Internet aplikacija za upravljanje ljudskim resursima

Loznica, 2011.

Tema: Internet aplikacija za upravljanje ljudskim resursima

Zadatak:.

1.) Opisati razvoj Web aplikacije „Kristal“ za upravljanje ljudskim resursima.

2.) Opisati tehnologiju izabranu za razvoj Web aplikacije „Kristal“.

3.) Opisati procese upravljanja ljudskim resursima koji su implementirani u Web aplikaciju „Kristal“.

.

Internet aplikacija za upravljanje ljudskim resursima

Sažetak: У овом раду је описан поступак развоја Интернет апликације за управљање људским ресурсима „Кристал“, употребом скупа .НЕТ технологија. Описане су карактеристике коришћених технологија и предности коришћеног приступа у односу на друге могућности. Поред тога, описани су процеси из области управљања људским ресурсима који су имплементирани у ову апликацију, на првом месту управљање каријером, лични план развоја и управљање временом. Кроз рад су разматране могућности и правци даљег развоја ове апликације.

Кључне речи: .NET, ASP.NET, ADO.NET, Upravljanje ljudskim resursima

Internet based application for human resource management

Abstract: In this work is description of the development process of Internet application for human resource management named „Kristal“, done by using the .NET technology. The characteristics of used technology and benefits of used access compared to other options described. Except that, processes in the field of human resource management which are implemented in this application are also described, career management first, than personal development plan and time management. Through this work the possibilities and directions for the future development of this application are considered.

Keywords: .NET, ASP.NET, ADO.NET, Human resource management

SADRŽAJ

1 Uvod ..... - 4 -

2 .NET Framework ..... - 6 -

2.1 Web programiranje ..... - 6 -

2.1.1 Serverski i klijentski programski model ..... - 7 -

2.1.2 Međujezik ..... - 8 -

2.1.3 Aplikacioni događaji ..... - 8 -

2.1.4 Konfiguracione XML datoteke ..... - 9 -

3 Razvoj Internet aplikacije „Kristal“ ..... - 11 -

3.1 ADO.NET ..... - 12 -

3.1.1 Baze podataka u Web aplikacijama ..... - 12 -

3.1.2 SQL Server ..... - 13 -

3.1.3 Provajderi podataka ..... - 15 -

3.1.4 Unos podataka ..... - 15 -

3.1.5 Pristup podacima ..... - 16 -

3.1.6 Prikazivanje podataka ..... - 16 -

3.2 Datoteke ..... - 18 -

3.3 Fokus ..... - 19 -

3.4 Konfigurisanje stanja sesije ..... - 19 -

3.4.1 Životni vek ..... - 20 -

3.5 Obrada grešaka ..... - 21 -

3.5.1 Obrada izuzetaka ..... - 21 -

3.6 Postavljanje ASP.NET aplikacije ..... - 22 -

3.6.1 Web server ..... - 22 -

3.6.2 Virtuelni direktorijum ..... - 23 -

3.6.3 URL adresa Web aplikacije ..... - 24 -

3.6.4 Web farme ..... - 24 -

3.7 Zaštita Web aplikacije „Kristal“ ..... - 25 -

3.7.1 XML ..... - 26 -
3.7.2 Parametri datoteke *Web.config* ..... - 27 -
3.7.3 Skladištenje podataka o korisnicima ..... - 29 -
3.7.4 Sistem zaštite zasnovan na korisničkim nalozima ..... - 30 -
3.8 Komponente ..... - 30 -
3.8.1 Komponenta *KristalBiblioteka.dll* ..... - 31 -
3.8.2 Klase sa pamćenjem stanja i klase bez pamćenja stanja ..... - 33 -
3.9 Keširanje ..... - 33 -
3.9.1 Kada koristiti keširanje ..... - 34 -
3.9.2 Keširanje u ASP.NET sistemu ..... - 35 -
3.9.3 Keširanje na strani klijenta ..... - 36 -
3.9.4 Keširanje i upitni string ..... - 36 -
3.9.5 Neposredno upravljanje keširanjem ..... - 37 -
3.9.6 Keširanje podataka ..... - 38 -
3.9.7 Keširanje sa kontrolama izvora podataka ..... - 39 -
3.9.8 Zavisno keširanje i SQL Server ..... - 40 -
3.10 Izgled Web aplikacije „Kristal“ ..... - 42 -
3.10.1 Stilovi ..... - 42 -
3.10.2 Teme ..... - 43 -
3.10.3 *Master* stranice ..... - 44 -
4 Upravljanje ljudskim resursima ..... - 46 -
4.1 Procesi u Web aplikaciji za upravljanje ljudskim resursima „Kristal“ .. - 47 -
4.1.1 Regrutovanje kandidata ..... - 47 -
4.1.2 Obuka i usavršavanje zaposlenih ..... - 48 -
4.1.3 Određivanje cilja karijere ..... - 49 -
4.1.4 Lična SWOT analiza ..... - 50 -
4.1.5 Razvojne potrebe ..... - 51 -
4.1.6 Upravljanje vremenom ..... - 51 -
5 Zaključak ..... - 53 -
Prilog ..... - 54 -
Akcije ..... - 55 -

Urođene radnje ..... - 56 -
Otkrivene radnje ..... - 57 -
Apsorbovane radnje ..... - 58 -
Uvežbane radnje ..... - 59 -
Mešovite radnje ..... - 59 -
Gestovi ..... - 60 -
Slučajni gestovi ..... - 60 -
Izrazni gestovi ..... - 61 -
Mimički gestovi ..... - 61 -
Šematski gestovi ..... - 62 -
Simbolički gestovi ..... - 63 -
Tehnički gestovi ..... - 63 -
Šifrovani gestovi ..... - 64 -
Literatura ..... - 65 -

1 Uvod

Opšte društvene okolnosti, savremene informacione tehnologije i nove poslovne tendencije, uzrokovale su brojne promene u modernom poslovanju. U nastojanjima da se poboljšaju poslovni rezultati, pored pitanja koja se bave organizacionim i tehnološkim temama, fokus je pomeran ka pitanjima koje se bave ljudskim potencijalima i psihološkim aspektima rada. Tim nastojanjima je doprinela i orijentacija psihologije u svom interesovanju za individualne osobine čoveka, sa svojim metodama za proučavanja razlika između ljudi i instrumentima kojima se te osobine i razlike mogu meriti. Posebno je značajan doprinos brojnih psihologa, koji su svojevremeno, postavili metodološke osnove za primenu psihologije u teoriji i praksi upravljanja ljudskim resursima. Primenom statistike u merenju individualnih razlika, razvijanjem metoda za predviđanje, postavljanjem standarda za primenu psiholoških testova i primenom sofisticiranih alata je formiran sistem koji se može upotrebiti za standardizaciju metoda i tehnika upravljanja ljudskim resursima.

Prvi pokušaji da se racionalno organizuje poslovanje, bili su vezani za empirijska istraživanja kojima su analizirani pokreti pri obavljanju različitih poslova. Već u samom početku se došlo do saznanja da se može ostvariti maksimalni učinak uz, istovremeno, ostvarivanje ušteda u vremenu i energiji, što je ukazalo na potrebu selekcije radnika. Pokazalo se da svaki radnik, u zavisnosti od svojih intelektualnih i fizičkih mogućnosti, može postići maksimalni učinak samo na onom radnom mestu koje odgovara takvim karakteristikama.

Razvoju znanja, vezanog za angažovanje ljudskih potencijala, značajno su doprineli brojni eksperimenti: na polju mikro klimatskih faktora, na polju samog organizovanja poslovnih aktivnosti, testiranja u vezi sa dužinom odmora, ili visinom novčane nadoknade, kao i eksperimenata vezanih za jačinu osvetljenja u radnim prostorijama, ali svi dobijeni rezultati, i ako veoma značajni, u velikoj meri nisu opravdavali očekivanja. Dešavalo se da su dobijeni rezultati u suprotnosti sa postavljenim hipotezama, na osnovu čega je zaključeno da čovek nije u potpunosti racionalno biće i da njegovo ponašanje ne zavisi isključivo od ekonomske stimulacije i fizičkih uslova rada. Takva saznanja su dovela do zaključka da su socijalne potrebe značajnije od uslova rada i zarada. Potrebe za poštovanjem, priznavanjem, pripadanjem, koje radnici realizuju kroz neformalne grupe postaju predmeti proučavanja, kao najznačajniji faktori ponašanja i radne produktivnosti.

Fond znanja, iz oblasti upravljanja ljudskim resursima, kojim danas raspolažemo, obiluje različitim tehnikama i metodama, uz pomoć kojih se znanje, veštine i sposobnosti zaposlenih, mogu usavršavati, uz istovremeno angažovanje, na način kojim se ostvaruju zadovoljavajući poslovni rezultati. Sistematski pristup analiziranju metoda i tehnika upravljanja ljudskim resursima i implementacija tih znanja i procesa u informacioni sistem za upravljanje ljudskim resursima, sigurno bi predstavljala krupan korak u praksi i nezaobilazan alat agencija za upravljanje ljudskim resursima.

U ovom diplomskom radu osmišljena je i realizovana Web aplikacije za upravljanje ljudskim resursima „Kristal“. Aplikacija je izgrađena uz pomoć ASP.NET tehnologije. Ova tehnologija je prvenstveno namenjena Web programiranju. Manipulacija podacima je realizovana uz pomoć ADO.NET sistema, dok je preostali deo izgrađen uz pomoć klasa koje su sastavni deo .NET *Framework*-a.

Metode i tehnike iz oblasti upravljanja ljudskim resursima, čine veoma kompleksan sistem i poseban programerski izazov, prvenstveno zbog same prirode podataka koji su predmet obrade. U ovoj fazi razvoja Web aplikacije „Kristal“, posebno su analizirane tri aktivnosti upravljanja ljudskim resursima: regrutovanje, razvoj kompetencija i upravljanje karijerom. Kada je u pitanju razvoj Web aplikacije i obzirom da su u pitanju prve verzije ove aplikacije, podržani procesi predstavljaju pravi izbor. Na prvom mestu, podržavanje ovih procesa otvara mogućnost za dalji razvoj aplikacije na jedan sistematičan i istovremeno, diskretan način, pored toga, podržani procesi daju ovoj aplikaciji jednu dozu zanimljivosti, koja se može pozitivno odraziti posetioce ove Web lokacije.

Naziv „Kristal“, sugeriše na jasnoću i cilj koji ova aplikacija treba da realizuje. Svi procesi kojima se usmeravaju i nadograđuju ljudski potencijali, prvenstveno su u interesu ostvarivanja boljih poslovnih rezultata i u interesu preduzeća u kojima se ti procesi realizuju. Obzirom da svako profesionalno angažovanje podrazumeva obostrano ostvarivanje interesa, Web aplikacija „Kristal“ treba da omogući da svi njegovi posetioci jasno vide svoje interese i koristi. Pored toga, ova aplikacija treba da istakne značaj samostalne inicijative, kada je u pitanju upravljanje karijerom i značaj intervencije preduzeća i rukovodilaca, i naravno, kada je u pitanju razvoj kompetencija.

Početna verzije ove aplikacije je osposobljena za prikupljanje podataka o korisnicima aplikacije, zatim podataka relevantnih za procenu sposobnosti zaposlenih, podataka o samim zaposlenim, o poslodavcima, o oglasima za zapošljavanje i o onima koji traže zaposlenje. Pored toga, ova aplikacija pruža mogućnost jednostavnog izveštavanja. Ti izveštaji u slučaju oglašavanja radnih mesta, ili postavljanja biografija, imaju ulogu posredovanja između poslodavaca i konkurenata za zaposlenje. Naredne verzije će biti proširene na sve aktivnosti upravljanja ljudskim resursima, a jednostavne izveštaje će zameniti rezultati analiza zasnovani na prikupljenim podacima.

Tumačenje ljudskih motiva, motivacionih faktora, proučavanje zadovoljstva poslom i radom, analiziranje međuljudskih odnosa i uopšte posmatranje ljudskog ponašanja ima veliki značaj za pravilno usmeravanje ljudskih potencijala. Brojna pitanja iz ove oblasti su obrađena sa velikim uspehom i velika količina znanja čeka da bude primenjena u praksi. Svaki proces upravljanja ljudskim potencijalima u svojoj osnovi podrazumeva posmatranje i tumačenje sa stanovišta psihologije. U tom smislu, početni koraci u kvalitetnoj analizi ponašanja ljudi i grupa ljudi u poslovnom okruženju, se odnose na posmatranje i tumačenje najsitnijih delova ljudskog ponašanja. Iz tog razloga u ovoj Web aplikaciji se nalaze i dva psihološka testa: test profesionalne ostvarenosti i zadovoljstva sobom, i test poverenja, a u prilogu ovom diplomskom radu se nalazi kratak pregled vrsta akcija i gestova koji čine osnovne delove ljudskog ponašanja.

2 .NET Framework

ASP.NET je *Microsoft*-ova platforma za razvoj Web aplikacija. Pomoću nje, korisnici mogu kreirati virtuelne Web prodavnice zasnovane na e-trgovini, platforme čiji je sadržaj zasnovan na podacima, kao i sve druge aplikacije koje se danas mogu sresti na Internetu.

U samoj osnovi ove platforme nalazi se sistem pod nazivom .NET *Framework*. .NET *Framework* je kolekcija različitih tehnologija prikupljenih pod jedinstveni marketinški naziv. Ovo okruženje obuhvata jezike kao što su *C#*, *VB*, *J#*, *F#*, zatim mehanizme programabilnih Web stranica i Web servisa (ASP.NET), kao i mehanizme za povezivanje sa bazama podataka (ADO.NET), zajedno sa bibliotekom klasa u koje su uključene alatke koje obavljaju najraznovrsnije poslove, od čitanja datoteka, do provere korisničkih šifara definisanih u okviru nekog od korisničkih naloga.

2.1 Web programiranje

Internet je nastao krajem šezdesetih godina prošlog veka kao rezultat jednog eksperimenta. Cilj eksperimenta je bio stvaranje potpuno elastične informacione mreže, koja bi bila u mogućnosti da nastavi sa radom i nakon gubitka jednog broja računara, bez uticaja na komunikaciju između preostalih računara. Finansijer eksperimenta je bilo američko ministarstvo odbrane, koje je razmatralo posledice eventualnih katastrofalnih događaja, kao što su prirodne nepogode, ili ratovi.

Prvobitni internet je bio ograničen na odbrambene i na pojedine obrazovne institucije. Mreža se razvila u akademsku alatku pomoću koje su naučnici širom sveta mogli da razmenjuju informacije. Početkom devedesetih godina prošlog veka napravljeni su *modemi* koji su mogli da komuniciraju pomoću postojećih telefonskih linija. Nakon toga, Internet je postao dostupan i komercijalnim korisnicima. 1993. godine kreiran je i prvi HTML Web čitač, sa čim je započela prava Internet revolucija.

Prvi Web sajtovi nisu imali karakteristike aplikacija. Više su podsećali na obične brošure sa nekoliko fiksnih HTML stranica, čiji sadržaj je ažuriran ručno. U osnovi su podsećale na dokumente nastale u programima za obradu teksta: na njima se nalazio formatirani sadržaj koji se mogao prikazivati na računaru, ali takve stranice, u suštini nisu obavljale nikakav posao.

HTML 2.0 je uveo osnovne elemente Web programiranja kroz tehnologiju koja je nazvana HTML formulari. Ovi formulari su proširili jezik tako što pored postojećih elemenata za formatiranje, uveli i elemente za grafičke dodatke, odnosno kontrole. Ove kontrole su obuhvatale klasične elemente, kao što su padajuće liste, tekstualna polja, ili komandna dugmad. Zahvaljujući ovim formularima, Web programeri su dobili mogućnost da kreiraju stranice za unos podataka, a same Web prezentacije su dobile osnovne karakteristike aplikacija.

Kontrole koje su uvedene u upotrebu pre više od deset godina i danas predstavljaju osnovu za kreiranje dinamičkih ASP.NET stranica. Razlika je u vrsti aplikacije koja se izvršava na serverima. Kada bi korisnik, u prošlosti, kliknuo na neko dugme, definisano na formularu stranice, podaci su slati putem elektronske pošte do administratora naloga ili do

aplikacije na serveru koja je koristila ambiciozni CGI (*Common Gateway Interface*) standard. Današnji korisnici rade sa znatno moćnijim i veoma elegantnim ASP.NET platformama.

2.1.1 Serverski i klijentski programski model

U prošlosti, Web serveri koji su koristili CGI standard su morali da pokrenu novu, posebnu instancu aplikacije za svaki primljeni zahtev. Serveri na popularnim Web sajtovima su na taj način morali da se izbore sa stotinama odvojenih kopija neke aplikacije, tako da su često postajali žrtve sopstvenog uspeha. Pored toga ta i slične tehnologije su nudile samo osnovno programsko okruženje. Operacije višeg nivoa, kao što su: identifikacija korisnika, skladištenje podataka, ili prikazivanje slogova izdvojenih iz baze podataka, zahtevale su pisanje obimnog koda „od nule“. Takav pristup Web programiranju je bio veoma naporan i podložan brojnim greškama.

Rešenje takvih problema je ponudio *Microsoft*, koji je razvio platformu višeg nivoa. Ona je omogućila kreiranje dinamičkih Web stranica bez upuštanja u sistemske probleme nižeg nivoa. Međutim, ni prve verzije ove platforme, nisu mogle da odgovore na sve zahteve modernog Web programiranja. U okviru prvobitnog dizajna ASP-a su se javili problemi u vezi sa performansama, zaštitom i definisanjem konfiguracije sistema.

To su bila pitanje koja su u prvi plan izbacila ASP.NET. On je razvijan kao industrijsko aplikativno okruženje koje će prevazići ograničenja ASP-a. Ova platforma je stekla veliku popularnost odmah nakon objavljivanja. Možda je, najvećim delom, za popularizaciju ove tehnologije, doprinela činjenica da je nekoliko velikih komercijalnih sajtova izgrađeno pomoću nje, dok je još bila u *beta* verziji.

Dok je programiranje na strani servera krčilo svoj put kroz skup novih tehnologija, još jedan stil programiranja je sticao sve više popularnosti. Programeri su unapređivali Web stranice, ugrađujući u njih procedure razvijene u *JavaScript*-u, *ActiveX*-u i *Flash*-u. Sve ove procedure su bile namenjene klijentskim računarima, gde su se i izvršavale, bez angažovanja servera. Korisnik bi preuzimao kompletnu aplikaciju na svoj računar, gde se ona izvršavala u lokalu.

Osnovni problemi ovakvog pristupa su se ogledali u činjenici da ovako definisane aplikacije nisu bile podržane u svim Web čitačima i u svim operativnim sistemima. Obzirom da Web programiranje svoju popularnost, prvenstveno, duguje činjenici da Web aplikacije ne zahtevaju nikakva dodatna preuzimanja programa sa interneta i zamorne instalacione postupke, ovakvo programiranje je pokazalo određene nedostatke i ako u određenoj meri može da doprinese poboljšanju ukupnog vizuelnog dizajna i racionalnijem korišćenju resursa Web servera.

Najistaknutiji nedostatak programiranja aplikacija, koje se delom izvršavaju na klijentskom računaru se ogleda u tome što razvoj takve aplikacije podrazumeva mukotrpno testiranje na različitim operativnim sistemima i Web čitačima. Pored toga izvršavanje tako definisanih aplikacija, podrazumeva distribuiranje ažurnih datoteka za različite Web čitače. Klijentski model u suštini žrtvuje najvažniju prednost Web programiranja, nezavisnost od sistemskih postavki klijentskih računara.

Međutim, i ako je ASP.NET dizajniran kao serverska tehnologija, postoji mogućnost da se aplikacija optimizuje na način koji smanjuje broj obraćanja klijentskog računara serveru i na taj način obezbedi racionalna upotreba serverskih resursa.

2.1.2 Međujezik

.NET okruženje je skup različitih tehnologija. Pored toga što podržava rad pomoću različitih programskih jezika, i pored bogate biblioteke, od više hiljada unapred kreiranih klasa, ovo okruženje raspolaže i jednom specifičnom komponentom koja omogućava rad u nekoliko programskih jezika.

Svi programi, napisani u nekom od podržavanih jezika, pre izvršenja se prevode u poseban jezik nižeg nivoa. On se naziva *Common Intermediate Language*, a u literaturi se ovaj jezik često može naći i pod nazivom *IL*. *CLR* sistem, izvršno okruženje .NET platforme, koristi isključivo *IL* programski kod. Obzirom da je dizajn svih programskih jezika zasnovan na ovom jeziku, jasno je da postoji i velika sličnost u performansama i opcijama između jezika *VB, C#, J#* i *F#*. Stepen kompatibilnosti je takav da Web stranica napisana u *C#* jeziku, može koristiti *VB* komponente, na isti način kao što i one napisane u C#-u mogu koristiti komponente napisane uz pomoć *VB*-a.

.NET Framework definiše još formalniji oblik dodavanjem kompatibilnosti ovoj komponenti uz pomoć CLS-a (*Common Language Specification*). *CLS* je u osnovi ugovor, koji, ukoliko se poštuje, garantuje da će komponente napisane u jednom .NET jeziku raditi i sa ostalim jezicima. Deo specifikacije koji definiše pravila za tipove podataka, zajednički je za sve .NET jezike. Pored toga *CLS* definiše i objektno orijentisane elemente kao što su klase, metodi, događaji i drugi.

2.1.3 Aplikacioni događaji

Serverske kontrole su uobičajen izvor događaja, ali pored njih postoji još jedna vrsta događaja koji u pojedinim situacijama mogu biti veoma značajni: aplikacioni događaji. Aplikacioni događaji nemaju takav značaj za ASP.NET aplikacije, kao što imaju događaji koji se okidaju u serverskim kontrolama, ali se mogu uspešno iskoristiti za obavljanje nekih dodatnih zadataka.

Uz njihovu pomoć možemo, na primer, napisati rutinu za vođenje dnevnika (*log*) koja će se aktivirati pri svakom prijemu zahteva, bez obzira koja stranica se traži. Osnovne opcije ASP.NET sistema, kao što su praćenje stanja sesije i identifikacija korisnika, koriste aplikacione događaje u procesu ASP.NET obrade.

Aplikacioni događaji se ne mogu obrađivati u odvojenim datotekama sa kodom za Web obrazce, umesto toga, takvu ulogu ima datoteka *Global.asax*. Ova datoteka omogućava pisanje programskog koda koji će reagovati na globalne aplikacione događaje. Ovi događaji se okidaju u različitim periodima životnog veka aplikacije, uključujući i prvo kreiranje aplikacionog domena.

Datoteka *Global.asax* izgleda kao i obična **.aspx* datoteka, s tim što se u njoj ne nalaze nikakvi HTML ili ASP.NET elementi. Ona se sastoji isključivo od rutina za obradu događaja. Svaka ASP.NET aplikacija može imati samo jednu ovakvu datoteku. Ukoliko se

ona postavi u odgovarajući direktorijum Web sajta, ASP.NET će je automatski prepoznati i upotrebiti.

Načini na koje se ova datoteka može upotrebiti zavise od kreativnosti programera. Sama njena svrha ukazuje na brojne namene, od izvršavanja različitih podešavanja na Web serveru, do postavljanja baze podataka, ili aktiviranja različitih servisa. Obzirom da su sva sistemska podešavanja, na serveru gde je postavljena Web aplikacija „Kristal“, zaključana za korisnike *hosting* usluge, ova datoteka još uvek nije našla primenu, ali prilikom postavljanja ove aplikacije na komercijalni server, biće upotrebljena za instalaciju baze podataka i za konfigurisanje postavki vezanih za *mail* server.

2.1.4 Konfiguracione XML datoteke

Svaka pojedinačna Web aplikacija poseduje jednu glavnu datoteku *web.config* i nekoliko *Web.config* datoteka koje su raspoređene po odgovarajućim direktorijumima u kojima su definisani osnovni parametri kojim se podešavaju neka od svojstava Web aplikacije, od načina prikazivanja poruka o greškama do parametara zaštite koji izoluju neželjene posetioce.

Datoteka *web.config* se najčešće nalazi u korenom direktorijumu i ona nikada nije zaključana, ali može da bude, za razliku od ostalih datoteka *Web.config*, koje se nalaze u direktorijumima koji su zaključani upravo zahvaljujući tim datotekama. Sadržaj ovih datoteka programer može ažurirati u bilo kom trenutku, čak i tokom izvršenja aplikacije. Ukoliko je neki zahtev trenutno u obradi, on će nastaviti rad sa starim vrednostima parametara, dok će novi zahtevi koristiti izmenjene vrednosti.

Datoteci *web.config* se lako pristupa i ona se može jednostavno kopirati, naravno, ukoliko programer ima odgovarajuća prava na mreži. Sadržaj ove datoteke može se menjati i sa udaljenog računara. Pored toga, ona se može kopirati i na taj način se mogu primeniti isti parametri i u drugoj aplikaciji ili na drugom Web serveru koji izvršava istu aplikaciju u scenariju Web farme.

Parametri se lako podešavaju i lako su shvatljivi. Oni imaju prepoznatljivu formu, tako da za njihovo podešavanje nisu potrebne specijalne konfiguracione alatke, jer datoteka *web.config*, koristi definisani XML format. Ove datoteke, kao i svi drugi XML dokumenti, prave razliku između malih i velikih slova. Svaka sekcija počinje malim slovom, tako da se, u slučaju sekcija za podešavanje parametara Web aplikacije, ne može koristiti element *AppSetings*, umesto elementa *appSetings*.

Celokupan sadržaj datoteke *web.config* je ugnježden u koreni element *configuration*. Unutar tog elementa se nalazi određeni broj sekcija od kojih su neke posebno značajne za Web programiranje, dok se neke druge nikada ne menjaju. Sekcija *connectionStrings*, na primer, omogućava definisanje konekcionih stringova za pristup bazama podataka. Sekcija *appSetings* pruža mogućnost programerima da naprave sopstveni sistem za manipulaciju podacima, ali je pristup zasnovan na bazama podataka znano fleksibilniji. Konačno, sekcija *system.web* koja sadrži veliki broja parametara, omogućava programeru da definiše brojna podešavanja u vezi sa zaštitom Web sajta, kontrolu serverskih ili klijentskih grešaka i slično.

