Univerzitet u Beogradu
Matematički fakultet

Razvoj web-aplikacije za bankarsko poslovanje zasnovane na WPF/Silverlight tehnologiji

Master rad

Student

Novak Marković

Mentor

prof. dr. Dušan Tošić

Beograd, februar 2010.

Sadržaj

I PREDGOVOR

Master rad pod nazivom „Razvoj web-aplikacije za bankarsko poslovanje zasnovane na WPF/Silverligth tehnologiji” („*Windows Presentation Foundation“)* ima za cilj da predstavi Microsoftovu tehnologiju Silverlight za razvoj kompleksnih internet aplikacija (RIA, „*Rich Internet Applications*“) kroz kreiranje softvera „Namenski računi“.

Softversko rešenje „Namenski računi” predstavlja jedan od nezaobilaznih programa za svaku brokersku kuću ili banku. Pomoću tog softvera brokeru je omogućeno da prati sve promene na klijentskim portfolijima kao i da na zahtev klijenata štampa odgovarajuće potvrde o stanju na računima.

U sledećem poglavlju upoznaćemo se sa osnovama WPF/Silverlight tehnologije, razlozima njenog uvođenja a u trećem poglavlju i sa samom aplikacijom “Namenski računi”, sa svim njenim funkcionalnostima.

II OPIS WPF/SILVERLIGHT TEHNOLOGIJE

2.1 Uvod

Inspiracija za uvođenje tehnologije kao sto je WPF dolazi od limita prethodnih tehnologija. Postoje tehnologije kao što je GDI („*Graphics Device Interface*“) koje omogućuju razvijanje osnovnog korisničkog interfejsa i postoje tehnologije kao sto su DirectX i OpenGL za razvijanje sofisticiranijeg i kompleksnijeg korisničkog interfejsa. U slučaju da je programeru, koji programira poslovne aplikacije, potreban neki kompleksniji korisnički interfejs, on bi bio doveden u situaciju da uči nekoliko različitih tehnologija da bi to odradio.

WPF je bogat skup biblioteka za programiranje korisničkog interfejsa koji je dostupan i programerima i dizajnerima preko deklarativnog jezika zvanog XAML („*Extensible Application Markup Language*“). WPF je jedna od četiri nove tehnologije predstavljene u .NET *framework*-u 3.0. Da bismo koristili WPF aplikaciju potrebno je da na računaru imamo instaliran najmanje .NET *framework* 3.0 ili noviju verziju.

Dosadašnji način programiranja GUI-ja („*Graphical User Interface*“ odnosno „Grafički Korisnički Interfejs“) uz korišćenje Microsoft-ovih tehnologija je podrazumevao Windows Forms biblioteku. Windows Forms se oslanja na Windows API, 20 godina staru biblioteku elemenata korisničkog interfejsa kao što su prozori, dugmad, dugmad sa čekiranjem, tekstualna polja i drugi. Svaki element ima svoj prikaz na ekranu i mora da čeka *Draw* event da bi se prikazao na ekranu. Proces iscrtavanja ne koristi GPU („*Graphics Processing Unit*“ odnosno „Procesor Grafičke Kartice”), već ceo posao odrađuje CPU („*Central Processing Unit*“), tako da ne postoji mogućnost iskorišćavanja prednosti modernih grafičkih kartica.
WPF sve ovo menja tako što uvodi novi model sa drugačijom logikom zasnovanom na DirectX-u.
Svaki element je kompozicija osnovnih elemenata vektorske grafike. Svaki element iscrtava sebe u okviru jedne operacije koja je ubrzana hardware-om grafičke kartice.

2.2 Osnovne karakteristike

Osnovne karakteristike i prednosti WPF-a su sledeće:

Vektorska grafika – Prelazak na veću rezoluciju u slučaju WPF aplikacije ne znači da sve postaje manje, umesto toga – grafika i tekst samo postaju jasniji. WPF koristi Direct3D za iscrtavanje – što znači da koristi GPU – procesor grafičke kartice. Tako se mogu razvijati korisnički interfejsi koji izgledaju dobro kako na malom ekranu mobilnog telefona, tako i na 50-inčnom televizoru.

Hardversko ubrzanje – kao što je pomenuto, WPF za iscrtavanje koristi Direct3D. To znači da, za razliku od GDI sistema, WPF aplikacije koriste prednosti grafičkog hardvera za kvalitetniji i brži prikaz. Međutim, WPF ne zahteva postojanje vrhunskog grafičkog podsistema jer takođe poseduje i softverski rendering.

Deklarativno programiranje – WPF donosi modifikovan XML, napravljen tako da je pogodan za deklarisanje korisničkog interfejsa – XAML. Korišćenjem XAML-a elementi korisničkog interfejsa su prikazani kao XML tagovi. Na taj način XAML omogućava aplikacijama da ga dinamički parsiraju i manipulišu elementima bilo u vreme kompajliranja ili u vreme izvršavanja.
Osnovna ideja XAML-a je razdvajanje korisničkog interfejsa i logike čime je omogućen paralelni rad grafičkih dizajnera i programera pri kreiranju korisničkog interfejsa.
Druga osnovna ideja jeste unifikovanje korisinčkog interfejsa različitih tipova aplikacija (windows, internet aplikacije).
Izgled korisničkog interfejsa pisan u C++, Java ili C# jeziku mora da bude kompletno urađen ponovo prilikom portovanja na Internet, dok se XAML kÔd portuje na sve platforme bez promena.

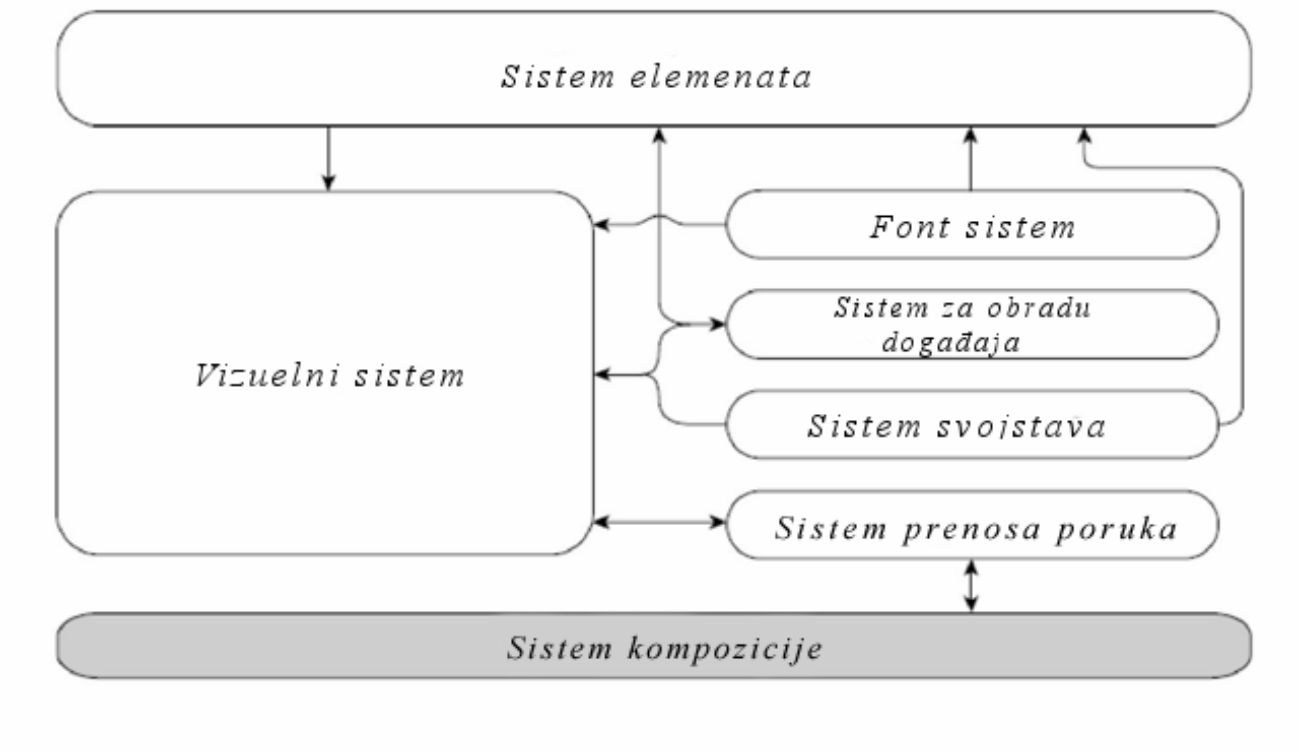
Kompozicija i prilagođavanje – WPF kontrole se mogu komponovati na izuzetno jednostavan način. Možemo kreirati padajuću listu u koju ćemo smestiti dugmad posebno dizajnirana u *Microsoft Expression Blend*-u sa raznim efektima (*MouseOver, Focus)* ili meni u kojem će stavke biti video klipovi. Takođe, moguće je definisati maske (*skin*) koje će drastično promeniti izgled aplikacije u skladu sa željama korisnika.

Jednostavna instalacija – WPF aplikacije mogu biti isporučene kao samostalne aplikacije ili kao web aplikacije kojima se pristupa kroz internet pretraživač (Internet Explorer, Mozilla Firefox).

Portabilnost dokumenata – u WPF je uključen novi set tehnologija za obradu i štampu dokumenata. Zajedno sa pojavom paketa Office 12, WPF koristi *Open Packaging Convention* koji nudi kompresiju dokumenata, prilagodljive meta-podatke, digitalne potpise i upravljanje pravima pristupa. Slično PDF-u, XPS („*XML Paper Specification*“ odnosno „XML Specifikacija Dokumenta“) omogućuje razmenu dokumenata između računara bez potrebe za instalacijom aplikacije u kojoj su napravljeni.

2.3 Arhitektura WPF-a

Anatomiju WPF-a čine servisi, podsistemi kao i kontrolisani programski interfejs („*Managed API*“) dostupan za korišćenje u WPF aplikacijama, koji se naziva „*Presentation Framework*“. Na slici 1 prikazana je osnovna arhitektura WPF.



Slika 1

Sistem elemenata (*„Element system“)* predstavlja površinu WPF-a sa kojom komuniciraju programeri. Sistem elemenata se sastoji od osnovnih komponenti korisničkog interfejsa kao što su stilovi, izgled (*layout)*, kontrole, povezanost kontrola i podataka (*binding)* i tekst. Elementi su u WPF aplikacijama grupisani u stabla.

Postoje dve vrste stabla elemenata, to su logičko stablo i vizuelno stablo. Logičko stablo je hijerarhijska struktura elemenata koji su definisani deklarativno u XAML datoteci ili imperativno u kodu. Za svaki element u logičkom stablu kreiraju se dodatni elementi koji predstavljaju njegove vizuelne aspekte. Kombinacija elemenata aplikacije i elemenata kreiranih za vizuelni prikaz čine vizuelno stablo.

Vizuelni sistem („*Visual system*“) predstavlja podsistem kroz koji aplikacija pristupa osnovnim prezentacionim servisima WPF-a. Vizuelni sistem prolazi kroz sve komponente (labele, dugmad, tekst, 2D i 3D elemente) i na osnovu njih renderuje grafički prikaz na ekranu (tj. prosleđuje poruke sistemu kompozicije).

Font sistem („*Font system*“) je potpuno iznova napisan za potrebe WPF-a i obezbeđuje jedinstveni mehanizam za kreiranje i keširanje informacija o fontovima uključujući *TrueType* i *OpenType* fontove. Sistem keširanja je napravljen kako bi se ubrzao proces prikazivanja fontova koji je inače prilično zahtevan.

