UN MODEL AVANSAT DE IMPLEMENTARE A APLICAŢIILOR WEB

Gheorghe CARMOCANU

LCŞ „Proiectarea sistemelor informatice"

The model of elaboration of Web applications by means of the modern tool ASP.NET 2.0 is considered in this work. The attention is focusing on the usage of the multiple managing systems of the data bases and of the store, on the object-oriented approach of the data management operations, on the usage of a module which should permit an easy and facile management of the configuration settings.

1. Problema

În prezent, foarte multe procese de prelucrare a informaţiei, cum ar fi procesele ce ţin de instruirea la distanţă, e-commerce şi altele, pot fi automatizate cu ajutorul aplicaţiilor Web. Astfel, îmbunătăţirea tehnologiilor de elaborare a aplicaţiilor Web prezintă o problemă de mare importanţă practică şi teoretică. De regulă, o aplicaţie Web are arhitectură de tipul client-server (Fig.1) [1,2].

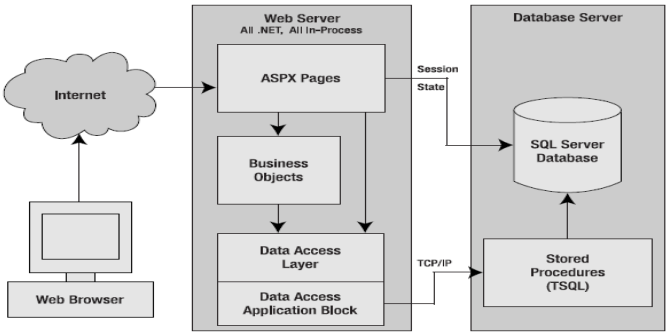

Fig.1. Structura funcţională a unei aplicaţii Web.

Implementarea în program a aplicaţiilor Web se realizează, de regulă, cu ajutorul unui sistem instrumental (de exemplu, ASP, PHP, Perl) şi al unor sisteme de gestionare cu Baze de Date (BD) (cum sunt MySQL, PostGree, SQL Server 2000/2005 etc). Realizarea unei aplicaţii Web cu ajutorul sistemelor instrumentale menţionate se încadrează în una din următoarele metodologii:

- aplicarea scripturilor care se interpretează de către o resursă software plasată pe partea server;
- aplicarea componentelor separate, care sunt executate de server atunci cand sunt apelate. Principalele dificultăţi întâmpinate la realizarea aplicaţiilor cu ajutorul sistemelor instrumentale enumerate mai sus consistă în următoarele: timp de dezvoltare mare din cauza tehnicilor de realizare a modulelor şi a interacţiunii dintre ele, abilitatea scăzută asupra controlului setărilor de securitate şi utilizarea ineficientă de resurse.

O altă problema a acestei metode rezidă în faptul că serverul Web trebuie sa creeze câte o instanţă a aplicaţiei pentru fiecare cerere a clientului. Acest model face ca aplicaţiile respective să fie mult mai puţin utilizate în medii cu un numar mare de utilizatori simultani, în afara faptului că realizarea codului cere o foarte mare atenţie. Aplicaţiile de acest tip de asemeanea pot fi greu de scris, de depanat şi integrat cu celelalte componente [2]. Această situaţie impune căutarea unui mod mai efectiv de implementare a unei aplicaţii Web. În continuare se face analiza unui model de implementare a unei aplicaţii Web cu aplicarea sistemului ASP.NET.

2. Posibilităţi oferite de sistemul ASP.NET

O parte din caracteristici prin care ASP.NET diferă de platformele anterioare de dezvoltare a aplicaţiilor Web sunt: - ASP.NET impune un model de programare complet orientată obiect şi o arhitectură a aplicaţiei bazată pe controlul şi reutilizarea codului; - ASP.NET oferă abilitatea de a codifica aplicaţia în orice limbaj suportat de platforma .NET Framework (inclusiv Visual Basic, C#, J# şi alte limbaje) [3]; - ASP.NET, în

cadrul mediului .NET, are acces la unităţi reutilizabile de cod; - paginile şi componentele ASP.NET sunt compilate la prima cerere în loc să fie interpretate de fiecare dată când sunt utilizate; -ASP.NET include, de asemenea, prin intermediul componentei ADO.NET, un bun model de acces la date şi flexibil în procesul utilizării memoriei **cache** pentru a creşte performanţa aplicaţiilor [4,9,12].