Unutar sekcije *system.web* smešteni su zasebni elementi za svaki pojedinačni konfiguracioni aspekt Web sajta. Programeru je na raspolaganju proizvoljan broj ovih elemenata. Ukoliko, na primer, želi da definiše posebne parametre za obradu grešaka, koristiće element *customErrors* koji je smešten u ovoj sekciji.

Datoteke *Web.config*, koje se nalaze u zaštićenim direktorijumima, i čija je prvenstvena svrha da obezbede zaštitu za te direktorijume, imaju samo *system.web* sekciju. Ukoliko nastojimo da definišemo način rada sistema zaštite, dodaćemo elemente *authentication* i *authorization*, u okviru kojih se mogu izvršiti brojna podešavanja.

3 Razvoj Internet aplikacije „Kristal“

Svaka Web aplikacija mora imati pažljivo isplaniranu strukturu direktorijuma. Datoteke sa slikama, na primer, se smeštaju u odvojeni direktorijum. Javne ASP.NET stranice obično se smeštaju u koreni direktorijum, dok one kojima je pristup ograničen, u posebne direktorijume. Na taj način se otvara mogućnost definisanja različitih nivoa zaštite za takve datoteke.

Prva verzija Web aplikacije „Kristal“, „Kristal 1.0“, je predviđena za četiri vrste korisnika: nezaposlene, poslodavce, rukovodioce i zaposlene. U skladu sa tim, a i zbog potrebe da se obezbedi odgovarajući sistem zaštite, za svaki tip korisnika je obezbeđen poseban direktorijum sa Web stranicama u koje su integrisane funkcije karakteristične za tu grupu korisnika. Pored toga u strukturi direktorijuma nalazi se i jedan, od nekoliko, unapred definisanih direktorijuma, pod nazivom *App_Data*.

Predefinisani direktorijum *App_Data* je predviđen da bude skladište podataka, između ostalog i za datoteke *SQL Server 2005 Express Edition* baze podataka, kao i XML datoteke. Datoteke se naravno mogu postaviti i u druge direktorijume, ali kako bi se izbegle sve neprijatnosti koje mogu da nastanu prilikom postavljanja aplikacije na Internet, bolji izbor je poštovati predefinisane vrednosti, gde god za to postoji mogućnost. Posebno zbog činjenice da sajtovi koji obezbeđuju besplatan *hosting* ograničavaju veliki broj opcija, koje su ponuđene u okviru IIS-a (*Internet Information Services*), prvenstveno zbog njihovih bezbednosnih postavki.

Uzimajući u obzir, da nakon postavljanja Web aplikacije na server, nastaje situacija u kojoj brojne mogućnosti ograničene, sama Web aplikacija se mora prilagoditi ponuđenim uslovima. Na prvom mestu, mora se obezbediti potpuna funkcionalnost svih komponenti. U tu svrhu najbolje je kreirati posebnu biblioteku klasa, u kojoj bi se nalazile sve potrebne klase i reference na klase iz .NET *Framework*-a. Web aplikacija „Kristal“ poseduje svoju biblioteku klasa pod nazivom *KristalBiblioteka.dll*. Ova biblioteka obezbeđuje normalno funkcionisanje ostalih komponenti ove aplikacije i pruža veliki stepen nezavisnosti, pa i od sistemske Windows *registry* baze, ili bilo koje druge konfiguracije.

Druga važna postavka je vezana za bazu podataka. Obzirom na ograničenja vezana za manipulaciju podacima na serveru, i obzirom na to da ADO.NET tehnologija raspolaže skupom kvalitetnih klasa za interakciju sa podacima, u ovoj aplikaciji je primenjena kombinacija, koja teoretski daje najbolje performanse: *SQL Server Express* i *SQL Server provajder*. Ovaj provajder je podešen tako da obezbedi najbolje performanse pri radu sa određenim izvorom podataka. On interno koristi *SQL Server TDS* (*Tabular Data Stream*) protokol za komunikaciju.

Web aplikacija „Kristal“ je uređena tako da raspolaže sistemom koji ograničava pristupa korisnicima koji nisu registrovani i prijavljeni na sistem. Zaštita ove Internet aplikacije funkcioniše zahvaljujući klasama *Membership* i *MembershipUser*, kao i XML dokumentima u okviru kojih su definisana pravila kontrole pristupa i prava korisnika na određene akcije.

Grafički dizajn Web stranica je uređen uz pomoć stilova koji su integrisani u glavnu stranicu (*Master page*). Sva definicije koje se nalaze u toj stranici vezane su za nekoliko značajnih funkcionalnosti i za izgled Web aplikacije. Sve ostale Web stranice, nasleđuju funkcionalnosti i izgled koji je definisan pomoću glavne stranice.

3.1 ADO.NET

ASP.NET je samo jedna komponenta kompleksne *Microsoft* .NET platforme. Ona uvodi novu filozofiju u integraciju brojnih .NET programskih jezika, donosi jednostavniji način za instalaciju programskih rešenja i mnoštvo klasa koje omogućavaju obavljanje najrazličitijih funkcija, od obrade grešaka, do analize XML dokumenata.

ADO.NET je tehnologija koju .NET aplikacije koriste radi ostvarivanja interakcije sa bazama podataka. Ova tehnologija je veoma značajna kada je potrebno optimizovati rutine za rad sa bazama podataka koje imaju visoke zahteve po pitanju performansi, kao i pri kreiranju samih komponenti za rad sa bazama podataka.

Praktično, svaki komadić softvera koji je ikada napisan radi sa podacima. Tipična Web aplikacija je, u svojoj osnovi, obična školjka u obliku korisničkog interfejsa, koja je postavljena preko sofisticiranog koda za rad sa podacima, Taj kod obezbeđuje očitavanje i ažuriranje slogova u bazi podataka. Posetioci Web sajtova obično nisu svesni, ili o tome ne razmišljaju, da prikazani podaci vode poreklo iz neke baze podataka. Oni najčešće imaju ambiciju da pretraže neki katalog proizvoda, ili da provere stanje na svom računu.

3.1.1 Baze podataka u Web aplikacijama

Upravljanje podacima se najdoslednije i na veoma fleksibilan način ostvaruje uz pomoć baza podataka. Tehnologija baza podataka je naročito prikladna za poslovni softver, koji najčešće radi sa skupom povezanih podataka. Tako će tipična baza podataka u sektoru prodaje, na primer, sadržati tabelu *Kupci*, tabelu *Proizvodi* i tabelu *Prodaja*, koja crpi podatke iz prve dve tabele. Takve strukture podataka se najtačnije opisuju relacionim modelom, koji je u osnovi svih savremenih proizvoda za rad sa bazama podataka, uključujući *SQL Server*, *Oracle*, ili *Microsoft Access*.

Tehnički bi bilo izvodljivo organizovati podatke u tabele i uskladištiti ih na hard disk u jednoj ili više datoteka, na primer upotrebom XML standarda. Takav pristup podrazumeva odricanje od određenih vidova fleksibilnosti. Umesto toga, upotreba zaokruženih sistema za upravljanje relacionim bazama podataka (*Relational Database Management System, ili RDBMS*) pruža zadovoljavajuće performanse uz prihvatljivu fleksibilnost. Ovi sistemi imaju još jednu značajnu mogućnost, oni rešavaju i pitanje istovremenog pristupa podacima od strane većeg broja korisnika, zabranu unosa pogrešnih podataka i grupisanje većeg broja aktivnosti u okviru jedinstvene transakcije.

Scenario pristupa bazama u Web aplikaciji se u velikoj meri razlikuje od sličnog scenarija u tipičnoj desktop aplikaciji tipa klijent-server. Brojna rešenja koja besprekorno funkcionišu u desktop aplikacijama, u slučajevima Web programiranja uzrokuju veliki broj grešaka ili potpuno otkazivanje sistema. Najjednostavnije rečeno, programiranje Web aplikacija sa bazama podataka, pokreće dva nova pitanja: skalabilnost i problem stanja.

Kada je u pitanju skalabilnost, problemi mogu nastupiti zbog izražene višekorisničke prirode Web aplikacija. Web aplikacije su izložene stotinama, pa i hiljadama korisnika, istovremeno. Pri izradi dizajna baze podataka mora se, na prvom mestu, voditi računa o tome koliko se zadržava veza prema bazi. Ukoliko bi aplikacija zadržavala vezu ka bazi samo nekoliko sekundi duže, korisnici bi uočili značajno i neprijatno usporenje u radu. Klase za manipulaciju podacima, Web aplikacije „Kristal“, zadržavaju vezu samo toliko, koliko je potrebno da se podaci upišu, ili isčitaju iz baze podataka. Sve provere unosa podataka se obavljaju u okviru samih formi za unos, a svi podaci uneti u te forme se keširaju do trenutka kada korisnik odluči da ih pošalje ka serveru.

Problemi koji su vezani za stanje su posledica same prirode Interneta, koji nije orijentisan ka uspostavi stalne veze sa korisnikom. HTTP protokol ne vodi računa o pamćenju trenutnog stanja. Kada korisnik zatraži neku stranicu iz ASP.NET aplikacije, Web server će obraditi odgovarajući kod i vratiti generisani HTML kod, nakon čega se veza prema korisniku odmah prekida. Korisnik će verovatno i dalje imati utisak da ima vezu sa aplikacijom koja neprekidno radi, mada on, u suštini, samo prima statičke stranice u nizu.

Upravo zbog toga, što se u HTTP protokolu ne pamti stanje, Web aplikacije moraju obaviti celokupan posao u toku obrade jednog zahteva. Ovaj pristup može uzrokovati brojne probleme ukoliko želimo da omogućimo korisniku da izmeni očitane podatke. U tom slučaju aplikacije moraju imati određenu dozu inteligencije kako bi mogle da pronađu korisnikov slog u bazi i kreiraju SQL naredbu radu njegove selekcije ili ažuriranja.

To je posebno izraženo u situacijama upravljanja korisničkim nalozima. Pored komunikacije sa bazom podataka, koja mora biti besprekorna u toj situaciji treba omogućiti i primenu određenog broja sigurnosnih postavki, bez narušavanja rada sistema za upravljanje bazom podataka.

Veoma elegantno rešenje nude funkcije klasa *MembershipUser* i *Membership*, koje se nalaze u paketu *Security*. Nakon unosa korisničkih podataka, slog o novom korisniku se smešta u bazu podataka, a nakon prijavljivanja na sistem, korisnički podaci se upoređuju sa onim koji su uneti prilikom kreiranja korisničkog naloga. Obzirom ove klase raspolažu metodama koje obezbeđuje vrednost primarnog ključa, iz baze podataka, na osnovu korisničkih podataka prijavljenog korisnika, ta vrednost se može koristiti za sve buduće unose u tabele baze podataka, ažuriranja, ili brisanja slogova. Zapravo, metoda koja se aktivira nakon prijavljivanja korisnika na sistem, obezbeđuje vrednost primarnog ključa iz tabele sa korisničkim podacima, koja prati korisnika prilikom rada u okviru ove aplikacije i koristi se za sve aktivnosti sa bazama podataka, koje su implementirane u ovoj Web aplikaciji.

3.1.2 SQL Server

Da bi smo mogli da pristupamo podacima iz baze podataka, potreban nam je server baze podataka koji može izvršavati naše komande. I pored toga što postoji više desetina različitih servera i što svi oni uspešno rade sa ADO.NET tehnologijom, najveći broj ASP.NET aplikacija koristi *Microsoft SQL Server*.

Pune verzije *SQL Server 2005* i *SQL Server 2008*, predstavljaju idealna rešenje, ali osiromašena i besplatna verzija servera, *SQL Server 2005 Express*, takođe pruža zadovoljavajuće performanse. Međutim, to podrazumeva i neka ograničenja. Ova verzija servera, može raditi samo sa jednim procesorom i sa maksimalno jednim gigabajtom radne memorije, pored toga, baze podataka ne mogu biti veće od četiri gigabajta i ova verzija se isporučuje bez grafičkih alata. Svejedno, to je i dalje pogodan i moćan sistem za veliki broj sajtova srednje veličine. Naravno, ovakve karakteristike su, kada je u pitanju, Web aplikacija „Kristal“, sasvim odgovarajuće, čak i dosta iznad potrebnih performansi.

Rad sa izvorima podataka iz ADO.NET okruženja se zasniva na sintaksi SQL-a (*Structured Query Language*). SQL je standardni jezik za pristup podacima koji se koristi u interakciji sa relacionim bazama podataka. Različite baze se razlikuju u načinu na koji podržavaju SQL standard , kao i po raznim dodatnim opcijama koje uvode, ali osnovne komande za selekciju, dodavanje i izmenu podataka imaju isti oblik kod svih. Pomoću ovog jezika se mogu kreirati složene SQL procedure i okidači, ali obzirom objektno orijentisani programski jezici, mogu pružiti veću slobodu, kompletna logika rada sa bazom podataka je realizovana u sloju poslovne logike uz pomoć C# programskog jezika. SQL je korišćen prilikom formiranja konkretnih naredbi za upis i izmenu podataka, kao i prilikom definisanja upita.

SQL definiše i posebne agregatne funkcije koje rade sa skupom vrednosti i vraćaju jednu vrednost, kao što su: *Avg, Sum, Min, Max, Count*, i slično. Pomoću njih se mogu prebrojati slogovi u tabeli, ili izračunavati prosečne vrednosti. Iako takav pristup obezbeđuje bolje performanse, taj posao je moguće realizovati uz pomoć klasa koje se nalaze u .NET okruženju, što je i slučaj kada je ova aplikacija u pitanju.

Sva literatura koja obrađuje problematiku manipulacije podacima uz pomoć baza podataka, naglašava značaj uvođenja identifikacionog polja sa automatskim uvećanjem vrednosti. To je najbrži, najjednostavniji i najpouzdaniji način za dobijanje jedinstvenog identifikacionog polja za svaki slog. Bez automatski generisanog identifikacionog polja, programer sam mora kreirati i održavati takvo polje, što može uzrokovati brojne probleme.

Programeri često greše tako što za jedinstveni identifikator koriste neko od polja sa podacima, poput matičnog broja ili ličnog broja osiguranja, ili imena i prezimena. Takav pristup neminovno dovodi do grešaka u nekom trenutku u budućnosti, kada u bazu pokušamo da unesemo podatke o licu koje nema zdravstveno osiguranje, ili ukoliko je određeno lice iz druge države i njegovi lični podaci, po formi ne odgovaraju predviđenim pravilima za unos takvog polja. Primena jedinstvenog identifikatora je znatno bolje rešenje, posebno ako sama baza podataka automatski generiše vrednost koja se dodeljuje svakom novom slogu jedne tabele u bazi podataka.

Kod tabela bez jedinstvene kolone identifikatora, mogu nastati problemi prilikom selekcije određenog reda, brisanja, ili ažuriranja. Selekcija slogova na osnovu sadržaja nekog tekstualnog polja može dovesti do problema ukoliko se u polju nalaze neki specijalni karakteri, poput apostrofa, ne primer. Pored toga, takav pristup izuzetno otežava kreiranje relacija između tabela. Sve novije verzije SQL Server-a imaju predefinisano identifikaciono polje tipa *uniqueidentifier*, kom se jednostavno dodaje nova vrednost prilikom unosa svakog novog sloga, zahvaljujući vrednosti koju vraća predefinisana funkcija *NEWID()*.

3.1.3 Provajderi podataka

ADO.NET se sastoji od manjeg skupa klasa čije funkcionalnosti omogućavaju upravljanje podacima i klasa koje se koriste za uspostavljanje veze sa određenim izvorom podataka. Svaki pojedinačni skup klasa za komunikaciju sa izvorom podataka se naziva ADO.NET provajder podataka. Najbolja praksa je koristiti onaj provajder koji preporučuje i sam proizvođač sistema za upravljanje bazom podataka. Ukoliko se koristi *Oracle* baza podataka, potrebno je koristiti i *Oracle* provajdere. Pored toga i nezavisni programeri i isporučioci baza nude sopstvene ADO.NET provajdere, koji prate iste konvencije i mogu se koristiti na isti način kao i provajderi koji se isporučuju u okviru .NET sistema.

Programer u svom radu može koristiti i različite provajdere, u odnosu na bazu podataka, jer oni nasleđuju iste osnovne klase, implementiraju iste interfejse i sadrže istu grupu metoda i svojstava. Pojedini provajderi nude i specifične opcije dostupne samo uz određene izvore podataka, kao što su opcije izvršenja XML upita, ali osnovna svojstva i metode klasa koji se odnose na pribavljanje i izmenu podataka su identična kod svih provajdera.

.NET sadrži četiri provajdera: *SQL Server*, *OLE DB*, *Oracle* i *ODBC* provajder. Po izboru treba pokušati sa onim provajderom koji je prilagođen konkretnom izvoru podataka. Ukoliko se takav provajder ne može pronaći, može se upotrebiti *OLE DB* provajder, pod uslovom da izvor podataka poseduje *OLE DB* drajver. *OLE DB* tehnologija je već duži niz godina prisutna u okviru ADO sistema, tako da najveći broj izvora podataka sadrži i *OLE DB* drajver, uključujući *SQL Server*, *Oracle*, *Microsoft Access*, *MySQL* i druge. U retkim situacijama, kada se desi da ni jedan od pomenutih provajdera ne funkcioniše, programer može koristiti *ODBC* provajder, koji radi u sprezi sa *ODBC* drajverom.

Microsoft SQL Server Data Provider, uliva najviše poverenja, jer ne koristi nikakve drajvere, i sve aktivnosti se događaju u okvirima samog SQL servera. Ovaj provajder podrazumeva upotrebu određenih klasa, koje su prilagođene, i posebno pisane za ovaj server, čime obezbeđuju najbolje performanse. Specifičnost, vezana za ove klase, je u tome što one isključuju dodatni *OLE DB* sloj, time što se direktno povezuju na prilagođeni *TDS* interfejs, čim se postižu bolji rezultati. Jedini nedostatak je u tome što ne podržavaju standardne SQL tipove podataka, kao što su *SqlMoney*, ili *SqlDateTime*, i što se ti podaci ne mogu koristiti bez konverzije, a svakom konverzijom podataka se rizikuju potencijalna oštećenja.

3.1.4 Unos podataka

Unos podataka u bazu, iz formi koje popunjavaju korisnici nosi sa sobom i određene rizike. Ti rizici se prvenstveno odnose na ispravno funkcionisanje sistema, a zatim i na zaštitu sistema. Ukoliko korisnici greškom unesu karaktere koji mogu da naruše rad SQL naredbi, generisani SQL string neće biti ispravan, što će uzrokovati pojavu greške, ili će, u boljem slučaju, biti lansiran izuzetak. Činjenica, na svakoj Web stranici može postojati provera validnosti unosa, najčešće i postoji, ali postoje i druga rešenja. Recimo, moguće je proveriti svaki unos i svaki put kada se pojavi apostrof (‘) izvršiti dupliranje, ili zamenu unetog znaka znakom navoda (“).

Pored toga, postoje i namerni pokušaji uzrokovanja grešaka, poznati pod nazivom *SQL injection attack*, odnosno, napad ubacivanjem dodatnog SQL koda. Primenom takvog napada, zlonamerni korisnici mogu narušiti rad sistema, ili izvršiti dodatne SQL naredbe i izbrisati neke od slogova u tabeli, ili još gore, doći do određenih podataka vezanih za druge korisnike, za koje u normalnom režimu nemaju pristup. Ovakav problem postaje izuzetno ozbiljan, ako se uzme u obzir da SQL Server sadrži čak i specijalnu sistemsku uskladištenu proceduru koja korisnicima omogućava da izvrše bilo koji program na lokalnom računaru.

Pojavljivanje ovih problema se može izbeći upotrebom parametarskih komandi, definisanih u okviru ADO.NET klasa. U zavisnosti od provajdera koji se koristi, sintaksa se malo razlikuje, ali sve parametarske komande, rade po istom principu. Parametarska komanda zamenjuje unete vrednosti sa rezervisanim pozicijama. Rezervisane pozicije se nakon toga dodaju odvojeno i automatski se kodiraju.

Svaki unos podataka u neku od baza podataka u Web aplikaciji „Kristal“, koristi parametarske komande. Na prvom mestu zbog sigurnosti, zatim i zbog obezbeđenja normalnog funkcionisanja ove aplikacije. Pored toga, upotrebom ovih komandi se dobija i na fleksibilnosti, jer svaka izmena u definiciji baze podataka, povlači korigovanje koda za manipulaciju podacima vezanim za tu bazu. Kada se kod organizuje uz pomoć parametarskih komandi, može se jednostavnim izbacivanjem, ili ubacivanjem redova, kod prilagoditi novoj definiciji baze, bez mukotrpnog izučavanja nepreglednih SQL naredbi i bez nepotrebnog testiranja.

3.1.5 Pristup podacima

Jedna od važnih karakteristika, kada su u pitanju aplikacije namenjene velikom broju korisnika, jeste ostvarivanje, što je moguće kraće veze sa bazom podataka. Pristup podacima se može izvršiti i uz prekid veze, a to omogućavaju klase ADO.NET sistema. Nakon preuzimanja, kopija podataka će biti smeštena u memoriju pomoću objekta *DataSet*. Veza sa bazom treba da traje samo onoliko koliko je potrebno za očitavanje podataka i njihovo prebacivanje u objekat *DataSet*, nakon čega veza prekida.

Objekat *DataSet* pamti izmene koje su izvršene nad slogovima koji su u njemu smešteni. Zahvaljujući tome, objekat *DataSet* možemo iskoristiti i za ažuriranje slogova u bazi. Osnovni princip podrazumeva da se objekat *DataSet* popuni podacima, zatim da se unesu odgovarajuće izmene u jedan ili više slogova, nakon čega se te izmene prebacuju u bazu uz pomoć objekta *DataAdapter*.

Ovakav pristup bi više odgovarao desktop aplikacijama, ali obzirom da se u Web aplikaciji „Kristal“ vrši i keširanje podataka, i obzirom na to da dužina sesije to dozvoljava, primenjen je ovakav pristup u kombinaciji sa ažuriranjem primenom direktnih komandi.

3.1.6 Prikazivanje podataka

ASP.NET kontrole se razlikuju međusobno kada su u pitanju mogućnosti njihovog povezivanja sa podacima. Nakon povezivanja sa podacima. Pojedinačne vrednosti se mogu automatski ubaciti u liste, ili neke druge tipove standardnih promenljivih članica. Pored

standardnih, postoje i napredne kontrole za prikazivanje podataka: *GridView*, *DetailsView*, *FormView* i *ListView*.

Za sve je karakteristično da raspolažu brojnim metodama za formatiranje podataka, za formatiranje kolona, ili celih tabela, kao i metodama za podešavanje izgleda. Pored toga, ove kontrole, kada se povežu sa podacima, omogućavaju selekciju, ažuriranje, ili pregled podataka na više stranica. Uz pomoć njih se mogu i brisati kompletni slogovi iz baze podataka, ali obzirom da je primarna svrha Web aplikacije „Kristal“, u ovoj fazi razvoja, prikupljanje podataka, ta opcija nije ponuđena korisnicima.

Ove kontrole su svakako dobre u situacijama postavljanja upita i kreiranja izveštaja. One omogućavaju postavljanje upita nad više tabela, takođe uz njihove funkcionalnosti se mogu napraviti dinamične forme za kretanje kroz bazu podataka u zavisnosti od izbora koji napravi korisnik. Naročito, kontrola *GridView*, koja omogućava prikazivanje više slogova istovremeno. Druga, manje značajna, ali sa stanovišta grafičkog dizajna veoma značajna pogodnost ove kontrole, je mogućnost primene različitih stilova, što svakako povoljno doprinosi ukupnom izgledu Web aplikacije.

Kontrola *DetailsView* omogućava prikazivanje određenog sloga iz baze podataka. Na nju se takođe mogu primeniti različiti stilovi, koji doprinose ukupnom izgledu. Međutim, ona je veoma povoljna za ažuriranje slogova. Aplikacija „Kristal“ je koncipirana tako da korisnik nakon prijavljivanja na sistem, a u zavisnosti od stranice na kojoj se nalazi, ima uvid u podatke koje je unosio i ima mogućnost da ih izmeni.

Obe, ukratko opisane kontrole, imaju opcije za dodavanje posebne „dugmadi“ za pregled u više stranica, ili za izmenu podataka, unos novih, ili brisanje. To pruža veliku fleksibilnost, ali u slučaju kontrole za pregled pojedinačnih slogova, svaki put kada korisnik pređe na novi slog, ili kada pokrene zahtev za manipulaciju podacima, aktivira se *postback* proces, odnosno vraćanje stranice ka serveru. Kada je u pitanju kontrola za detaljan pregled svih slogova, to može u velikoj meru opteretiti resurse Web servera, pa u tom slučaju treba biti oprezan.

Svako pretraživanje baze podataka za određenim slogom podrazumeva prolazak kroz sve slogove jedne, ili više tabela iz baze podataka. U tom slučaju bolje je koristiti kontrolu *GridView* koja po pokretanju zahteva uzima sve slogove iz baze podataka i snima ih u memoriju radi dalje obrade. U zavisnosti od prirode Web aplikacije i u zavisnosti od podešavanja vezanih za keširanje podataka, to se može realizovati na različite načine, od kojih neki daju dobre rezultate kada su u pitanju ograničeni resursi koje stavlja na raspolaganje Web server.

Upotreba ovih kontrola u svom izvornom obliku nepotrebno opterećuje servere. Rešenje postoji i ono se ogleda u upotrebi parametarskih komandi i klasa sa pamćenjem stanja. Kontrolni parametar za svakog ulogovanog korisnika je njegov identifikacioni broj. Prilikom učitavanja stranice, ta vrednost stoji u pripravnosti i čeka pokretanje zahteva za podacima. Nakon pokretanja zahteva i nakon uzimanja konkretnog sloga iz baze podataka, sve preuzete vrednosti se pamte u predviđenim promenljivim članicama i spremne su za dalju obradu, a veza ka bazi podataka se prekida.