Sistem za obradu događaja („*Input/Event system*“) donosi značajna unapređenja u odnosu na Win32. Input/Event sistem obezbeđuje integraciju korisnikovih aktivnosti i vizuelnog stabla. WPF prihvata događaj koji je izazvala akcija korisnika i zatim se na osnovu vrste događaja i kontrola koje su taj događaj registrovale odlučuje koji i koliko događaja će biti podignuto. Štaviše, WPF podržava tzv. rutirane događaje, koji pružaju mogućnost elementima vizelnog stabla da osluškuju i reaguju na događaje kako roditelja tako i ugnježdenih kontrola. To se postiže na dva načina: aktiviranje i prosleđivanje događaja ka (*tunneling)* ili od *(bubbling)* izvorišnog (*source*) elementa. *Tunneling* predstavlja situaciju kada se događaj (*event*) aktivira u korenu vizuelnog stabla, a potom na svakom od elemenata niz stablo sve dok se ne stigne do izvorišnog (*source*) elementa (elementa koji je stvarno aktivirao događaj) ili dok metod (*handler)* ne prekine *tunneling* i označi da je događaj obrađen. *Bubbling* je situacija u kojoj se događaj diže u izvorišnom (*source*) elementu a potom i u svakom elementu iznad sve dok se ne dođe do korena, ili dok metod (*handler*) ne prekine *bubbling* i označi da je događaj obrađen.

Sistem svojstava („*Property system*“) je takođe značajno nadograđen. WPF uvodi novi tip svojstava koji se naziva svojstvo zavisnosti („*dependency property*“). Takvo svojstvo zavisi od više objekata kako bi se utvrdila njegova vrednost u svakom trenutku u vremenu. Najveća vrednost zavisnih svojstava je njihova mogućnost da pruže notifikaciju o promeni. Na taj način se može postići velika funkcionalnost korišćenjem deklarativnog jezika. Pored notifikacije o promeni, sistem svojstava nadograđen je nasleđivanjem vrednosti svojstava tj. prenošenjem vrednosti svojstava niz stablo elemenata.

Sistem prenosa poruka („*Message transport*“) je ključna komponenta WPF arhitekture zadužena za povezivanje vizuelnog sistema i sistema kompozicije. Sistem prenosa poruka pruža kanale za komunikaciju između dva pomenuta sistema preko protokola za udaljeno pozivanje metoda („*.NET Remoting protocol*“). Ovakav pristup je posebno koristan prilikom tzv. terminal klijent scenarija (kao što je „*Remote Desktop*“) gde će se procesiranje grafike preneti na klijentsku mašinu i time rasteretiti server.

Sistem kompozicije („*Composition system*“) predstavlja podsistem (zasnovan na „*unmanaged*“, nebezbednom kodu) koji prihvata instrukcije od vizuelnog sistema („managed“, bezbedni, kontrolisani kod) i takve instrukcije pretvara u grafiku koju vidimo na ekranu. Sistem kompozicije predstavlja deo WPF-a koji je u komunikaciji sa grafičkim hardverom računara preko Direct3D tehnologije.

2.4 XAML

XAML („*Extensible Application Markup Language*“) predstavlja deklarativni jezik koji se koristi za instanciranje .NET objekata. Iako je XAML tehnologija koja se može primeniti u više domena, njena primarna uloga je konstrukcija WPF korisničkog interfejsa.
Drugim rečima, XAML dokumenti definišu raspored panela, dugmadi i drugih kontrola koje čine UI aplikacije.
Iako je takav slučaj moguć, XAML se ne piše „ručno“. Postoji niz alata čija uloga je generisanje XAML koda. Koji alat će biti korišćen, zavisi od uloge korisnika u razvojnom procesu aplikacije. Programeri će, naravno, koristiti Visual Studio, dok će se grafički dizajneri puno bolje snaći u Microsoft Expression Blend-u. Upravo u toj činjenici leži i jedna od najvećih prednosti XAML-a – on predstavlja tehnologiju koju će deliti programeri i dizajneri korisničkog interfejsa. Pre pojave XAML-a proces kreiranja korisničkog interfejsa, koji bi uključivao dizajnere i programere, bio je frustrirajući za obe strane i na kraju se obično svodio na sledeće: grafički dizajner bi pripremio model korisničkog interfejsa, koji bi programeri kasnije morali da prevode u kod. Upotrebom XAML-a proces može izgledati ovako:
programer kreira osnovni korisnički interfejs i potom ga predaje dizajn timu koji dalje radi na njemu.
Važno je napomenuti da XAML nije neophodan za WPF. Moguće je nastaviti sa korišćenjem Windows Forms pristupa i kreirati kod koji će sadržati izraze za generisanje WPF prozora. Takav WPF prozor
bio bi zarobljen u Visual Studio okruženju i ne bi se mogao menjati van njega, što znači da bi bio dostupan samo programerima.
I pored saradnje sa grafičkim dizajnerima postoji mnogo drugih razloga za korišćenje XAML-a:
XAML je koncizan način za predstavljanje korisničkog interfejsa ili druge hijerarhije objekata.Korišćenje XAML-a ohrabruje razdvajanje korisničkog interfejsa od pozadninske logike.XAML je jezik koji gotovo svi WPF alati generišu.XAML je relativno jednostavan deklarativni jezik opšte namene koristan za konstruisanje i inicijalizaciju .NET objekata. .NET *framework* 3.5 uključuje kompajler i *run-time* parser za XAML, kao i *plug-in* pomoću koga XAML datoteke možemo videti preko Internet Explorera.
XAML kôd koji definiše deo korisničkog interfejsa aplikacije „Namenski računi“ može se videti u prilogu 6.1 .

2.5 Učitavanje i kompajliranje XAML-a

XAML i WPF, kao što je ranije napomenuto, su dva odvojena koncepta iako su komplementarni. To znači da je moguće napisati WPF aplikaciju bez upotrebe XAML jezika. Postoje dva različita stila kodiranja koja se mogu koristiti pri razvoju WPF aplikacija:

Samo kôd – tradicionalni način generisanja korisničkog interfejsa koji se koristi u Windows Forms aplikacijama. Korisnički interfejs se generiše kroz proceduralni kôd. Očigledan nedostatak ovakvog pristupa je taj što može biti prilično zamoran. WPF kontrole ne poseduju parametrizovane konstruktore tako da dodavanje najobičnijeg dugmeta zahteva pisanje nekoliko linija koda. Takođe, ne može se ostvariti saradnja na liniji grafički dizajner – programer. S druge strane, način razvoja WPF aplikacija korišćenjem samo koda ima određenih prednosti npr. možemo generisati formu punu kontrola na osnovu podataka iz baze, ili koristiti uslovnu logiku za kreiranje kontrola itd...

Kôd i nekompajlirani XAML – Scenario u kojem se delovi interfejsa učitavaju za vreme izvršavanja programa preko XamlReader klase. Koristi se pri kreiranju dinamičnih korisničkih interfejsa. Ova tehnika donosi fleksibilnost u generisanju korisničkog interfejsa, mogućnost korišćenja uslovne logike, itd... Naravno, ovakav pristup donosi pad performansi aplikacije naročito ako je korisnički interfejs kompleksan.

2.6 Tipovi WPF aplikacija

Ukoliko bi se WPF aplikacija sastojala samo od XAML stranica i njihovog pozadinskog koda, funkcionalnosti koje bi posedovala bile bi ograničene i nedovoljne za većinu potreba. Za ozbiljniju aplikaciju potrebno je imati mehanizam čuvanja stanja stranice, razmenu podataka i konteksta među stranicama, detekciju događaja učitavanja stranice, upravljanja globalnim promenljivim, itd...
WPF aplikacioni model podržava objedinjavanje XAML stranica u jednu aplikaciju na tradicionalni način, analogno WinForms aplikacijama koje objedinjuju nekoliko formi u jednu izvršnu datoteku.

WPF aplikacije se mogu izvršavati na jedan od sledećih načina:
XAML installed – standardne Windows aplikacije koje se izvršavaju lokalno i korisnički interfejs prikazuju u jednom ili više prozora. Ovakve aplikacije se

izvršavaju u Win prozoru ili u WPF navigacijskom prozoru. WPF navigacijski prozor sadrži navigacijsku traku na vrhu prozora koja sadrži Nazad (Back) i Napred (Forward) komande za jednostavnije kretanje kroz aplikaciju. Navigacijska traka se može posebno implementirati (*override*) upotrebom S*ystem.Windows.Navigation.NavigationWindow* klase.

Kao što je napomenuto, XAML instalirane („*installed*“) aplikacije mogu biti standardne Windows aplikacije ili aplikacije koje se izvršavaju u WPF navigacijskom prozoru.
Kada u Visual Studio okruženju kreiramo standardnu Windows aplikaciju, kreiraju se dve XAML datoteke (sa pozadinskim kodom) koje čine srž
same aplikacije: App.xaml i Window.xaml. One sadrže Application i Window objekat koji su najbitniji delovi Win WPF aplikacije.

Application klasa predstavlja jedinu ulaznu tačku u aplikaciji i stara se o prosleđivanju poruka u aplikaciji (message dispatcher). Takođe, ona sadrži i nekoliko događaja i metoda kao što su StartUp, Exit, Activated, Deactivated i dr. Aplikacija se napušta onog trenutka kada se svi prozori koji su deo nje zatvore ukoliko ShutDown mod nije podešen drugačije – recimo interesantno je podešavanje po kojem se aplikacija gasi tek kada se Shutdown eksplicitno pozove, pa će klik na Close samo spustiti
aplikaciju u System Tray.

Application objekat predstavlja drugu bitnu komponentu WPF *installed* aplikacije. Unutar *Application* klase nalazi se Main() metod koji predstavlja ulaznu tačku aplikacije.
Thread-ovi unutar WPF aplikacije se izvršavaju u tzv. STA („*Single Threaded Apartment*“) okruženju. To znači da Main() metod mora biti obeležen STAThread atributom.
Application klasa takođe definiše i Run() metod koji aplikaciju drži pokrenutom i obrađuje poruke koje operativni sistem šalje MainWindow klasi sve dok se ne pokrene njeno zatvaranje.
Osnovna implementacija izgleda ovako:

```
 [STAThread]
public static void Main()
{
   Application app = new Application();
   MainWindow window = new MainWindow();
   window.Show();
   app.Run(window);
}
```

XAML browser – aplikacije koje se izvršavaju unutar Web browser-a. Takve aplikacije nazivaju se XBAP („*XAML Browser Application*“) i imaju .xbap ekstenziju. Ovakve aplikacije rade samo pod Windows-om, samo u Internet Exploreru i samo ako je na sistemu instaliran .NET *framework* 3.5. Ovakva ograničenja su prevaziđena plugin-om MS Silverlight koji je u toku razvoja bio poznat pod imenom WPF/E („WPF Everywhere“ odnosno „WPF Svuda“). Silverlight predstavlja *framework* nezavisan od platforme i browsera.

2.7 Silverlight

Silverlight je novi Microsoft *framework* koji je zasnovan na WPF-u (originalno ime Silverlighta je bilo WPF/E). Osnovni cilj je pravljenje RIA-aplikacija („*Rich Internet Applications*“, sofisticiranih web aplikacija). Ovo sve je dovelo do web aplikacija koje imaju jednostavnost, performanse i odziv WinForms aplikacija. Od verzije 2.0, ovo okruženje podržava klijentski .NET kôd, što znači da se .NET kôd izvršava u okviru web pretraživača, dopuštajući mnogo više procesiranja na klijentskoj strani bez odlazaka do servera (slično Javascriptu).