3. Principii de elaborare a arhitecturii unei aplicaţii Web

O aplicaţie Web, realizată cu o astfel de tehnologie, este formată dintr-un număr de module separate între ele, pentru a putea administra conţinut dinamic, cum ar fi: cursurile, forumurile, ştirile, sondajele, mesajele de e-mail. Problemele comune referitoare la proiectarea unei aplicaţii implică: - ***proiectarea multistrat*** (n-*tier*) – separarea codului de acces la date de codul de tranzacţionare (prelucreare) a datelor şi de codul de prezentare a acestora (interfaţa utilizator); - ***decuplarea nivelurilor*** – izolarea procedurilor de acces la date, astfel încât să fie permisă utlizarea de sisteme diferite de baze de date, fără a fi nevoie de schimbări la nivelul obiectelor de tranzacţionare sau să implice schimbări în nivelul de prezentare; - ***maparea datelor relaţionale în clase orientate obiect*** – proiectarea obiectelor de tranzacţionare pentru a expune datele extrase din nivelul de acces la date într-un format reprezentat obiect; - ***suportul de cachinng al obiectelor de tranzacţionare*** (administrarea memoriei cache) – salvarea datelor extrase din baza de date în memoria cache, pentru eventuala reutilizare a acestora pe parcursul unei sesiuni de lucru, acesta având ca rezultat creşterea performanţelor aplicaţiei prin utilizarea scăzută a CPU şi reducerea traficului de reţea. Celor enumerate mai sus li se alătură: ***administrarea şi comunicarea excepţiilor*** şi a unor evenimente importante, cum ar fi ştergerea unei înregistrări, ajutor în caz de eşec al sistemului, realizarea ***monitorizării stărilor şi auditul sistemului***. De asemenea, s-au avut în vedere problemele referitoare la stocarea setărilor de configurare a modulelor aplicaţiei într-o locaţie din care pot fi uşor citite şi modificate, crearea claselor de ajutor destinate să simplifice accesul la aceste setări, ataşarea controalelor interfeţei utilizator (User Interface (UI)) la datele extrase la nivelul logic de tranzacţionare, trasferând astfel rolul de administrare a datelor nivelului logic de tranzacţionare, nivelul UI focalizându-se mai mult pe prezentarea datelor.

4. Proiectarea unei aplicaţii pe niveluri

Proiectarea multistrat (n-*tier*), a aplicaţiilor ASP.NET, divide funcţionalitatea componentelor şi codul aplicaţiei în niveluri separate. În general, pot există patru niveluri (Fig.2): 1) ***Nivelul de date (sursa de date)*** – acesta poate fi o bază de date, un fişier XML, un fişier text sau orice alt sistem de stocare a datelor; 2) ***nivelul de acces la date*** **(DAL – Data Access Layer)** – presupune extragerea şi administrarea datelor stocate în baza de date; 3) ***nivelul logic de tranzacţionare*** **(BLL – Business Logical Layer)** – include preluarea datelor extrase de nivelul de acces la date, le prelucrează şi le expune clientului prin mai multe căi abstracte, ascunzând detaliile nivelul de jos. 4) ***nivelul de prezentare*** **(UI – User Interface)** – codul care specifică ceea ce trebuie să vadă utilizatorul pe ecran, inclusiv formatarea datelor şi meniul de navigare în sitem. Aceste niveluri sunt proiectate să ruleze în interiorul unui browser Web [4].