3.2 Datoteke

Klasične datoteke se u Web programiranju znatno manje upotrebljavaju u odnosu na programiranje desktop aplikacija. Baze podataka su tako dizajnirane da podržavaju veliki broj istovremenih korisnika i kada bi ih koristili za smeštanje datoteka, to bi u velikoj meri uticala na smanjenje performansi. Činjenica je da *SQL Server 2008 Express* raspolaže mogućnostima za smeštanje fotografija u samu bazu, ili velikih tekstualnih polja, koja mogu poslužiti za smeštanje tekstualnih datoteka, ali kako bi se obezbedio nesmetan rad korisnicima, bez neprijatnog čekanja, bolje je primeniti nešto drugačije rešenje.

.NET raspolaže sa određenim brojem klasa koje se uspešno mogu koristiti za čitanje i izmene podataka u okviru sistema datoteka. Uz pomoć njih se mogu kreirati jednostavne tekstualne ili binarne datoteke, ili se može napraviti pretraživač datoteka. Verovatno, najznačajnija mogućnost, sa stanovišta Web programiranje je ta što se uz pomoć ovih klasa može omogućiti korisnicima da postavljaju sopstvene datoteke na Web server.

Moja namera je bila da omogućim svakom korisniku da pored podataka koje unosi, može da snimi i svoju fotografiju. Tu fotografiju bi u kasnijem radu mogao da pridruži svojoj biografiji, ukoliko konkuriše za neko radno mesto, ili da jednostavno na svojoj ličnoj stranici, pored ličnih podataka ima i svoju fotografiju. Isto tako, rukovodioci bi prilikom ocenjivanja sposobnosti zaposlenih, pored podataka unetih od strane njihovih radnika, imali i njihove fotografije, što bi stvaralo potpuniji utisak o radniku.

Kada je Web programiranje u pitanju, tu postoje određena ograničenja. Svaka nova datoteka, ne sme imati isti naziv kao i neka postojeća datoteka koja se nalazi u istom direktorijumu. Problem se može rešiti uz pomoć sistema koji generiše nazive datoteka na bazi slučajnosti, ili na osnovu slučajnog broja u kombinaciji sa tekućim datumom i vremenom. U oba slučaja, naziv datoteke bi bio statistički jedinstven, što znači da bi verovatnoća dupliranja naziva bila ekstremno niska. Takvi nazivi, međutim bi bili veoma nepraktični za upotrebu. Sa druge strane, ukoliko bi korisnici sami određivali nazive takvih datoteka, javio bi se problem jednakih unosa, verovatno u samom startu, pored toga, to bi napravilo niz bezbednostnih rizika.

Obzirom da svaki korisnik ima svoj jedinstveni identifikator, odlučio sam da njegovu vrednost, pretvorenu u string, iskoristim za formiranje imena direktorijuma i za formiranje imena datoteke. Na taj način se kreira sistem u kom svaki korisnik ima jedinstven naziv fotografije. Obzirom da se postupak ponavlja svaki put kada korisnik snima podatak na server, ne može doći do situacije da jedan korisnik ima više unetih datoteka jer unos svake nove fotografije podrazumeva njeno snimanje pod imenom koje je bilo određeno za staru fotografiju i pokreće brisanje stare fotografije. Na taj način se izbegavaju greška i štedi stalna memorija na serveru, a kako bi se osiguralo da korisnici stvarno unose fotografije, unos je ograničen ekstenziju **.jpg*. Ovo ograničenje omogućava unos samo jedne datoteke koja nosi prosleđen naziv i ima pomenutu ekstenziju.

Relacione baze podataka podržavaju opcije kao što su zaključavanje i transakcije, koje obezbeđuju konzistentnost podataka i u uslovima kada više korisnika radi istovremeno sa istim podacima. Sistem datoteka, Web servera, ne nudi takve mogućnosti, ali obzirom da je baza podataka ta koja omogućava pristup podacima, istovremeno će biti moguće da jedan korisnik pregleda unete podatke, dok drugi korisnik, menja te podatke. To podrazumeva da će korisnik koji pregleda u jednom kratkom periodu, dok se ne

pokrene novi događaj, manipulisati sa neažurnim podacima. Pored toga,logika aplikacije je takva da ne dozvoljava korisnicima da jedan dugom menjaju podatke , što sprečava nastanak nekih grešaka.

Korisnici nisu u mogućnosti da samostalno određuju nazive datoteka i putanja do njih. Na taj način se zlonamernim korisnicima onemogućava da pristupe zaštićenim sistemskim datotekama i da izmene, ili obrišu njihov sadržaj. Obzirom da ni jedan korisnik nema mogućnosti snimanja više datoteka, obezbeđeno je da ne dođe do opterećenja Web servera prekomernim zauzimanjem stalne memorije na hard disku.

3.3 Fokus

Sve Web kontrole u okviru .NET tehnologije poseduju metodu *Focus()*. Ovaj metod deluje samo za ulazne kontrole, odnosno kontrole koje prihvataju korisnički unos sa tastature. Nakon što klijentski Web pretraživač generiše stranicu, korisnik počinje svoj rad od kontrole koja ima fokus. Na primer, na stranicama gde korisnik unosi lične podatke, nakon pozivanja ovog metoda, fokus će biti na prvom tekstualnom polju ove stranice i korisnik će moći odmah da počne sa unosom prve vrednosti.

Obzirom da ovaj metod poseduje parametre za definisanje hijerarhije između kontrola koje služe za unos ili za pokretanje određenih događaja, uz pomoć njega se mogu definisati redosled aktiviranja svih polja na stranici, a korisnik će biti u mogućnosti da jednostavnim pritiskom na taster *Tab*, preskače iz jedne kontrole u drugu. Na taj način se kod korisnika stvara navika da unosi podatke određenim redosledom. Tako se ubrzava rad i izbegava se zamorno „nišanjenje“ mišem.

Fokus se može upotrebiti i za definisanje podrazumevanog komandnog dugmeta. Podrazumevano komandno dugme je ono koje će biti pritisnuto kada korisnik pritisne taster *Enter*. Ukoliko se na nekoj Web stranici nalazi obrazac, poželjno je da dugme za slanje stranice ka serveru, *Submit* dugme, bude i podrazumevano. Tako korisnik u svakom trenutku može poslati stranicu ka serveru jednostavnim pritiskom na taster *Enter*, čime se okida događaj dugmeta za slanje i snimanje podataka.

Ponekad se javlja potreba da na jednoj stranici bude više podrazumevanih dugmadi. Situacije u kojim stranice imaju više grupa ulaznih kontrola, pri čemu svaka grupa ima svoje podrazumevano dugme. To se može postići podelom stranica na delove uz pomoć panela. Za svaku kontrolu tipa *Panel*, može se definisati posebno podrazumevano dugme.

3.4 Konfigurisanje stanja sesije

Konfiguracija stanja sesije se može obaviti pomoću datoteke *web.config*. Ova datoteka omogućava podešavanje nekih naprednih opcija kao što je životni vek, ili režim stanja sesije.

Smeštanje velikih količina podataka vezanih za sesije može izazvati dodatne probleme. Stanje sesije obezbeđuje odlične performanse kada se radi sa manjim brojem korisnika, ali povećanje njihovog broja može dovesti do problema. Privremeni podaci o sesijama se mogu smeštati i u bazu podataka, čime se obezbeđuje dugoročno smeštanje

podataka: nedelje i meseci, u poređenju sa minutima. Time se, međutim, narušavaju performanse, zato što svaki zahtev za stranicom, pokreće i pretraživanje baze podataka.

ASP.NET omogućava upotrebu identifikatora sesija. Ukoliko pošaljemo zahtev sa upitnim stringom koji sadrži *ID* prekinute sesije, ASP.NET će kreirati novu sesiju koristeći taj *ID*. Međutim, i tu postoji problem, *ID* sesije se može pojaviti na nekom javnom mestu, na primer, na stranici sa rezultatima pretraživanja nekog Web *Browser*-a. Time se otvara mogućnost da više korisnika pristupe serveru sa istim identifikatorom sesije, čime bi se našli u istoj sesiji sa istim deljenim podacima.

Takav bezbednosni problem se može rešiti upotrebom opcionog atributa koji regeneriše sesiju, pod uslovom da su sesije podešene da rade bez kolačića (*cookies*). Ukoliko korisnik uspostavi vezu sa *ID*-em sesije koji je ranije korišćen i koji je istekao, sistem će u tom slučaju generisati novi *ID*. Ni u ovom pristupu nije sve savršeno, tekuća stranica će tada izgubiti sve podatke iz obrazaca, prvenstveno iz bezbednosnih razloga, a ASP.NET će izvršiti novo usmeravanje kako bi obezbedio da pretraživač dobije novi identifikator sesije.

Obzirom da je Web aplikacija „Kristal“ karakteristična kada je u pitanju unos podataka, odnosno, obzirom da ima nekoliko stranica sa velikim brojem polja za unos podataka, sesija je konfigurisana da traje 30 minuta. Ovo vreme trajanja sesije je malo više od optimalnog, ali se opet može dogoditi da u nekim situacijama sesija bude prekinuta. Zato su u slučaju ove aplikacije sesije konfigurisane tako da se *ID* obnavlja nakon isteka sesije. Ovakav pristup u ovom slučaju, ne stvara bezbednosni problem jer svaki pristup resursima aplikacije je uslovljen registracijom i prijavom korisnika na sistem. To je omogućilo i da sesije funkcionišu bez kolačića (*cookies*).

<sessionState cookieless=“UseUri“ timeout=“30“ regenerateExpiredSessinoId=“true“>
<sessionState>

3.4.1 Životni vek

Parametar vezan za životni vek *timeout* je još jedna veoma bitna opcija iz datoteke *web.config*. Ovaj parametar određuje koliko će vremena proći pre nego što ASP.NET ugasi sesiju, ukoliko u međuvremenu ne bude novih zahteva.

Parametar *timeout* podrazumeva nalaženje kompromisa kada je u pitanju stanje sesije. Razlika od samo nekoliko minuta može imati drastičan uticaj na opterećenje Web servera i performanse neke aplikacije. Poželjno je da se izabere dovoljno kratko vreme koje će obezbediti brzo oslobađanje preko potrebne memorije servera kada korisnik napusti aplikaciju, ali i dovoljno dugačko vreme kako bi klijent mogao da napravi pauzu bez nepotrebnog gubljenja sesije.

Životni vek sesije se može eksplicitno podesiti i iz programskog koda. Ukoliko neka sesija sadrži neuobičajeno veliku količinu podataka, potrebno je ograničiti životni vek sesije i upozoriti korisnika na preostalo vreme. Ukoliko se neki korisnik zadrži na stranici duže od tog vremena, sesija će biti obnovljena. Obzirom da će broj korisnika Web aplikacije „Kristal“ biti dosta manji u odnosu na prosečne Web aplikacije, mislim da će biti dovoljno da se podaci o sesiji čuvaju u okviru procesa u kom se nalaze i radne niti ASP.NET sistema.

3.5 Obrada grešaka

Ni jedan program nije otporan na greške. Izvesno je da će u nekom trenutku u budućnosti programski kod biti prekinut usled greške u samom programu, neispravnih podataka, nepredviđenih okolnosti, ili neke greške u samom hardveru. Programeri su potpuno svesni da su greške neizbežne u softverskim aplikacijama i zbog toga primenjuju defanzivni stil pri pisanju programa, uključujući u sopstveni kod posebne funkcije za hvatanje grešaka.

Greške se mogu pojaviti u različitim situacijama, od pokušaja deljenja sa nulom, zbog nepostojećih podataka, ili pogrešnih unosa, do pokušaja uspostavljanja veze sa ograničenim resursima kao što su datoteke ili baze podataka, usled isteka vremena veze sa bazom podataka, ili ako programski kod nema odgovarajuća prava pristupa.

Izuzetak tipa nulte reference predstavlja jedan od ozloglašenih tipova grešaka, a javlja se kada program pokuša da upotrebi neki objekat koji nije inicijalizovan. Ukoliko se pojavi ovakva, ili bilo koja druga greška, .NET će proveriti da li je definisana bilo kakva rutina za obradu grešaka u tekućoj oblasti važenja. Ukoliko do greške dođe u okviru nekog metoda , .NET će potražiti lokalnu rutinu za obradu greške, a zatim će pokušati da nađe takvu rutinu u kodu iz kog je taj metod pozvan. Ukoliko ne naiđe ni na jednu rutinu za obradu grešaka, sistem će zaustaviti proces obrade stranice i prikazati stranicu sa porukom o grešci u Web *browser*-a.

U zavisnosti od toga da li je zahtev za stranicom došao sa lokalnog računara, ili sa udaljenog klijentskog računara, stranica sa porukom o grešci može sadržati detaljan opis, ili će to biti samo poruka generičke prirode. Web server na koji je postavljena Web aplikacija „Kristal“ je prvenstveno namenjen u edukativne svrhe i pruža veoma detaljan sistem izveštavanja o greškama. Pomenuti sajt, *AspSpider.com*, omogućava izveštavanje koje je generiše sam .NET sistem. To je od velikog značaja, obzirom da to omogućava tačan uvid u uzroke grešaka koje nastaju tokom razvoja Web aplikacije i tokom njene implementacije na Web server. Naročito, zbog česte pojave da stranice koje na lokalnom računaru funkcionišu besprekorno, na Web serveru prijavljuju različite greške.

Sa druge strane, kada su korisnici sajta u pitanju, ukoliko se dozvoli da .NET sistem sam prijavi grešku, korisnik će dobiti izveštaj o tome, ali takvim pristupom se značajno umanjuje kvalitet Web aplikacije. Svaka pojava stranice sa porukom o grešci, narušava profesionalni izgled aplikacije, čak i onda kada je greška rezultat pogrešnog unosa, ili njen uzrok leži u nekom hardverskom problemu. Korisnici, u svakom slučaju, stiču utisak da se radi o nestabilnoj i nesigurnoj aplikaciji, što je delimično i istina. Međutim, poruke o greškama se mogu u potpunosti izbeći pažljivim dizajnom i konstrukcijom ASP.NET aplikacija. Greške se i dalje mogu pojaviti usled nepredviđenih okolnosti, ali će one blagovremeno biti uočene i identifikovane.

3.5.1 Obrada izuzetaka

Skoro svi .NET jezici podržavaju strukturnu obradu izuzetaka. Ukoliko dođe do greške u aplikaciji, .NET Framework će kreirati objekat tipa izuzetka koji detaljno opisuje nastali problem. Programer može „uhvatiti“ takav objekat pomoću rutine za obradu izuzetaka i to je prvi korak u borbi protiv eventualnih grešaka. Ukoliko ne postoji rutina za

obradu izuzetaka, doći će do prekida rada programa i korisnik će na ekranu dobiti generičku stranicu o grešci. Obzirom da takve stranice obiluju podacima koji za korisnika ne nose neku značajnu informaciju, najbolje je „uhvaćene“ objekte uposliti na drugačiji način.

Svaki pojedinačni izuzetak sadrži veliku količinu dijagnostičkih podataka koji su upakovani u uređeni objekat. Takvi objekti sadrže i svojstvo pod nazivom *InnerException*. Pomoću ovog svojstva, specifične greške možemo upakovati u šire, generičke greške. Naravno, postoji i mogućnost kreiranja sopstvenih objekata izuzetaka, koji se zatim mogu lansirati u određenim situacijama.

Obzirom da je hvatanje izuzetaka zasnovano na njihovom tipu, kod za obradu grešaka se može pravilno usmeriti i na osnovu njega se mogu formirati stranice koje će korisnika obaveštavati o grešci koja se pojavila, bez pregledanja nerazumljivih kodova grešaka. Za sada, dok je aplikacija „Kristal“ u fazi testiranja, generički izveštaji ukazuju na propuste i veoma dobro opisuju greške koje se javljaju usled neadekvatne konfiguracije. U fazi kada aplikacija bude spremna za puštanje u rad, generičke izveštaje će zameniti stranice sa porukom o grešci, koje će biti prilagođene i upućene korisnicima.

Zahvaljujući mogućnosti da se formira blok struktura za hvatanje grešaka, mogu se postaviti okidači koji aktiviraju, ili deaktiviraju različite rutine za obradu grešaka u različitim delovima programskog koda i na taj način se može obezbediti zasebna obrada grešaka za svaki deo programa. Pored toga, rutine za obradu izuzetaka podržavaju višeslojni rad, što omogućava da se postavi nekoliko rutina, jedna za drugom, od kojih će samo pojedine reagovati ukoliko se pojavi određena vrsta greške.

3.6 Postavljanje ASP.NET aplikacije

Postavljanje, odnosno instalacija ASP.NET aplikacije u okviru .NET okruženja je krajnje pojednostavljena. Posao se svodi na kopiranje direktorijuma sa Web aplikacijom na Web server, nakon čega se taj direktorijum definiše kao virtuelni. Brojne teškoće iz prošlosti kao što su, registracija komponenti, ili rešavanje konflikta između različitih verzija, su trajno otklonjene. Zahvaljujući tome, instalacija Web sajta se može obaviti jednostavno, u potpunosti ručno i bez upotrebe složenih alata.

ASP.NET aplikacije uvek rade u sprezi sa Web serverom, specijalizovanim softverskim paketom koji prihvata zahteve koristeći HTTP (*Hypertext Transport Protocol*) i obezbeđuje odgovarajuće sadržaje. Prilikom izvršavanja ASP.NET aplikacija na lokalnom računaru, koristi se virtuelna verzija Web servera, ali ukoliko Web sajt želimo da prikažemo širem spektru korisnika, potrebna nam je posebna verzija Web servera, koja raspolaže *IIS*-om (*Internet Information Services*). Pored toga, potreban je i specijalizovani softver, kojim raspolažu Web serveri, a koji omogućava razmenu elektronske pošte, manipulaciju datotekama i slično.

3.6.1 Web server

Slanje jednostavnih HTML stranica je najlakši posao koji neki Web server može da obavi. Kada klijent zatraži takvu datoteku, Web server će je jednostavno očitati sa hard diska, ili iz memorijskog keša i poslati kompletan dokument Web *browser*-u koji je aktivan

na klijentskom računaru, koji će ga zatim prikazati na ekranu. U takvom scenariju, Web server obavlja funkciju običnog servera datoteka koji prima zahteve sa mreže i isporučuje odgovarajuće dokumente.

Ukoliko Web server upravlja dinamičkim sadržajima poput ASP.NET stranica, scenario je malo složeniji. Web server ne može sam obraditi ASP.NET elemente, niti izvršavati programski kod. Za takve poslove, Web server će angažovati ASP.NET sistem i kada neki korisnik zatraži stranicu, Web server će poslati odgovarajući zahtev prema ASP.NET sistemu, koji se pokreće automatski kada se za to ukaže potreba. ASP.NET će učitati traženu stranicu, izvršiti njen programski kod, nakon čega će kreirati HTML dokument koji će biti vraćen *ISS* serveru. *ISS* server će takav dokument poslati klijentu.

ASP.NET sistem se aktivira u zavisnosti od ekstenzije tražene stranice. On poredi tipove traženih stranica, sa njegovim spiskom ekstenzija, čime određuje da li je zadužen za taj sadržaj. U spisku se, primera radi, nalazi podatak da je za datoteku sa ekstenzijom *Kristal.aspx* zadužena komponenta *KristalBiblioteka.dll*, koja se nalazi u određenom direktorijumu.

Za postavljanje Web aplikacije na Internet potreban je Web server na kom je adekvatno konfigurisan *IIS*. Softver Web servera se izvršava neprekidno i to je obično računar koji je posvećen samo tom poslu. Drugim rečima, Web server je uvek spreman za obradu HTTP zahteva i za slanje Web stranica ka klijentima, koji su povezani sa njim preko lokalne mreže ili preko Interneta.

3.6.2 Virtuelni direktorijum

Prilikom postavljanja Web aplikacije na Web server, postupak počinje formiranjem virtuelnog direktorijuma. Najjednostavnije rečeno, virtuelni direktorijum je javna strana osnovnog direktorijuma neke Web aplikacije. Između virtuelnog i osnovnog direktorijuma Web aplikacije postoji veza koja se definiše uz pomoć *IIS*-a. Praksa je da i virtuelni i osnovni direktorijum imaju isti naziv, ali to nije obavezujuće.

Da bi udaljeni korisnici mogli da pristupe sajtu pomoću svog Web *browser*-a, oni će na prvom mestu unositi adresu Web servera, zatim adresu virtuelnog direktorijuma, a nakon toga će formirati nastavak adrese i zavisnosti od same strukture direktorijuma i datoteka koje u fizičkom smislu postoje u korenom direktorijumu. Hipotetički primer takve adrese: http://WebServer/VirtuelniDirektorijum/FizickiDirektorijum/NekaStranica.aspx. Web server će obezbediti da, na osnovu konfiguracije virtuelnog direktorijuma, korisnik ostvari vezu sa fizičkim direktorijumom i da dobije željenu stranicu.

Web sajt, kreiran uz pomoć ASP.NET tehnologija, se postavlja na Web server uz pomoć *IIS*-a koji je prethodno instaliran na njemu. Prvi korak je kreiranje fizičkog direktorijuma u koji se smeštaju datoteke budućeg Web sajta. Nakon toga se taj fizički direktorijum povezuje sa virtuelnim direktorijumom i uz pomoć *IIS*-a, se definiše poistovećivanje ta dva direktorijuma. Na taj način sajt postaje javno vidljiv drugim računarima koji su povezani sa serverom. Udaljeni računari ne mogu pristupati fizičkom direktorijumu, međutim, nakon mapiranja ovog direktorijuma na neki virtuelni direktorijum, korisnici mogu slati zahteve za različitim datotekama preko *IIS*-a.

Postupak počinje kopiranjem osnovnog direktorijuma sa kompletnom strukturom stranica i direktorijuma, pri tom one mogu biti na proizvoljnom nivou dubine, nakon čega se kreira virtuelni direktorijum koji se povezuje sa fizičkim. Takvim postupkom se kreira adresa koja se sastoji na prvom mestu od naziva servera, a zatim i od naziva virtuelnog direktorijuma, nakon čega se nadovezuju nazivi direktorijuma i nazivi stranica. Kreiranjem virtuelnog direktorijuma se kreira paralela između osnovnog direktorijuma Web aplikacije i virtuelnog i svaki ugnježđeni direktorijum i svaka datoteka se može pozvati jednostavnim navođenjem njenog naziva u Web *browser*-u klijenta.

3.6.3 URL adresa Web aplikacije

ASP.NET aplikacije se mogu koristiti u različitim okruženjima i mogu se realizovati kroz različitu infrastrukturu. Samo konfigurisanje ASP.NET aplikacije se ne razlikuje kada je u pitanju vrsta mreže za koju je namenjena, uključujući lokalne mreže računara, odnosno *LAN* (*Local Area Network*), ili globalne mreže računara kao što je Internet. ASP.NET tretira mrežu kao skup uređaja povezanih komunikacionim kanalima. Klasična lokalna mreža povezuje uređaje na manjem prostoru, dok sa druge strane, Internet predstavlja brzu magistralu koja povezuje više miliona lokalnih mreža.

Internet je zasnovan na *IP*-u (*Internet Protocol*). Svaki računar u *IP* mreži dobija jedinstveni broj koji predstavlja njegovu *IP* adresu. *IP* obično definišu četiri broja u opsegu od 0 do 255, pri čemu su ti brojevi odvojeni tačkama. Pristupanje nekom od računara na mreži se obavlja uz pomoć te adrese.

Takve adrese nije baš jednostavno pamtiti, pa Web serveri na Internetu koriste jedinstvene nazive domena, poput http://aspspider.org/Crystal. Ovi domeni se jednoznačno preslikavaju u IP adrese uz pomoć specijalnih kataloga koje održava posebna mreža servera na Internetu. Ova mreža, koja se naziva DNS (*Domain Name Service*), predstavlja jednu od osnovnih komponenti infrastrukture Interneta. Kada korisnik otkuca neku adresu u Web pretraživač, pretraživač će uspostaviti vezu sa DNS serverom gde će potražiti IP adresu koja odgovara unetom domenu, nakon čega će uspostaviti vezu sa računarom koji ima tu adresu.

ASP.NET aplikacije ne moraju biti dostupne preko Interneta. Veliki broj takvih aplikacija funkcioniše samo u okvirima interne mreže. U tim situacijama nema potrebe za bilo kakvom DNS registracijom. Ostali računari mogu pristupati takvoj mrežnoj aplikaciji ili preko IP adrese računara domaćina ili, što je još verovatnije, preko naziva računara na mreži.

3.6.4 Web farme

Neke aplikacije se izvršavaju na Web farmama, odnosno na grupi servera koji zajednički opslužuju pristigle zahteve. One se obično koriste da opsluže zahtevne Web aplikacije koje moraju izdržati velika opterećenja. U takvim okruženjima, grupa računara može obaviti značajno veći broj korisničkih zahteva u odnosu na pojedinačni Web server. Web farme, su međutim, nepotrebna investicija kada su u pitanju manji ili srednji Web sajtovi.

Kada je ASP.NET u pitanju, rad Web farmi je značajno pojednostavljen. Umesto da određene datoteke neke aplikacije budu postavljene na odvojene servere koji igraju određene uloge u pružanju radnog okruženja za neki sajt, kopije celokupnih sajtova se mogu postaviti na sve servere koji sačinjavaju Web farmu. Pristigli zahtevi se, u tom slučaju, usmeravaju na jedan od servera, i to na onaj sa najmanjim opterećenjem. Međutim, sam postupak postavljanja Web aplikacije na Web farmu se prilično razlikuje od postavljanja na jedan računar koji ima ulogu Web servera.

Na prvom mestu, brojni problemi mogu nastati ukoliko se koristi stanje sesije, tada se mora upotrebiti režim *StateServer* ili *SQLServer*. U suprotnom, korisnički podaci sesije se mogu blokirati na jednom od servera. Ukoliko naredni zahtev bude preusmeren na neki drugi server, podaci će biti izgubljeni i sistem će kreirati novu sesiju.

Do problema može doći i pri radu sa stanjem pogleda i identifikacije. U oba slučaja se dešava ista stvar, ASP.NET kodira deo podataka kako bi sprečio zlonamerne izmene i proverio njihovu tačnost kasnije. To podrazumeva, da pri radu sa stanjem pogleda, ASP.NET ubacuje *hash* kod koji se proverava nakon vraćanja stranice na server, radi utvrđivanja eventualnih izmena u skrivenim poljima stanja pogleda. Ukoliko je došlo do nekih izmena, zahtev se odbacuje. Različiti serveri na Web farmi kreiraju različite *hash* kodove, tako da se može desiti da prilikom preusmeravanja na drugi računar dođe do neslaganja kodova jer svaki računar kreira kod u skladu sa tajnim ključem kojim raspolaže.

