metoda Connect iz klase CanavsPage Facebook Graph Toolkita uzima identifikator trenutnog korisnika i referencu polje Api te vraća listu korisnika prisutnih na chatu.

Ta se lista ispisuje unutar web frome ListBoxFB stranice Default.aspx. Prije popunjavanja forme ListBoxFB ona se prazni.

List<FacebookTable> table = new List<FacebookTable>();

table = FacebookClient.Connect(Api, FBUserID);

ListBoxFB.Items.Clear();

foreach (FacebookTable item in table)

{

ListBoxFB.Items.Add(item.Name + " " + item.FBStatus);

}

Sama metoda Connect generira upit prema Facebookovoj bazi podataka. Facebookova baza podataka prihvaća samo upite u posebnom upitnom jeziku koji se zove Facebook Query Language. Strukturom je sličan SQL-u, ali onemogućava neke mogućnosti SQL-a poput spajanja tablica i slično. Status na chatu se može dohvatiti isključivo korištenjem ovakvog upita i nije dostupan kao osnovni podatak o korisniku. Kako bi se smanjio broj upita na bazu, a time i ubrzao rad aplikacije ne dohvaćaju se pojedinačni prijatelji i obavljaju pojedinačni upiti za svakog od njih nego se u jednom složenom upitu dohvaćaju imena svih dostupnih korisnika. Pod dostupne korisnike podrazumijevaju se korisnici sa statusom *active* i *idle*. Klasa FacebookTable služi za zapisivanje podataka o pojedinačnom korisniku. Polje *name* služi za zapis imena, a polje *fbStatus* za zapisivanje poruke o status. Odgovor dobiven od bazi nije u obliku prihvatljivom za ispis, pa se parsira u metodi ParseUserFriends u kojoj se iz odgovora izvlači ime korisnika te se vraća lista tako pripremljenih imena prijatelja.

string query = "SELECT name FROM user WHERE online\_presence IN ('active', 'idle') AND uid IN (SELECT uid2 FROM friend WHERE uid1 =" + userID + ")";

JsonArray result = Api.Fql(query);

List<string> friendsName = ParseUserFriends(result);

List<FacebookTable> onlineFriends = new List<FacebookTable>();

foreach (string name in friendsName)

{

onlineFriends.Add(new FacebookTable(name, " is online"));

}

return onlineFriends;

Upit na bazu je konstruiran kao složeni upit u kojem se iz tablice podataka o korisniku *user* dohvaćaju svi korisnici kojima je status aktivan i istovremeno su prijatelji trenutnog korisnika. Prijatelji trenutnog korisnika zapisani su unutar tablice *friend* u obliku liste identifikatora korisnika i njemu pridruženih identifikatora prijatelja te se dohvaćaju unutar podupita. Metoda Fql iz polja Api šalje upit prema Facebookovoj bazi podataka.

Kada korisnik odabere sličicu Facebookovog simbola unutar aplikacije pokreće se metoda ImageButtonFacebook\_Click koja sadrži funkcionalnost potpuno jednaku onoj u metodi Page\_Load.

Preusmjeravanje sa ostalih stranica na Default.aspx stranicu se obavlja posebnom metodom unutar Facebook Graph Toolkita IframeHelper.IframeRedirect koja služi za preusmjeravanje između Iframe stranica.

3.2.3. Dizajn programskog rješenja

3.2.3. Dizajn programskog rješenja

*Dijagram Razreda*

**

*Sekvencijski dijagram*

**

**3.3. Literatura**

[1] Wikipedia: *Facebook,* <http://en.wikipedia.org/wiki/Facebook>, datum pristupa 13.prosinca.2011

[2] Wikipedia: *Facebook features,* <http://en.wikipedia.org/wiki/Facebook_features>, datum pristupa 13.prosinca.2011.

[3] Wikipedia: *Facebook platform,* <http://en.wikipedia.org/wiki/Facebook_Platform>, datum pristupa 13.prosinca.2011

[4] Facebook developers: *Graph* AP,I<http://developers.facebook.com/docs/reference/api/>, datum pristupa 10.prosinac.2011.