Veličina Silverlighta je mala (4.6 mb) i potrebno je malo vremena da se instalira na računaru koji ga nema. Izvršavanje Silverlight aplikacija ne zahteva instaliran .NET *framework*. Programeri mogu da pišu Silverlight aplikacije koristeći bilo koji .NET jezik (C#, VB, Javascript, IronPython i IronRuby). Silverlight poseduje bogat skup osobina koje uključuju:

1. WPF UI *framework*
   Silverlight uključuje bogat UI (*user interface*) *framework* koji omogućava razvijanje aplikacija mnogo jednostavnije. Uključuje moćan grafički i animacijski model, zatim bogatu podršku visokog nivoa UI kao što su kontrole, organizaciju izgleda (*layout*), povezanost kontrola i podataka (*data-binding*), stilove i maske (*skin*). WPF UI *framework* u Silverlight-u je kompatibilni podskup WPF UI karakteristika i omogućava programerima da ponovo iskoriste znanja, kontrole i kôd da bi razvijali bogate web multi-platformske aplikacije kao i napredne Windows aplikacije.

2. Bogat skup kontrola
   Silverlight uključuje bogat skup ugrađenih kontrola koje programeri kao i dizajneri mogu da koriste. Uključuje osnovne kontrole (*TextBox, CheckBox, RadioButton, ComboBox*,...), ugrađene okvire za sofisticiraniji izgled

(*StackPanel, Grid, Panel,..*), standardne kontrole (*Slider, ScrollViewer, Calendar, DatePicker,...*) kao i kontrole za manipulaciju podacima (*DataGrid, ListBox,...*). Sve Silverlight-ove kontrole podržavaju rad sa šablonima (*template*) što omogućava programerima i dizajnerima da zajedno prave krajnje sofisticirana i specifična rešenja.

3. Široka podrška komunikaciji
   Silverlight podržava komunikaciju preko REST, WS/SOAP, POX,RSS i preko standarnog HTTP servisa. Omogućava klijentima direktan pristup resursima na Internetu. Takođe ima ugrađenu podršku za sokete.

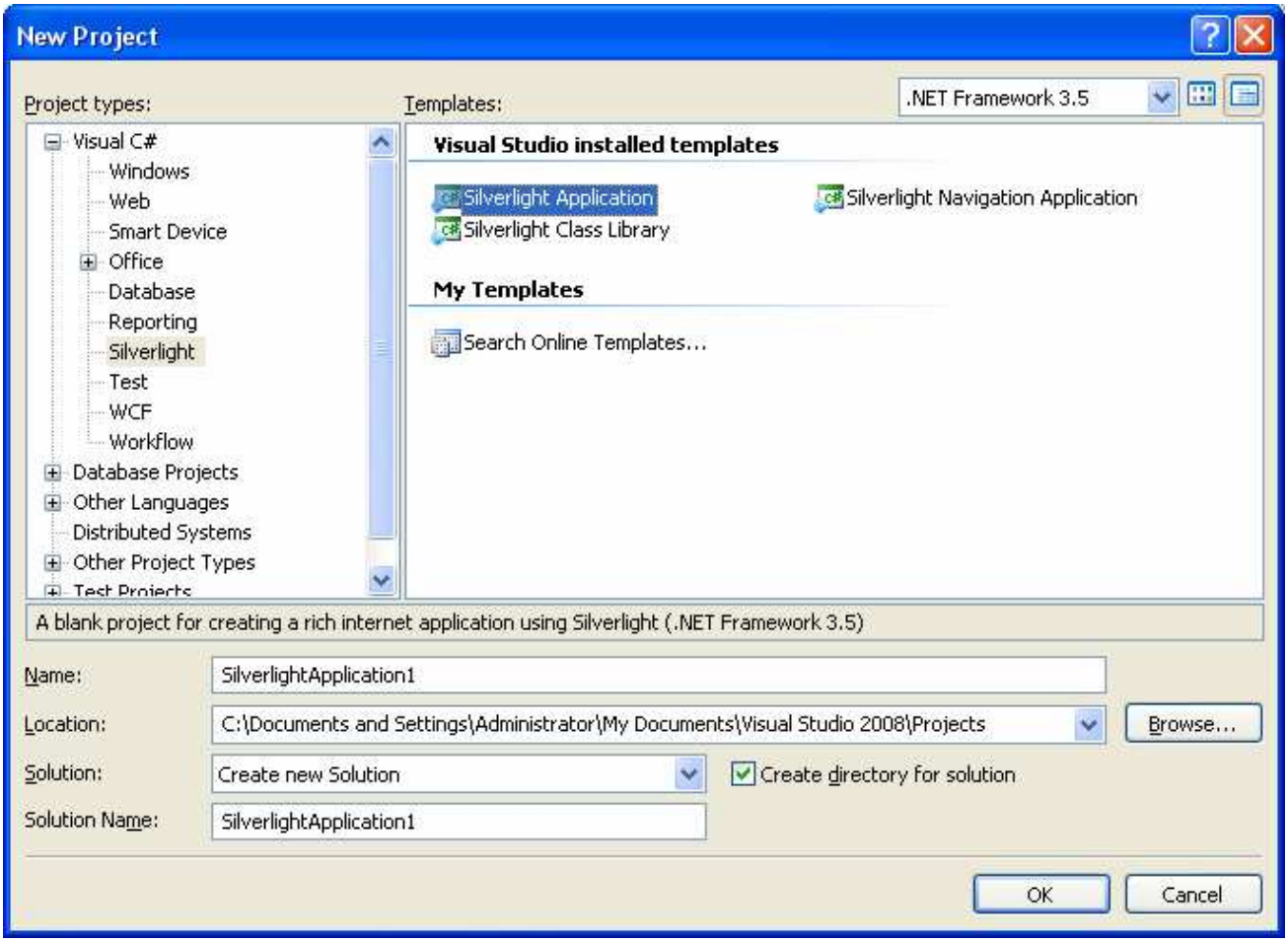
4. Bogata osnovna biblioteka klasa
   Silverlight uključuje bogatu biblioteku osnovnih klasa/funkcionalnosti (kolekcije, ulaz-izlaz, generičnost, niti, globalizacija, xml) . Takođe, omogućuje integraciju html dom/javascripta sa .NET kodom. Uključuje LINQ i LING to XML biblioteku (laka transformacija i upitovanje podataka).

5. Bogatu podršku za multimediju
   Silverlight dolazi sa ugrađenim podrškom za puštanje video materijala visoke definicije, kao i za uživo puštanje video materijala preko interneta (*streaming*, kako *live* tako i na zahtev (*on-demand)).* Podržava promenu video *bitrate*-a u realnom vremenu u zavisnosti od uslova mreže, kao i zaštitu sadržaja.

6. CLR („*Common Language Runtime*“, „Izvršno okruženje .NET-a“)
   Silverlight uključuje prilagođenu verziju CLR-a, zajedno sa osnovnim klasama, *garbage collector*-om (mehanizam za dealokaciju i čišćenje memorije), JIT („*Just-in-time*“, dinamičko kompajliranje) kompajlerom, podrškom za konkurentno izvršavanje (niti) i dr.

Silverlight stremi ka tome da iskombinuje snagu i multi-platformnost Flash-a sa programerskom platformom zasnovanom na konceptima .NET-a.
Flash trenutno ima nadmoć nad Silverlight-om prvenstveno zbog široke i dugogodišnje upotrebe i zrelosti, međutim Silverlight poseduje nekoliko arhitekturalnih prednosti koje Flash ne poseduje.
Najvažnija prednost je to da je zasnovan na prilagođenoj verziji CLR i da omogućava programerima da razvijaju klijentski-kod u C#.
Za programiranje i razvoj Silverlight aplikacija neophodno je da na računaru bude instaliran Microsoft Visual Studio 2008, standardni Microsoft-ov alat za razvoj aplikacija u .NET okruženju, kao i dodatak web pretraživaču za Silverlight (*plug-in*).
Moguće je umesto Visual Studio-a koristiti Expression Blend, koji je više orijentisan ka dizajnerima i aplikacijama sa dosta grafike i animacije, a manje poslovne logike.

Da bismo kreirali Silverlight aplikaciju potrebno je da, nakon pokretanja Visual Studio-ja u meniju *File -> New -> Project*, odaberemo sa leve strane tip projekta Silverlight i unesemo ime projekta, slika 2.
Time smo napravili osnovno okruženje za razvoj ozbiljnih i modernih web aplikacija.



Slika 2

Nakon kreiranja Silverlight projekta, automatski su dodate dve XAML datoteke sa pozadinskim kodom: App.xaml (App.xaml.cs) i Page.xaml (Page.xaml.cs).
App.xaml klasa predstavlja ulaznu tačku aplikacije i sadrži događaje *Startup*, *Exit* i *UnhandledException*. Na sledećoj slici je prikazan sadržaj klase App.xaml.cs.

```
public partial class App : Application
{
    public App()
    {
        this.Startup += this.Application_Startup;
        this.Exit += this.Application_Exit;
        this.UnhandledException += this.Application_UnhandledException;

        InitializeComponent();
    }

    private void Application_Startup(object sender, StartupEventArgs e)
    {
        this.RootVisual = new Page();
    }

    private void Application_Exit(object sender, EventArgs e)
    {

    }
    private void Application_UnhandledException(object sender, ApplicationUnhandledExceptionEventArgs e)
    {
        // If the app is running outside of the debugger then report the exception using
        // the browser's exception mechanism. On IE this will display it a yellow alert
        // icon in the status bar and Firefox will display a script error.
        if (!System.Diagnostics.Debugger.IsAttached)
        {

            // NOTE: This will allow the application to continue running after an exception has been thr
            // but not handled.
            // For production applications this error handling should be replaced with something that wi
            // report the error to the website and stop the application.
            e.Handled = true;
            Deployment.Current.Dispatcher.BeginInvoke(delegate { ReportErrorToDOM(e); });
        }
    }
```

Slika 3

Kao što se na slici vidi u okviru metoda Application_Startup postavlja se početna stranica Page.xaml. Page klasa predstavlja Silverlight korisničku kontrolu (*user control*), tj. nasleđuje klasu UserControl.
Dakle, sam izgled aplikacije ćemo definisati u okviru Page.xaml datoteke, dok ćemo obradu događaja definisati u Page.xaml.cs.

2.8 Ostale korišćene tehnologije

Pored tehnologije WPF/Silverlight, čije upoznavanje je i jedan od ciljeva ovog master rada, za razvoj aplikacije „Namenski računi“ korišćeno je još nekoliko tehnologija novog .NET *framework*-a.

LINQ

LINQ to SQL („*Language-integrated query for relational data*“, jezički-integrisani upiti) predstavlja novu komponentu već postojeće ADO.NET biblioteke za pristup bazi podataka.
LINQ predstavlja okruženje za manipulaciju relacionim podacima kao objektima bez gubitka mogućnosti za postavljanje upita.LINQ radi tako što jezički-integrisane upite konvertuje u SQL upite spremne za izvršavanje nad bazom i obratno, retultate SQL upita pretvara u odgovarajuće .NET objekte.

WCF

WCF („*Windows communication foundation*“) predstavlja novi Microsoft-ov okvir za razvoj servisno-orijentisanih aplikacija, tj. razvoj distribuiranih sistema.
Omogućava razvoj komponenti koje imaju transakcione mogućnosti, pouzdan sistem poruka, nezavisnost od protokola i vremena.
Na taj način se ujedinjuju do sada postojeće tehnologije: Web Services, .NET Remoting, COM+ i druge.