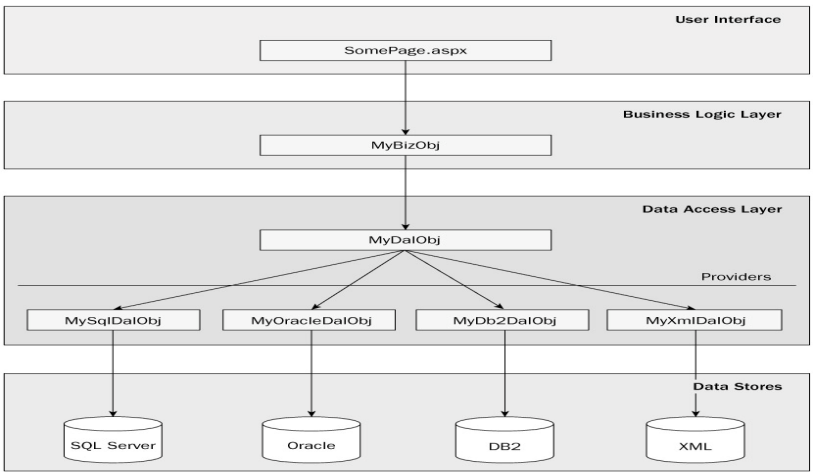

Fig.2. Schema funcţională pe straturi a unei aplicaţii Web.

În funcţie de dimensiunea şi complexitatea aplicaţiei pot fi încluse niveluri suplimentare sau nivelurile menţionate pot fi combinate. Pentru organizarea claselor în modelul obiectual al aplicaţiei se vor utiliza spaţii de nume [5].

4.1. Alegerea mecanismului de memorare a datelor

Pentru realizarea acestui nivel se propune de a fi utilizat serverul de baze de date „SQL Server 2005" al firmei Microsoft, care oferă mai multe avantaje: include excelente unelte din mediul Visual Studio; oferă preţ relativ scăzut şi nivel ridicat al performanţelor sub platforma Windows; asigură administrarea uşoară a datelor. O versiune SQL Server 2005 cu performanţe ridicate în realizarea aplicaţiilor Web este ediţia *free* SQL Server 2005 Express Edition [6]. Toate ediţiile SQL Server 2005 sunt echivalente funcţional, principalele diferenţe fiind legate de uneltele de administrare GUI. Puternica integrare a componenţei Runtime cu mediul NET. (SQL Server având comportament de gazdă pentru motorul CLR – Common Language Runtime) permite scrierea de funcţii UDS (User Defined Function) şi UDT (User Defined Type) în cod VB.NET sau C#, introducerea tipurilor de date native XML [13] şi suport pentru paginaţie. Limitările ediţiei Express sunt cele referitoare la: interzicerea lucrului cu mai mult de o unitate CPU, admiterea numai 1 GB RAM; mărimea fizică a bazei de date poate fi de maximum 4 GB [7].

4.2. Proiectarea nivelului de acces la date

Se preferă ca codul de acces la date să fie separat de alte coduri ale aplicaţiei Web, din mai multe motive. În cazul aplicaţiilor Web mijlocii şi mari, realizatorii UI pot fi diferiţi faţă de cei care creează codul de acces la date. Păstrând datele de conţinut în obiecte separate asigurăm abstractizarea ***high-level*** a tabelelor, procedurilor stocate, numelor câmpurilor şi a interogărilor SQL care lucrează cu ele. Menţinând codul de acces la date în clasele comune ale modulului DAL, atunci când vom avea nevoie să facem modificări (schimbarea unei interogări, tipuri de sortare etc.), vom modifica codul numai în aceste clase, paginile care le apelează rămânând neschimbate. Plasarea codului puternic structurat la nivelul paginilor Web ar face destul de dificilă migrarea către un alt sistem RDBMS (Relational Data Base Management System) sau asigurarea unui suport pentru mai multe sisteme RDBMS [15]. Nivelul DAL este alcătuit dintr-un număr de clase care extrag datele din baza de date prin intermediul procedurilor stocate şi le returnează ca o colecţie de clase. Fiecare model de aplicaţie Web va avea propria clasă abstractă de bază şi una sau mai multe clase provider, care realizează implementarea concretă a sistemului RDBMS specificat.