[5] Facebook Graph Toolkit dokumentacija: *Facebook Graph Toolkit documentation* <http://computerbeacon.net/facebookgraphtoolkit/documentation/>, datum pristupa 15.prosinac.2011.

**4. Društvena mreža Google+**

tu ide za g+

**5. Društvena mreža Twitter**

2006. godine Jack Dorsey osnovao je Twitter, društvenu internetsku mrežu namijenjenu razmjeni kratkih poruka koje su prema imenu mreže popularno nazvane *tweetovima*. Ova društvena mreža danas broji preko 200 milijuna korisnika, te pruža pristup korisničkim podacima na različite načine, a jedan od njih je i razvoj aplikacija korištenjem nekih od dostupnih sučelja za programiranje aplikacija, tj. API-ja (eng. *Application Programming Interface*). Danas je dostupan niz API-ja za razne programske jezike i radne okvire, pa tako i za .NET Framework 3.5 i 4.0 što nam omogućava razvoj Twitterovih aplikacija u programskom jeziku C#. Neki od njih su *Twitterizer* i *TweetSharp*. U nastavku ćemo opisati usluge koje pruža *TweetSharp* API i način na koji je pomoću njih ostvarena provjera prisutnosti korisnikovih prijatelja [1].

**5.1. Funkcionalni zahtjevi**

Twitter kao društvena mreža ne pruža mogućnost uvida u trenutnu prisutnost korisnika, a upravo je to cilj projekta: razviti aplikaciju koja će pratiti trenutnu prisutnost na društvenim mrežama. Iz tog razloga koristit ćemo se kratkim porukama, tj. *tweetovima*, koji su objavljeni od strane korisnikovih prijatelja. Budući da za svaki *tweet* poznajemo vrijeme nastanka možemo ustanoviti koji je od korisnika nedavno bio prisutan na mreži. Tako će naša aplikacija za sve korisnikove prijatelje ispisivati *tweetove* koje će poredati po vremenu nastanka. Tako će korisnik dobiti jasan uvid u aktivnost prijatelja te kada je ona bila ostvarena. Također omogućit ćemo korisniku postavljanje novog *tweeta* putem naše aplikacije. Ove jednostavne funkcionalne zahtjeve ćemo u nastavku opisati koristeći tzv. slučajeve korištenja, tj. *use caseove* [1,4].

Korisnik pregledava aktivnosti prijatelja na društvenoj mreži Twitter

1. Korisnik odabire opciju za pregledavanje aktivnosti prijatelja na društvenoj mreži Twitter.

2. Sustav poziva Twitterovu uslugu za autorizaciju aplikacije.

3. Korisnik autorizira aplikaciju.

4. Sustav vraća popis *tweetova* korisnikovih prijatelja s pridruženim vremenom nastanka.

Ekstenzija:

2a. Korisnik je već autorizirao aplikaciju

1. Sustav dohvaća podatke o obavljenoj autorizaciji.

2. Sustav vraća popis *tweetova* korisnikovih prijatelja s pridruženim vremenom nastanka.

Korisnik postavlja novu kratku poruku (*tweet*)

Preduvjet: Korisnik je uspješno autorizirao aplikaciju.

1. Korisnik odabire opciju za postavljanje novog *tweeta*.

2. Korisnik upisuje tekst *tweeta*.

3. Korisnik pregledava *tweet*.

4. Korisnik potvrđuje postavljanje *tweeta*.

5. Sustav postavlja korisnikov *tweet* na Twitterovu društvenu mrežu.

Ovi jednostavni *use caseovi* kratko upisuju funkcionalnost koju je potrebno ostvariti programskim rješenjem. U nastavku ćemo vidjeti na koji način je ostvarena pojedina funcionalost korištenjem *TweetSharp* API-ja i ASP .NET platforme.