2.9 WPF i Win Forms

Tehnologija WPF nije predstavljena da bi zamenila postojeću i vrlo uspešnu tehnologiju razvoja windows aplikacija Win Forms.
WPF ne predstavlja samo tehnologiju za razvoj vizuelno dopadljivih aplikacija. Ovo je jedna od najvećih zabluda vezanih za WPF.
WPF je odlična platforma za razvoj aplikacija koje koriste razne vrste medija. U slučaju da imamo potrebu za prikazom video snimaka, dokumenata, 3D sadržaja ili animacija onda je WPF idealan izbor. Takođe je dobar izbor ako želimo da kreiramo korisnički interfejs sa temama (*skin).*

Sa druge strane, Win Forms je i dalje aktivna i popularna tehnologija, pogodna za razvoj aplikacija koje nemaju potrebu za modernim funkcionalnostima WPF-a. Takođe, postoji mnogo više kontrola razvijenih od strane različitih firmi (*3rd party controls), online* resursa i programerskih zajednica za Win Forms, nego za WPF.

Pored svih razlika i situacija u kojima treba koristiti jednu ili drugu tehnologiju, moguće je razvijati aplikacije koje koriste obe tehnologije.

Kao zaključak se nameće činjenica da će Win Forms trajati još nekoliko godina, jer postoji dosta poslovnih aplikacija (LOB, „*Line Of Business Applications*“) razvijenih u Win Forms tehnologiji, a da će razvoj WPF aplikacija doživeti svoj procvat tek u narednom periodu.

III APLIKACIJA NAMENSKI RAČUNI

3.1 Opis i namena aplikacije

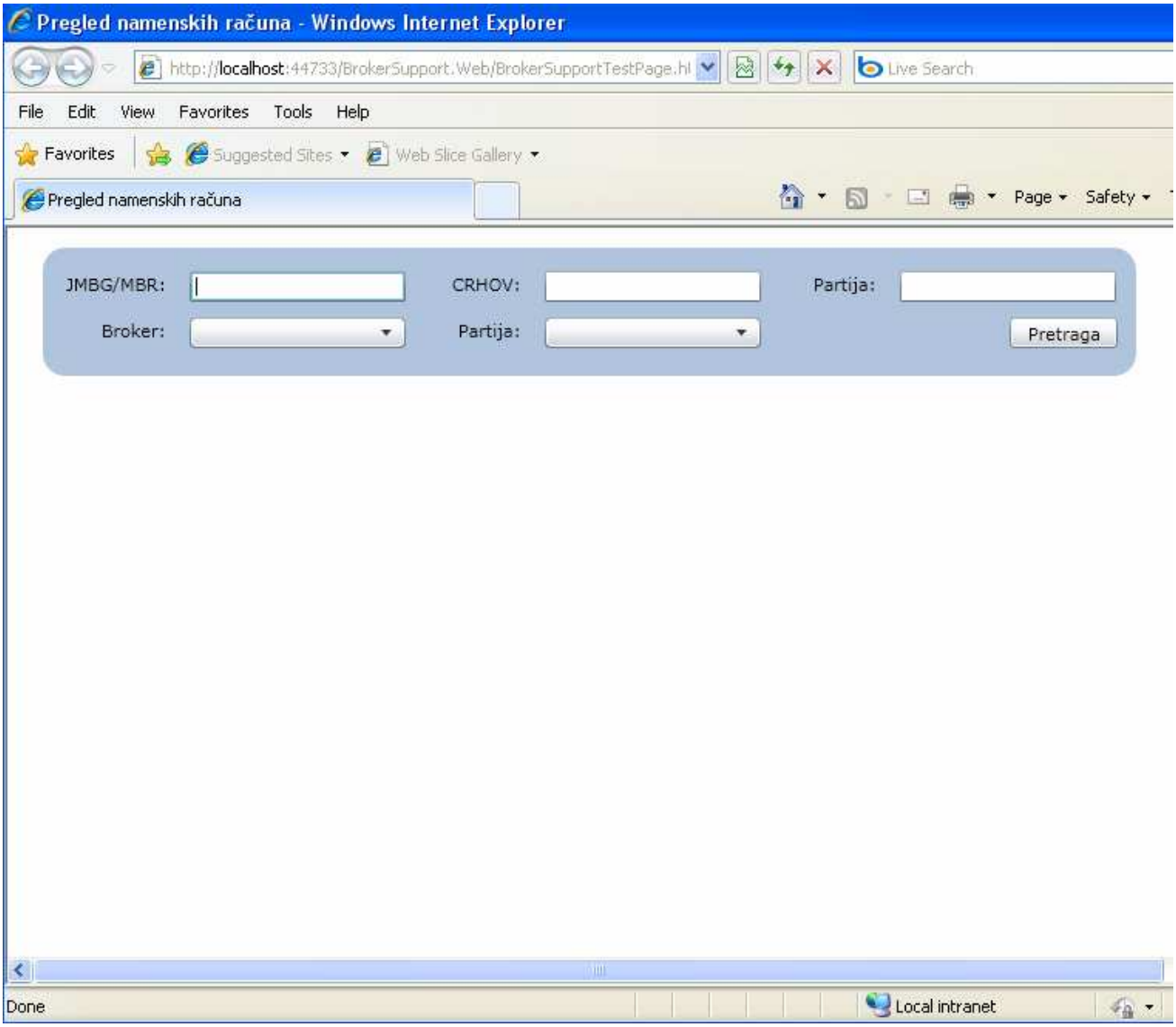
Aplikacija „Namenski računi" predstavlja web aplikaciju namenjenu brokerskim kućama i ona omogućava uvid u novčano pokriće transakcija kupovine HOV („Hartije od vrednosti") klijenata brokerske kuće, koji imaju otvoren namenski novčani račun u Komercijalnoj Banci.

Aplikacija omogućava uvid u raspoloživa sredstva (stanje na računu umanjeno za rezervisana sredstva po osnovu realizovanih a nesaldiranih transakcija, prema zaključnicama-transakcijama CRHOV-a („Centralnog registra hartija od vrednosti")), kao i štampu potvrde o stanju sredstava.

Pretragom korisnika po matičnom broju ili po broju partije prikazuju se podaci podeljeni u nekoliko delova.
Podaci o korisniku prikazuju osnovne podatke o korisniku uz mogućnost prikaza i štampanja Potvrde o stanju sredstava na namenskom računu.
Podaci o računu, kao što i naziv kazuje, prikazuju osnovne podatke o računu kao i promet po partiji, rezervisana sredstva po partiji i stanje partije po valutama.
I poslednji deo sadrži osnovne podatke o brokeru.

3.2 Pretraga korisnika

Aplikacija „Namenski računi” predstavlja web aplikaciju kojoj se pristupa i izvršava u okviru web pretraživača (*web browser*-a).
Na sledećoj slici je prikazana početna strana aplikacije.



Slika 4

Omogućena su tri načina za upit stanja po klijentu tj. računu:

1. Unosom matičnog broja klijenta (fizičko/pravno lice) u odgovarajuće polje (pored natpisa JMBG/MBR).

2. Unosom matičnog broja (kastodi ID-a) pod kojim se klijent vodi u centralnom registru hartija od vrednosti (pored natpisa CRHOV)

3. Unosom broja partije u odgovarajuće polje (pored natpisa partija)

Ovde je značajno napomenutio sledeće:

- Ukoliko je MB isti i u Komercijalnoj banci i u CRHOV, pretraga se može vršiti bilo po partiji bilo po MB u Komercijalnoj banci
- Ukoliko postoji veza 1:1 za matične brojeve u Komercijalnoj banci i CRHOV, pretraga se može vršiti bilo po partiji, bilo po matičnom broju u Komercijalnoj banci, bilo po matičnom broju CRHOV-a
- Ukoliko postoji veza 1:n za matične brojeve u Komercijalnoj banci i CRHOV, pretraga isključivo mora da ide preko CRHOV MB. U protivnom sistem ne bi mogao da odluči za koji CRHOV MB se traže rezervisana sredstva.

Dakle, za kreiranje početnog izgleda aplikacije, kao na prethodnoj slici, u okviru Page.xaml datoteke dodaćemo *Grid* kontrolu. *Grid* kontrola u Silverlight-u predstavlja pandan html tabeli, dakle služi za organizovanje izgleda (*layout*) aplikacije.
Na sledećoj slici se nalazi *Grid* kontrola dodata u Page.xaml datoteku.

```
<Border CornerRadius="15" BorderThickness="1" Height="85"  Width="710"  Background="LightSteelBlue"
    <Grid x:Name="LayoutRoot" Background="LightSteelBlue" Grid.Column="6" Grid.Row="2" ShowGridLines=
        <Grid.RowDefinitions>
            <RowDefinition Height="30"></RowDefinition>
            <RowDefinition Height="30"></RowDefinition>
            <RowDefinition ></RowDefinition>
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="80"/>
            <ColumnDefinition Width="150"/>
            <ColumnDefinition Width="80"/>
            <ColumnDefinition Width="150"/>
            <ColumnDefinition Width="80"/>
            <ColumnDefinition Width="150"/>
        </Grid.ColumnDefinitions>
        <TextBlock Text="JMBG/MBR:" Grid.Row="0" Grid.Column="0" HorizontalAlignment="Right" Padding=
        <TextBox x:Name="txtJMBG" Grid.Row="0" Grid.Column="1" Height="20" Width="140" ></TextBox>
        <TextBlock Text="CRHOV:" Grid.Row="0" Grid.Column="2" Width="auto" HorizontalAlignment="Right
        <TextBox x:Name="txtCRHOV" Grid.Row="0" Grid.Column="3" Height="20" Width="140" ></TextBox>
        <TextBlock Text="Partija:" Grid.Row="0" Grid.Column="4" Width="auto" HorizontalAlignment="Rig
        <TextBox x:Name="txtPartija" Grid.Row="0" Grid.Column="5" Height="20" Width="140"></TextBox>

        <TextBlock Text="Broker:" Grid.Row="1" Grid.Column="0" HorizontalAlignment="Right" Padding="2
        <ComboBox x:Name="cmbBroker" Grid.Row="1" Grid.Column="1" Height="20" Width="140" SelectionCh
        <TextBlock Text="Partija:" Grid.Row="1" Grid.Column="2" Width="auto" HorizontalAlignment="Rig
        <ComboBox x:Name="cmbPartija" Grid.Row="1" Grid.Column="3" Height="20" Width="140" SelectionC
        <Button Grid.Row="1" Name="btnPretraga" Grid.ColumnSpan="6" Grid.Column="6" Width="70" Horizo
            <TextBlock Text="Pretraga"></TextBlock>
        </Button>
    </Grid>
</Border>
```