4.2.1. Proiectarea modelului de acces la date

Un lucru de care trebuie să se ţină cont atunci când se proiectează aplicaţii Web este că pot fi utilizate surse de gestionare a datelor multiple, cum ar fi: SQL Server, Oracle, MySQL, DB2 etc. Sistemul RDBMS are funcţii şi dialecte SQL diferite, procedurile de stocare şi parametrii pentru instrucţiunile SQL sunt transmişi prin sintaxă diferită, au tipuri de date diferite etc. Adăugarea suportului pentru surse de gestionare de date multiple se face prin utilizarea unui model **provider** (furnizor).

Pentru a evita dezavantajele descrise mai sus, se va utiliza o clasă de bază abstractă, în care va fi definită sintaxa publică a metodelor **CRAŞ** (Creare, Redenumire, Actualizare, Ştergere), şi vor fi implementate unele metode de ajutor. Codul de acces la date se va implementa prin intermediul unor clase, care moştenesc clasa de bază abstractă şi realizează implementarea concretă a metodelor abstracte. Aceste clase secundare se numesc **provider-i (furnizori**) şi sunt de obicei specifice unui anumit mediu de stocare a datelor [8]. Clasa de bază are o proprietate statică Instanţa care creează şi returnează o instanţă a clasei provider, în concordanţă cu care este specificat în fişierul de configurare **Web.config**. Această proprietate este utilizată de clasele de tranzacţionare pentru a obţine o referinţă la obiectele modulului **DAL** pentru a extrage/modifica date. Clasele de tranzacţionare, în schimb, sunt utilizate de nivelul UI, care nu accesează niciodată direct nivelul DAL de acces la date. Acest mod de abordare, numit **model provider**, permite de a crea rapid codul, deoarece dispunem de toate optimizările disponibile ale suportului de date ales. Modulele referitoare la autentificare, comunitatea de membrii profilul utilizatorului, administrarea conţinuturilor din această aplicaţie Web utilizează acest model provider.

4.2.2. Obiectul DataSet şi entităţi definite de utilizator

Atunci când clasele modului BLL apelează unele metode ale modului DAL pentru a extrage date, se pune problema de a defini formă de recepţionare a acestora. Datele pot fi recepţionate ca un obiect DataSet/DataTables sau ca o colecţie de obiecte ale claselor entităţi definite de utilizator. Obiectele DataSet/DataTables au unele dezavantaje care se referă la: performanţa şi scalabilitatea scăzută, reprezentarea datelor şi regulile de validare a tranzacţiilor [7]. Totuşi, în această prezentare a aplicaţiei Web, se propune utilizarea obiectului DataSet/Data Tables pentru a transfera date între niveluri DAL şi BLL, dar nu şi din nivelul BLL în nivelul UI. Nivelul BLL va adăuga toată logica de validare şi va transfera datele de interes general din obiectul DataSet/DataTables într-o colecţie adaptată de obiecte de tranzacţionare, care va fi transmisă nivelului UI sub forma unui set de date specific claselor orientate obiect.

Un ***obiect entitate definit de utilizator*** este o clasă care conţine datele extrase dintr-o bază de date printr-o cale orientată obiect abstractizând schema de stocare a datelor şi celelalte detalii. În această prezentare a aplicaţiei Web se propune utilizarea colecţiilor particularizate de entităţii definite de utilizator pentru a transfera şi prelucra date între niveluri DAL si BLL şi între niveluri BLL şi UI. În primul caz, clasele entitate sunt foarte simple, deoarece ele cuprind doar datele extrase din baza de date şi fără metode de inserare, actualizare şi ştergere. În al doilea caz, clasele sunt mult mai complexe: ele conţin date, dar au şi alte proprietăţi care se referă la obiectele-fiu sau la obiectele-părinte şi metode pentru a manipula datele. Aceste clase sunt, deseori, numite *obiecte domeniu*. Obiectele domeniu sunt uşor de întreţinut, rapid de încărcat din baza de date, uşor de utilizat în memorie, deoarece permit încărcarea numai a înregistrărilor de care în mod real avem nevoie şi numai atunci când sunt necesare, în loc să încarce totul în acelaşi timp printr-o singură cerere. Permit adăugarea unei logici particularizate de validare, deoarece datele sunt conţinute în proprietăţi pe care le putem extinde conform logicii de care avem nevoie [3].