**5.2. Programsko rješenje**

5.2.1. Autorizacija aplikacije

Da bi preko web-aplikacije pristupili osobnim podacima korisnika Tweeterove društvene mreže, potrebno je prvo autorizirati aplikaciju. Svi programi pisani za rad u okviru različitih operacijskih sustava koji koriste Tweeterove usluge (tzv. third-party aplikacije), moraju se autorizirati koristeći OAuth, otvoreni standard za autorizaciju. OAuth (eng. *Open Authorization*) je način autorizacije na koji se omogućava pristup third-party aplikacijama podacima Twitterovog korisničkog profila bez prethodnog unošenja korisničkog imena i lozinke. Ovo osigurava veću sigurnost povjerljivim podacima korisnika s obzirom da se podaci za prijavljivanje na Twitterov korisnički profil neće moći koristiti ni na jednom drugom mjestu osim na twitter.com. Sve third-party aplikacije koje žele koristiti osobne podatke Twetterovih korisnika moraju se povezati s Twitterovim profilom isključivo koristeći **OAuth** proceduru.



Slika 5.2.1. *Open Autorization*

Da bi omogućili autorizaciju naše aplikacije, moramo ju prvo registrirati na Tweeterovoj mreži. Registracijom aplikacije dobivamo tzv. potrošački ključ (eng. *Consumer key*) te potrošačku tajnu (eng. *Consumer secret*), koji trebaju ostati poznati samo nama koji razvijamo aplikaciju. Na temelju tih podataka aplikacija generira tzv. *request* token pomoću kojeg od Twittera dobivamo URL adresu na kojoj nas čeka pin koji korisnik predaje aplikaciji. Aplikacija zatim na temelju pina traži tzv. *access* token, tj. token za pristup korisničkim podacima, te se konačno pomoću njega autorizira sama aplikacija. Opisani postupak je detaljno ilustriran na sljedećoj shemi (slika 5.2.2.). *Tweetsharp* API pruža vrlo dobru podršku otvorenoj autorizaciji pa ćemo se nastavku upoznati s klasama i metodama koje koristimo u programskom rješenju da bi ostvarili autorizaciju aplikacije [2,3].



**Slika 5.2.2.** Postupak autorizacije aplikacije korištenjem OAuth

Da bi autorizirali aplikaciju korištenjem *TweetSharp* API-ja moramo iskoristiti potrošački ključ i tajnu koje smo dobili prilikom registracije aplikacije. Na temelju tih podataka stvaramo objekt tipa TwitterClientInfo koji nam služi za identifikaciju aplikacije te stvaranje novog objekta tipa TweeterService pomoću kojeg komuniciramo s Tweeterovom društvenom mrežom i manipuliramo podacima korisnika.

TwitterClientInfo twitterClientInfo = new TwitterClientInfo();

twitterClientInfo.ConsumerKey = ConsumerKey;

twitterClientInfo.ConsumerSecret = ConsumerSecret;

TwitterService twitterService = new TwitterService(twitterClientInfo);

Sljedeći korak je stvaranje već prije spomenutog *request* tokena pomoću kojeg ćemo dohvatit URL adresu za autorizaciju, te ćemo ju otvoriti u web-pregledniku kako bi korisnik mogao unijeti dobiveni pin te tako dobiti *access* token. *Request* token i URL adresa za dobivanje pina generiraju se prilikom pritiska na gumb „Get Pin“, a *access* token generira se pritiskom na gumb „Authorize“ (slika 5.5.3.). URL adresa s pinom otvara se u novom skočnom prozoru.

OAuthToken requestToken = twitterService.GetRequestToken();

Session["TwitterRequestToken"] = requestToken;

authUrl = twitterService.GetAuthorizationUrl(requestToken);

authUrl = Page.ResolveClientUrl(authUrl);

-------------------------------------------------------------------------------

OAuthToken accessToken;

accessToken = twitterService.GetAccessToken((OAuthToken) Session["TwitterRequestToken"], this.TextBoxPin.Text);

Session["TwitterAccessToken"] = accessToken;

OAuthToken requestToken = twitterService.GetRequestToken();

Session["TwitterRequestToken"] = requestToken;

authUrl = twitterService.GetAuthorizationUrl(requestToken);

authUrl = Page.ResolveClientUrl(authUrl);

OAuthToken accessToken;

accessToken = twitterService.GetAccessToken((OAuthToken) Session["TwitterRequestToken"], this.TextBoxPin.Text);

Session["TwitterAccessToken"] = accessToken;

pin.png

auth.png

*Access* i *request* token spremamo u tzv. sesijske varijable kako korisnik ne bi trebao više puta autorizirati aplikaciju. Naime, kada klijent prvi puta pristupa web-aplikaciji, generira se jedinstveni SessionID broj i sprema se na korisnikovo računalo u obliku *cookie*-a ili ga dodaje u svaki korisnikov zahtjev. Vezano uz taj ID možemo stvarati sesijske (eng. *session*) varijable koje se čuvaju na serveru.