Ovaj problem bi se mogao rešiti isključivanjem *hash* koda iz stanja pogleda. Međutim, takav pristup bi stvorio bezbednosni problem. Umesto toga, bolje rešenje nudi konfiguracija koja podrazumeva da svaki server koristi isti ključ za dobijanje *hash* koda.

Ovakav pristup, takođe, otvara mogućnost zloupotrebe, ali se podatak o tome da svi serveri koriste isti ključ ne navodi, tako da zlonamerni korisnici mogu samo da pretpostave takav scenario, a i u toj situaciju, obzirom na složenost *hash* funkcija, ne otvara se, ili je jako mala, mogućnost zloupotrebe.

Podešavanje nekoliko Web servera da koriste isti ključ se može ostvariti unošenjem izmena u konfiguracionu datoteku *machine.config*, gde se uz pomoć običnog tekstualnog editora, može dodati stavka *machineKey* u okviru sekcije *system.web*. U toj sekciji se mogu definisati ključevi za šifrovanje i dešifrovanje.

Na ovaj način se eksplicitno definišu ključevi. Tako podešeni serveri jedne Web farme koriste isti ključ i to im omogućava da razmenjuju stanja pogleda i razne druge opcije, na primer, identifikaciju korisnika. Naravno takav ključ se ne može definisati ručno. Svaki pokušaj će rezultovati ključem koji nema dovoljan nivo slučajnosti. Umesto toga, najbolje je koristiti neki generator ključeva.

3.7 Zaštita Web aplikacije „Kristal“

Sistem zaštite jedne Web aplikacije se može realizovati na nekoliko načina i svaki ima svoje prednosti i nedostatke. Najkvalitetnije rešenje je realizovanje zaštite u sprezi sa samim Web serverom i upotrebom opcija za kreiranje korisničkih naloga u *Windows* okruženju. To podrazumeva i nesmetan pristup svim opcijama i programima jednog Web servera, što nije bio slučaj kada je u pitanju postavljanje Web aplikacije „Kristal“, tako da je korišćen drugačiji pristup.

ASP.NET nudi mogućnost identifikovanja korisnika putem obrazaca. Takav pristup omogućava da se putem obrazaca utvrdi identitet korisnika i da se ograniči pristup stranicama koje nisu njemu namenjene. Međutim takav pristup podrazumeva upotrebu *cookies*-a u svrhu identifikacije što svakako ne pruža dovoljan nivo sigurnosti. U tom slučaju dovoljno bi bilo da zlonamerni korisnik upotrebi računar kojim se već pristupalo resursima Web sajta i da zloupotrebi svoj „novostečeni identitet".Pored toga, identifikacija putem obrazaca nije kompletno rešenje, a i stvara nekoliko ograničenja.

Web aplikacije „Kristal" koristi višeslojni sistem zaštite koji se zasniva na nalozima korisnika. To otvara mogućnost svakom korisniku da formira korisnički nalog. Svaki nalog u sadrži korisničko ime, korisničku lozinku, sigurnosno pitanje ukoliko korisnik zaboravi lozinku, odgovor na sigurnosno pitanje i adresu elektronske pošte na koju aplikacija šalje zaboravljenu lozinku, ukoliko je u pitanju validna adresa i ukoliko se tačno odgovori na sigurnosno pitanje.

Sva navedena polja za unos korisničkih podataka su obavezna. Obzirom da klase koje služe upravljanju korisničkim nalozima pružaju veliki broj mogućnosti, naredne verzije ove aplikacije će sigurno koristiti složeniji sistem prijave korisnika, i upravljanja korisničkim nalozima. Sledeće složenije verzije ove aplikacije će imati korisnički interfejs i korisničke opcije koje će svakom korisniku dozvoljavati, odnosno sprečavati određen broj aktivnosti, u skladu sa korisničkim potrebama i svrhom posete ovoj aplikaciji. Zbog toga je onemogućeno kreiranje korisničkog naloga bez unosa svih relevantnih podataka.

3.7.1 XML

XML je čvrsto utkan u osnovu .NET sistema i predstavlja osnovu ključnih elemenata ASP.NET okruženja. Uz pomoć XML-a može se vršiti očitavanje podataka kreiranih u drugim aplikacijama, može se takođe koristiti i kao praktična zamena za jednostavne tekstualne datoteke.

XML u .NET sistemu ima skrivenu ulogu. Svaka Web aplikacija se oslanja na infrastrukturu XML-a i to je njegova primarna svrha. Svaki takav dokument se može definisati da obavlja određenu ulogu, zahvaljujući tome što nudi opštu sintaksu za skladištenje bilo kakvih podataka na konzistentan i standardizovan način uz pomoć predefinisanih XML elemenata.

Polazna osnova XML standarda je prilično jednostavna, mada konkretne implementacije mogu biti veoma komplikovane. XML je dizajniran kao format opšte namene za organizaciju podataka. Izborom ovog standarda, u principu se opredeljujemo za standardizovan način skladištenja podataka. Odatle potiču i skoro histerična zalaganja jednog dela programera za što širu primenu XML jezika i njegovu zastupljenost u Web aplikacijama. To bi donekle podrazumevalo i to da se programeri u tom slučaju bave ponovnim izmišljanjem već pronađenih stvari. Kod napisan slobodnom upotrebom ovog standarda, u zavisnosti od sposobnosti programera, može biti veoma dobro optimizovan, fleksibilan, i funkcionalan, ali pored kreatora tog koda, retko koji programer će se složiti sa tim da je takvo rešenje dobro ili bolje od nekog drugog.

XML je standard opšte namene koji predstavlja bilo kakav tip podataka pomoću elemenata, odnosno standardnih XML *tagova*. Elementi (*tagovi*) koriste isti format kao i

onaj koji se koristi i u HTML datotekama. Ipak, dok HTML elementi definišu format, XML elementi su okrenuti ka sadržaju. U osnovi, bez određenih podešavanja, XML datoteka je okrenuta isključivo podacima, tako da ne postoji standardizovan način za direktan prikaz tih podataka u Web *browser*-u.

Međutim, i pored osnovne svrhe XML-a, on ne može biti odgovarajuća zamena za baze podataka, jer ovaj standard unosi ista ograničenja kao i ostale tehnike pristupa datotekama. Samo jedan korisnik u Web aplikaciji može raditi sa jednom datotekom u određenom trenutku vremena, bez obzira da li se radi o XML ili binarnoj datoteci. Sistemi koji koriste baze podataka, sa druge strane, nude značajno fleksibilnije opcije za konkurentni višekorisnički rad, uz istovremeno obezbeđenje optimalnih performansi. Naravno, XML podatke možemo smestiti u bazu podataka, što je već uobičajen pristup u brojnim savremenim sistemima za rad sa bazama, između ostalih i *SQL Server 2005 Express*.

Web aplikacija „Kristal“ koristi XML dokumente za mnoge namene. Najveći deo parametara vezanih za aplikaciju koristi predefinisane vrednosti i zahvaljujući tim vrednostima, Web aplikaciji je obezbeđen normalan rad. Pored predefinisanih vrednosti u XML dokumentima su definisani brojni drugi parametri, počevši od podataka za povezivanje sa bazom podataka do podataka kojima se definišu pravila pristupa resursima.

3.7.2 Parametri datoteke *Web.config*

Režim identifikacije se može definisati u datoteci *Web.config* preko sekcije *authentication*. Svi elementi ove datoteke se nalaze u početnoj sekciji *configuration*, u okviru koje se nalaze sve ostale sekcije. Sekcija u kojoj se definišu parametri vezani za pravila pristupa resursima sajta se nalazi u okviru sekcije *system.web*.

Ukoliko želimo da ograničimo pristup Web sajtu, korisnicima koji nisu registrovani u bazi podataka sa korisničkim podacima, moramo definisati parametre koji upravljaju pristupom u sekciji *authorization* datoteke *Web.config*. Sledi kod:

```
<configuration>
    <system.web>
        ...
        <authentication mode="Forms">
            <forms lginUrl="~/Nalozi_Korisnika/PrijaviteSe.aspx"
        </authentication>

        <authorization>
            <allow users="*" />
        </authorization>
    </system.web>
    ...
</configuration>
```

Zvezdica (*) predstavlja džoker znak koji eksplicitno omogućava pristup aplikaciji svim korisnicima, čak i korisnicima koji nisu prošli kroz proces identifikacije. Takav način rada će biti zastupljen čak i kada ne unesemo prethodni kod u datoteku *Web.config*, zato što podrazumevani parametri koji se nasleđuju iz datoteke *machine.config* dozvoljavaju

pristup svim korisnicima. Takav način pristupa se može izmeniti ubacivanjem restriktivnog pravila kojim se ograničava pristup. Sledi kod:

```
<authorization>
        <allow users="*">
        <deny users="?"  />
</authorization>
```

Upitnik (?) je džoker znak koji označava sve anonimne korisnike. Ukoliko u datoteku *Web.config* unesemo prikazano pravilo, anonimni korisnici neće moći da pristupe Web aplikaciji. Svaki korisnik aplikacije u ovom slučaju mora biti identifikovan, a svi zahtevi korisnika se izvršavaju u skladu sa podacima o korisničkim pravima, unetim u bazu podataka o korisnicima. Ukoliko neki korisnik zatraži stranicu iz direktorijuma aplikacije, ASP.NET će utvrditi da zahtev nije prošao proces identifikacije i pokušaće da preusmeri korisnika na stranicu za prijavljivanje na sistem, gde će se nakon prijavljivanja na sistem, omogućiti korisniku da pristupi željenim resursima.

ASP.NET vrši analizu pravila iz liste po određenom redosledu, od vrha ka dnu, nakon čega će nastavlja sa analizom pravila iz eventualno nasleđene datoteke *Web.config*, iz roditeljskog direktorijuma, završavajući sa analizom parametara iz osnovne datoteke *machine.config*. Čim naiđe na neko pravilo koje se može primeniti, ASP.NET će prekinuti pretragu i primeniti pravilo na koje je naišao. U skladu sa takvim redosledom, na primeru otkucanog koda, sistem će zaključati pravilo <*allow users="*"*> i odmah ga zatim primeniti, ne vršeći analizu redova koji slede. Obzirom da navedeno pravilo definiše dozvolu pristupa svim korisnicima, sistem će dozvoliti pristup svim korisnicima, uključujući i anonimne. Međutim, ukoliko se promeni redosled pravila, sistem će zabraniti pristup anonimnim korisnicima i dozvoliti pristup svim ostalim korisnicima. Sledi kod:

```
<authorization>
        <deny users="?">
        <alow users="*">
</authorization>
```

Pravila pristupa korisnika se mogu definisati i na nivou direktorijuma, ili, na nivou datoteka. Prilikom projektovanja Web aplikacije sa kontrolom pristupa, najbolje je sve stranice takve aplikacije organizovati po direktorijumima u koje će se smeštati posebne *Web.config* datoteke. U okviru svake od njih će biti definisana pravila pristupa koja važe samo za taj direktorijum i skup Web stranica koje su smeštene u njega. Kod je identičan kodovima koji su već navedeni, jedina razlika je u tome što se te datoteke fizički nalaze u samim direktorijumima koji se štite od anonimnih korisnika. To omogućava da se za svaki direktorijum definišu specifična pravila pristupa.

Najjednostavniji i najčistiji način za regulisanje prava pristupa datotekama je preko direktorijuma. Prava se, međutim, mogu definisati i na nivou pojedinačnih datoteka ubacivanjem elemenata <*location*> u datoteku *Web.config*. Ovaj element se smešta u osnovnu sekciju <*configuration*>. Uz pomoć ovih elemenata se kreira posebna sekcija koja definiše pravila za određenu datoteku, čiji su naziv i putanja definisani kao vrednost jednog od parametara u ovoj sekciji. Pored navođenja lokacije i naziva datoteke, navode se i pravila pristupa kao i u navedenim kodovima.

Kontrola pristupa se može definisati i za svakog od korisnika pojedinačno. XML dokumenti podrazumevaju da se pravila *allow* i *deny* ne moraju koristiti isključivo u kombinaciji sa džoker znakovima u obliku zvezdice ili upitnika. Umesto toga, u njima se može navesti i konkretno ime korisnika ili kompletna lista imana razdvojenih zarezima.

Međutim, ovakav pristup se ne može iskoristiti za povezivanje sa korisničkim nalozima kreiranim u *Windows*-u koji su definisani na Web serveru, što je bilo više nego dobro kada je u pitanju Web aplikacija „Kristal". Sigurnosni model ove aplikacije se oslanja na identifikaciju korisnika putem obrazaca, i on je potpuno odvojen od sistema korisničkih naloga na Web serveru. Obzirom da besplatne *hosting* usluge u najvećem broju slučajeva ne dozvoljavaju mogućnost otvaranja korisničkih naloga na samom serveru, ovakvo rešenje se pojavilo kao najbolje.

3.7.3 Skladištenje podataka o korisnicima

Osnovna opcija sistema za rad sa nalozima i korisnicima je mogućnost skladištenja korisničkih akreditiva u bazu podataka. Teoretski spisak korisnika možemo sačuvati na bilo koji način, od XML datoteka do *Oracle* baze podataka. Svako skladištenje podrazumeva drugog provajdera podataka. ASP.NET nudi, kada je zaštita sajta u pitanju, dva provajdera: *SQL Server* i *Active Directory*.

Prilikom postavljanja sajta, prvo pitanje koje se nametnulo je bilo vezano sa softverske mogućnosti i prava koja korisnici mogu da ostvare na tom serveru. Obzirom da je bila omogućena upotreba *SQL Server*-a, a da puna verzija ovog servera u potpunosti podržava sve osobine *Express* verzije, ponuđeni provajder koji radi u sprezi sa SQL bazom podataka je bio dobro rešenje. To je omogućilo postavljanje baze podataka namenjene čuvanju podataka o korisničkim nalozima. Pored toga je bilo potrebno obezbediti da string za povezivanje sa bazom bude adekvatno podešen. Obzirom da prilikom postavljanja sajta nije moguće podešavati parametre u datoteci *machine.config*, potrebno je bilo podesiti string da odgovara konfiguraciji servera. Taj problem sam rešio zahvaljujući naprednim funkcijama ASP.NET sistema. Svako povezivanje sa bazom koristi relativne putanje, što smanjuje broj potencijalnih problema na minimum.

Svaki novi korisnik kreira svoj nalog preko kontrole za kreiranje korisničkog naloga, a uz pomoć funkcija klase *Membership*. Podrazumevani provajder sistema za rad sa nalozima će ubaciti u bazu podataka slog sa podacima o novom korisniku. Nakon završetka tog procesa kontrola koja se sastoji od nekoliko koraka će prikazati poruku da je korisnički nalog uspešno kreiran, a ukoliko neko od polja nije popunjeno korektno, ili ukoliko je ostalo nepopunjeno, kontrola će prikazati poruku upozorenja.

Sa druge strane, u bazi podataka, korisnički uneti podaci će se razvrstati u nekoliko tabela. Svako korisničko ime se unosi dvostruko, prvi unos je identičan onom koji je sam korisnik uneo, dok je drugi unos konvertovan u mala slova. Isto se događa i sa korisničkim lozinkama, s tim što se one ne čuvaju u svom originalnom obliku, već se transformišu uz pomoć *MD5*, ili *SHA hash* funkcija.

3.7.4 Sistem zaštite zasnovan na korisničkim nalozima

Osnovne funkcije sistema za rad sa korisničkim nalozima donose znatne vremenske uštede. Zahvaljujući tim funkcijama, nije potrebno dodatno opterećivati sistem zaštite dodatnim funkcijama. Sve što je potrebno nalazi se u klasama višeg nivoa *Membership* i *MembershipUser*. Pored toga, što se upotrebom funkcija ovih klasa mogu pojednostaviti poslovi vezani za razvoj sistema zaštite, time se postiže i standardizacija tih poslova. Na taj način ostale komponente i kontrole mogu koristiti klasu *Membership* radi sopstvene integracije u sigurnosni model, bez složenih podešavanja.

Posebna pogodnost, pored jednostavnosti, jeste u tome što se ovaj pristup može upotrebiti i kada je u pitanju puna verzija *SQL Server*-a i kada je u pitanju besplatna verzija *SQL Server Express*, koju koristi Web aplikacija „Kristal“. Pored toga ovakve kontrole zaštite omogućavaju fleksibilan i pristupačan način za rad sa korisnicima i organizaciju osetljivih sadržaja. Isto tako, prednost ovog pristupa se ogleda u mogućnosti da se rad sa korisničkim nalozima realizuje u sprezi sa korisničkim ulogama, preko kojih se mogu definisati prava korisnika koji su se prijavili na sistem. Obzirom na potencijal koji imaju klase namenjene radu sa korisničkim nalozima, razvoj sistema kontrole pristupa i sistema definisanja prava za korisnika, će predstavljati pravi izazov. Naredne verzije Web aplikacije „Kristal“ će imati preciznije i detaljnije definisan sistem zaštite.

3.8 Komponente

Programiranje pomoću komponenti predstavlja elegantno i jednostavno rešenje. Pravilna primena komponenti omogućava dobijanje organizovanog i konzistentnog koda koji se može višestruko upotrebiti. Pored toga, komponente se mogu uspešno primenjivati u .NET aplikacijama, čime se obezbeđuje visok stepen nezavisnosti aplikacije.

U osnovi, svaka komponenta se sastoji od određenog broja klasa koje su kompajlirane u odvojenu **.dll* datoteku. Te klase predstavljaju logičku celinu koja obuhvata međusobno povezane funkcionalne elemente. Značajna karakteristika svake komponente jeste da se može primenjivati i u više aplikacija. Svaka Web stranica i svaka druga predefinisana .NET komponenta, koristi klase iz korisnički definisane komponente kao i iz svake druge. Pored toga, komponente su i učaurene i pružaju samo one opcije koje su potrebne, dok su svi ostali detalji skriveni.

Web aplikacija „Kristal“ je realizovana kroz tri nivoa. Prvi nivo je vezan za korisnički interfejs, odnosno sloj prezentacije. U njemu su prikazane kontrole koje prihvataju vrednosti unete od strane korisnika, uz određene provere. Pored toga, tu se nalaze i funkcije koje ispunjavaju neke od kontrola podacima, kako bi korisnici imali primer vrednosti koje bi trebali da unesu. Dugi nivo, nivo poslovne logike, zadužen je za logiku same aplikacije. U okviru njega se realizuju svi događaju vezani za pripremanje podataka i upravljanje korisničkim nalozima. U trećem sloju, odnosno, u sloju podataka, je smeštena sva logika vezana za manipulaciju podacima. On sadrži logiku koja reguliše očitavanje i ažuriranje i brisanje podataka uz pomoć SQL upita i uskladištenih procedura.

Međutim, kada je ova aplikacija u pitanju, teško je napraviti jasnu granicu između slojeva. Bez obzira na to, troslojni dizajn aplikacije treba slediti kada god je to moguće. Komponente koje su namenski kreirane se koriste isključivo u drugom nivou i

predstavljaju vezu između podataka i korisničkog interfejsa. Svaki put kada se u neku od kontrola unese podatak i nakon pokretanja događaja za manipulaciju podacima poziva se određena metoda koja će, ili pribaviti određene podatke i formirati izveštaj, ili koja će unete podatke snimiti u bazu podataka, odnosno u određeni direktorijum.

Takav pristup omogućava brojne prednosti. Na prvom mestu sigurnost. Obzirom da programski kod nije smešten u Web stranicama, stranice su ograničene na opcije koje nude komponente. Na taj način neku komponentu za rad sa bazom podataka možemo definisati tako da omogućava izvršenje samo određenih operacija nad određenim tabelama, poljima, ili redovima. Takav pristup je i jednostavniji u odnosu na definisanje kompleksnih prava pristupa u samoj bazi podataka. Pošto se čitava aplikacija odvija preko takve komponente, ona će biti prinuđena da poštuje pravila nametnuta samom komponentom.

Pored toga, razvoj Web aplikacije koji se oslanja na komponente pruža mogućnost za bolju organizaciju. Obzirom da je broj mesta gde se kod ponavlja veoma mali, jednostavno je uneti izmene ili izvršiti testiranje. Jednostavno otklanjanje grešaka značajno štedi vreme i pomaže razvoju stabilnije aplikacije. Tim se postiže i veća kontrola, obzirom da svaka komponenta može obavljati svoje zadatke bez obzira na druge komponente, više različitih programera može raditi na razvoju aplikacije istovremeno.

Komponente mogu obavljati više povezanih zadataka za jedan isti zahtev klijenta, upis nekoliko slogova u bazu podataka, otvaranje i čitanje datoteke u jednom koraku, pa čak i pokretanje i kontrola transakcije sa bazom podataka. Pored toga, pravilno definisane komponente brinu o svim detaljima. One se mogu kasnije upotrebljavati bez ponovnog preispitivanja naziva baze podataka, lokacije servera, ili korisničkog naloga koji je neophodan za uspostavljanje veze i pristup resursima Web sajta.

3.8.1 Komponenta *KristalBiblioteka.dll*

Sa tehničkog aspekta, *KristalBiblioteka.dll* je biblioteka koja sadrži određeni broj klasa koje su posebno kreirane za potrebe ove aplikacije. Svaka od tih klasa koristi funkcije i metode gotovih klasa koje su implementirane u .NET sistem. Na prvom mestu to su klase koje obezbeđuju normalno funkcionisanje sistema zaštite, zatim i napredne klase ADO.NET sistema zadužene za manipulaciju podacima i podrazumevane klase .NET sistema koje su integrisane u svaku Web stranicu i koje obezbeđuju normalno ponašanje tih stranica na Web serverima.

.NET je u potpunosti objektno orijentisan. Svaka ASP.NET aplikacija je uređena po principu objekata. Pored toga što .NET omogućava rad sa objektima, on izričito zahteva njihovu upotrebu. Web stranice kreirane u skladu sa standardima ovog sistema su posebne klase. One nasleđuje klasu *System.Web.UI.Page* i sadrže uglavnom rutine za obradu događaja. Upravo te rutine u kombinaciji sa određenim događajima se mogu upotrebiti za aktiviranje neograničeno velikog broja raznovrsnih funkcija koje su definisane u namenski pravljenim klasama.

Postupak počinje kreiranjem nove biblioteke klasa u koju se naknadno dodaju nove datoteke sa klasama u zavisnosti od potreba koje se jave prilikom razvoja aplikacije. Dovoljno je da se nakon kreiranja takve datoteke doda referenca na nju i odmah nakon toga se takva klasa može pozivati po potrebi na drugim mestima.

Nove klase dobijaju svoju funkcionalnost zahvaljujući javnim metodima i svojstvima. Svaka klasa u prvoj verziji ove aplikacije podržava jednu od tabela iz baze podataka. Obzirom da se svi podaci smeštaju u promenljive koje su privatne, za pristup njima se koriste članovi koji su definisani kao javni i imaju mogućnost prihvatanja i davanja vrednosti promenljivih koje su u njima deklarisane. Sledeće verzije će raspolagati složenijim klasama, koje će umesto jednostavnog traženja podataka iz baze vršiti različite analize pre formiranja izveštaja koji će se prikazivati na Web stranicama, odnosno na sloju prezentacije. Pored toga *KristalBiblioteka.dll* raspolaže i jednom klasom koja komunicira sa klasama koje upravljaju korisničkim nalozima.

Upotreba komponente na ASP.NET Web stranici je prilično jednostavna. Web sajt mora imati kopiju komponente u svom *Bin* direktorijumu. ASP.NET automatski nadgleda ovaj direktorijum i obezbeđuje da sve Web stranice iz aplikacije mogu prići klasama smeštenim u njega. Kako bi klase koje čine stranice Web aplikacije mogle da koriste funkcionalnosti ove komponente mora se dodati referenca na tu komponentu, a kako bi aplikaciji bile dostupne i druge biblioteke iz okvira .NET sistema, ovoj komponenti se moraju dodati reference i na te biblioteke. Nakon prevođenja komponente je spremna za upotrebu.

Takvim pristupom se dobija zaokružena celina koja nesmetano funkcioniše, nezavisno od okruženja Web aplikacije. Sve izmene izvršene na delovima komponente, na postojećim klasama, dodavanje, izmena ili brisanje nekih klasa, se ažuriraju automatski, zahvaljujući naprednim funkcijama ASP.NET sistema. Međutim, pre postavljanja Web aplikacije na Web server, komponentu treba ponovo prevesti i obezbediti da ona poseduje reference na sve potrebne biblioteke iz .NET sistema.

Nakon dodavanja reference na komponentu može se kreirati neograničeno veliki broj primeraka klasa koje su implementirane u nju. Obzirom na izražen višekorisnički rad kada su Web aplikacije u pitanju, rešenje sa uvođenjem komponenti daje najbolje rezultate uz malu verovatnoću pojave grešaka. Pored toga, potrebno je i same namenski formirane klase, definisati tako da podrže višekorisnički rad.

U osnovi postoje dva pristupa definisanju i upotrebi klasa: metod instance ili statički metod. Statički metodi otvaraju nova pitanja. Klasa u tom slučaju mora pripadati grupi *stateless* klasa, odnosno klasa bez pamćenja stanja, što znači da ne može pamtiti nikakve dodatne podatke u svojim promenljivim članicama. Sa takvim pristupom može doći do konflikta ukoliko se komponenta istovremeno koristi na više mesta u programskom kodu.

Opšte pravilo je da se metod instanci koristi kada treba kreirati nekoliko primeraka klase istovremeno. Takvi metodi predstavljaju pravo rešenje za klasu *SqlConnection*, na primer, zato što se u okviru jedne operacije često mora otvoriti nekoliko veza prema različitim bazama podataka. Metod instanci odgovara situacijama kada konfiguraciju nekog objekta treba izvršiti jednom, a zatim ga upotrebiti više puta. U klasi *SqlConnection* potrebno je samo jednom definisati string za povezivanje sa bazom podataka, nakon čega je moguće otvarati i zatvarati veze prama bazi neograničen broj puta. Statičke metode, sa druge strane, treba uzeti u razmatranje kada obavljamo neki diskretan zadatak koji ne zahteva da neki objekat bude inicijalizovan. Tipičan primer su proračuni koji koriste klasu *Math*, ili procesi registracije novog korisnika u nekoj komponenti iz nivoa poslovne logike.