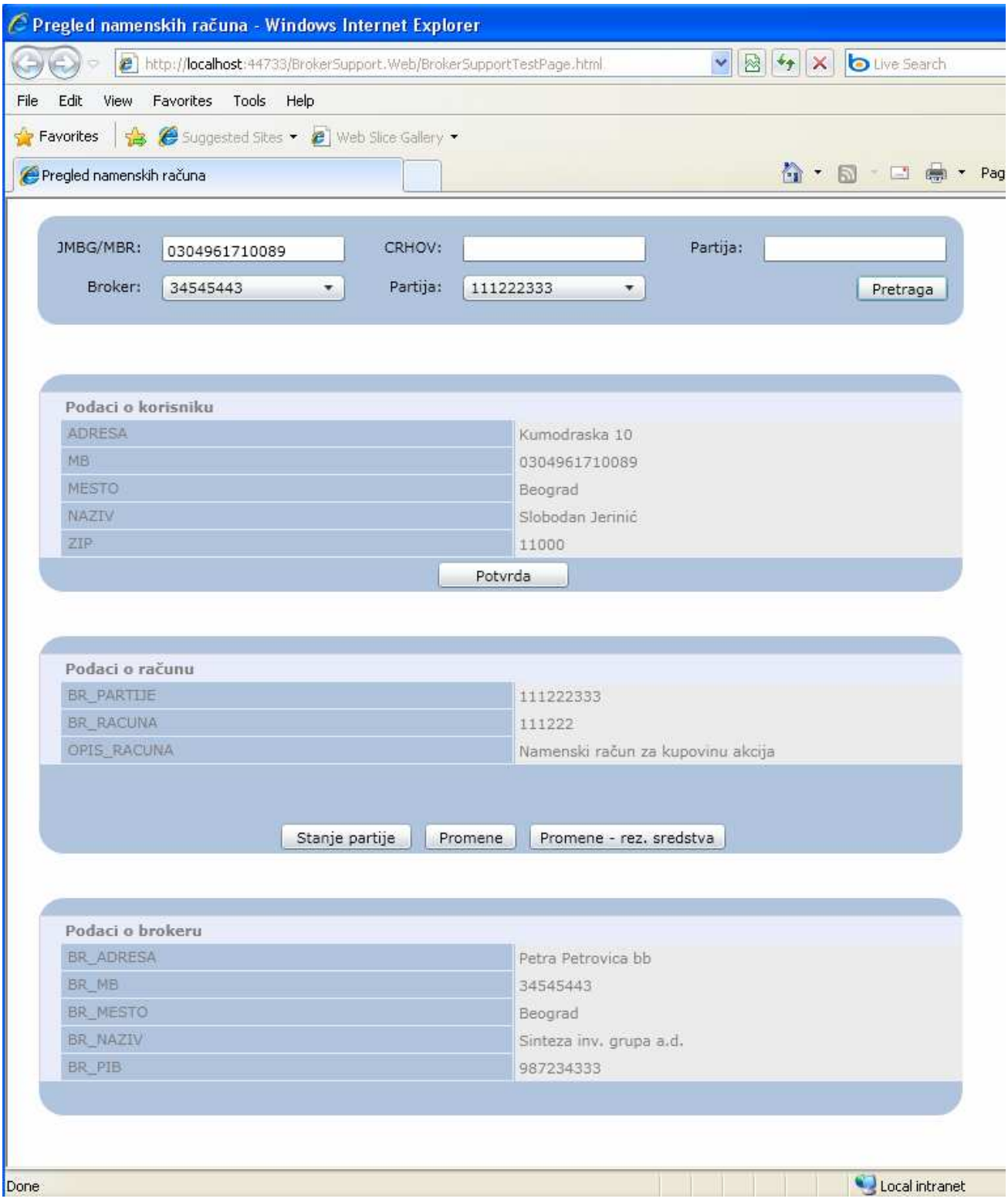
Slika 5

Pre dodavanja *Grid* kontrole, dodali smo *Border* kontrolu za definisanje ivica *Grid* kontrole, koje su u ovom slučaju blago zakrivljene (*rounded corners*). Broj redova i broj kolona u okviru *Grid* kontrole definiše se pomoću *Grid.RowDefinitions* i *Grid.ColumnDefinitions* svojstva.
Početni *Grid* je definisan sa 3 reda i 6 kolona. U okviru svakog elementa (*cell*) *Grid*-a dodata je odgovarajuća kontrola (*TextBlock*, *TextBox*, *Button*). Način na koji se kontrola dodaje u odgovarajući element *Grid*-a se realizuje preko pridodatih svojstava („*attached properties*“). Kao što se na slici 5 može videti, npr. prvi *TextBlock* ima pridodata svojstva, dakle svojstva koja ne pripadaju *TextBlock*-u, već *Grid*-u, *Grid.Row=”0”* i *Grid.Column=”0”*, čime se *TextBlock* postavlja u prvi red i prvu kolonu *Grid*-a.

Klikom na dugme „Pretraga“ prikazuju se u padajućim listama spisak registrovanih brokera za klijenta, kao i spisak partija registrovanih na odgovarajućeg brokera. Automatski se selektuju prve vrednosti iz liste i prikazuju se odgovarajući detaljni podaci koje možemo podeliti na:

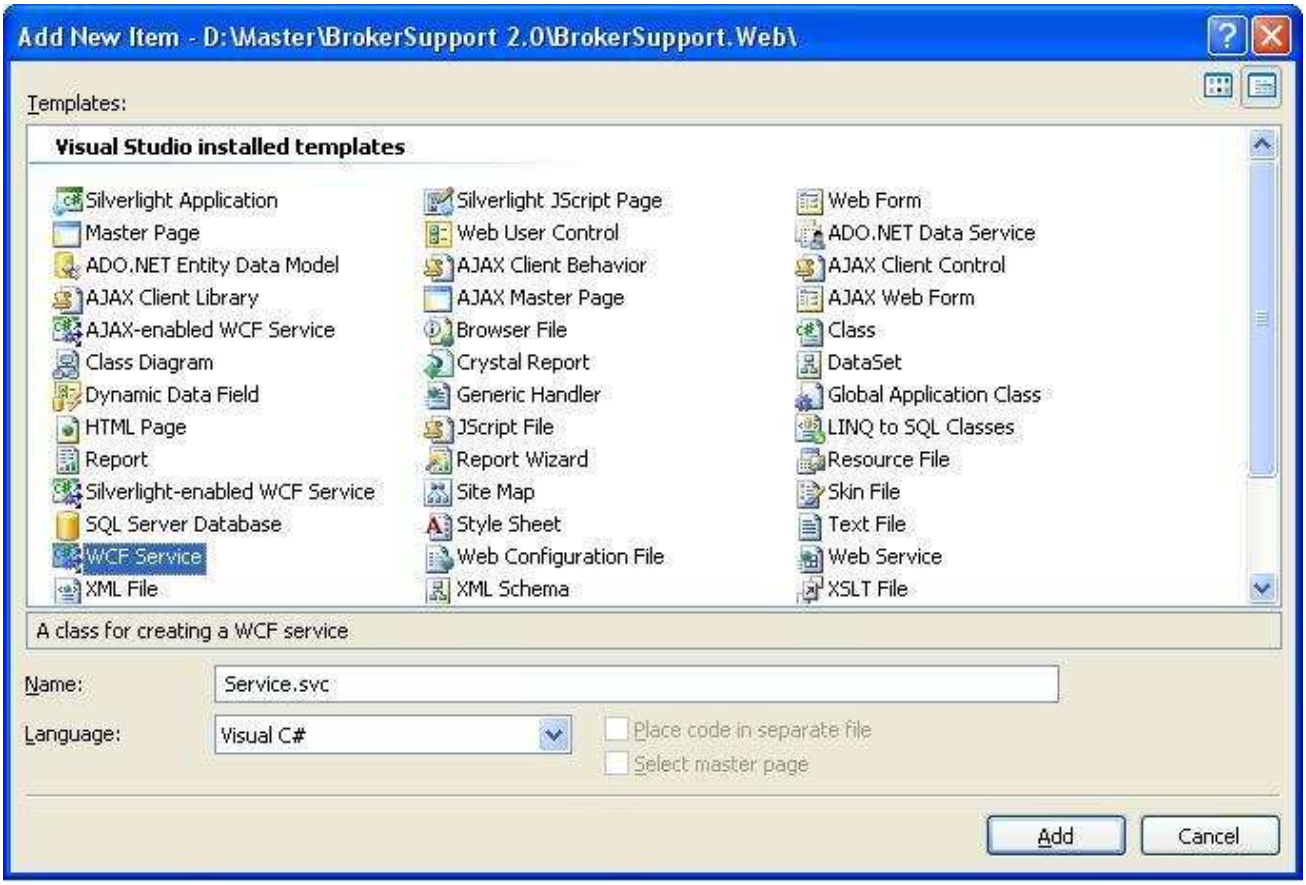
- Podaci o korisniku
- Podaci o računu
- Podaci o brokeru

Na sledećoj slici se nalazi strana sa rezultatom pretrage, slika 6.



Slika 6

U nastavku opisaćemo kako se preko WCF-a i LINQ-a dohvataju podaci iz baze podataka i zatim prikazuju na silverlight kontrolama.
WCF servis se dodaje u projekat klikom na *Add New Item* i odabirom odgovarajućeg šablona (template), kao na slici 7.



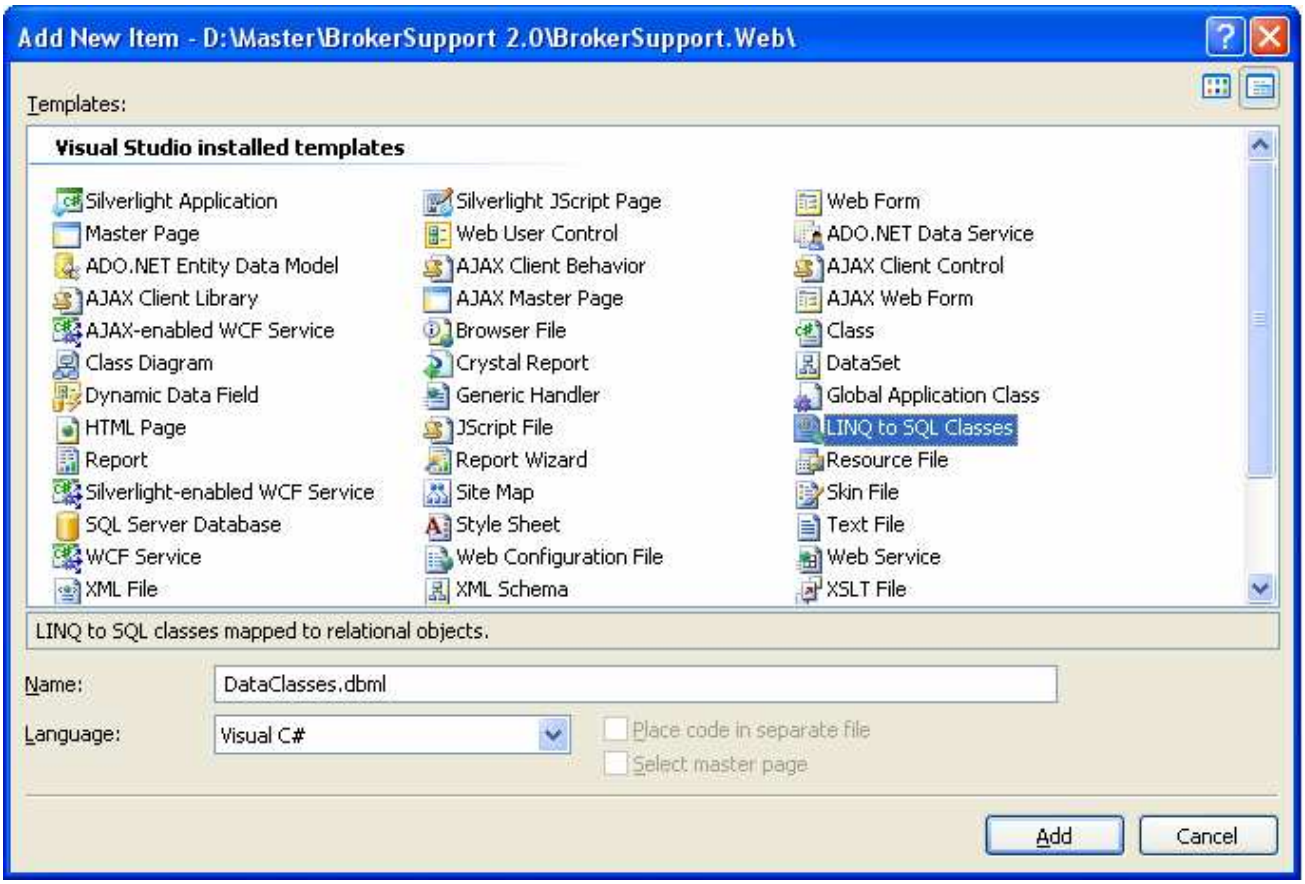
Slika 7

U okviru servis klase definišu se metodi za pristup i ažuriranje baze podataka, kao na slici 8.

```
[ServiceContract(Namespace = "")]
[AspNetCompatibilityRequirements(RequirementsMode = AspNetCompatibilityRequirementsMode.Allowed)]
public class Service
{
    [OperationContract]
    public KORISNIK GetClientDetailsByMB(string mb)
    {
        DataClassesDataContext datacontext = new DataClassesDataContext();

        return (from korisnik in datacontext.KORISNIKs
                        where korisnik.MB == mb
                        select korisnik).First();
    }
```