**Slika 5.2.3.** Autorizacija aplikacije

5.2.2. Pristup podacima s Tweeterovog korisničkog profila

Sljedeći korak je pristupiti korisnikovim podacima kako bi dohvatili njegove prijatelje i njihove *tweetove*. Prije dohvaćanja podataka korištenjem *access* tokena koji je spremljen u sesijskoj varijabli potvrdit ćemo autoriziranost aplikacije. Podatke s Tweeterove društvene mreže dohvaćat ćemo jednostavnim pozivima metoda klase TwitterService. Metoda ListTweetsOnFriendsTimeline() dohvaća *tweetove* prijatelja kao objekte tipa TwitterStatus. Svakom tom objektu pridruženo su svojstva: CreatedDate (vrijeme nastanka *tweeta* po kojem se sortiraju svi dohvaćeni *tweetovi*), User (korisnik koji je postavio *tweet*), te Text (tekst *tweeta)* [4].

protected void refreshFriendTweets()

{

IEnumerable<TwitterStatus> tweets = twitterService.ListTweetsOnFriendsTimeline();

foreach (TwitterStatus tweet in tweets)

{

ListBoxTweets.Items.Add("[" + tweet.CreatedDate + "] --> " + tweet.User.ScreenName + " says: " + tweet.Text + ".");

}

}

5.2.3. Dizajn programskog rješenja

U ovom poglavlju opisan je dizajn programskog rješenja koje ostvaruje pristup korisnikovim podacima s Twitterovog korisničkog profila. Rješenje je ostvareno koristeći dvije web stranice (eng. *web form*). Svaka od stranica predstavljena je jednom ASPX datotekom s pridruženim HTML-om i C# kodom. Prva stranica je nazvana TwitterLogIn.aspx i njome je ostvarena autorizacija korisnika. Druga stranica je TwitterTable.aspx, te ona dohvaća odgovarajuće podatke s korisničkog profila i upisuje ih u tablicu. Kada korisnik odabire Twitter kao društvenu mrežu s koje želi dohvatiti podatke o prisutnosti prijatelja, sustav automatski preusmjerava poziv na TwitterTable.aspx. Ukoliko u sesijskoj varijabli postoji *access* token, sustav u tablici ispisuje odgovarajuće podatke. No, ako je sesijska varijabla prazna, sustav korisnika preusmjerava na stranicu TwitterLogIn.aspx koja se dalje bavi autorizacijom korisnika kako je opisano u poglavlju 5.2.1. Autorizacija aplikacije. Sljedeća shema slikovitije opisuje opisano rješenje.



if (Session["TwitterAccessToken"] == null)

{

Response.Redirect(@"~\TwitterLogIn.aspx");

}

else

{

OAuthToken accessToken = (OAuthToken)Session["TwitterAccessToken"];

twitterService.AuthenticateWith(accessToken.Token, accessToken.TokenSecret);

ListBoxTweets.Items.Clear();

refreshFriendTweets();

}

Upoznat ćemo se još s klasama koje koristimo u ovom djelu programskog rješenja. Budući da smo aplikaciju registrirali na Tweeteru potrebno je negdje spremiti podatke o potrošačkom ključu i tajni. Ti podaci spremljeni su kao statički atributi u klasi TwitterAppInfo, a na temelju njih klasa vraća objekt tipa TwitterClientInfo. Klasu TwitterClientInfo već smo ranije spominjali. Prisjetimo se: ona nam služi za stvaranje objekta tipa TwitterService kojime komuniciramo s Twitterom i dohvaćamo odgovarajuće podatke. Također spominjali smo i tokene. Oni su zapravo objekti klase OAuthToken. Nakon autorizacije dohvaćamo podatke s korisničkog profila. Prilikom dohvaćanja *tweetova* dobivamo objekte klase *TwitterStatus*, a preko njihovih atributa dobivamo podatke o korisniku, vremenu nastanka i samom sadržaju *tweeta*. U nastavku ćemo prikazati sve ove klase dijagramom razreda (slika 5.2.4.). Dijagram razreda je pojednostavljen pa tako ne sadrži sve metode i atribute pojedinih klasa.