Sve klase svoju funkcionalnost nude preko javnih metoda. Svaka klasa poseduje određen broj promenljivih kojim se pristupa preko javnih svojstava. U ove promenljive zahvaljujući tim svojstvima mogu čuvati razne podatke i ti podaci se mogu menjati pozivom javnih svojstava. Na taj način klasa *OsnovniPodaci.cs* može imati svojstva *Ime*, ili *Prezime,* koja se koriste u svrhu snimanja ili učitavanja podataka.

Dizajn programiranja koji koristi svojstva i skladišti podatke u promenljivim članicama nazivamo i *stateful* dizajn, ili dizajn sa pamćenjem stanja. Klase u takvom dizajnu imaju punu odgovornost za čuvanje određenih podataka. U dizajnu bez pamćenja stanja, klasa ne čuva nikakve podatke između dva pozivanja metoda.

3.8.2 Klase sa pamćenjem stanja i klase bez pamćenja stanja

Programeri koji razvijaju velike aplikacije, poput profesionalnih Web aplikacija, često raspravljaju o tome koji je pristup bolji: sa pamćenjem stanja ili bez, *stateful* ili *stateless*. Dizajn sa pamćenjem stanja prirodni objektno orijentisani pristup, mada i on ima nekoliko nedostataka. Pre bilo kakvog poziva metoda radi realizacije običnih zadataka uvek moramo definisati vrednosti nekoliko svojstava. Svaki takav korak dodatno opterećuje izvršenje aplikacije. Dizajn bez stanja, sa druge strane, celokupni posao najčešće obavlja kroz samo jedan poziv metoda. Ipak, pošto nema nikakvog pamćenja podataka u sklopu stanja, mora se definisati određeni broj parametara, što otežava sam proces pisanja programa.

Ne postoji jednostavan odgovor na pitanje da li je bolji dizajn sa ili bez pamćenja stanja, jer odgovor uvek zavisi od prirode konkretnog posla. Komponente od kojih se zahtevaju visoke performanse, komponente koje koriste transakcije ili troše ograničene resurse kao što su veze prema bazama podataka, kao i one komponente koje se pozivaju daljinski, poput Web servisa, obično koriste dizajn bez pamćenja stanja, što je najjednostavniji i najpouzdaniji pristup. Pošto se u memoriji ne čuvaju nikakvi podaci, ove komponente troše manje serverskih resursa, a ne postoji ni mogućnost gubljenja dragocenih podataka usled softverskih ili hardverskih grešaka.

3.9 Keširanje

ASP.NET aplikacije nose u sebi određene kontradiktornosti. Sa jedne strane, zahvaljujući tome što su predviđene radu na Internetu, one imaju jedinstvene zahteve, tačnije, one moraju opslužiti na stotine klijenata jednostavno i brzo, baš kao da se radi o jednom klijentu. Sa druge strane, ASP.NET nudi nekoliko opcija koje omogućavaju razvoj Web aplikacije i pisanje njenog koda na isti način kao i pri kreiranju desktop aplikacije. Te opcije su veoma korisne, ali mogu doneti i određene probleme. Pri radu u ASP.NET okruženju veoma lako se zaboravlja da se radi na razvoju Web aplikacije, i ubrzo se počinje sa primenom tehnika koje usporavaju rad aplikacije ukoliko ona radi sa većim brojem korisnika.

Srećom, postoji i srednje rešenje koje omogućava upotrebu opcija koje štede vreme kao što su stanje pogleda, Web kontrole, ili stanje sesije, a da se pri tom dobije robusna Web aplikacija. Da bi se završio posao na korektan način, neophodno je i dodatno vreme za optimizaciju Web sajta. Keširanje je jedan od najjednostavnijih načina za poboljšanje performansi Web aplikacije. To je tehnika koja privremeno skladišti dragocene podatke u

memoriji servera ili u memoriji klijentskog računara, čime se omogućava njihova ponovna upotreba.

Keširanje može biti i nepotreban ukras, međutim, ukoliko se upotrebi na pravi način, kešriranje može dvostruko, trostruko, pa čak i desetostruko poboljšati performanse putem skladištenja važnih podataka u kratkim vremenskim intervalima. Najčešće su u pitanju podaci koji se preuzimaju iz baze podataka. Takav pristup je sasvim opravdan jer pribavljanje podataka iz baze zahteva dosta vremena. Uz pažljivu optimizaciju može se smanjiti vreme angažovanja i ukupno opterećenje baze podataka. Na takav način se obezbeđuju podaci i bez uspostavljanja veze sa bazom podataka i bez realizacije upita jer se podaci pribavljaju direktno iz memorije servera.

Obzirom da se na sajtu Web aplikacije „Kristal“ ne očekuje velika posećenost, keširanje podataka se primenjuje u retkim slučajevima i stoji kao opcija za naredne verzije aplikacije. Nepotrebno je učitavati podatke iz baze i zauzimati memoriju servera, bez obzira na njihovu malu količinu, ako takav pristup nema efekta na smanjenje ukupne zauzetosti resursa, usled malog broja korisnika aplikacije. Međutim, „Kristal“ koristi keširanje podataka na strani klijenta, što je korektan pristup kada je u pitanju racionalno korišćenje resursa Web servera.

Skladištenje podatak u memoriji servera ne mora biti pravo rešenje, naročito ako su u pitanju veoma velike baze podataka. Memorija servera je ograničen resurs, ukoliko se pretera sa keširanjem, deo podataka će biti smešten u virtuelnu memoriju na hard disku, što će uzrokovati usporavanje rada celog sistema. Pored toga, kada neki podatak smestimo u keš, sa velikom verovatnoćom očekujemo da ćemo ga tamo naći i kasnije. Životni vek takvog podatka, međutim, nije neograničen i u nadležnosti je servera. Ukoliko keš postane prepun, ili neka druga aplikacija zauzme veliku količinu memorije, server će početi selektivno da ga prazni, vodeći računa da pri tom ne naruši performanse aplikacije.

3.9.1 Kada koristiti keširanje

Tajna pravilne primene sistema keširanja leži u određivanju pravog vremena za njegovu upotrebu. Kvalitetna strategija keširanja utvrđuje najčešće korišćene podatke čije kreiranje zahteva najviše vremena, nakon čega se ti podaci smeštaju u keš. Ukoliko se u keš smeštaju nepotrebni podaci, lako se može desiti da bude popunjen nepotrebnim podacima i da na taj način bude sprečeno skladištenje podataka koji su zaista neophodni.

Treba keširati one podatke čije vreme kreiranja zahteva dosta vremena. Tipičan primer su upiti nad bazom podataka. Pored toga što uspostava veze sa bazom ili otvaranje datoteke zahteva dosta vremena, takve operacije mogu isključiti druge korisnike koji pokušavaju da obave istu operaciju u istom trenutku. Učestanost upotrebe određenih podataka diktira i potrebu za keširanjem. Trošenje memorije za skladištenje podataka koji su retko potrebni nema smisla. Uvažavajući ovaj princip, nema smisla keširati podatke o svim korisnicima aplikacije jer ne postoji takva potreba, ali keširanje podataka o konkretnom, trenutno prijavljenom korisniku ima značaja jer njegovi podaci imaju određenu ulogu prilikom pokretanja većine događaja.

Performanse Web aplikacije su određene brzinom rada Web stranica za pojedinačnog korisnika. Keširanje poboljšava performanse, zato što se na taj način

izbegavaju uska grla kao što je pristup bazi podataka. Zahvaljujući keširanju Web stranice se znatno brže obrađuju i šalju klijentu.

Skalabilnost se meri degradacijom performansi Web aplikacije usled povećanja broja istovremenih korisnika. Keširanje popravlja i skalabilnost Web aplikacije, zato što klijenti mogu ponovo upotrebiti keširane podatke u okviru zahteva koji stižu u relativno kratkim vremenskim intervalima. Sa keširanjem raste i broj ljudi koji mogu istovremeno pristupiti Web sajtu, mada se broj veza sa bazom podataka neće drastično smanjivati, ukupno opterećenje sistema će ostati relativno konstantno.

3.9.2 Keširanje u ASP.NET sistemu

Najjednostavniji vid keširanja je keširanje izlaza. Pri njemu se u memoriju smešta konačna verzija generisane HTML stranice koja se šalje klijentu. Kada naredni klijent zatraži istu stranicu, server neće pokrenuti njeno generisanje, već će automatski poslati keširanu verziju iz memorije. Time se štedi kompletno vreme potrebni za pokretanje stranice i izvršavanje njenog koda.

Prilikom ponovne kompilacije keširane stranice ASP.NET automatski uklanja stranicu iz memorije. Time se izbegavaju problemi koji mogu nastati ukoliko sadržaj nije ažuran prilikom upotrebe već keširane verzije stranice. Pored toga, za svaku stranicu se može eksplicitno navesti vreme osvežavanja. Vreme od dvadeset sekundi je optimalno. Iako period od dvadeset sekundi premalo, na sajtovima koji beleže veliku posećenost, to vreme drastično povećava performanse. Ovaj limit od ne predstavlja i garanciju da će stranica zaista toliko i izdržati u memoriji. Stanica može biti izbačena i ranije, ukoliko sistem zaključi da nema dovoljno memorije, ili ukoliko dođe do izmena u podacima koji se prikazuju na stranici.

Keširanje podataka se, sa druge strane, inicira ručno iz programskog koda. Pri tome se u memoriju smeštaju važniji podaci čija rekonstrukcija zahteva dosta vremena, poput objekta *DataSet* dobijenog selekcijom podataka iz baze. Druge stranice mogu proveriti da li u memoriji već postoje takvi podaci i eventualno ih iskoristiti, čime se izbegavaju uobičajeni koraci neophodni za njihovo pribavljanje. Keširanje podataka je u konceptualnom smislu identično stanju aplikacije, ali je više naklonjeno serveru, zato što će podaci biti uklonjeni iz memorije automatski kada njihova veličina pređe određenu granicu i eventualno ugroze performanse sistema.

Specijalni vidovi keširanja su delimično i fragmentirano keširanje. Umesto keširanja HTML koda kompletne stranice, u memoriju se smešta HTML kod samo jednog dela stranice, odnosno samo HTML kod korisničkih kontrola koje se nalaze na stranici. Pri narednom izvršavanju stranice biće aktivirani isti događaji koji su vezani za stranicu, uključujući i programski kod, ali bez izvršavanja koda koji je vezan za korisničke kontrole.

Keširanje izvora podataka je tip koji je ugrađen u kontrole izvora podataka, uključujući i kontrole *SqlDataSource*, *ObjectDataSource*, ili *XmlDataSource*. U tehničkom smislu, ove kontrole koriste keširanje podataka. Prednost je u tome što su u ovom slučaju ne mora eksplicitno voditi računa o procesu keširanja, umesto toga dovoljno je da se definišu određena svojstva, nakon čega će ove kontrole same upravljati smeštanjem podataka u memoriju i njihovim učitavanjem.

3.9.3 Keširanje na strani klijenta

Web stranicu je moguće u potpunosti keširati na strani klijenta. U tom slučaju Web pretraživač čuva kopiju stranice na lokalnom računaru i automatski koristi tu kopiju kada klijent ponovo dođe na nju ili ponovo zada njenu URL adresu. Ukoliko klijent osveži sadržaj stranice, njena kopija se briše iz memorije i šalje se ponovni zahtev serveru, koji će ponovo izvršiti odgovarajući programski kod.

Keširanje na stani klijenta se ređe primenjuje u odnosu na keširanje na strani servera. Pri klijentskom keširanju stanica se mora ponovo kreirati za svakog pojedinačnog korisnika, tako da uštede u izvršenju programskog koda i u pristupu bazi podataka nisu ni približno tolike kao pri keširanju na strani servera, jer u tom slučaju keširanu verziju stanice dobijaju svi korisnici.

„Kristal“, obzirom da skoro svaka stranica nosi određenu količinu ličnih podataka, koristi keširanje na strani klijenta. To je realizovano primenom delimičnog keširanja i smeštanjem karakterističnih podataka za konkretnog klijenta na njegov računar. Prilikom prijavljivanja na sistem, korisnik pokreće i procedure koje selektuju podatke potrebne dalje unose u bazu podataka. Obzirom da stranice na kojima se generišu izveštaji nemaju prioritet kada je u pitanju brzina, i obzirom da količina podataka za izveštavanje u budućnosti može biti velika, pokretanje tih stranica ne podrazumeva automatsko povlačenje podataka iz baze, ali kada su u pitanju događaji koji pokreću funkcije za unos podataka u bazu, određeni podaci su uvek dostupni. Kada korisnik popuni formu za unos podataka, oni se na osnovu podataka vezanih za korisnički nalog konkretnog korisnika, koriste za ažuriranje određenog sloga u bazi podataka. Svako slanje podataka ka Web serveru se izvršava tek kada korisnik pokrene događaj za snimanje podataka u bazu. U međuvremenu svi podaci se nalaze na korisničkom računaru i tu čekaju dalju obradu.

3.9.4 Keširanje i upitni string

Jedno od najvažnijih pitanja na koje sistem keširanja mora da odgovori glasi: „Kada se može koristiti keširana verzija stranice, a kada je potrebno koristiti sveže podatke?“. Programeri koji su skloni nedostatku strpljenja i po prirodi naklonjeni trenutnim zadovoljstvima, često dodatno naglašavaju značaj podataka u realnom vremenu. Keširanje se obično može primeniti bez ikakvih problema pri radu sa relativno statičnim podacima, pri čemu je moguće ostvariti značajna poboljšanja performansi.

Međutim, često se javlja potreba za dinamičkim podacima. Keširanje čitave stranice u takvim uslovima nije dobro rešenje, zato što se ista stranica ne može upotrebiti kao odgovor na zahteve koji potiču od različitih korisnika. Dinamički sadržaj imaju i stranice koje primaju podatke od drugih stranica putem upitnih stringova. Sadržaj takvih stranica je previše dinamičan da bi mogao da se kešira, ali postoje rešenja.

Ukoliko ASP.NET naloži da se sadržaj jedne stranice smesti u keš, i pri tome druga stranica pošalje zahtev sa URL adresom keširane stranice, ASP.NET će poslati već keširanu stranicu. U takvom scenariju se stvari neće promeniti, ASP.NET će i dalje koristiti istu keširanu verziju stranice sve dok ne istekne njen životni vek. Na osnovu toga bi se moglo zaključiti da keširanje izlaza nije pravo rešenje za stranice koje koriste upitne stringove, međutim, ASP.NET nudi jednu opciju koja bi mogla da reši takve probleme.

Ukoliko u atribut *VaryByParam* ubacimo (*), ASP.NET će znati da stranica koristi upitne stringove, zbog čega će u memoriju smeštati odvojene kopije stranica za različite argumente iz upitnog stringa. Sledi kod:

<%@ OutputCache Duration="20" VaryByParm="" %>*

Kada korisnik zatraži stranicu sa dodatnim parametrima u upitnom stringu, ASP.NET će prvo proveriti sadržaj upitnog stringa. Ukoliko string odgovara prethodnom zahtevu i u memoriji postoji kopija te stranice, ona će biti poslata korisniku. U suprotnom, kreira se nova kopija stranice, koja će biti smeštena odvojeno u memoriji.

Keširanje izlaza funkcioniše odlično sa stranicama koje su zavisne isključivo od podataka koji su smešteni na serveru i od podataka iz upitnog stringa. Ova varijanta, međutim, ne odgovara situacijama u kojima sadržaj stranice zavisi od podataka koji su karakteristični za stanje sesije ili podataka iz *cookies*-a, zato što sistem u tom slučaju ne može menjati keširane verzije. Keširanje izlaza nije pogodno ni za dinamičke stranice koje menjaju svoj sadržaj u skladu sa kontrolnim događajima. U takvim situacijama uspešno se primenjuje delimično keširanje pri čemu se u keš smešta samo deo stranice, ili keširanje podataka radi skladištenja samo jednog dela podataka.

Postavljanjem znaka (*) u atribut *VaryByParam* unosi se nepotrebna neodređenost. Umesto toga, u ovom atributu se može navesti naziv konkretnog parametra iz upitnog stringa. Sledi kod:

<%@ OutputCache Duration="20" VaryByParm="korisnik_id" %>

U ovom slučaju ASP.NET će proveriti sadržaj upitnog stringa i pokušati da pronađe parametar *korisnik_id*. Zahtevi sa različitim parametrom *korisnik_id* keširaju se odvojeno, dok će svi ostali parametri iz upitnog stringa biti ignorisani. Takvo rešenje je veoma praktično ukoliko se stranici mogu proslediti i drugi parametri koje ona ne koristi. ASP.NET ne može sam razlikovati „važne“ i „nevažne“ parametre bez prethodne intervencije programera. ASP.NET sistem omogućava da se u atribut ubaci više parametara, ali oni moraju biti razdvojeni znakom tačka-zarez (;). Tu, takođe, važi isto pravilo, ASP.NET kešira odvojene stranice, pod uslovom da se upitni stringovi međusobno razlikuju po vrednostima.

<%@ OutputCache Duration="20" VaryByParm="korisnik_id;status;datum" %>

3.9.5 Neposredno upravljanje keširanjem

Parametri upitnog stringa nisu jedini način za dobijanje više različitih keširanih verzija neke stranice. ASP.NET omogućava i kreiranje zasebnih procedura koje odlučuju o tome da li treba kreirati novu keširanu verziju stranice ili upotrebiti postojeću. Takva procedura vrši proveru svih neophodnih podataka i vraća određeni string čija vrednost zavisi od rezultata provere. ASP.NET će zatim upotrebiti taj string radi keširanja stranice. Ukoliko programski kod u proceduri generiše isti string za različite zahteve, ASP.NET će ponovo upotrebiti već keširanu stranicu. Nakon generisanja stringa koji ima drugačiju vrednost, ASP.NET će generisati novu keširanu verziju stranice i smestiti je u keš odvojeno.

Na primer, korisnički definisano keširanje možemo upotrebiti radi keširanja u zavisnosti od vrste Internet *browser*-a, ili za dobijanje različitih verzija stranice u zavisnosti od jezika koji koristi pretraživač klijenta. Pored toga, ovo otvara mogućnost da se napravi opcija kojom će korisnik birati jezik kojim će biti ispisan tekst na stranicama i kontrolama. Na taj način će korisnik koji odabere određeni jezik, dobiti stranice koje su prevedene na željeni jezik, ili će se sadržaj prilagođavati tipu *browser*-a koji koristi klijent. Da bi ovakvo, selektivno, keširanje bilo moguće u sve stranice koje treba keširati potrebno je dodati direktivu *OutputCache*. U atributu *VaryByCustom* potrebno je navesti naziv željenog korisničkog keširanja.

```
<%@ OutputCache Duration= ”20” VaryByParm=”None”
    VaryByCustom=”Browser” %>
```

Funkcija koja definiše ovakvo keširanje prosleđuje string sa vrednošću *VaryByCustom* u parametru *arg*. To omogućava da se kreira aplikacija koja će primenjivati više vrsta korisnički definisanog keširanja u istoj funkciji. Svaka od tih vrsta mora imati drugačiji *VaryByCustom* naziv. Funkcija, zatim, pronalazi ovu vrednost u parametru *arg*, nakon čega vraća odgovarajući string keširanja. ASP.NET će u skladu sa svojim procedurama, kreirati i uskladištiti odvojenu verziju stranice za svaki string keširanja na koji naiđe.

Direktiva *OutputCache* poseduje i treći atribut koji se može koristiti u podešavanjima keširanja. To je atribut *VaryByHeader*, koji omogućava skladištenje različitih verzija kopija stranice na osnovu vrednosti iz HTTP zaglavlja u okviru zahteva za stranicom. Ovo se može upotrebiti za izgradnju višejezičkog sajta, gde bi se ovakvim pristupom keširale različite verzije stranica u zavisnosti od jezika koji koristi pretraživač klijenta.

```
<%@ OutputCache Duration= ”20” VaryByParm=”None”
    VaryByHeader=”Accept-Language” %>
```

3.9.6 Keširanje podataka

Keširanje podataka je znatno fleksibilnije u odnosu na keširanje stranica i za razliku od keširanja stranica, zahteva dodatne rutine koje se moraju implementirati u programski kod. Keširanje podataka funkcioniše tako što se objekti čije kreiranje zahteva dosta vremena smeštaju u poseban, objekat pod nazivom *Cache*. Keš je svojstvo klase *Page* koje vraća primerak klase *System.Web.Caching.Cache*. On ima globalni karakter i dostupan je svim zahtevima, svih klijenata o okviru aplikacije.

Objekat *Cache* funkcioniše u zaštićenoj niti, što isključuje potrebu za eksplicitnim zaključavanjem i otključavanjem ovog objekta pre dodavanja ili uklanjanja pojedinačnih elemenata. Unutrašnji objekti koji su smešteni u objekat *Cache,* nemaju takav stepen zaštite. Ukoliko se prilikom izvršavanja nekog dela aplikacije kreira neki objekat može se desiti da više korisnika pokuša da mu priđe istovremeno, što može dovesti do pojavljivanja nekonzistentnih podataka. Međutim, taj problem se može rešiti u programskim kodom koji pravi određeni broj objekata, u zavisnosti od broja klijentskih zahteva.

Elementi se u okviru objekta *Cache* brišu automatski nakon isteka njihovog životnog veka, ili ukoliko dođe do izmene nekog objekta, ili datoteke od koje objekat

zavisi, ili ukoliko server ostane bez raspoložive memorije. Obzirom na ovo, prilikom upotrebe keširanja, programski kod mora biti organizovan tako da svaki put proveri da li željeni objekat postoji u kešu, pre nego što pokuša da ga upotrebi. U suprotnom, sistem će generisati izuzetak tipa prazne, odnosno *null* reference.

3.9.7 Keširanje sa kontrolama izvora podataka

Kontrole *SqlDataSource*, *ObjectDataSource*, i *XmlDataSource* sadrže ugrađenu podršku za keširanje podataka. Keširanje sa ovim kontrolama je neophodno, zato što, za razliku od korisnički definisanog koda za pristup podacima ove kontrole uvek iznova izvršavaju upite nakon svake *postback* operacije. Pored toga, one izvršavaju i dodatni upit nad izvorom podataka za svaku povezanu kontrolu. Ukoliko stranica sadrži tri kontrole vezane za istu kontrolu izvora podataka, pre generisanja stranice nad bazom podataka će biti izvršena tri odvojena upita. Čak i minimalni nivo keširanja može znatno smanjiti opterećenje sistema.

Ukoliko je aktivirana opcija keširanja u kontroli *SqlDataSource*, u keš će biti smešteni rezultati komande *SelectCommand*. Ukoliko, međutim, definišemo upit sa dodatnim parametrima, *SqlDataSource* će keširati odvojene rezultate za svaku pojedinačnu grupu parametara. Ova mogućnost se uspešno može koristiti za pravljenje jednostavne verzije padajućeg menija.

Pretpostavimo da želimo da napravimo stranicu koja ima dve kontrole, prava kontrola je namenjena pregledanju podataka dobijenih iz tabele i vezana je za komandu koja je predstavlja upit nad tabelom u zavisnosti od vrednosti parametra. Druga kontrola je *DropDownList* u kojoj korisnik može da selektuje neku od vrednosti koja će preko parametara biti prosleđena prvoj kontroli i na osnovu koje će kreirati izveštaj.

Ukoliko imamo dve tabele, jednu sa adresama stanovanja zaposlenih i drugu sa ličnim podacima zaposlenih, u kontrolu *DropDownList* možemo učitati nazive gradova. Svaki put kada korisnik selektuje neki grad, program će izvršiti novi upit radi dobijanja spiska zaposlenih koji su iz tog grada. Upit se koristi radi popunjavanja kontrole *DataSet*, koja se zatim smešta u keš memoriju sa maksimalnim vremenom od šest stotina sekundi. Ukoliko korisnik selektuje neki drugi grad, proces se ponavlja, a novi objekat *DataSet* se smešta u keš memoriju nezavisno od prethodnog. Ukoliko korisnik selektuje grad koji je već selektovao, ili koji je selektovao neki drugi korisnik, program će očitati odgovarajući objekat *DataSet* iz memorije,pod uslovom da životni vek tog objekta nije istekao.

Na ovaj način i uz pomoć panela se može urediti da na jednoj stranici postoji više različitih izveštaja iz više različitih tabela. Jedino što je potrebno je da na jednoj stranici postoji više izvora podataka i više kontrola koje mogu da proslede vrednost. Ono što preostaje jeste da se u okviru svakog od panela definiše potreban broj izvora podataka i odgovarajući broj kontrola za manipulaciju podacima. Pored toga, potrebno je obezbediti da se unete ili odabrane vrednosti, preko parametara, proslede kontrolama koje imaju zadatak da formiraju izveštaj u zavisnosti od vrednosti parametra koji se dodeljuje SQL upitu nakon klauzule WHERE.

3.9.8 Zavisno keširanje i SQL Server

SQL Server pruža mogućnost automatskog održavanja stanja keša. Ta mogućnost se može uključiti, aktiviranjem sistema izveštavanja u bazi podataka, na sledeći način. ASP.NET raspolaže programom pod nazivom *aspnet_regsql.exe*, koji se nalazi u okviru instalacionog direktorijuma .NET Framework-a. Program se poziva iz komandne linije i pri pozivanju se ubacuje prekidač *–ed* u komandnu liniju. Pored toga, u komandnoj liniji se mora identifikovati server (*-E* se koristi za proverenu vezu, a *–S* radi izbora drugog servera koji nije na lokalnom računaru) i bazu podataka (prekidač *-d*). Sledi naredba:

aspnet_regsql -ed -E -d KristalDB

Nakon izvršenja komande u bazu *KristalDB*, koja je kreirana ranije i čija se struktura neće menjati, će biti dodata tabela *SqlCacheForChangeNotification*. Tabela *SqlCacheForChangeNotification* ima tri kolone: *tableName*, *notificationCreated* i *changeId*. U ovoj tabeli se registruju eventualne izmene u bazi podataka *KristalDB*. Kada dođe do neke izmene, SQL Server upisuje novi slog u ovu tabelu, a ASP.NET-ov sistem prozivanja povremeno ispituje njen sadržaj.