4.2.3. Proceduri de stocare şi interogări SQL text

Există două moduri de proiectare pentru accesul la baze de date. Unul este cel în care utilizăm instrucţiunile SQL prin intermediul procedurilor stocate şi utilizarea obiectelor SqlCommand pentru execuţia acestor proceduri. Celălalt mod constă în stocarea instrucţiunilor SQL sub formă de text care să fie executate prin intermediul obiectelor SqlCommand. Performanţele procedurilor stocate şi ale interogărilor SQL text sunt asemănătoare în cele mai multe cazuri. Procedurile stocate sunt mai performante decât instrucţiunile SQL text, deoarece sunt precompilate (nu ne referim la compilare ca la un program binar, ci analizate şi utilizate să genereze un plan de execuţie) şi stocate în memoria cache prin serverul de baze de date. Însă, numeroşi autori uită să menţioneze că acest lucru se întâmplă şi cu interogările SQL text, atunci când utilizăm instrucţiuni parametrizate (acestea permit evitarea atacurilor de securitate de tip injection SQL) [7,11]. În aplicaţiile Web procedurile stocate oferă o serie de avantaje. Procedurile stocate au numele mai scurt şi invariabil, ceea ce permite motorului SQL Server să găsească mai rapid planul de execuţie în memoria cache; permit un control al securităţii de acces la date mult mai fin; reduc traficul în reţea; permit adăugarea de securitate la nivel de înregistrare. Un alt avantaj al procedurilor stocate este că furnizează o stratificare a codului. Nu există cod SQL la nivelul modului DAL. Acest lucru este de dorit, deoarece ne permite să distribuim nivelul DAL compilat, iar eventualele ajustări şi optimizări în codul procedurilor stocate nu vor avea nici un impact asupra codului DAL, nemaifiind nevoie să-l recompilăm.

Nu în toate cazurile procedurile stocate reprezintă cea mai bună alegere. Cel mai mare avantaj al utilizării instrucţiunilor SQL text este că sunt mult mai flexibile. În aplicaţiile Web sunt utilizate cu succes, la prelucrarea şi prezentarea datelor prin intermediul nivelului UI instrucţiunile SQL text. Pentru implementarea formularelor referitoare la filtrări şi căutări avansate prin UI, în care utilizatorul are să introducă parţial text în controalele de tip text, el trebuie să filtreze conţinutul unor câmpuri şi să ordoneze rezultatele altor câmpuri ale bazei de date. Instrucţiunile SQL text permit implementarea caracteristicii de paginaţie prin care rezultatele sunt afişate pe mai multe paginii, păstrând astfel performanţele şi scalabilitatea şi eliminând plictiseala utilizatorului, prin reducerea timpului de încărcare, în cazul tabelelor cu foarte multe înregistrări.

4.3. Proiectarea nivelului logic de tranzacţionare

Datele returnate de nivelul DAL sunt încă brute, chiar dacă sunt conţinute în clase, deoarece aceste clase entitate nu adaugă nimic, ele sunt numai nişte tipuri de containere utilizate pentru a muta datele dintr-o parte în alta. Nivelul BLL prelucrează aceste date, adaugă logica de validare şi proprietăţi calculate, schimbă spe-

cificatori de acces pentru unele proprietăţi, adaugă instanţe şi metode statice ca să extragă datele, pentru inserare, editare, ştergere. Această reprezentare orientată obiect şi puternic tipizată a oricăror date furnizează o extrem de puternică abstractizare a bazei de date, care nu doar stochează datele, ci furnizează şi un set puternic de clase pe care constructorii UI le pot utiliza, fără chiar să cunoască detalii despre locul în care şi cum sunt stocate datele brute, câte tabele utilizează baza de date şi care sunt relaţiile dintre ele. Acest mod de abordare uşurează muncă constructorilor UI, oferă rezultate uşor de administrat şi o siguranţă crescută, dă posibilitatea de a modifica la nivelul low-level baza de date, fără să producă o ruptură cu codul interfeţei utilizator şi, de asemenea, susţine motivaţia utilizării proiectării multistrat.