**Slika 5.2.4.** Dijagram razreda

Na samom kraju opisa programskog rješenja sekvencijskim dijagramom ćemo dati uvid u slijed događaja između korisnika, sustava i društvene mreže prilikom dohvaćanja *tweetova* korisnikovih prijatelja.



**Slika 5.2.5.** Sekvencijski dijagram

**5.3. Literatura**

[1] Wikipedia: Twitter, rujan 2011., *Twitter,* http://en.wikipedia.org/wiki/Twitter, 2. studenog 2011.

[2] Wikipedia: OAuth, listopad 2011., *OAuth,* http://en.wikipedia.org/wiki/OAuth, 2. studenog 2011.

[3] Twitter: Devlopers, srpanj 2011., *Get started with API,* https://dev.twitter.com/docs, 3. studenog 2011.

[4] Dan's Development Blog, Dan Swains, 13. veljače 2011., *A simple Twitter client tutorial in C# with OAuth using TweetSharp*, http://www.d80.co.uk/post/2011/02/13/A-Simple-Twitter-Client-in-C-with-OAUTH-using-TweetSharp.aspx, 3. studenog 2011.

[5] Tweetsharp, Daniel Crenna, svibanj 2011., *Tweetsharp - Short, sweet, social.*, https://github.com/danielcrenna/tweetsharp, 6. studenog 2011.

**6. Društvena mreža Gowalla**

Gowalla je zamišljena kao lokacijski orijentirana društvena mreža. Glavna ideja Gowalle je podijeliti priče, slike i iskustva o mjestima koje posjećujemo sa drugim ljudima, npr. gradove, muzeje i sl. Upravo je to imao na umu jedan od osnivača (Josh Williams, Scott Raymond) kada je 2007. godine pokrenut projekt pod nazivom Gowalla. Prije svega, ona je prvenstveno mobilna aplikacija jer se nije moguće „prijavljivati“ na raznim lokacija sa standardne web stranice[1].

Nažalost, početkom prosinca 2011. godine službeno je objavljeno da je Gowalla kupljena od strane Facebooka, te se krajem siječnja ova mreža u potpunosti gasi[2].

**6.1. Funkcionalni zahtjevi**

Na Gowalli nije moguće dobiti uvid u status prisutnosti korisnika jer on uopće ne postoji. Takav status nije ni potreban za funkcioniranje Gowalle. Ono što zanima njezine korisnike je da li je netko od njihovih prijatelja bio negdje te to objavio. Zbog toga smo kao zamjenu za statuse uzeli vrijeme zadnjeg objavljivanja „prijavljivanja“ na nekom mjestu. Prema tome korisnik koji koristi našu aplikaciju može lako zaključiti da li se je dogodilo nešto zanimljivo vrijedno odlaska na samu Gowallu. Tipičan *use case*, odnosno slučaj korištenja opisan je u nastavku.

Korisnik pregledava aktivnosti prijatelja na Gowalli

1. Korisnik odabire pregled aktivnosti prijatelja na Gowalli
2. Sustav ga traži korisničko ime i lozinku
3. Dohvaćaju se i dostavljaju podaci o zadnjim aktivnostima korisnikovih prijatelja

**6.2. Programsko rješenje**

Za programsko ostvarivanje zadatka korišten je Gowalla API. API poštuje REST konvenciju te nije ovisan o specifičnom programskom jeziku. Za dohvat npr. korisnikovih podataka potrebno je poslati određen HTTP zahtjev. URL-ovi kojima dohvaćamo različite podatke popisani su u dokumentaciji (API Explorer)[3]. Odgovor koji dobijemo je u JSON formatu – tekstualno baziranom standardu koji je čitljiv ljudima.

Da bismo uopće mogli koristiti API potrebno je dobiti tzv. *API* *key.* Da bi smo ga dobili potrebno je registrirati aplikaciju. Ključ koji dobijemo moramo dodati u zaglavlje HTTP upita kojeg koristimo. Ovjera (eng. *authentication*) korisnika se vrši pomoću osnovne ovjere korisnika (eng. *basic access authentication*) koju podržava većina web preglednika.