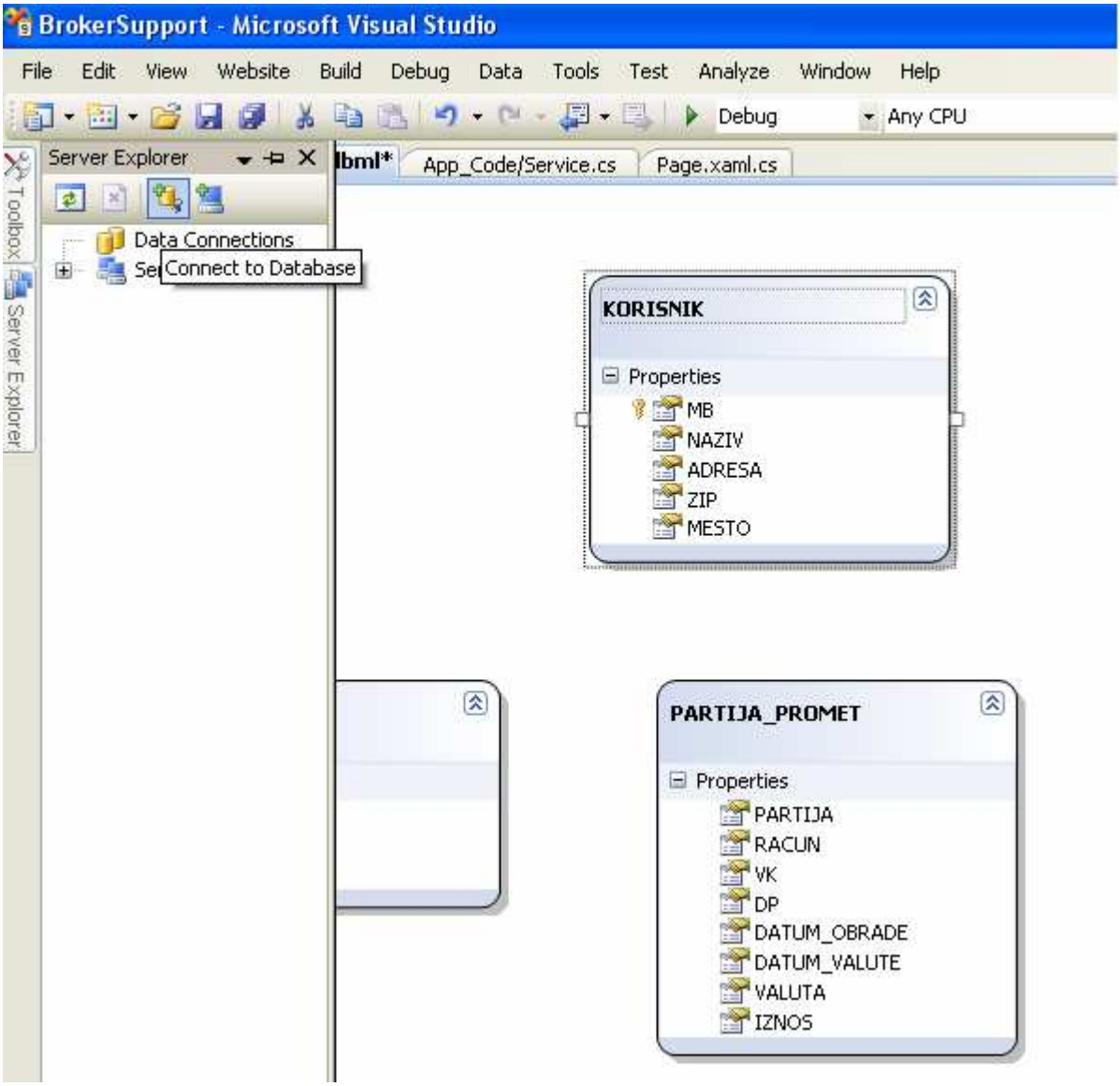
Slika 8

Metod *GetClientDetailsByMB* na osnovu matičnog broja klijenta dohvata preko LINQ-a podatke o tom klijentu iz baze.
Atribut *ServiceContract* ukazuje da je ovaj WCF servis moguće koristiti i pristupati mu iz drugih aplikacija, i to samo onim metodima u okviru servisa koji imaju atribut *OperationContract*.
Povratna vrednost metoda *GetClientDetailsByMB* je klasa KORISNIK, koja predstavlja LINQ klasu i kreira se kada u projektu kliknemo na *Add New Item* i odaberemo odgovarajući šablon (*template*) kao na slici 9.



Slika 9

Zatim se u okviru *Server Explorer*-a, klikne na *Connect to Database* i unesu se odgovarajući parametri povezivanja na bazu (ime servera, ime baze, korisničko ime i lozinka) i jednostavnim prevlačenjem tabela dobijaju se C# klase KORISNIK, PARTIJA_PROMET i druge, kao na slici 10.



Slika 10

Sada, kada smo napisali WCF servis koji preko LINQ-a dohvata podatke iz baze podataka, ostaje da te podatke prikažemo preko neke silverlight kontrole.
U okviru glavne silverlight stranice, Page.xaml.cs, definisaćemo WCF servis. Pozivi metoda WCF servisa se odvijaju asinhrono tako da je potrebno da definišemo povratne (*callback*) funkcije za obradu rezultata tih poziva, slika 11.

```
public partial class Page : UserControl
{
    public BrokerSupportService.ServiceClient sc = null;

    public Page()
    {
        InitializeComponent();

        sc = new BrokerSupport.BrokerSupportService.ServiceClient();
        sc.GetClientDetailsByMBCompleted += new EventHandler
                                            <BrokerSupport.BrokerSupportService.GetClientDetailsByMBCompletedEventArgs>
                                            (sc_GetClientDetailsByMBCompleted);
```

Slika 11

Kao što se vidi sa slike 11, naziv povratne (*callback*) funkcije, koja će obraditi rezultat poziva metoda *GetClientDetailsByMB*, je *sc_GetClientDetailsByMBCompleted*.

U okviru funkcije *GetClientDetailsByMBCompleted*, podatke o korisniku ćemo prikazati preko silverlight kontrole.U ovom slučaju izabrana je silverlight kontrola *PropertyGrid* koja se ne nalazi u osnovnoj kolekciji silverlight kontrola, već je preuzeta sa CodePlex-a i prilagođena potrebama aplikacije. CodePlex je Microsoft-ova zajednica projekata otvorenog koda („*Open Source Project Community*“) na kojoj ima dosta zanimljivih i korisnih aplikacija i kontrola za Microsoft platformu. Definicija funkcije *sc_GetClientDetailsByMBCompleted* prikazana je na slici 12.

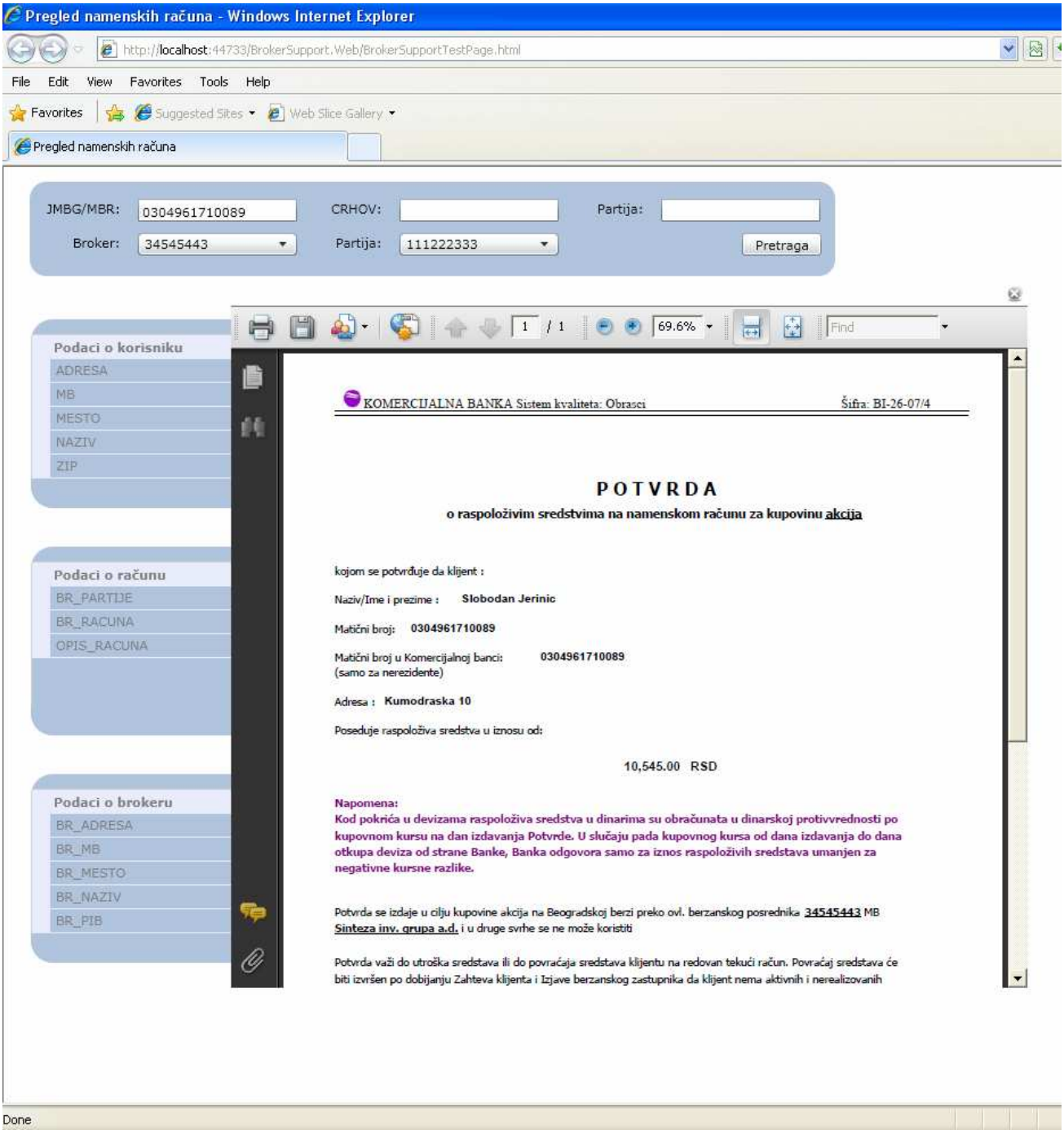
```
private void sc_GetClientDetailsByMBCompleted(object sender, GetClientDetailsByMBCompletedEventArgs e)
{
    KORISNIK korisnik = e.Result;

    this.pgClientData.Height = 135;
    this.pgClientData.headline = "Podaci o korisniku";
    this.pgClientData.SelectedObject = korisnik;
    this.brdClientData.Height = 170;
}
```

Slika 12

U okviru funkcije, *PropertyGrid* kontroli postavljamo visinu, naslov i objekat koji će prikazati, tj. korisnika.