5. Setările de configurare

Administrarea configurărilor nu poate fi încadrată în nici unul dintre cele trei niveluri, dar este folosită de toate nivelurile. La prezentarea nivelurilor DAL şi BLL am menţionat că aplicaţia Web va avea un număr de setări de configurare, cum ar fi: numele provider-ului nivelului DAL, starea memoriei cache, durata de păstrare a datelor în memoria cache şi şirul de conectare la baza de date. Nivelul UI va avea nevoie de locaţia unde se vor salva adresele de e-mail trimise din formularul Contact al aplicaţiei, textul predefinit pentru subiectul mesajului etc. Toate aceste setări sunt salvate în fişierul de configurare Web.config din directorul rădăcină al aplicaţiei Web. Datorită reprezentării în format text, acesta este foarte uşor de administrat de către un administrator utilizând un simplu editor de text. Odată modificat acest fişier, ASP.NET îl va încărca automat şi noile modificări vor putea fi folosite chiar de utilizatori online la primul refresh, fără să fie nevoie de a reseta serverul IIS [12].

5.1. Secţiuni de configurare definite de utilizator

Aplicaţia Web de tip multistrat, realizate cu ASP.NET 2.0, are un număr de setări pentru fiecare modul, astfel încât am definit o singură secţiune de configurare cu câte un subelement pentru fiecare modul. Fiecare modul va avea propriile sale setări pentru provider, şirul de conectare şi administrare a memoriei cache. Cu toate acestea, este bine să se precizeze unele valori predefinite pentru aceste setări la nivel de secţiune, astfel încât, dacă dorim să utilizăm acelaşi şir de conectare pentru toate modulele, nu va trebui specificat pentru fiecare modul în parte, ci numai odată pentru întreaga aplicaţie Web.

În noua versiune ASP.NET 2.0 este mult mai uşor să utilizăm secţiunile definite de utilizator, aceasta fiind şi metoda cea mai preferată. Acest lucru poate fi realizat prin crearea unei clase particularizate, care să moştenească clasa ConfigurationSection din spaţiul de nume System.Configuration, şi declararea proprietăţilor sale ca fiind publice, cu atributul ConfigurationProperty, pentru a preciza că este nevoie ca valorile lor să fie citite din fişierul Web.config. Pentru citirea setărilor prin cod se va utiliza metoda GetSection a clasei de bibliotecă WebConfigurationManager [9].

6. Modul de gestionare a comunităţii de membri

Modelul de dezvoltare analizat în prezentul articol este orientat spre elaborarea unei aplicaţii Web dinamice. Deci, este considerată o aplicaţie Web bazată pe conţinut, în care părţi semnificative ale acesteia pot fi uşor modificate sau actualizate de către unii utilizatori privilegiaţi. Această funcţionalitate este în mod general numită **Content Management System** (CMS - Sistem de Management al Conţinutului) [3]. Un sistem al unei comunităţi de membri este necesar pentru cele mai multe site-uri, nu numai pentru gestionarea utilizatorilor, dar şi pentru administrarea drepturilor de acces la conţinut. Componenta de administrare poate fi la fel de complexă ca o aplicaţie.

Un sistem al unei comunităţi de membri trebuie: să permită administrarea unor activităţi; crearea conturilor utilizator să poată fi realizată independent utilizând un formular de înregistrare online; să permită utilizatorilor schimbarea drepturilor de autentificare (parolă, profil etc.) sau recuperarea acestora, la necesitate. Administratorul trebuie să poată modifica drepturile de acces la anumite componente ale site-lui sau anumite pagini individuale, pentru un utilizator. Drepturile de autorizare ar trebui să fie editabile şi după distribuirea site-ului, fără a fi necesară modificarea codului. Sistemul trebuie să permită administratorului site-ului să acceseze informaţii statistice (numărul total de utilizatori înregistraţi, numărul utilizatorilor online, date referitoare la administrarea conturilor). Sistemul trebuie să permită fiecărui utilizator înregistrat să salveze informaţiile referitoare la datele personale în sursa de date, astfel încât setările să persiste între sesiunile de lucru.