**6.3. Primjer dohvaćanja korisnikovih podataka**

Da bi smo mogli dohvatiti korisnikove podatke prvo se moramo prijaviti na Gowallin sustav. Prilikom svakog upita na Gowallin sustav gradi se objekt tipa *HttpWebRequest* koji je sastavni dio kazala *System.Net.*  U njemu definiramo potrebna zaglavlja , kao što je *API Key* koji smo spomenuli ranije. Metoda prima dva parametra – URL traženog resursa i enumerator *authentication.* Pošto se ova metoda koristi za sve upite samo je kod prijavljivanja korisnika na Gowallin sustav potrebno postaviti vrijednost te varijable na *required.*

public HttpWebRequest createRequest(string url, authentication aut)

{

HttpWebRequest request = (HttpWebRequest)WebRequest.Create(url);

if (aut == authentication.required)

{

request.Credentials = new NetworkCredential(userName, passWord);

request.Headers.Add("Authorization", "Basic" +

Convert.ToBase64String(new

ASCIIEncoding().GetBytes(userName + passWord)));

}

request.ContentType = "application/json";

request.Accept = "application/json";

request.Headers.Add("X-Gowalla-API-Key:" + Gowalla.key);

return request;

}

Sljedeći odsječak koda zapravo poziva gore navedenu metodu sa odgovarajućim parametrima. Nakon što je zahtjev kreiran, odnosno imamo objekt tipa *HttpWebRequest,* možemo zatražiti odgovor od Gowallinog poslužitelja. Kao odgovor dobiti ćemo JSON objekt. Dio odgovora slijedi u nastavku:

"stamps\_count": 0,

"url": "[/users/2898661](http://gowalla.com/users/2898661)",

"bookmarked\_spots\_url": "[/users/2898661/bookmarks](http://gowalla.com/users/2898661/bookmarks)",

**"friends\_activity\_url": "**[**/checkins/recent**](http://gowalla.com/checkins/recent)**",**

"can\_post\_to\_facebook": false,

"\_auto\_connect\_addressbook\_friends": null,

"current\_checkins": [],

**"following\_count": 7,**

"friend\_requests\_count": 0,

"\_is\_friend": true,

"\_visited\_spots\_urls\_url": "[/users/2898661/visited\_spots\_urls](http://gowalla.com/users/2898661/visited_spots_urls)",

"can\_post\_to\_twitter": false,

…

Iz navedenog odgovora je vrlo lako izvući željene podatke – bilo da se radi o jednostavnom podatku, kao što je npr. *following\_count,* odnosno broj prijatelja čiju aktivnost pratimo, ili o kolekciji podataka npr. *friends\_activity\_url* za koju dobijemo URL preko kojeg možemo dohvatiti elemente kolekcije. Konkretno, ovaj kod postavlja korisnikovo ime, prezime, te korisničko ime koje može biti proizvoljno.

Na sličan način se dohvaćaju podaci o korisnikovim prijateljima te spremaju u objekte tipa *Friend* koji sadrže sve potrebne podatke o njima.

private void fetchBasicUserData(authentication aut)

{

HttpWebRequest request = createRequest("http://api.gowalla.com/users/me", aut);

using (HttpWebResponse response = (HttpWebResponse)request.GetResponse())

{

StreamReader reader = new StreamReader(response.GetResponseStream());

string result = reader.ReadToEnd();

String[] data = result.Split(',');

this.firstName = data.First(x =>

x.Contains("first\_name")).Split('"')[3];

this.lastName = data.First(x => x.Contains("last\_name")).Split('"')[3];

this.alias = data.First(x => x.Contains("username")).Split('"')[3];

}

}

**6.4. Tehničke poteškoće vezane uz Gowalla API**

Nažalost, API ne radi ispravno u potpunosti te mnogi URL-ovi dobiveni u odgovoru rezultiraju pogreškom o nepostojanju traženog resursa (*404 Not Found).* Zbog toga ne možemo pristupiti popisu korisnikovih prijatelja preko API-a. Kako bi zaobišli ovaj problem, nakon što smo prijavili korisnika na sustav koristeći API, popis korisnikovih prijatelja učitavamo izravno filtrirajući HTML stranicu na kojoj bi oni bili prikazani kada bi korisnik koristio Gowallu izravno. To može predstavljati problem - ako se nešto promijeni u vezi URL-a koji vodi do te HTML stranice aplikacija više neće ispravno raditi.