3.3 Podaci o korisniku

Podaci o korisniku, čije stanje se pregleda, prikazuju se u posebnom delu. Osnovne informacije koje se prikazuju jesu naziv korisnika, matični broj, mesto, adresa i poštanski broj.
U okviru ovog dela postoji dugme „Potvrda“.
Klikom na na dugme „Potvrda“, korisniku se otvara potvrda o stanju na računu, za štampu. Potvrda predstavlja formu na kojoj se prikazuje potvrda o stanju na namenskom računu, u skladu sa sistemom kvaliteta obrazaca banke, definisanim od strane odeljenja brokersko-dilerskih („*custody*“) poslova.
Na potvrdi se nalaze osnovni podaci o korisniku, stanje na računu, kao i podaci o brokeru, datum i mesto. Potvrda važi bez potpisa.
Potvrda se otvara u potpuno funkcionalnom PDF-čitaču u kome je moguće sačuvati potvrdu, odštampati i dr (slika 13).

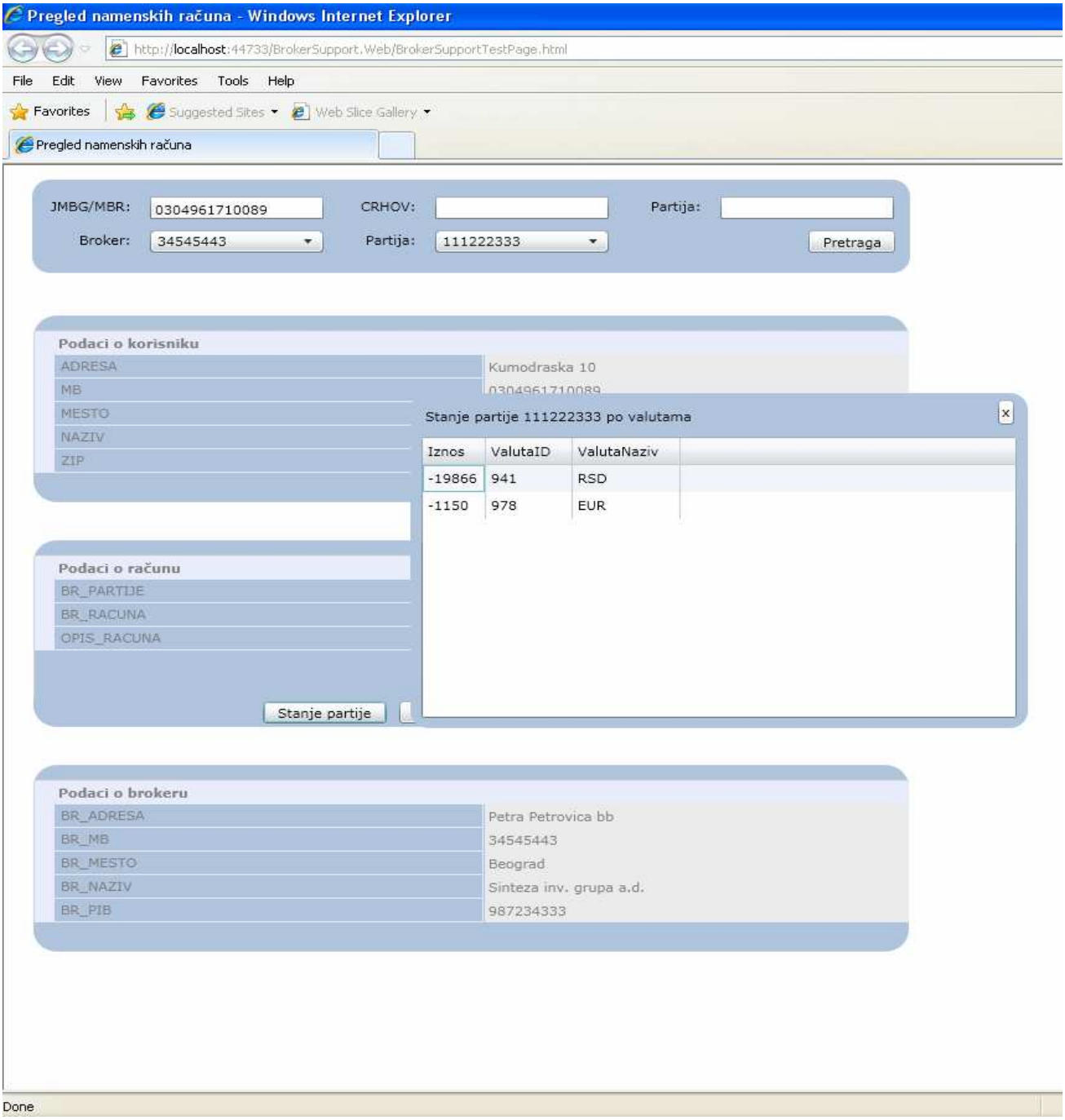

Pregled namenskih računa - Windows Internet Explorer
http://localhost:44733/BrokerSupport.Web/BrokerSupportTestPage.html
File Edit View Favorites Tools Help
Favorites Suggested Sites Web Slice Gallery
Pregled namenskih računa
JMBG/MBR: 0304961710089
CRHOV:
Partija:
Broker: 34545443
Partija: 111222333
Pretraga
Podaci o korisniku
ADRESA
MB
MESTO
NAZIV
ZIP
Podaci o računu
BR_PARTIJE
BR_RACUNA
OPIS_RACUNA
Podaci o brokeru
BR_ADRESA
BR_MB
BR_MESTO
BR_NAZIV
BR_PIB
1 / 1
69.6%
Find
KOMERCIJALNA BANKA Sistem kvaliteta: Obrasci
Šifra: BI-26-07/4
POTVRDA
o raspoloživim sredstvima na namenskom računu za kupovinu akcija
kojom se potvrđuje da klijent :
Naziv/Ime i prezime : Slobodan Jerinic
Matični broj: 0304961710089
Matični broj u Komercijalnoj banci: 0304961710089
(samo za nerezidente)
Adresa : Kumodraska 10
Poseduje raspoloživa sredstva u iznosu od:
10,545.00 RSD
Napomena:
Kod pokrića u devizama raspoloživa sredstva u dinarima su obračunata u dinarskoj protivvrednosti po kupovnom kursu na dan izdavanja Potvrde. U slučaju pada kupovnog kursa od dana izdavanja do dana otkupa deviza od strane Banke, Banka odgovora samo za iznos raspoloživih sredstava umanjen za negativne kursne razlike.
Potvrda se izdaje u cilju kupovine akcija na Beogradskoj berzi preko ovl. berzanskog posrednika 34545443 MB
Sinteza inv. grupa a.d. i u druge svrhe se ne može koristiti
Potvrda važi do utroška sredstava ili do povraćaja sredstava klijentu na redovan tekući račun. Povraćaj sredstava će biti izvršen po dobijanju Zahteva klijenta i Izjave berzanskog zastupnika da klijent nema aktivnih i nerealizovanih
Done

Slika 13

3.4 Podaci o računu

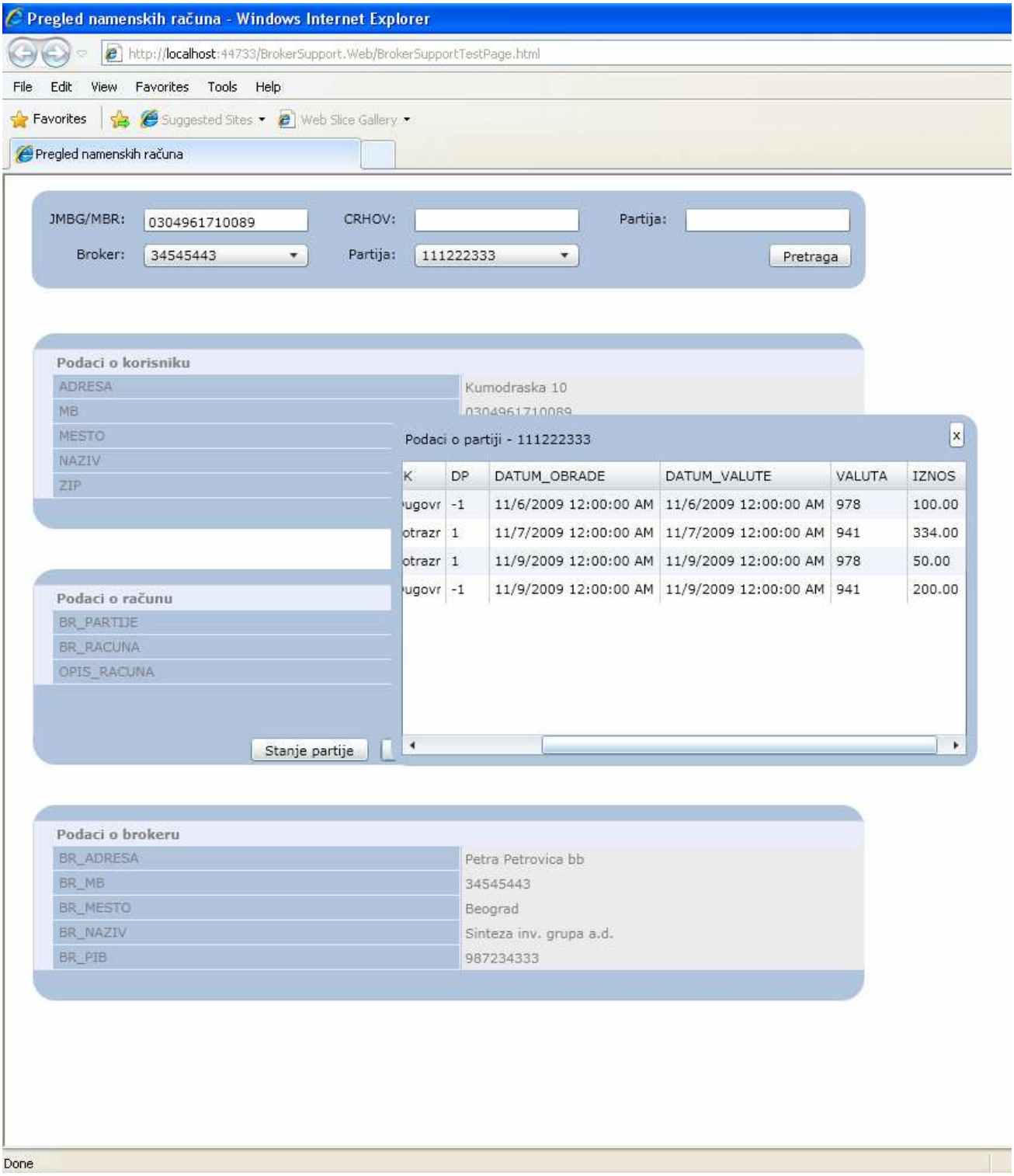
Podaci o računu sadrže podatke po izabranoj partiji ili sumarno po svim partijama ukoliko partija pripada odgovarajućem tipu računa.
U okviru ovog dela postoje sledeće opcije:

- Stanje na partiji po valutama u realnom iznosu, bez kursiranja,slika 14



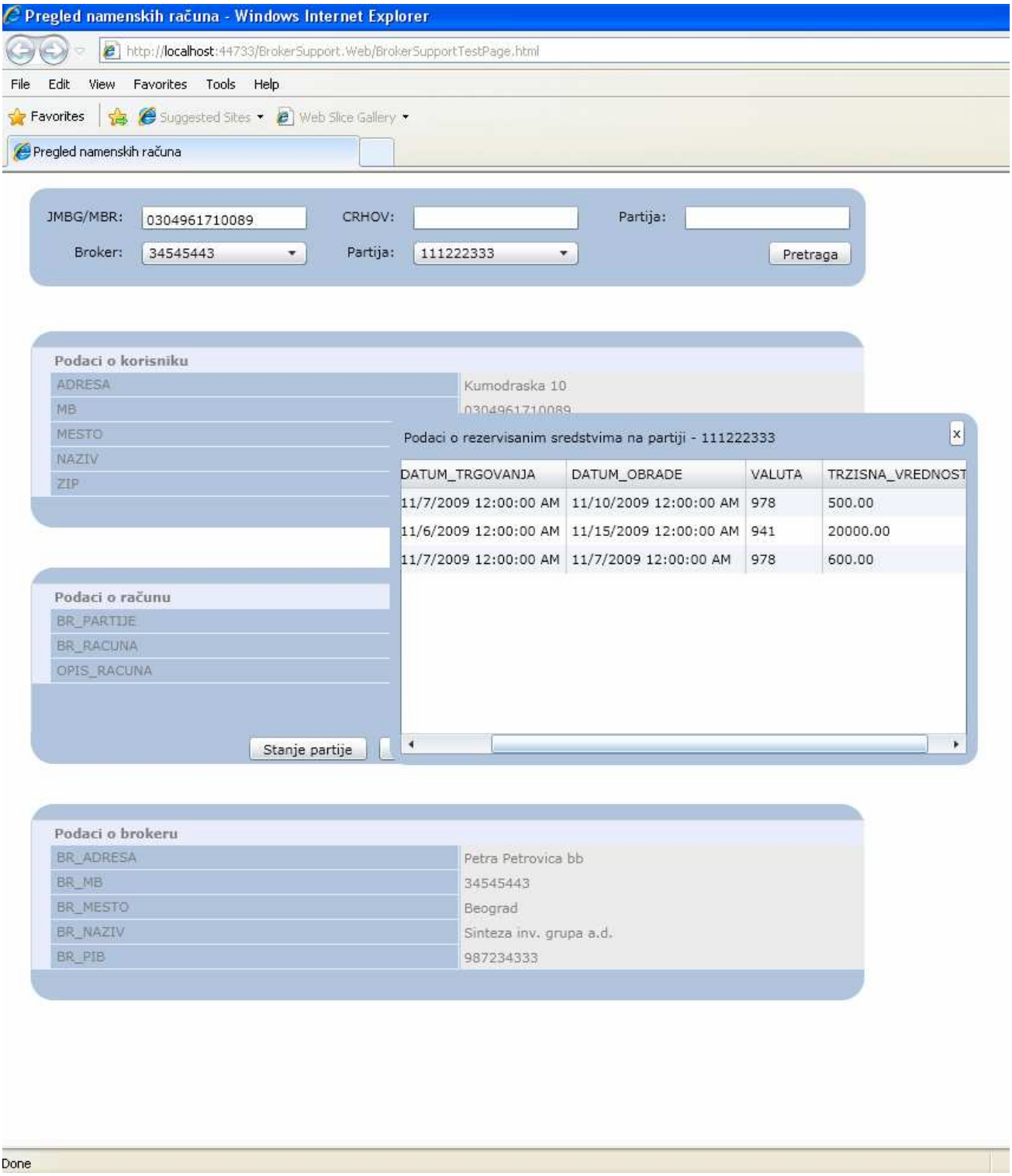
Slika 14

- Istorijski pregled promena po partiji, slika 15



Slika 15

- Istorijski pregled promena rezervisanih sredstava po partiji, slika 16



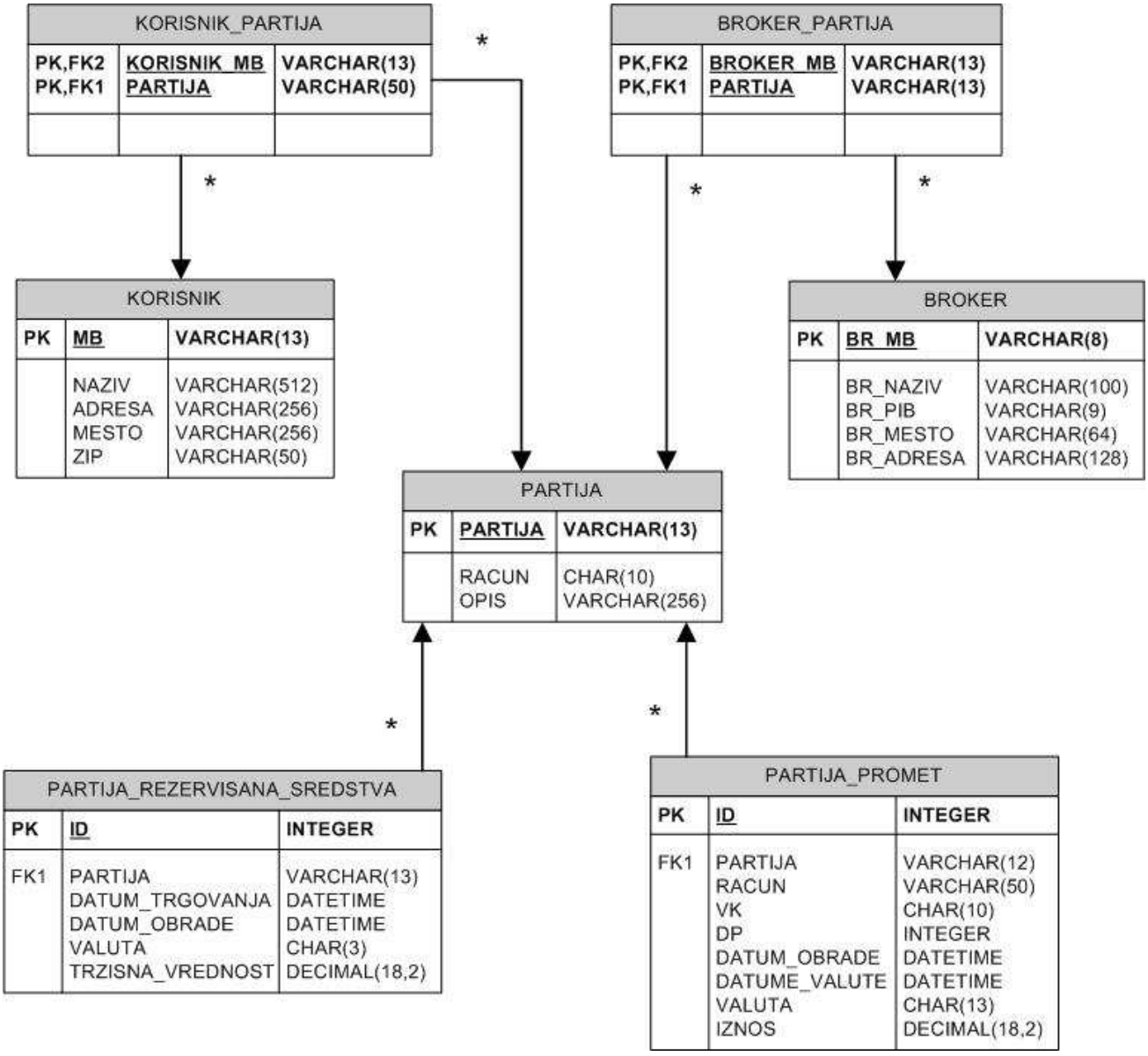
Slika 16

3.5 Podaci o brokeru

Broker predstavlja berzanskog posrednika, koji ima pristup berzi i ovlašćen je da skapa berzanske poslove u ime nalogodavca. Broker ne snosi nikakav rizik za ishod trgovanja, već samo nalogodavac.
Podaci o brokeru sadrže osnovne podatke o brokeru, kao što su naziv brokera, pib, matični broj, mesto i adresa.

3.6 Model baze podataka

Dijagram koji predstavlja model podataka za aplikaciju “Namenski računi” predstavljen je na slici . Veze među tabelama predstavljaju standardne relacione odnose roditelj-dete, slika 17.



Slika 17

IV ZAKLJUČAK

Uvođenjem WPF-a Microsoft je promenio dosadašnji način razvoja kako windows tako i internet (*web*) aplikacija. Sada se jasno razdvajaju razvoj bogatog i sofisticiranog korisničkog interfejsa na koji su dizajneri fokusirani, od razvoja ključnih segmenata u okviru informacionog sistema, kao što je poslovna logika, sigurnost, manipulacija podacima, što je zadatak programera.

Silverlight, kao tehnologija koja je zasnovana na WPF-u, predstavlja odličan Microsoft-ov odgovor trenutnom vladaocu na polju razvoja modernih i brzih web aplikacija, Flash-u (Flex-u), koji se koristi u preko 90% web pretraživača i postoji više od 10 godina.

Softversko rešenje “Namenski računi” je univerzalna aplikacija koju mogu da koriste sve brokerske kuće, uključujući i banke, kao i njihovi klijenti, obezbeđujući uvid u kompletan portfolio klijenta.

Uz moderne tehnologije i okruženja koje nudi Microsoft, moguće je razvijati vrlo složene veb aplikacije sa komplikovanim poslovnim procesima koje su u isto vreme lake za korišćenje i vrlo performantne što se tiče brzine i sigurnosti.

V LITERATURA

1. Chris Andrade, Shawn Livermore – WPF Programming, Wrox 2007
2. Chris Anderson – Essential WPF, Addison Wesley 2007
3. Matthew MacDonald – Pro Siverlight 3 in C#, Apress 2009
4. WPF Wiki - http://en.wikipedia.org/wiki/Windows_Presentation_Foundation
5. MSDN WPF - http://msdn.microsoft.com/en-us/library/ms754130.aspx
6. Silverlight homepage - http://silverlight.net/
7. Silverlight Brad Abrams blog - http://blogs.msdn.com/brada/archive/tags/Silverlight/default.aspx

VI PRILOG

6.1 XAML kod

XAML kod koji definiše deo korisničkog interfejsa aplikacije, deo vezan za podatke o računu korisnika.

```
<Border x:Name="brdPodaciRac" CornerRadius="20" BorderThickness="2" Height="0"
        Width="710"  Background="LightSteelBlue"  Margin="10,10,10,10"
        VerticalAlignment="Top" Grid.Row="2" Grid.Column="0" >
  <Grid ShowGridLines="False" >
         <Grid.RowDefinitions>
            <RowDefinition Height="140"></RowDefinition>
            <RowDefinition Height="23"></RowDefinition>
         </Grid.RowDefinitions>
         <Grid.ColumnDefinitions>
            <ColumnDefinition Width="710"/>
         </Grid.ColumnDefinitions>

         <slpg30:PropertyGrid  x:Name="pgPodaciRac" HorizontalAlignment="Left"
                  VerticalAlignment="Stretch" IsTabStop="False" Height="135"
                  Width="710" UseLayoutRounding="True" Grid.Row="0"
                  Grid.Column="0"
                  Margin="0,13,0,0">
         </slpg30:PropertyGrid>

         <StackPanel Orientation="Horizontal" Grid.Row="1" Grid.Column="0"
                  HorizontalAlignment="Center">
            <Button Name="btnStanje" Width="100" Height="20"  Margin="5,3,5,0"
                 Click="btnStanje_Click">
                <TextBlock Text="Stanje partije"></TextBlock>
            </Button>
            <Button  Name="btnPromene" Width="70" Height="20"  Margin="5,3,5,0"
                  Click="btnPromene_Click">
                <TextBlock Text="Promene"></TextBlock>
            </Button>
            <Button Name="btnPromeneRezSredstva" Width="150" Height="20"
                  Margin="5,3,5,0" Click="btnPromeneRezSredstva_Click">
                <TextBlock Text="Promene - rez. sredstva"></TextBlock>
            </Button>
         </StackPanel>
  </Grid>
</Border>
```