6.1. Administrarea conturilor utilizator

Principala clasă din cadrul securităţii ASP.NET 2.0 este **System.Web.Security. Membership,** care expune un număr de metode statice pentru crearea, ştergerea, actualizarea şi extragerea utilizatorilor înregistraţi. Metodele acestei clase acceptă sau returnează instanţe ale clasei a doua **System.Web.Security.MembershipUser**, care reprezintă de fapt un singur utilizator şi furnizează detalii despre acesta prin intermediul metodelor acestei clase. Aceste două clase au fost utilizate pentru implementarea zonei de administrare a aplicaţiei, pentru: extragerea numărului de utilizatori, determinarea utilizatorilor online, implementarea funcţiei de căutare după o mască dată (nume parţial, adresă de e-mail etc.), detalii de logare. De asemenea, pentru operaţiile de prelucrare a datelor reprezentând conturile se va utiliza modelul provider .NET de abstractizare a datelor prin intermediul claselor. Clasa Membership utilizează o clasă secundară pentru a implementa suportul logic de acces la datele conturilor stocate în baza de date. Utilizând ca suport pentru date serverul SQL Server Exprees 2005, poate fi ales ca provider de date SQLServerProvider. Există şi posibilitatea să se editeze propriul provider sau să se utilizeze un altul, predefinit de terţi (Oracle, MySQL, sau chiar pentru fişiere XML). Selectarea unuia sau altuia dintre aceştia făcându-se prin anumite setări în fişierul web.config. Considerăm utilizarea modelului provider destul de flexibilă, deoarece putem schimba providerul folosit de interfaţa API a comunităţii de membri fără să afectăm codul [10].

6.1.1. Constituirea şi utilizarea rolurilor

Completarea unui sistem de autentificare/autorizare se face prin intermediul rolurilor. Rolurile sunt utilizate pentru gruparea utilizatorilor cu scopul utilizării împreună a unui set de permisiuni sau autorizări. ASP.NET 2.0 are propriul suport pentru roluri şi oferă performanţe şi flexibilitate destul de bună în ceea ce priveşte securitatea. Sistemul de roluri, ca, de altfel, mai toate componentele acestui modul, se bazează pe şablonul de proiectare a modelului provider. Opţiunile de configurare (activare roluri, activare memorie cookie) vor fi precizate prin intermediul atributelor elementului <roleManager> din fişierul de configurare Web.config. Administrarea informaţiilor despre roluri prin program se va face prin intermediul metodelor statice ale clasei System.Web.Security.Roles [10].

6.1.2. Utilizarea rolurilor pentru protejarea paginilor împotriva accesului neautorizat

Există două căi pentru a controla şi a proteja accesul la paginile sensibile. O putem face într-un mod imperativ (algoritmic) sau într-un mod declarativ (folosind fişiere Web.config). Sistemul de roluri se integrează perfect cu interfaţa standard de securitate IPrincipal, care este implementată ca un obiect returnat de proprietatea User a paginii. Pentru a preîntâmpina dezavantajul securităţii imperative, în cazul securizării unui întreg director, caz în care codul trebuie copiat în fiecare fişier de cod ascuns al paginii, sau dezavantajul creat de schimbarea ACL-ului (lista de control a accesului) pentru o pagină sau un director, caz în care va trebui să schimbăm codul în toate fişierele, utilizăm securitatea declarativă. Am definit o secţiune <authorization> în fişierul Web.config pentru tot site-ul şi câte una pentru fiecare subdirector, în care am specificat utilizatorii şi rolurile care au acces pe un anumit director sau pagină [4,14].