**7. Društvena mreža LinkedIn**

LinkedIn je najveća profesionalna mrežna usluga. Profesionalna mrežna usluga(ili jednostavno profesionalna mreža) je vrsta društvene mreže koja je usmjerena isključivo na interakciju i odnose poslovne prirode, a ne uključuje osobne, neposlovne odnose. Pokrenuta je 2003 godine, a osnivać je Reid Hoffman sa članovima osnivačkog tima iz PayPal-a i Socialnet.com-a. Jedna svrha stranice je da omogući registriranim korisnicima da održava popis kontaktnih podataka ljudi s kojima imaju neki poslovni odnos, u LinkedIn-u nazvan veza tj. *Connection*. Korisnik može pozvati bilo koga da postane njegova veza, što korisnik koji je pozvan može prihvatiti ili odbiti. LinkedIn ima više od 135 milijona korisnika diljem svijeta.

**7.1. Funkcionalni zahtjevi**

LinkedIn poput Twitter-a i Gowalle također ne pruža mogućnost uvida u trenutnu prisutnost korisnika. Kao što smo već spomenul da čemo kod Twittera koristiti

tweetove za provjeru prisutnosti tako čemo kod LinkedIn-a koristiti status koji korisnik može ažurirati, te čem tako dobiti uvid u aktivnosti korisnika s

kojima smo povezani. Slučaj korištenja je sličan kao kod Twittera. Pogledajmo kako izgleda.

Korisnik pregledava aktivnosti veza na LinkedIn-u

1. Korisnik odabire opciju za praćenje aktivnosti svojih veza na LinkedIn-u.

2. Sustav poziva LinkedIn-ovu uslugu za autorizaciju aplikacije.

3. Korisnik autorizira aplikaciju.

4. Sustav vraća popis statusa korisnikovih veza.

Nakon toga korisnik ima mogućnost ažurirati svoj status, pa pogledajmo slučaj korištenja.

1. Korisnik upisuje željeni status.

2. Korisnik odabire opciju za ažuriranje statusa.

3. Sustav ažurira status.

**7.2. Programsko rješenje**

Način autenikacije je isti kao kod Twittera, samo postoji jedna iznimka. Kad autoriziramo aplikaciju LinkedIn nam vraća verifikator s kojim se dalje pristupa bez ponovne autorizacije.

Kad smo jednom autorizirali aplikaciju i kad smo dobili dozvolu za korištenje API-ja, onda možemo slati zahtjeve za resursima koje nam pruža LinkedIn. Kod izrade ovog programa slali smo zahtjeve za dohvat statusa korisnika, te zahtjev za postavljanje statusa. Kad pošaljemo zahtjev za dohvat statusa korisnika

LinkedIn nam vraća rezultat u xml obliku. Također u zahtjevu za ažuriranje statusa željeni status čemo morati staviti u prikladan xml oblik. Pogledajmo kako izgleda metoda za ažuriranje statusa.

protected void Update\_Status(object sender, EventArgs e)

{

try

{

string xml = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>";

xml += "<current-status>" + TextBox1.Text + "</current-status>";

string response = OAuthObject.\_oauth.APIWebRequest("PUT", "http://api.linkedin.com/v1/people/~/current-status", xml);

if (response == "")

TextBox1.Text = "";

}

catch (Exception exp)

{

TextBox1.Text += "\nException: " + exp.Message;

}

}

Dakle, kreiramo xml u obliku koji će LinkedIn prihvatiti, te na mjesto gdje treba ići novi status, postavljamo ono što se nalazi u polju za unos statusa. Nakon toga pozivamo metodu APIWebRequest koja će kreirati zahtjev, poslati zahtjev i vratiti nam odgovor u obliku niza znakova. Ako je taj niz znakova prazan niz, onda je status uspješno ažuriran.

**8. Upute za korištenje**

Upute