Ovakvo rešenje je najbolje od svih koje nudi .NET Framework. Na prvom mestu zbog brzine, jer tabela u kojoj se registruju izmene u bazi je znatno manja od tabele sa keširanim podacima, tako da je izvršavanje upita znatno brže. Pored toga, ova tabela se ne koristi za druge poslove, tako da očitavanje njenih slogova nije ugroženo eventualnim problemima oko zaključavanja i konkurentnosti. Obzirom da više tabela iz ove baze podataka koristi istu tabelu za registrovanje izmena, istovremeno se mogu pratiti izmene u većem broju tabela, bez preopterećivanja sistema pozivanja.

U svakoj pojedinačnoj tabeli iz baze podataka potrebno je aktivirati podršku sistemu izveštavanja o promenama, i pored toga što je tabela za keširanje podataka o izmenama već kreirana. Podrška se aktivira uz pomoć iste alatke *aspnet_sqlreg* i parametara *–et* za otključavanje izveštavanja i *–t* za određivanje tabele po nazivu. Ovom alatkom se može kreirati okidač koja uvodi novu tabelu u sistem, u ovom slučaju, tabelu sa osnovnim podacima o korisniku, koja omogućava praćenje promena u pomenutoj tabeli. Sledi naredba:

aspnet_regsql -et -E -d KristalDB -t TabelaOsnovniPodaci

Klasičan upit *SELECT * FROM TabelaOsnovniPodaci*, vraća sve slogove sa svim kolonama iz ove tabele, ali da bi smo utvrdili da li je bilo operacija nad tom tabelom potrebno je kreirati okidač koji će registrovati svaku promenu. Kod okidača:

```
CREATE TRIGGER kristalDBObject.[TabelaOsnovniPodaci_SqlCache_Trigger]
  ON [TabelaOsnovniPodaci]
  FOR INSERT, UPDATE, DELETE AS BEGIN

  SET NOCOUNT ON
  EXEC kristalDBObject.SqlCacheChangeProcedure N 'TabelaOsnovniPodaci'
END
```

Kada dođe do bilo kakve izmene u tabeli koja se nadzire, okidač će aktivirati uskladištenu proceduru *SqlCacheChangeProcedure*. Ova procedura će uvećati vrednost *changeId* u odgovarajućem redu tabele za praćenje izmena u bazi. Kod procedure:

```
CREATE PROCEDURE kristalDBObject.SqlCacheChangeProcedure
  @imeTabele NAVCHAR (450)
AS

  BEGIN
  UPDATE kristalDBObject.SqlCacheForChangeNotification WITH (ROWLOCK)
    SET changeId = changeId + 1
    WHERE imeTabele = @imeTabele
  END
GO
```

Tabela *SqlCacheForChangeNotification* sadrži po jedan slog za svaku tabelu koja se nadzire. Kao što možemo videti, kada dođe do izmene u tabeli, odgovarajuća kolona *changeId* se uvećava za jedan. ASP.NET redovno proverava sadržaj tabele i pamti najnovije vrednosti *changeId* za svaku tabelu. Ukoliko se prilikom provere utvrdi da je neka *changeId* vrednost izmenjena, ASP.NET će automatski znati da je došlo do promene u odgovarajućoj tabeli baze. Ukoliko dođe do izmene u bilo kom slogu, sistem će registrovati promenu i proglasiti keširanje rezultate upita nevažećim.

Pored svega navedenog, postoji još jedan korak koji će ASP.NET-u naložiti da vrši redovne kontrole baze podataka. Svaka aplikacija koja koristi sistem za praćenje stanja keša mora uspostaviti sopstvenu vezu i samostalno povremeno zvati tabelu za praćenje promena.

Servis pozivanja se može aktivirati preko elementa *sqlCacheDependency* u datoteci *web.config*. Servis se uključuje postavljanjem vrednosti *true* u atribut *enabled*, dok se u atribut *pollTime*, ubacuje broj milisekundi između dva pozivanja. Vreme treba da bude optimalno i zavisi od same aplikacije. Pored toga, potrebno je definisati i vezu prema bazi podataka upotrebom stringa za povezivanje.

```
<configuration>
        ...
        <system.web>
                <SqlCacheDependency enabled="true" pollTime="15000">
                        <databases>
                                <add name="KristalDB" connectionStringName="Kristal">
                        </databases>
                </SqlCacheDependency>
                ...
        </system.web>
</configuration>
```

Preostaje još da se doda programski kod. To se radi tako što se kreira novi objekat *SqlCacheDependency*, koji se zatim prosleđuje metodu *Cache.Insert()*. Konstruktor objekta *SqlCacheDependency* zahteva dva parametra u obliku stringa. Prvi sadrži naziv

baze podataka koji je definisan u elementu *add* sekcije *sqlCacheDependency* datoteke *web.config*. Drugi je naziv povezane tabele. Sledi kod:

```
SqlCacheDependency tabelaOPDependency  = new
        SqlCacheDependency(“KristalDB”, “TabelaOsnovniPodaci”);

Cache.Insert(
“TabelaOsnovniPodaci“,dataSetTabelaOsnoviPodaci,tabelaOPDependency);
```

Sloj podataka je veoma osetljiva oblast po pitanju performansi u većini Web aplikacija. Činjenica je da se uz pomoć keširanja i uz veoma mali deo programskog koda može drastično smanjiti opterećenje baze podataka i povećati skalabilnost Web aplikacije. Bilo koja vrsta optimizacija, sa druge strane, zahteva dosta stres testiranja i podešavanja radi postizanja dobrih rezultata. Bez toga, lako se može dogoditi da i uz veliki trud i složena ispravljanja programskog koda, dobiju samo minimalna poboljšanja performansi.

3.10 Izgled Web aplikacije „Kristal“

Stilovi su deo CSS (*Cascading Style Sheet*) standarda. Uz pomoć stilova se može podesiti izgled Web stranica. Oni nisu direktno vezani za ASP.NET sistem, ali mogu biti veoma korisni prilikom formiranja konzistentnog formata na čitavom Web sajtu. Pomoću stilova je moguće definisati grupu opcija formatiranja, nakon čega se one mogu upotrebiti za formatiranje različitih elemenata na većem broju stranica.

Stilovi su izuzetno praktični, ali postoje i stvari koje oni ne mogu obaviti. Stilovi su zasnovani na HTML standardu, tako da ne mogu raditi sa ASP.NET konceptima kao što su svojstva kontrola. ASP.NET otklanja taj nedostatak uvođenjem tema, koje imaju sličnu ulogu kao i stilovi, ali rade isključivo sa serverskim kontrolama i uz pomoć njih se mogu automatski definisati svojstva ASP.NET kontrola.

Standardizacija Web sajta se može obaviti i primenom *master* stranica. One su šabloni za pojedine delove Web sajta i njihova primena omogućava podešavanje nekih funkcionalnosti koje su potrebne na svakoj od stranica i definisanje izgleda Web stranica, definisanjem standardnih detalja kao što su zaglavlje, meni i podnožje. Jednom napravljena *master* stranica se kasnije koristi za kreiranje svih ostalih stranica sa sadržajem. Svaka stranica sa sadržajem automatski preuzima izgled i sadržaj glavne stranice sa kojom je povezana.

Primenom stilova, tema i *master* stranica možemo postići standardizovan izgled svih stranica na našem Web sajtu. Standardizovan izgled spada u one detalje koji čine razliku između prosečnog i profesionalnog Web sajta.

3.10.1 Stilovi

U prvim danima Interneta dizajneri Web sajtova su koristili HTML opcije formatiranja radi poboljšanja izgleda stranica. Te opcije su imale ograničene mogućnosti, nisu bile dosledne, što je i danas slučaj, i nisu bile dovoljno podržane. Pored toga, takvo formatiranje je davalo veoma zapetljan *markup* kod.

Rešenje je ponuđeno u CSS standardu, koji podržava većina savremenih Web čitača. CSS obezbeđuje omogućava širok spektar konzistentnih svojstava formatiranja, koji se mogu primeniti na bilo koji HTML element. Stilovi omogućavaju postavljanje okvira, definisanje izgleda fonta, promenu boja, određivanje margina i slično.

CSS podržava i nasleđivanje. Zahvaljujući tome, pojedine opcije formatiranja, poput familije fontova se prosleđuju od roditeljskog elementa ka drugim ugneždenim elementima. Drugim rečima, ukoliko je definisan određeni tip fonta za celu stranicu, svaki element na toj stranici će naslediti to svojstvo, ukoliko se u okviru konkretnog elementa ne definiše neki drugi font. Ostala svojstva, kao što su margine i praznine, ne koriste nasleđivanje.

U Web aplikaciji „Kristal“ postoje tri datoteke koje definišu izgled stranica. Jedna od njih je zadužena za formatiranje izgleda celog sajta, druga definiše izgled stranica koje izveštavaju o greškama, dok je treća zadužena za formatiranje izgleda navigacionog menija.

Podešavanje izgleda uz pomoć CSS standarda podleže određenim pravilima. Doslednim poštovanjem tih pravila se može obezbediti da stranice imaju željenu formu i da izgledaju isto, bez obzira na vrstu Web *browser*-a koji se koristi na klijentskom računaru.

3.10.2 Teme

Pravila formatiranja uz pomoć CSS-a su ograničena na fiksni skup atributa. Pomoću njih je moguće definisati razna podešavanja izgleda, ali ona ne mogu upravljati brojnim aspektima ASP.NET kontrola. Kontrola *CheckBoxList*, na primer, sadrži svojstva koja regulišu način organizacije stavki u redove i kolone. I ako je u pitanju svojstvo koje je vezano isključivo za izgled te kontrole, ono je izvan domašaja CSS stilova. Pored toga, uz formatiranje često treba definisati i način rada, što sigurno nije moguće uz pomoć CSS stilova.

Baš kao i stilovi, i teme omogućavaju definisanje raznih opcija formata koje se mogu primeniti na kontrole raspoređene na različitim Web stranicama. Za razliku od CSS stilova, Web *browser*-i ne podržavaju teme, a i nema potrebe za tim jer se one obrađuju u okviru ASP.NET sistema, na Web serveru, tokom kreiranja Web stranice.

Obzirom da je svaka stranica Web aplikacije „Kristal“ specifična, na svakoj od njih je definisana veličina polja za unos podataka, veličina kontrolnih dugmadi, raspored polja za čekiranje i slično. Upravo zbog toga, Web aplikacija „Kristal“ nema definisanu temu koja se primenjuje na aplikaciju u celini. Pored toga, istovremena upotreba tema i stilova može dovesti konflikta, ali ukoliko je to jedino rešenje za dobijanje željenog izgleda stranica, postoji opcija da se u okviru datoteke *web.config* definiše polje delovanja određene teme. To je posebno korisno ukoliko postoji želja da se definiše nekoliko različitih izgleda u okviru jedne Web aplikacije. Umesto standardizacije Web sajta, teme se mogu upotrebiti i za definisanje izgleda koji se može prilagođavati potrebama svakog pojedinačnog korisnika.

Sledeće verzije aplikacije „Kristal“ će posedovati korisničke opcije za definisanje fontova, boje pozadine, veličine fotografija i slično. Obzirom da ova aplikacija zahteva

brojne lične podatke, korisnički interfejs je uređen tako da korisnici stiču utisak da rade sa desktop aplikacijom. To stvara intimniji utisak, što navodi korisnike da iskrenije odgovaraju na pitanja. Pored toga, nakon prijavljivanja na sistem, podaci o korisniku se učitavaju na stranicu, što dodatno pojačava taj utisak jer korisnik stiče utisak da ima svoj „kutak“. Dodavanje opcija za podešavanje izgleda i veličine fontova, boje pozadine, ili nekih drugih karakteristika vezanih za izgled, predstavlja još jedan korak dalje ka kreiranju korisnički orijentisane aplikacije u pravom smislu.

3.10.3 *Master* stranice

Kvalitetni Web sajtovi ne deluju kao običan niz Web stranica, umesto toga oni daju iluziju aplikacije koja radi neprekidno. To se može uočiti prilikom posete nekom od kvalitetnih sajtova. Tokom pretrage stranica, u vidokrugu se stalno nalazi isti korisnički interfejs sa standardnim zaglavljem i grupom navigacionih kontrola.

Postavljanje korisničkog menija na svaku stranicu se može obaviti i tako što ćemo kopirati isti kod na svaku stranicu i tako definisati raspored kontrola na svakoj od stranica. Međutim, pomeraj od samo nekoliko piksela, može pokvariti kompletan utisak, jer će korisnici odmah uočiti da nisu sve stranice jednake. U situaciji kada se postigne da svi zajednički elementi stranica izgledaju isto, takav vizuelni dizajn je i dalje veoma lomljiv. Različiti Web čitači mogu drugačije tumačiti opcije formatiranja i definisati drugačiji izgled stranica. Pored toga, to uzrokuje još jednu teškoću, ukoliko poželimo da promenimo neko od svojstava neke kontrole, to moramo uraditi na svakoj od stranica gde se ona pojavljuje i na taj način obezbediti jedinstveni vizuelni dizajn.

Kreiranje takvog interfejsa je moguće i iz ASP.NET-a, vrlo jednostavno, uz pravi pristup. *Master* stranice ASP.NET-a, koje omogućavaju definisanje šablona stranice i njegovu upotrebu na čitavom Web sajtu, predstavljaju pravo rešenje. One su slične običnim ASP.NET stranicama. Baš kao i obične stranice, i *master* stranice su tekstualne datoteke u kojima se nalaze HTML i Web kontrole i kod. One, međutim, imaju drugačiju ekstenziju, *.master*, i ne mogu se videti direktno iz Web *browser*-a. Umesto toga, one se koriste u drugim stranicama, stranicama sa sadržajem. U osnovi, *master* stranica definiše strukturu stranice i zajedničke elemente. Stranice sa sadržajem prihvataju takvu strukturu i samo je popunjavaju određenim sadržajem.

Takav pristup omogućava da sve stranice koriste taj šablon, čime je njihova organizacija identična, uključujući isti naslov, podnožje stranice, navigacione kontrole, i slično. Svaka pojedinačna stranica, naravno, i dalje može ubacivati specifične sadržaje u takav šablon, poput polja za unos podataka, različitih izveštaja, slika, kontrolnih dugmadi i slično.Međutim, prava moć *master* stranica je u tome što one imaju zaseban kod i kome se mogu definisati reakcije na različite događaje. Obzirom da se ti događaji realizuju u životu stranica koje nasleđuju svojstva *master* stranice, oni se mogu iskoristiti za pokretanje brojnih metoda i rutina.

„Kristal“ koristi ovakvu stranicu pod nazivom *KristalGlavnaStranica.master*, na prvom mestu za obezbeđenje logike prijavljivanja i registracije korisnika. Za nju su vezane funkcije koje u zavisnosti od identifikacije korisnika,obezbeđuju razne korisničke podatke, značajne za rad same aplikacije i za unos i čuvanja podataka, kao i za formiranja izveštaja. Pored toga, pomoću *master* stranice je definisan položaj svih kontrola koje dodaju stranice sa sadržajem.

4 Upravljanje ljudskim resursima

Upravljanje ljudskim resursima se, pod ovakvim nazivom javlja u organizacionoj teoriji i praksi, početkom devedesetih godina dvadesetog veka. Menadžment ljudskih resursa je deo oblasti rukovođenja, a zasniva se na teorijskim i praktičnim znanjima iz oblasti psihologije rada. Pitanje na koje se često može naići je: da li je u pitanju nova grana organizacionih nauka, ili je samo u pitanju novi naziv za kadrovske poslove.

Mnogi humanistički orijentisani kritičari savremenih trendova i globalnih društvenih pojava ističu da postoji dublji smisao u samoj suštini ove grane rukovođenja, koji predstavlja jedno specifično, novo tretiranje čoveka kao nosioca rada. Činjenica je da se ljudi, profesionalci i stručnjaci iz različitih oblasti, nazivaju resursom, što možda na prvi pogled zvuči nehumano. Međutim, ne može se zamisliti poslovanje bilo kog preduzeća bez odgovarajućih resursa, a ljudi, sa svojim ukupnim potencijalom, takođe predstavljaju jedan od tih resursa. Danas, kada su visoko sofisticirane tehnologije dostigle stadijum na kom su dovoljno savršene i istovremeno dovoljno pristupačne, ne postavlja se pitanje kako povećati efikasnost mašina, već kako povećati efikasnost čoveka.

Brojna pitanja koja se danas postavljaju i rešavaju u okviru menadžmenta ljudskih resursa, postoje još od davnina. Postoje radovi koji su se bavili problemima tipičnih bolesti vezanih za određene profesije, rudare ili zanatlije, na primer, ili proučavanjem pokreta pri radu, ili prirodom umora, što sa savremenim radovima iz ove oblasti, čini jedan fond znanja koji omogućava vrtoglavi razvoj ove grane upravljanja.

Upravljanje ljudskim resursima koristi ta teorijska znanja, kao i brojne tehnike i metode u cilju efikasnog angažovanja ljudskih potencijala. Upravljanje ljudskim potencijalima primarno je usmereno ka realizaciji ciljeva preduzeća, ali je nemoguće ostvariti dobre rezultate bez razmatranja i ugradnje interesa pojedinaca u opšti sistem ciljeva preduzeća. Ostvarivanjem ciljeva preduzeća, zaposleni u tom preduzeću ostvaruju i svoje ciljeve. Pored novčane nadoknade, zaposleni se razvijaju i ostvaruju u okvirima profesije, zadovoljavaju brojne socijalne potrebe i ostvaruju niz koristi i privilegija za sebe i svoje najbliže. Činjenica da je zadovoljstvo jedan od najznačajnijih motivacionih faktora, jasna je odavno, ali ostvariti takav sistem rukovođenja u kom bi svaki pojedinačni cilj vodio ostvarivanju misije preduzeća nije ni malo jednostavan posao.

Ukoliko upravljanje ljudskim resursima, ili ljudskim potencijalima, definišemo kao sistem, možemo reći da ga u osnovi čini pet manjih sistema: analiza posla, regrutovanje kandidata, testiranje i selekcija zaposlenih, obuka i razvoj zaposlenih i upravljanje radnim rezultatima uz procenu učinka i zavisnu procenu srazmernih materijalnih i nematerijalnih nadoknada. Svaka od ovih celina razlikuje određene mehanizme i tehnike koje se u informacionom ili računarskom smislu mogu razmatrati kao procesi. Obzirom da svaki informacioni sistem, na nivou modela, podražava jedan sistem iz realnog sveta, sve ove podsisteme možemo, sa manje ili više uspeha realizovati i u informacionom i računarskom smislu.

Ovo je ujedno i polazna tačka u razvoju informacionog sistema za upravljanje ljudskim resursima, u ovom slučaju zasnovanog na Internet tehnologiji. Obzirom na kompleksnost procesa upravljanja ljudskim resursima, prva verzija Web aplikacije

„Kristal“ je samo početak u razvoju jednog kompleksnog informacionog sistema i namenjena je isključivo u edukativne svrhe. Buduće verzije će, po svojoj složenosti, dostići kompleksnost procesa koji postoje u okviru sistema upravljanja ljudskim potencijalima.

4.1 Procesi u Web aplikaciji za upravljanje ljudskim resursima „Kristal“

„Kristal“, među svojim posetiocima treba da stvori atmosferu, koja jasno ukazuje na pojedinca, njegove kvalitete i sistem vrednosti u kom se visoko kotira kreativnost i trud pojedinca da preuzme odgovornost za razvoj sopstvene karijere i ostvari sve moguće ciljeve i koristi u svom profesionalnom razvoju. Na prvom mestu adekvatnu novčanu nadoknadu od poslodavca, koja odgovara, kako doprinosu zaposlenog, tako i tržišnoj vrednosti, zatim adekvatan sistem nadogradnje znanja i razvoja veština uz mogućnost napredovanja u profesiji i karijeri uopšte.

Pored toga treba da ukaže na kratkoročnu prirodu poslovnih odnosa između pojedinaca i preduzeća u kojima su angažovani. Ti odnosi bi trebali da traju dok postoje obostrani interesi, što podrazumeva postojanje mogućnosti za profesionalno usavršavanje pojedinaca potrebno za dalji napredak u karijeri. Na takav sistem vrednosti se može naići u preduzećima koja ulažu u razvoj sposobnosti i kompetencija svojih zaposlenih i koja imaju razvijen sistem savetovanja svojih zaposlenih. Dakle, u okruženjima koja u pravom smislu razumeju da je zadovoljan radnik, produktivan radnik i koja u skladu sa tom činjenicom ulažu u sve procese vezane za upravljanje ljudskim resursima, poštujući pri tom moralna i etička načela, koja su u osnovi svih međuljudskih odnosa i koja su osnov za izgradnju kvalitetnih poslovnih odnosa.

4.1.1 Regrutovanje kandidata

Jedan od osnovnih zadataka upravljanja ljudskim potencijalima je pronalaženje pravih ljudi za obavljanje poslova u preduzeću. Pronalaženje dostupnih kandidata na tržištu rada, odnosno pronalaženje prave osobe za posao unutar preduzeća, je izazov sa kojim se suočava svaka radna organizacija. U većini preduzeća se u procesu zapošljavanja angažuje stručna osoba, koja raspolaže odgovarajućim znanjem i instrumentima za procenu kandidata, ili se angažuju specijalizovane agencije, koje čitav niz procesa preuzimaju na sebe, od osmišljavanja detaljnog oglasa za konkurs, preko detaljne procene svih kandidata, do davanja preporuka za zapošljavanje. Veća preduzeća, sa druge strane raspolažu sopstvenim odeljenjima za upravljanje ljudskim resursima, što je i najkvalitetnija, ali i najskuplja varijanta.

Regrutovanje kandidata, kao i većina procesa u okviru upravljanja ljudskim resursima, je kvalitativna disciplina celina koja u osnovi sadrži i jednu kvantitativnu kategoriju, a to je planiranje radne snage, odnosno planiranje popunjavanja budućih radnih mesta u preduzeću. Planiranje se zasniva na procenjivanju potreba za slobodna radna mesta ili za nova radna mesta i vezano je za odluku da li će se ta radna mesta popunjavati kandidatima koji već rade u preduzeću, ili kandidatima izvan preduzeća. U oba slučaja je potrebno razraditi više scenarija i napraviti više planova.

U slučaju popunjavanja radnih mesta iz internih izvora, licima koja su već zaposlena će možda trebati dodatna obuka, usavršavanje i pomoć kako bi se pripremili za nova radna mesta, pa su u tom slučaju potrebni i planovi obuke. U drugom slučaju

preduzeća se oslanjaju na različite vidove informisanja i na ograničenu ponudu sa tržišta rada. U tom poslu im može pomoći „Kristal“, jer on podržava proces posredovanja između dve vrste korisnika ove aplikacije, poslodavaca, koji mogu da oglase upražnjena radna mesta, i nezaposlenih, koji traže radna mesta, i koji uz pomoć ove aplikacije mogu da postave svoje radne biografije i ostale podatke, sa ciljem da ih poslodavci pozovu na razgovor u vezi sa zaposlenjem.

Poslodavci će biti u mogućnosti da ostave podatke o svojim preduzećima i o radnim mestima za koja traže radnike. Pored toga će moći da pretraže bazu podataka sa podacima o konkurentima za zaposlenje i eventualno nađu odgovarajuće kandidate koje mogu kontaktirati radi razgovora. Sa druge strane, kandidati mogu ostaviti svoje lične podatke, podatke o svom osnovnom i tehničkom obrazovanju, o svojoj radnoj biografiji, o profesionalnoj ostvarenosti i zadovoljstvu sobom i druge. Naravno i oni su u mogućnosti da pretraže bazu podataka u eventualno samostalno kontaktiraju poslodavce i informišu ih o svojoj zainteresovanosti za obavljanje poslova u preduzećima poslodavaca.

4.1.2 Obuka i usavršavanje zaposlenih

Obuka ili trening zaposlenih predstavlja planski napor rukovodstva da poboljša performanse zaposlenih na njihovom radnom mestu. Pod obukom se podrazumevaju promene u specifičnim znanjima, veštinama i stavovima ili ponašanju zaposlenih. U novije vreme se fokus obuke pomera sa tradicionalnog shvatanja, ka stvaranju uslova da zaposleni razumeju veze i razloge zbog čega se nešto radi, zašto, na određeni način, i da budu motivisani da inovativno razmišljaju i neprestano unapređuju kvalitet svog rada.

Najjednostavniji način za uvođenje novih radnika je usmeravanje, koje podrazumeva pružanje podataka potrebnih za obavljanje posla i upoznavanje sa pravilima preduzeća. Usmeravanje ima komponentu socijalizacije koja podrazumeva stalan proces ispoljavanja stavova, prikazivanje standarda, i ukazivanja na vrednosti i obrazace ponašanja koje preduzeće očekuje. Samo usmeravanje može biti formalno organizovano, ili može imati oblik kratkog neformalnog uvođenja u posao. U oba slučaja novi radnik dobija formalne podatke o radnom vremenu, očekivanom učinku, isplati ličnog dohotka, godišnjem odmoru, beneficijama, bezbednosnim merama, propisima i slično.

Uspešnim usmeravanjem se postiže nekoliko ciljeva: zaposleni stiču osećaj da su dobrodošli, stiču uvid u politiku kompanije, njena pravila i procedure, i dobijaju jasnu sliku kakav rad i ponašanje se očekuje od njih na poslu. „Kristal“ može pomoći u tom poslu jer otvara mogućnost da novi zaposleni, ili radnici koji već duže vreme rade u okviru preduzeća, izradom lične SWOT analize, uvide koji su to faktori kojima treba posvetiti posebnu pažnju. Na koje faktore se može uticati i na koji način se mogu ostvariti obostrane koristi, tretiranjem tih faktora , odnosno na koje faktore se ne može izvršiti uticaj i gde je najbolje predupreditи negativne posledice tih faktora.