7. Concluzii

ASP.NET înlătură dezavantajele vechilor tehnologii de dezvoltare a aplicaţiilor Web care se referă în general la: abilitatea scăzută asupra controlului setărilor de securitate şi utilizarea ineficientă a resurselor. De asemenea, o altă problemă o constituie faptul că server-ul Web trebuie sa creeze câte o instanţă a aplicaţiei pentru fiecare cerere a clientului. Acest model face ca aceste aplicaţii să fie mult mai puţin utilizate în medii cu un număr mare de utilizatori simultani, plus la faptul că realizarea codului cerea o foarte mare atenţie. Acest tip de aplicaţie poate fi de asemeanea greu de scris, de depănat şi integrat cu celelalte componente. ASP.NET este o platformă de dezvoltare a aplicaţiilor Web perfect adaptabilă pentru proiectarea, implementarea aplicaţiilor multistrat (n-tier), susţine un model de programare complet orientată obiect şi o arhitectură bazată pe controlul şi reutilizarea codului, oferă abilitatea de scriere a codului în orice limbaj suportat de platforma .NET Framework (inclusiv Visual Basic, c#, J#, etc). ASP.NET este, de asemenea, o platformă pentru dezvoltarea serviciilor Web, care sunt unităţi de cod reutilizabile pe care alte aplicaţii le pot accesa prin intermediul platformei de pe computerul care asociază aceste servicii.

Referinţe:

1. Selly D., Troelsen A., Barnaby T. Expert ASP.NET 2.0. Advanced Application Design. Apress, 2006. - 440 p.
2. Carmocanu Gh. Tehnologia ASP.NET în elaborarea aplicaţiilor pe reţele // Conferinţa corpului didactico-ştiinţific „Bilanţul activităţii ştiinţifice a USM în anii 2000-2002, 30 septembrie – 6 octombrie 2003". - Chişinău: CEP USM, 2003, p.146-147.
3. Moroney L., MacDonald M., et al. Pro ASP.NET 2.0 in VB 2005. Apress, 2006, ISBN-13: 978-1-59059-776-7. - 360 p.
4. Bellinaso M. ASP.NET 2.0 WebSite programming – Problem – Design – Solution, Wrox Press, 2006. ISBN: 978-0-7645-8464-0. - 600 p.
5. Ferguson J., Patterson B., et al. C# Bible. Wely Publishing Inc., Indianapolis, 2006, ISBN: 978-0-7645-4834-5. - 830 p.
6. Schneider R. SQL Server 2005. Express Edition for Dummies. Wiley Publishing, Inc, Indianapolis, 2006, ISBN: 978-0-7645-9927-9. - 408 p.
7. Rajesh G., Lance D. Wrox's SQL Server 2005 Express Edition Starter Kit. Wiley Publishing, Inc., Indianapolis, 2006. - 384 p.
8. Esposito D. Programming Microsoft ASP.NET 2.0 Core Reference. Microsoft Press, 2006, ISBN 9780735621763. - 784 p.
9. Evjenet B., et al. Professional ASP.NET 2.0 Special Edition. Wrox Press, 2006. - 1584 p.
10. Schackow S. Professional ASP.NET 2.0 Security, Membership, and Role Management. Wiley Publishing, Inc. Indianapolis, 2006. - p.648.
11. Wallace B. McClure, Gregory A. Beamer et al. Professional ADO.NET 2.0 Programming with SQL Server 2005, Oracle, and MySQL. Wiley Publishing, Inc, Indianapolis, 2006, SBN: 978-0-7645-8437-4. - 648 p.
12. Esposito D. Introducing Microsoft ASP.NET 2.0. Microsoft Press, Redmond, Washington 98052-6399, 2005, ISBN 9780735620247. - 448 p.
13. Felea V., Gorea D. Bazele de date native XML în contextul sistemelor de e-learning pe Web // Lucrările Conferinţei Naţionale de Învăţământ Virtual: ediţia a III-a, Bucureşti, 28-30 octombrie, 2005. Software educaţional. – Bucureşti: Editura Universităţii din Bucureşti, ISBN 973-737-097-X , 2005, p.2-3.
14. Armstrong D. Pro ASP.NET 2.0 Website Programming. Apress, 2005, ISBN10: 1-59059-546-7. - 672 p.
15. Thangarathinam Th. Professional ASP.NET 2.0 Databases. Wiley Publishing, Inc., Indianapolis, 2007, ISBN: 978-0-470-04179-6. - 504 p.

Prezentat la 19.04.2008