Mnogo kompleksniji proces je obuka zaposlenih. Obuka se koristi da novi radnici, ili lica koja već rade u preduzeću, nauče nove veštine koje su im potrebne za obavljanje posla. Obučavanju se u praksi posvećuje veća pažnja, jer je ona sastavni deo politike preduzeća, njegovih strateških planova i jedan od ključnih segmenata kvaliteta preduzeća. U sklopu obuke, zaposleni ne dobijaju samo tehnička i tehnološka znanja, već osposobljavaju za timski rad, donošenje odluka, uspešnu komunikaciju, analizu procesa i njihovo poboljšavanje. Obuka ima ključnu ulogu u upravljanju efikasnošću zaposlenih što

utiče i na povećanje radnog učinka, sve kroz ocenjivanje i nagrađivanje zaposlenih, stvaranjem sistema koji zadovoljavajući interese radnika, zadovoljava potrebe i ciljeve preduzeća i krajnjih korisnika.

4.1.3 Određivanje cilja karijere

Web aplikacija „Kristal“ omogućava svojim korisnicima da upotrebe ponuđeni model za upravljanje karijerom. Za dobro upravljanje karijerom, važno je na dobar način definisati i postaviti ciljeve. Cilj mora biti jasan, merljiv, dostižan, vremenski određen i dovoljno izazovan. Model koji se uspešno može koristiti za definisanje misije karijere i za postavljanje ciljeva je poznat kao GROW model i implementiran je u ovu aplikaciju.

Naziv ovog modela, GROW je nastao od početnih slova engleskih reči, koje po svom značenju ukazuju na suštinu ovog modela: *Goal*, *Reality*, *Options* i *Will*. Ovaj model polazi od pretpostavke da sva pitanja, i akcije koje preduzmemo, treba posmatrati kroz prizmu ciljeva, koje smo sami sebi postavili. Takav pristup nam omogućava da sagledamo realnosti i odlučimo se između opcija koje nam stoje na raspolaganju, preduzmemo konkretne korake i utičemo da se naša karijera usmeri ka ostvarenju naše profesionalne misije.

Goal – Misija karijere je finalni cilj, ono čemu trebaju da budu podređeni svi ciljevi koje nastojimo da ostvarimo tokom našeg profesionalnog razvoja. Misija bi trebala da bude ogledalo nas samih, može biti u službi svih nas ili opšteg dobra, a može biti i na planu postizanja ličnog bogatstva. Suština je da se ovim pristupom kreira hijerarhija ličnih ciljeva na čijem vrhu će se nalaziti misija karijere.

Reality – Objektivno stanje, ili realna slika o nama samima. Ovo je možda i najteže pitanje na koje treba dati odgovor. Retki su ljudi koji mogu da na objektivan način sagledaju stvarne prilike u kojim se nalaze i stvarne kvalitete kojim raspolažu, ali nije nemoguće. Dobar motiv može biti i činjenica, da od kvalitetnog sagledavanja realnosti, zavise i dalje akcije po pitanju razvoja karijere. Pored toga, brojne informacije o nama samima možemo saznati od ljudi koji nas okružuju ili jednostavnim vraćanjem u prošlost i sagledavanjem naših prethodnih poslovnih angažovanja. Pored svega, „Kristal“ nudi mogućnost izrade lične SWOT analize koja, ukoliko se primeni na dosledan i iskren način, može pružiti potrebne podatke za sagledavanje trenutnog stanja.

Otions – Opcije, ili mogućnosti koje nam stoje na raspolaganju. Uvek postoji način da se stvari urade na dobar način i da se ostvare pozitivni rezultati. Od svakog od nas zavisi da li ćemo i u kojoj meri uspeti da uočimo mogućnosti koje nas okružuju, i da li ćemo uspeti da uvidimo koliko smo sposobni da odgovorimo na zahteve tih mogućnosti. Pomoću lične SWOT analize se može steći uvid u mogućnosti koje nam pruža okruženje i snage kojima raspolažemo kako bi te mogućnosti iskoristili. Svi moramo pronaći načine da unapredimo svoje sposobnosti, veštine, profesionalna znanja, da povećamo našu spremnost da sarađujemo i timski radimo. Pri tome uvek moramo imati ideju o tome koliko od nas samih ulažemo i koliko očekujemo da nam se vrati.

Will – Volja da se nešto uradi je ključ svakog uspeha. Naravno, volja da se ostvare profesionalni ciljevi je u osnovi upravljanja karijerom. Lična predstava o uspehu je većini ljudi najznačajniji motivacioni faktor. Upravo iz tog razloga svaki postavljeni cilj treba da bude merljiv i dovoljno precizno definisan. To omogućava da se svaki profesionalni uspeh

okarakteriše kao nešto opipljivo zbog čega smo se trudili i što smo na kraju i uspeli da realizujemo. Uspeh se u skladu sa tim meri kroz kvalitativnu i kroz kvantitativnu dimenziju. Cilj je u potpunosti ostvaren ako je realizovan u predviđenom vremenu i ako je učinjeno sve što je bilo moguće i opravdano, po pitanju njegovog ostvarenja.

4.1.4 Lična SWOT analiza

Jedna od opcija koju nudi Web aplikacija „Kristal“ je izrada lične SWOT analize. Ova analiza omogućava sistemski pristup sagledavanju unutrašnji i spoljnih faktora koji zajedno utiču na naš profesionalni razvoj. Uz pomoć ovog alata se može steći bolji uvid u snage, ili slabosti koje posedujemo, kao šanse i pretnje koje nam nameće ili pruža naše poslovno okruženje.

Svakom aspektu SWOT analize treba prići, uz razmišljanje o razlozima izrade ove analize i rezultatima koje nastojimo da ostvarimo. U skladu sa tim na prvom mestu treba razmisliti o unutrašnjim faktorima: snagama (*Strenghts*) i slabostima (*Weaknesses*), koji su podložni promenama i našem uticaju, zatim i o spoljnim faktorima, uticajima okruženja: šansama (*Opportunities*) i pretnjama (*Threats*), koje su izvan naše moći uticaja.

Strenghts – Naše snage, iskustvo, znanje, sposobnosti, veštine, su verovatno naš najznačajniji resurs. Svi posedujemo predstavu o tome da postoje poslovi u kojima smo posebno dobri, koje volimo da obavljamo i na čijem razvoju treba raditi u daljem razvoju karijere. Koje su to snage kojima raspolažemo? Radno iskustvo, kvalifikacije, obuka kroz koji smo prošli, obrazovanje i slično. Pored toga treba uzeti u obzir i komunikativnost, kreativnost, odnos sa kolegama i sa pretpostavljenima.

Pored toga treba razmotriti još jedan aspekt, one osobine ili kvaliteti koji su u jednom preduzeću uočeni kao prednost, u nekom dugom poslovnom okruženju mogu biti shvaćeni kao nedostatak. Aspekti privatnog života, kao što su stabilna porodična situacija, uspešna adaptacija prilikom premeštanja na drugo radno mesto, ili prelazak u drugo preduzeće, finansijska sigurnost i dobro zdravstveno stanje, takođe ima značajnu ulogu.

Weaknesses – Lične slabosti mogu imati poguban uticaj na razvoj karijere i na profesionalni život. Upravo zbog toga, potrebno je biti veoma iskren i u potpunosti objektivan, pri ocenjivanju ličnih slabosti. Karijera sa prekidima, finansijska nestabilnost, problemi sa nadređenim, nedovoljno obrazovanje, ili nedostatak iskustva, su slabosti koje mogu uticati negativno na naš profesionalni život. Međutim, ukoliko pokušamo i uspemo da ih sagledamo na objektivan način, već smo u prednosti, jer tada smo jasno definisali šta je to protiv čega se treba boriti i jednostavno možemo usmeriti naše napore ka otklanjanju tih slabosti. U identifikovanju ličnih slabosti nam najviše mogu pomoći ljudi iz našeg poslovnog okruženja, kolege ili pretpostavljeni. Pored toga, dobri rezultati se postižu ukoliko posmatramo sebe iz perspektive ljudi iz našeg okruženja.

Opportunities – Šta je to što bi smo želeli da radimo u narednih nekoliko godina? Da li postoji određena oblast kojoj bi smo želeli da se posvetimo i u koju bi trebalo ulagati? Da li razmišljamo o specijalizaciji u nekoj oblasti? Da li postoje izazovi u okvirima postojećeg poslovnog angažovanja? Kakve su ti šanse koja nam pruža naše poslovno okruženje?

Moguće je da nismo raspoloženi da napustimo funkciju na kojoj smo, ali ako višegodišnji plan karijere, ne ispunjava naša profesionalna očekivanja, možda je vreme da razmislimo i da krenemo dalje? Da li postoji hobi koji može postati naš životni poziv?

Koji je najbolji način da povežemo naše snage sa mogućnostima koje nam pruža poslovno okruženje?

Threats – Bilo koji faktor koji nas sprečava u ostvarivanju profesionalnih ciljeva, predstavlja opasnost i pretnju našem profesionalnom razvoju. Pretnju predstavlja loša finansijska situacija preduzeća u kom radimo, intenzivna konkurencija na radno mesto na kom želimo da radimo, smanjivanje broja zaposlenih usled uvođenja nove tehnologije i slično. Takođe, radno mesto koje ne pruža dovoljno izazova, može delovati kao veoma negativan faktor uticaja na razvoj karijere.

4.1.5 Razvojne potrebe

Web aplikacija „Kristal“ omogućava da se definišu razvojne potrebe i da se stvori dijalog između radnika i poslodavaca u smislu identifikovanja razvojnih potreba, predloga određenih akcija za ostvarivanje profesionalnog napretka, merenja ostvarenosti postavljenih ciljeva, definisanja potrebnih resursa za ostvarivanje ciljeva karijere, određivanje vremenskog perioda u kom bi trebalo ostvariti te ciljeve i evidentiranja ostvarenosti ciljeva.

Polazna tačka za određivanje razvojnih potreba su podaci o slabostima, navedeni u ličnoj SWOT analizi. U skladu sa tim, ciljevi su vezani za dostizanje vrednosti na ličnom planu kojim se te slabosti pretvaraju u snage, ili bar neutralizuju. Ti ciljevi se ostvaruju određenim aktivnostima, koje se mogu preduzeti zahvaljujući ličnim resursima i ličnoj inicijativi, ili uz pomoć rukovodstva i resursa preduzeća u kom smo zaposleni. Naravno, svaki cilj mora biti jasno definisan i jednostavno merljiv. Zato je potrebno ciljeve definisati tako da se jednostavno može zaključiti da li je cilj ostvaren, na prvom mestu, a zatim i da li je ostvaren u predviđenom vremenskom periodu.

Ovakav pristup, pored toga što omogućava stvaranje jasne ideje od daljim akcijama na planu ličnog profesionalnog razvoja, uklanja barijere koje postoje između zaposlenih i rukovodilaca, otvarajući mogućnost za razvoj otvorene komunikacije uz atmosfere iskrenosti i uzajamnog poverenja.

4.1.6 Upravljanje vremenom

Upravljanje vremenom je veština koju nije lako savladati. Vreme predstavlja važan resurs u organizovanju, određivanju prioriteta, ostvarivanju profesionalnih ciljeva, ali isto tako i u odmaranju, zabavi, druženju i slično. U osnovi potrebno je da razvijemo sposobnost da ostvarimo postavljene ciljeve u predviđenom vremenskom roku.

Niko od nas ne može da uradi sve. Zato je važno biti vešt u određivanju prioriteta. Pravilnim određivanjem prioriteta, možemo kvalitetnije organizovati ograničeno vreme i obezbediti da sve obaveze izvršimo na kvalitetan način. Svaka od obaveza ima svoje karakteristike i može se definisati kao važna ili manje važna, ili kao hitna ili manje hitna. U skladu sa tim, važnije obaveze, koje uz to moraju da se izvrše u određenom vremenskom

periodu imaju najviši prioritet. Slede obaveze koje su manje hitne, ali je njihovo izvršavanje važno i za njima obaveze koje su manje važne.

Nakon postavljanja prioriteta, potrebno je izvršiti kvalitetnu analizu vremena. Analiziranje vremena je u toliko važnije, ako se uzme u obzir da je vreme resurs na koji apsolutno ne možemo uticati. Web aplikacija „Kristal“ omogućava, da se uz pomoć ponuđene forme, napravi plan aktivnosti i vremenskog angažovanja prilikom izvršavanja važnih aktivnosti u periodu od nedelju dana. Jednostavna grafička forma ima za cilj da prikaže na jasan način vremensko angažovanje u toku jedne nedelje i da putem tabele shvatimo koliko sati nedeljno trošimo na obavljanje određenih aktivnosti.

5 Zaključak

Svedoci smo nezaustavljivog trenda razvoja informacionih sistema baziranih na Web tehnologijama. Savremeni trendovi, diktiraju razvoj savremenog poslovanja. Sve je veći broj preduzeća koja raspolažu virtuelnim kancelarijama i koja su zahvaljujući savremenim tehnologijama prisutna na svim krajevima zemaljske kugle. Decentralizacija u fizičkom smislu je kompenzovana centralizacijom u informacionom i računarskom smislu.

Evidentno je da su sva uspešna preduzeća odavno shvatila prednosti koje nose informacione tehnologije. Implementacijom takvih tehnologija u velikoj meri su smanjila troškove, na jednoj strani, istovremeno ostvarujući niz boljitaka sa druge strane. Za ovakva preduzeća se može reći da shvataju značaj informacija. Njihova ulaganja u informacione sisteme govore o tome. Njihovi zaposleni u svakom trenutku imaju pristup podacima koji su verifikovani od strane njihovih rukovodilaca, znači, podacima koji su tačni, koji su uvek dostupni i na osnovu kojih mogu da preduzmu konkretne korake prilikom obavljanja svojim poslova.

Sistem vrednosti u kom su informacije i ljudski potencijali, dva najznačajnija resursa preduzeća, neminovno vodi ka ostvarivanju zavidnih poslovnih rezultata. Upravljanje ljudskim resursima na način koji kroz ostvarivanje ciljeva zaposlenih vodi ka ostvarivanju ciljeva preduzeća je idealno rešenje, ali ne mora biti samo ideal.

Kvalitetno razvijen informacioni sistem može učiniti dostupnim sve potrebne podatke i može obezbediti da se na sistematski način skladište podaci relevantni za poslovanje i za upravljanje ljudskim resursima, dok kvalitetno upravljanje ljudskim resursima može stvoriti profesionalne međuljudske odnose u kojima postoji saradnja u pravom smislu reči. Takva kombinacija bi pored visokih performansi poslovanja, obezbedila i određeni nivo zadovoljstva zaposlenih koji bi bili motivisani da u potpunosti shvate svoje angažovanje u okviru preduzeća i u skladu sa tim se adekvatno prilagode svakoj novonastaloj promeni.

Preduzeća koja su svoje poslovanje, protok informacija i upravljanje ljudskim resursima organizovala na ovaj način, su već dva koraka ispred konkurentskih preduzeća. Ovakav pristup im omogućava da ostvare prednosti koje će ubrzo postati nedostižne za preduzeća čije poslovanje se još uvek zasniva na prevaziđenim principima. U odnosu na preduzeća u kojima zaposleni svoju nesigurnost kompenzuju pravljenjem monopola nad informacijama, ili ostvaruju poslovne prednosti neplanskom upotrebom resursa preduzeća.

Prilog

Kao što osoba koja se bavi proučavanjem nebeskih tela, astronom, posmatra zvezde i planete, tako i osoba koje se bavi proučavanjem čoveka, posmatra sve ljude. On je istraživač i posmatrač ljudskog ponašanja, a ne ljubopitljivi ispitivač. Za njega je način na koji neka starija osoba mahne svom prijatelju isto tako uzbudljiv kao i način na koji neka mlada devojka prekrsti noge. On je terenski posmatrač čovekovih akcija, a teren mu je sve što ga okružuje – radna mesta, autobuske stanice, tržnice, aerodromi, trgovi, parkovi. Na svakom mestu gde se ljudi „ponašaju“, posmatrač čoveka ima nešto da nauči o ljudima kao što je on, dakle i o samom sebi.

Svi smo mi donekle posmatrači čoveka. Mi ponekad zapazimo neki određeni stav ili gest drugog čoveka i pitamo se kako je do toga došlo, ali se retko potrudimo da to i saznamo. Na primer, kažemo: „Meni je neprijatno u njihovom društvu – ne znam zašto, ali tako je.“, ili „Zar ne smatrate da se ona sinoć čudno ponašala?“, ili „Ja sam u društvu njih dvoje potpuno opušten – to je zbog nečeg u njihovom držanju.“, i tu prestajemo dalje da razmišljamo o tome, ali ozbiljan posmatrač čovekovog ponašanja želi da sazna zašto se takva osećanja javljaju. On želi da sazna šta uzrokuje određena ponašanja. To podrazumeva pažljivo posmatranje ljudi na jedan poseban način.

U takvom pristupu nema ničeg, izuzetno stručnog. Sve što je tu potrebno jeste razumevanje izvesnog broja jednostavnih koncepata i obrazaca zastupljenih u ljudskom ponašanju. Svaki nam govori o jednoj posebnoj vrsti ponašanja, ili o jednom posebnom načinu na koji se to ponašanje razvija, na koji ono nastaje, ili se menja. Poznavanje tih koncepata omogućava nam da jasnije uočimo određene obrazce ponašanja kada se sa njima ponovo suočimo, a posmatrača osposobljava da prodre ispod površine onoga što se događa u međuljudskim odnosima.

Sve ljudske radnje, ako se posmatraju iz određenog ugla, postaju gestovi, a gestovi prenose poruke. Kao vrsta, mi možemo biti tehnološki superiorni i filozofski bistri, ali mi nismo izgubili našu životinjsku osobinu: fizičku aktivnost. Upravo je ta telesna aktivnost ono što prvenstveno zanima posmatrača čoveka. Čovek je često nesvesan svojih akcija i to je baš ono što čini da nam te akcije više otkrivaju. Čovek je tako pomno fokusiran na svoje reči da najčešće zaboravlja da pri tome, njegovi pokreti, stavovi i izrazi lica „pričaju“ posebnu priču.

Sebičnost i želja za dominacijom, ne dozvoljavaju prosečnom čoveku da ovlada prvom veštinom posmatranja ljudi. Svako je bar nekad poželeo da ovlada situacijom čitajući skrivene poruke i gestove svojih sagovornika. Posmatrač nebeskih tela, astronom, ne proučava zvezde u nameri da ih prisvoji. Na isti način, posmatrač čoveka ne zloupotrebljava svoju prednost u tome što ima jedan poseban uvid u ljudsko ponašanje. Istina, jedan stručan, objektivan posmatrač može da se svojim znanjem koristi u smislu menjanja neke dosadne društvene okolnosti u uzbudljivu „ekskurziju“ na samom terenu, ali njegov prvenstveni cilj je jeste da stekne dublje razumevanje ljudskih uzajamnih akcija i značajne uzročnosti jednog dela ponašanja ljudskih bića.

Akcije

Osnova ljudskog uspeha je zasnovana na evolucijom stečenoj mogućnosti apstraktnog razmišljanja. Postoje dokazi koji ukazuju na to da su neke vrste životinja sposobne da apstraktno misle, ali stvaranje veštačkih tvorevina podržano apstraktnim načinom razmišljanja samo su kod čoveka postale neograničene. To je suština ljudske priče o uspehu. Čovek, zahvaljujući svojoj moći povezivanja i složenog razmišljanja, sve više pridaje unutrašnji ili subjektivni karakter svom ponašanju preko složenih procesa apstraktnog mišljenja, preko jezika, filozofije i matematike. On je svojim slabašnim telom dramatično ispoljio svoje ponašanje, ostavljajući upečatljiv trag na površini zemljine kugle u vidu oruđa, mašina, umetničkih dela, vozila, građevina, sela i gradova.

Moglo bi se pretpostaviti da je akcija, prosta životinjska akcija, ispod čoveka, i da je ona preživela samo kao ostatak njegove drevne prošlosti, ali nije tako. Od samog početka do danas čovek je ostao stvorenje akcije, biće koje se kreće, koje gestikuliše, zauzima stavove, koje je ekspresivno. On je danas isto toliko daleko od bestelesnog bića naučne fantastike, koliko je daleko od svoje praistorijske lovačke prošlosti. Filozofija i graditeljstvo nisu zamenili animalnu aktivnost, već su joj po nešto i dodali. Činjenica da smo razvili koncept sreće i da imamo reči kojima ga izražavamo, ne sprečava nas da izvodimo akcioni obrazac rastezanja naših usana u osmeh. Činjenica da poznajemo koncepte brodova ne sprečava nas da plivamo.

Ljudsko ponašanje nije nešto što teče nezavisno i kontinuirano. Ono je podeljeno na duge serije posebnih zbivanja. Svako zbivanje ima svoja posebna pravila i ritmove. Svako od tih zbivanja je, opet, podeljeno na mnogobrojne zasebne činove. U samoj osnovi, ti činovi se nižu u formi stav-pokret-stav-pokret.

Većinu telesnih stavova koje zauzimamo i pokreta koje činimo, zauzimali smo i činili hiljadama puta ranije. Mnoge od njih činimo nesvesno, spontano i bez analiziranja. U mnogim slučajevima oni su nam toliko bliski, usvojeni, toliko naši, da čak i ne znamo kada ih vršimo. Jednostavan primer govori o tome: kada čovek spoji svoje šake i uplete prste, jedan palac leži na drugom. Kod svake osobe je jedan palac dominantan pri toj radnji, i svaki put kada se leva i desna šaka spoje na taj način, isti palac je odozgo. Retki su ljudi koji mogu pogoditi koji je palac dominantan kod njih bez spajanja šaka u takav položaj. Svaka osoba je kroz čitav jedan niz godina razvijala i fiksirala jedan obrazac upitanja prstiju, a da nije bila svesna toga. Ako pokušamo da preinačimo položaj palčeva i dominantan palac stavimo ispod onog drugog, osetićemo se čudno i nelagodno.

Ovaj jednostavan, istovremeno veoma upečatljiv primer, govori u prilog tome da gotovo svaka telesna radnja odraslih ima jedan karakterističan utvrđeni obrazac. Takvi „fiksirani“ akcioni obrasci čine osnovne jedinice ponašanja na koje se terenski posmatrač može osloniti i pozvati u svom radu, posmatranjem njihovog oblika, konteksta u kom se događaju i poruka koje prenose.

Tu se takođe postavlja i jedan niz pitanja. Na prvom mestu koji su od tih obrazaca stečeni. Da li, su oni urođeni? Da li, im je bilo potrebno neko stečeno iskustvo? Da li, su otkriveni ličnim pokušajima i greškama? Da li, su apsorbovani dok se čovek nesvesno takmičio sa svojim suparnicima? Ili su možda stečeni svesnim vežbanjem, ili naučeni

promišljenim naporom zasnovanim na specifičnoj analitičkoj opservaciji ili aktivnim učenjem?

Urođene radnje

Čovekov najveći genetski dar je njegova sposobnost da uči od sredine koja ga okružuje. Neki dokazuju da čovek, zahvaljujući ovoj urođenoj sposobnosti, nema potrebe za bilo kojom drugom. Oni koji drugačije misle, tvrde da je čovekovo ponašanje, naprotiv, bogato urođenim obrascima i da se njegovo ponašanje može potpuno razumeti samo ako se ova činjenica uvažava.

Kao podrška shvatanju da ljudski mozak većinu stvari uči, a tek mali deo nasleđuje, iznosi se zapažanje da različita društva širom sveta u velikoj meri pokazuju različite obrazce ponašanja. Kako svi pripadamo istoj vrsti, to ukazuje da ljudi pre nauče da se ponašaju nego što se vladaju u skladu sa nekim utvrđenim zbirom genetskih podataka.

Nasuprot ovome iznose se zapažanja da kulture ne razlikuju u tolikoj meri međusobno. Ukoliko ljudi traže razlike, naći će ih, ali ukoliko potraže sličnosti, i njih će naći u velikom broju. Na žalost, prirodna težnja čoveka je da uočava razlike, a da iz vida gubi sličnosti.

To su dva suprotna gledišta. Niko se ne suprotstavlja činjenici da mi u toku života zaista dosta toga naučimo, ali niko, takođe, ne osporava ni činjenicu da određen deo podataka nosi i naš genetski materijal.

Kako jedna urođena radnja deluje? U osnovi je predstava da je naš mozak programiran, da pojedine reakcije vezuje za određene impulse iz okoline. Impuls „okida“ reakciju bez bilo kakvog prethodnog iskustva. Takve radnje su unapred „isplanirane“ i takav mehanizam uspešno deluje kada se prvi put u životu suočimo sa nekim podstrekom.

Klasičan primer je odojče. Ono instinktivno traži hranu i odlično se snalazi u svojim prvim pohodima. Mnoge infantilne reakcije, zasnovane na instinktu, odvijaju se bez greške, očigledno zbog toga što su od presudnog značaja za opstanak. Tu nema vremena za učenje. Međutim, postoje radnje koje se javljaju kasnije, u periodu kada smo već imali dosta vremena za učenje. Recimo osmehivanje ili mrštenje. Da li, malo dete i u tome podražava svoju majku, ili je i to urođeno? Stvaranje eksperimentalno kontrolisanih uslova, zarad traženja odgovora na ovo pitanje, bi predstavljalo pravu monstruoznost, mada istorija pamti i takve primere, ali ako pogledamo decu koja su rođena slepa i gluva, vidimo da se ona osmehuju i mršte u odgovarajućim trenucima njihovih svakodnevnih života. Ona i plaču, iako nisu u stanju da čuju sebe.

Te radnje su očigledno urođene, ali šta reći o obrascima ponašanja odraslih. Ovde čak ni oni koji su rođeni bez čula sluha ne nose informaciju o urođenim radnjama, jer su do tog perioda naučili da komuniciraju pomoću jezika gluvonemih, i kao takvi su suviše rafinirani, i suviše obavešteni, ili slepi ljudi koji su do tog perioda naučili da pomoću svojih prstiju osećaju izraze na licima. Takvi metodi definitivno ne pružaju informacije i dokaze koji idu u prilog teze o urođenim radnjama.

Jedini način kojim se može podržati shvatanje da su neke radnje odraslih urođene, je da dokažemo da se one događaju u svakom ljudskom društvu, bez obzira na različite

uticaje, ili pritiske pojedinih kultura. Da li, svi ljudi udaraju nogom o pod kada su ljuti, ili pokazuju zube kada su razjareni, ili naizmenično podižu i spuštaju obrve kada se pozdravljaju sa prijateljima? Postojali su, a možda još među nama ljudi koji su obišli zemaljsku kuglu u potrazi za odgovorima i koji su uspeli da potvrde da čak i pripadnici plemena koji nikada nisu videli civilizovanog čoveka, vrše mnoge male radnje na način na koji ih vrše i svi drugi ljudi.

Ukoliko pripadnici dalekih urođeničkih plemena poseduju prepoznatljive i svima svojstvene obrazce ponašanja, da li možemo biti sigurni da to znači da su te reakcije „ugrađene“ u nas i pre našeg rođenja. Sigurno i dalje, ne. Čak ni u prošlosti nije postojao pravi argument koji bi ukazivao da postoje radnje koje nismo mogli „svi“ da naučimo. Danas naročito, razvoj brojnih novih komunikacionih tehnologija, uticao je da svi budemo, u informacionom smislu, jedan organizam.

Čak i kada bi smo na neki način uspeli da dokažemo da neka od radnji za koje se veruje da su urođene, nije zastupljena u okvirima neke kulture, takva informacija, ne bi imala veliki značaj. Moguće je, pa i kada su u pitanju radnje koje su istinski urođene, da pod uticajem određene kulture, takva ponašanja budu potisnuta, a to bi novonastalim ponašanjima davalo lažan vid nečega što ima lokalni karakter.

Sve do trenutka kada budemo u mogućnosti da gene čovekovog ponašanja čitamo kao knjigu, gotovo da i nema smisla zadržavati se dugo na ovom pitanju, sve dok genetika ne učini krupne korake u svom razvoju, mi možemo „pouzdano“ govoriti o urođenim radnjama kod čoveka samo u slučajevima novorođenčadi, ili u slučajevima dece sa posebnim potrebama. Ovakva restrikcija u skoro svemu ograničava tu kategoriju aktivnosti, ali je na današnjem nivou znanja neminovna.

Otkrivene radnje

Ukoliko postoji nedoumica u tome da li je neka naša radnja urođena ili ne, teško da može biti sumnje u to da smo mi i naša anatomska struktura i njeni oblici genetski nasleđeni. Mi ne možemo da „naučimo“ neki deo tela, kao što možemo da naučimo pozdrav, ili hodanje. Profesionalni sportista i dete u razvoju imaju isti sistem mišića. Oni su kod profesionalnog sportiste bolje razvijeni, ali su to ipak isti mišići.

Sredina u kojoj čovek živi ne može da izmeni njegov osnovni anatomski sastav tokom njegovog života, prirodnim putem. Iz toga sledi da ćemo svi mi, ako svi nasleđujemo u osnovi slične ekstremitete, biti sposobni da gestikulišemo gotovo isto onako kao što se radi u svakoj kulturi.

Drugim rečima, kada posmatramo nekog urođenika na Novoj Gvineji kako skršta ruke na isti način na koji to radi švajcarski bankar, ili tibetanski zemljoradnik, mi možda ne gledamo vršenje jedne istinski urođene radnje, već pre *otkrivene radnje*. Sva tri čoveka su nasledila po dve ruke istog oblika. U nekoj fazi svojih života, kroz proces samostalnih pokušavanja i grešaka, svako od njih je otkrio da može da skrsti ruke na grudima. Tako su *ruke* ono što je nasleđeno, a ne radnje. Međutim, imajući te i takve ruke, gotovo je neizbežno da, i bez podražavanja svoje okoline, dođu do radnje skrštanja ruku. Sve *otkrivene radnje* se mogu okarakterisati kao radnje zasnovane na „genetskoj sugestiji“, preko anatomije, više nego na nekoj genetskoj instrukciji.

Do vršenja otkrivenih radnji dolazimo nesvesno, dok spoznajemo naše telo kroz dugi proces samospoznaje. Mi čak nismo svesni ni toga da su te radnje bile dodatak našem repertoaru u detinjstvu, a u većini slučajeva nemamo tačnu predstavu o tome kako ih obavljamo, kako prekrštamo ruke, koju preko koje, ili kako gestikuliramo rukama dok govorimo.

Apsorbovane radnje

Najkraće, *apsorbovane radnje* su one kojima nesvesno podražavamo druge ljude. Ovde, kao i kod *otkrivenih radnji*, ne znamo kako smo ih i kada stekli, ali za razliku od *otkrivenih radnji* ovde se zapažaju varijacije od grupe do grupe, od kulture do kulture, od nacije do nacije. Mi kao vrsta imamo veliku sposobnost podražavanja i nemoguće je da i jedan pojedinac koji je odrastao i živi u zajednici ne bude razvijen i uobličen uticajima sredine i njenim tipičnim obrascima ponašanja. Način na koji hodamo i stojimo, na koji se smejemo, ili pravimo grimase, sve je to pod tim uticajem.

Mnoge radnje vršimo prvi put samo zato što smo ih zapazili u ponašanju ljudi iz našeg okruženja. Takve radnje je veoma teško razabrati u našem sopstvenom ponašanju, jer je proces apsorpcije veoma suptilan, mi smo retko svesni činjenice da se dogodila radnja koja je uticala na promenu u našem sopstvenom ponašanju, ali je takve radnje lako otkriti u određenim grupama u našem vlastitom okruženju.

Kod ljudi se jasno vidi da je tu važan status pojedinih uglednih članova zajednice. Što je njegov ili njen status u grupi viši, to su ostali skloniji da nju ili njega podražavaju. Mi, u našem sopstvenom društvu, najviše apsorbujemo od onih kojima se divimo. Ovakav princip je naviše delotvoran u bliskim susretima, mada i u našim vezama sa masovnim medijima, mi apsorbujemo radnje dalekih slavnih ličnosti, istaknutih idola i popularnih ličnosti.

Oponašanje istaknutih i voljenih stanovnika naše planete je već predmet globalne kulture. Rasprostranjenost ove pojave, u drugoj polovini prošlog veka, je uzrokovala nastanak duboke promene u ljudskom pogledu na društveni život uopšte. Brojni predstavnici mlađe populacije na visokom mestu u svom sistemu vrednosti, su negovali opuštenost i neposrednost. U njima se razvila duhovna opuštenost i ležeran, opušten način mišljenja. Takvo stanje je u startu okarakterisano kao *patos*[1], ali je vreme i mediji su učinili da takav stav postane standard i uobičajeno ponašanje. Njihovi telesni stavovi i akcije što prate ovo ponašanje, starijima su ličili na aljkavost, ali za objektivnog posmatrača sve to čini jedan način određenog držanja i ponašanja. Stil, a nikako nedostatak stila.

Patos – (gr. Πάσχω patim, πάγοζ patnja, bolest) patnja, bolest; bolesno stanje duše; strast, strasna uzbuđenost; strasna uzvišenost i dirljivost u izrazu i govoru, velika osećajnost pesnika ili govornika; čuvstvenost, toplina, žar, polet, zanos; strastan i zanosan govor u drami, naročito u tragediji; svaki viši način izražavanja; takođe i uobraženost.

Među predstavnicima mlade populacije danas, veoma važnu ulogu u životu i međusobnoj komunikaciji, čini emotivni život. Otvoreno pokazivanje emocija i otvoreni razgovori o tome, deluju kao logičan odgovor na pritisak koji savremeno društvo i nove tehnologije, vrše na ljude, prvenstveno mlade. Brojni sociolozi su u svojim radovima istakli da je generaciju „X“, nasledila generacija „Y“, a nju generacija koja i sama sebe naziva „Emo“.

Svakom prilikom stavovi i gestovi su se ljudima na razne načine menjali i procesom brže apsorpcije novi stilovi ponašanja su se širili kao divlja vatra, čim bi na jednom mestu počinjala da se gasi, na drugom bi buknula. Već postoje znaci da se moderan „Emo“ stil menja, ali niko ne može predvideti koje će stilove ponašanja omladina dvadeset prvog veka pomno podražavati i usvajati.

Uvežbane radnje

Uvežbane radnje se svesno stiču, ili učenjem od drugih, ili samo analitičkom opservacijom i praksom. Na jednom kraju skale se nalaze teška fizička dostignuća kao što su hodanje na rukama, ili cirkuske akrobacije. Vršenjem tih radnji mogu da ovladaju ljudi koji su posvećeni vežbanju.

Na drugom kraju skale nalaze se jednostavne radnje kao što su namigivanje, ili rukovanje. U nekim slučajevima one bi mogle da se svrstaju u kategoriju *apsorbovanih radnji*, ali kada pažljivo posmatramo decu, ubrzo nam postaje jasno da dete prvo mora samo sebe da nauči, svesno i namerno, mnogim onim radnjama koje mi odrasli smatramo za date. Rukovanje, toliko prirodno u očima odraslih, maloj deci deluje kao nešto neprijatno i nezgodno i na samom početku, njih moramo da nagovaramo da nekome pruže ruku i da im onda pokazujemo kako se to radi. Posmatranje deteta kada prvi put u životu pokušava da ovlada lukavim namigivanjem, omogućava nam da se ponovo živo setimo koliko neke prividno proste radnje mogu da budu teške.

Mešovite radnje

Dakle, u osnovi postoje četiri načina na koje stičemo sposobnosti za realizovanje različitih radnji: genetskim nasleđem, samostalnim otkrivanjem, apsorpcijom od društva i svesnim obučavanjem, ali kada pravimo razliku između ta četiri odgovarajuća tipa akcija, ne stiče se utisak da su one jasno odvojene jedne od drugih. Za svoj punoletni oblik mnoge radnje duguju uticajima ne samo jedne od tih kategorija.

Urođene radnje često bivaju drastično modifikovane pod društvenim pritiscima. *Otkrivene radnje* često podležu uticajima na isti način, bivajući snažno modifikovane nesvesnim podražavanjem društvenih običaja. To se događa gotovo neprimetno, pa čak i ako se primeti, neće biti analizirano ili shvaćeno. Kada se član jedne grupe pomeša sa drugom, on će se osećati neprijatno, iako ne shvata zašto. Razlog će biti u tome što se drugi kreću, druže i gestikuliraju na neki njemu tuđ način. Razlike mogu biti veoma male, ali će se one otkriti i registrovati. Takođe se često može čuti da član jedne grupe govori o članovima druge grupe kao o lenjim, feminiziranim ili sirovim ljudima, a kada treba da argumentuje svoj stav, najčešći odgovor je: „Pa samo treba da ih pogledate.“ Skoro je sigurno da on nesvesno pogrešno tumači njihove radnje.

Gestovi

Gest je svaka radnja kojom se neki vidni znak daje posmatraču. Da bi neki čin mogao da se posmatra kao gest, njega moraju da vide drugi ljudi i on njima mora da prenosi neku informaciju. U osnovi, svi gestovi se mogu okarakterisati kao *primarni*, ili kao *slučajni*. Mahanje rukom je *primarni gest*, jer on nema drugu egzistenciju, ili funkciju. On je od početka do kraja jedan delić komunikacije. Nasuprot tome, kijanje je sekundaran, ili *slučajan gest*. Njegova prvenstvena funkcija je fiziološka, međutim u svojoj sekundarnoj funkciji, kijanje šalje jasnu poruku okolini da treba da obrati pažnju i zaštiti se od eventualne kijavice.

Mnogi ljudi su skloni da svoju upotrebu termina „gest“ ograniče na njegov primarni oblik, ali se tako gubi iz vida jedna značajna dimenzija. Ono što je važno u gestikuliranju nije u tome kakve znake mi mislimo da dajemo, već kakvi se znaci primaju. Naši slučajni gestovi su, na neki način, ilustrativniji od onih drugih, ako ni po čemu drugom, onda po samoj činjenici da mi o njima ne razmišljamo kao o gestovima, pa ih i ne cenzurišemo i njima ne manipulišemo tako strogo i dosledno kao sa svesnim.

Konvencionalan način za pravljenje razlike između slučajnih i primarnih gestova je u postavljanju pitanja: „Da li bih ja to činio i kad sam potpuno sam?“ Ako je odgovor *ne,* onda je u pitanju *primarni gest*. Mi ne mašemo rukama, ne namigujemo, niti pokazujemo prstom kada smo sami; to jest, ne, ukoliko nismo došli u ono neobično stanje živahnog pričanja sa samim sobom.

Slučajni gestovi

Iako u osnovi sve radnje koje radimo, radimo za sopstveno dobro, mi nismo uvek bez nečije pratnje dok to radimo. Ljudi iz našeg okruženja veliku količinu informacija o nama dobijaju posmatranjem našeg ponašanja. Iz našeg ponašanja se vidi kakve smo mi ličnosti i u kakvom smo raspoloženju pri tome.

Mnogi naši *slučajni gestovi* pružaju neku vrstu informacije, koje najčešće nismo svesni, ni mi, *ni oni oko nas*. Poput tajnog sistema komunikacije ispod same površine naših susreta u društvu. Vršimo neku radnju i ona je zapažena. Njeno značenje je protumačeno, ali ne svesno. Ponekad radnja ove vrste postaje za određenu situaciju toliko karakteristična da je mi vremenom identifikujemo, i na taj način ukazujemo na pojedine činjenice, ali ovaj tip komunikacije češće funkcioniše ispod granice svesnog.

Tamo gde su te veze jasnije, mi svakako možemo da manipulišemo situacijom i da naše slučajne gestove upotrebimo na neki domišljat način. Ukoliko student koji sluša predavanje nije umoran, ali želi da uvredi predavača, on može namerno da zauzme mlitavo držanje čoveka kome je dosadno, znajući da će njegova poruka stići na pravo mesto. To je *stilizovan slučajni gest* – mehanička radnja koja se veštački vrši kao čist signal. Mnoge opšte „ljubaznosti“ takođe spadaju u tu kategoriju. Vladanje našim *slučajnim gestovima* na ovaj način je jedan od procesa kroz koji svako dete, mora da prođe dok raste i uči da se prilagođava pravilima ponašanja društva u kom živi.

Izrazni gestovi

Primarni gestovi se dele na šest glavnih kategorija. Pet kategorija su karakteristične samo za čoveka, i one su jedinstvene jer zavise od njegovog kompleksnog, visoko razvijenog mozga. Izuzetak je kategorija koju možemo nazvati *izraznim gestovima*. Oni su zajednički svim ljudima, ali se mogu primetiti i kod nekih vrsta životinja. Oni uključuju važne znake izraza lica, koji su od najvećeg značaja u svakodnevnim ljudskim odnosima.

Ljudi su facijalno ekspresivni, a i među ljudima postoje razlike po pitanju razređenosti mišića lica i samim tim, mogućnošću izražavanja na ovakav načim. Visoka razređenost tih mišića omogućava davanje čitavog niza različitih facijalnih znakova sa suptilnim varijantama i sa sigurnošću se može reći da glavni deo neverbalnog signaliziranja vrši čovekovo lice.

Ljudske ruke su takođe važne, one pomoću svojih manuelnih gestikulacija, daju znake o mnogim malim promenama raspoloženja pomeranjem, pokretima i menjanjem svog položaja, naročito tokom konverzacionih susreta.

Reč „gestikulacija“ bi trebalo nezavisno definisati od „gesta“, kao jednu manuelnu radnju koju nesvesno vršimo tokom društvenih interakcija, kada kao gestikulator naglašavamo neku verbalnu poentu koju iznosimo.

Ovi prirodni gestovi su obično spontani i umnogome podrazumevani, i mi se nakon nekog razgovora sećamo značajnih delova i naravno same teme, ali se i uz veliki trud ne možemo setiti pokreta obrva, ili zamišljenih oblika koje su formirali prsti sagovornika. Mi smo sve to videli i naši mozgovi su registrovali ono što smo videli. Jednostavno nismo imali potrebe da analiziramo radnje našeg sagovornika, kao što reči koje smo čuli nismo morali da izgovaramo da bi smo ih razumeli. U tom pogledu, ovi gestovi su slični slučajnim gestovima, ali se razlikuju, jer ovde ne postoji mehanička funkcija, već samo signalizacija. To je svet osmeha i podrugljivih grimasa, sleganja ramenima i durenja, smeha i trzanja, crvenila i bledila, mahanja i klimanja, mrštenja i gunđanja. To su gestovi koje prave gotovo svi ljudi, gotovo svugde u svetu. Oni mogu da se razlikuju, od mesta do mesta, po pojedinostima i kontekstu, ali u osnovi, to su radnje koje su nam svima zajedničke.

Mimički gestovi

Mimički gestovi su oni pokreti kojima izvođač pokušava da što vernije podražava neku osobu, predmet, ili radnju. Oni nisu predmet našeg genetskog nasleđa, već su u potpunosti vezani za našu ljudsku sferu. Suštinski kvalitet jednog *mimičkog gesta* je u tome što pomoću njega pokušavamo da prikažemo stvar koju nastojimo da opišemo. Tu se ne primenjuju stilizovane konvencije. Uspešan mimički gest je zato razumljiv i nekome pred čijim očima nikad ranije nije bio izveden. Predhodno znanje nije potrebno i ne mora da postoji utvrđena tradicija u pogledu načina na koji se određeni subjekat, ili objekat prikazuje. Postoje četiri vrste *mimičkih gestova*:

Prvo, tu je *društvena mimikrija* ili „stavljanje prikladne maske na lice“. To svi radimo. Svi se osmehujemo u veselom društvu i ako na u stvari ništa ne izaziva smeh, a na nekoj sahrani izgledamo i tužnije nego što zapravo jesmo, prosto zato što se to od nas

očekuje. Lažemo pomoću simuliranih gestova da bi smo zadovoljili očekivanja našeg okruženja. To ne treba mešati sa onim što psiholozi nazivaju „igranje uloga“. Kada se upuštamo u društvenu mimikriju mi obmanjujemo samo druge, a kada igramo neku ulogu mi obmanjujemo i sebe.

Drugo, tu je teatralna mimikrija, svet glumaca i glumaca koji simuliraju sve, za našu zabavu. U suštini, ona obuhvata dve posebne tehnike. Jedna je proračunato nastojanje da se na specifičan način podražavaju zapažene radnje. Druga je tehnika fokusiranja na zamišljeno raspoloženje karaktera koji treba opisati, nastojanje da se uživi u to duševno stanje i osloni na njega, kako bi se, nesvesno, ostvario potreban stil telesnih radnji.

U praksi, svi glumci koriste kombinaciju ove dve tehnike, mada, kada govore o svojoj umetnosti, znaju da naglase samo jedan od ta dva metoda. U prošlosti je gluma bila visoko stilizovana, ali danas, sa izuzetkom pantomime, opere i farse, postiže se visok stepen realizma. Kada gledamo neki komad, mi moramo da verujemo da se to stvarno događa. Drugim rečima, *teatralna mimikrija* postala je najzad isto tako realistična kao što je svakodnevna *društvena mimikrija*. U tom pogledu ova prva dva tipa mimičkog predstavljanja upadljivo se razlikuju od trećeg, koji se može okarakterisati kao *parcijalna mimikrija*.

U *parcijalnoj mimikriji* izvođač pokušava da podražava nešto što on nije i nikada ne može da bude, kao što su, na primer, meteorološke pojave. Tu su obično ruke uposlene, ali ovde ruke sasvim realistički pristupaju predmetu koji mogu da opišu. Ukoliko se opisuje kiša, one prikazuju njeno padanje i prskanje na što je moguće plastičniji način.

Četvrta vrsta mimičkog gesta najbolje bi se mogla nazvati: *vakumska mimikrija*, jer se tu radnja vrši u odsustvu predmeta na koji se odnosi. Dobar primer je, možda, traženje alata u bučnoj proizvodnoj hali, gde radnik podiže ruku i pokazuje karakteristične pokrete za upotrebu alata koji traži od svojih kolega.

Važna crta *parcijalne mimikrije* i *vakumske mimikrije* je u tome što one, kao i *društvena* i *teatralna mimikrija*, teže ka realnosti. Ovo znači da se one mogu uzeti kao zajedničke svim nacijama, da se mogu shvatiti internacionalno. U tom pogledu one čine veliki kontrast sledećim dvema vrstama gestova, koji pokazuju izrazita kulturna ograničenja.

Šematski gestovi

Šematski gestovi su skraćene, ili sažete verzije *mimičkih gestova*. Oni nastoje da nešto opišu izdvajanjem samo jedne od karakterističnih crta onoga što se opisuje. Tu nema više nikakve težnje ka realizmu.

Šematski gestovi se obično javljaju kao neka vrsta stenografije u polju gestikulacije zbog potrebe da se neka imitacija obavi brzo i u mnogim prilikama. Kada jedan gest izdvojimo, a ostale elemente skratimo, ili izostavimo, taj gest će i dalje biti lako shvatljiv, kad se vidi prvi put, ali stilizacija može da ode čak dotle da postane besmislena za one koji nisu u „toku stvari“. Onda šematski gest postaje lokalna tradicija sa ograničenim geografskim opsegom. Ako su izvorni gestovi bili kompleksni, pa su uključivali više karakterističnih crta, različiti lokaliteti mogu da izdvoje različite ključne crte za svoje sažete verzije, a kada se te različite forme jednom učvrste u svakom regionu, onda će ljudi

koji se njima služe postajati sve manje sposobni za tumačenje nekih drugih, koji u osnovi prenose isto značenje.

Simbolički gestovi

Simbolički gest ukazuje na jedan apstraktan kvalitet koji nema prost ekvivalent u svetu predmeta i pokreta. Oni su za jedan stepen dalje od očiglednosti odigravanog *mimičkog gesta*.

Kada bi smo, na primer, želeli da bez reči ukažemo na nečiju tupoglavost, imali bi smo nekoliko mogućnosti. Jedna od njih je kuckanje kažiprstom po slepoočnici, ali tu nedostaje tačnost. Potpuno istu stvar bi učinili i kada bi smo hteli da ukažemo na nečiju pamet, ili mudrost. Alternativno, mogli bi smo da vrtimo prst uz samu slepoočnicu, kao da nešto bušimo, dajući tako znak da se unutra nalazi praznina, ili neka jednostavna aktivnost, ili možda nešto treće.

Situacija se dodatno komplikuje činjenicom da neki signali koji označavaju tupoglavost znače nešto sasvim drugo u drugim zemljama. Na primer, u Saudiskoj Arabiji tupoglavost se signalizira dodirivanjem donjeg očnog kapka vrhom kažiprsta, a ta ista radnja, u raznim drugim zemljama, može da znači nevericu, odobravanje, slaganje, sumnju, tajnost, lukavost, opasnost, ili krivicu. Razlog za ovaj prividni haos značenja je sasvim jednostavan. Kada pokazujemo na oko, organ čula vida, samo naglašavamo simboličko značenje upravo toga, da vidimo, odnosno ne vidimo.

Jasno je da *tumačenje simboličnih* gestova nema naučnu osnovu, jer pored toga što postoji veoma veliki broj faktora koji utiču na njihovo formiranje, bilo kakva detaljna analiza bi dala rezultate koji bi bili kratkoročni i u velikoj meri netačni, a i sam terenski posmatrač čoveka bi se češće nalazio u području jalove imaginacije, nego u sferi istorijskih, ili terenskih podataka.

Tehnički gestovi

Tehničke gestove definiše stručna manjina samo za upotrebu u granicama njene specifične aktivnosti. Oni su besmisleni za skoro svakog ko se nalazi van te specijalnosti. Tim gestovima se operiše na veoma uskom polju delovanja i može se reći da su u osnovi karakteristični samo za kulturu vizuelnog komuniciranje u okviru nekih poslova.

Oni su obično nastali u prvim danima razvoja neke profesije i, mada su i van granica konkretnih profesija rasprostranjeni, svoj pravi značaj i značenje imaju samo u okviru uskog kruga ljudi koji komuniciraju uz pomoć tih znakova.

Sve *tehničke gestove* nalazimo gde god neka aktivnost zabranjuje, ili sprečava verbalni kontakt. Ronioci, na primer, ne mogu da razgovaraju međusobno, pa su im zato potrebni prosti znaci sporazumevanja u potencijalno opasnim situacijama. Zapravo, potrebni su im posebni gestovi za opasnost, hladnoću, grč, prepreku, ili zamor. Ostale poruke kao što su *da, ne, dobro, loše, gore, dole,* mogu se veoma lako slati i razumeti i uz pomoć običnih, svakodnevnih radnji i tu nisu potrebni *tehnički gestovi*.

Šifrovani gestovi

Šifrovani gestovi, za razliku od svih ostalih, predstavljaju deo formalnog sistema znakova. Oni stoje u uzajamnom odnosu, jedan je u vezi sa drugim na složen i sistematski način, tako da formiraju jedan pravi jezik. Posebna karakteristika ove kategorije jeste u tome što su njene pojedinačne jedinice beznačajne ako se ne odnose na ostale jedinice kodeksa. *Tehnički gestovi* mogu da se sistematično planiraju, ali, kod njih, svaki znak može da dejstvuje sasvim nezavisno od ostalih. Nasuprot tome, sve jedinice šifrovanih vezuju se jedna za drugom na strogo formulisanim principima, kao slova i reči u govornom jeziku.

Najznačajniji primer za ovo je *jezik znakova gluvonemih* koji se sastoji i od jednoručne i od dvoručne verzije. Takođe postoji *mornarski semaforski jezik* ručnih znakova, kao i „tik - tak“ jezik hipodroma za sporazumevanje onih koji se klade na konjskim trkama. Svi oni zahtevaju znatnu veštinu i uvežbanost i pripadaju jednom sasvim drugačijem svetu od onog poznatih gestova koje pravimo u svakodnevnom životu. Oni nas, pored ostalog, i te kako dobro podsećaju na onaj neverovatno senzitivni potencijal kojim svi mi raspolažemo za vizuelno komuniciranje. Sve to ukazuje na činjenicu da mi reagujemo, sa većom osetljivošću nego što smo svesni, na one obične gestove čiji smo svedoci svaki dan.

Literatura

[1] Matthew, MacDonald, **ASP.NET 3.5 sa C#**, drugo izdanje, Kompjuter biblioteka, Svetlost Čačak, 2009.

[2] Dezmond, Moris, **Otkrivanje čoveka kroz gestove i ponašanje**, Izdavački zavod JUGOSLAVIJA, Beograd, 1980.

[3] Vikipedija, http://sr.wikipedia.org