Vysoká škola ekonomická v Praze
Fakulta informatiky a statistiky
Vyšší odborná škola informačních služeb Praha

Miroslav Jandák

Návrh a vývoj aplikace pro internetový obchod

Bakalářská práce

2007

Prohlašuji, že jsem diplomovou bakalářskou práci na téma Internetový obchod zpracoval samostatně a použil pouze zdrojů, které cituji a uvádím v seznamu použité literatury.

V Praze dne 22.8.2007

Obsah

Úvod ... 1
Specifikace požadavků ... 1
Struktura aplikace ... 5
Požadavky na server ... 6
Konfigurace aplikace ... 6
Struktura databáze ... 7
Stromová struktura a relační databáze ... 9
Katalog produktů ... 13
  Vytvoření stromu kategorií ... 13
  Zobrazení obsahu kategorie ... 16
Zobrazení detailů o produktu ... 18
Uživatelé ... 20
  Anonymní uživatelé ... 24
  Přihlášení uživatelé ... 25
Košík ... 27
  Funkce košíku ... 30
  Přenesení obsahu košíku anonymního uživatele do košíku přihlášeného uživatele ... 31
Objednávky ... 32
  Vytvoření objednávky ... 32
  Zobrazení objednávek ... 33
  Detaily objednávky ... 33
  Zrušení objednávky ... 34
Jednotný layout aplikace ... 35
Další vývoj ... 37
  Administrační rozhraní ... 37
  Elektronické platby ... 37
  Další fukčnosti ... 39
Seznam použitých zdrojů ... 40
  Odborná literatura: ... 40
  Internetové zdroje: ... 40

Úvod

Hlavním cílem této práce je vytvoření základního řešení WWW aplikace, která bude sloužit jako internetový obchod, tj. k objednávání zboží prostřednictvím Internetu. Zdůrazňuji, že se jedná pouze o základní řešení zaměřené zejména na část přístupnou zákazníkům. Nicméně již v této podobě by aplikace teoreticky mohla být nasazena a plnit svůj účel.

Aplikace bude vytvořena pomocí technologií ASP.NET 2.0 a MS SQL 2005. Součástí práce je i stručný popis některých nových vlastností ASP.NET 2.0. Konkrétně Membership a Profiles.

Výsledek je možno shlédnout na http://bakalarka.sprinx.cz.

Specifikace požadavků

Jedná se o klasickou webovou aplikaci, jejíž uživatelské rozhraní bude dostupné pomocí pomocí webových prohlížečů Microsoft Internet Explorer a Mozilla Firefox (a možná i jiných). Jeho prostřednictvím si zákazníci mohou zobrazovat informace o produktech, přidávat je do nákupního košíku a vytvářet objednávky.

Produkty jsou rozděleny do kategorií, což umožňuje jejich jejich snažší vyhledávání. Každá kategorie může obsahovat podkategorie, které rozdělují skupinu produktů na konkrétnější podskupiny. Zákazník může procházet jednotlivé kategorie a zobrazovat si jejich obsah. Když zákazník vstoupí do obchodu, tak se mu zobrazí produkty označené jako novinky a sepciální akce.

Aplikace bude uchovávat informace o produktech, zákaznících a jejich objednávkách v relační databázi.

Uživatelské rozhraní je k dispozici pouze zákazníkům, u administrátora systému se buď předpokládá, že ovládá SQL, nebo že použije nějakou další aplikaci umožňující zaobrazovat databázová data, např. MS Excel.

Na všech stránkách aplikace bude zákazníkovi k dispozici přehledné navigační menu, s jehož pomocí se bude moci pohybovat po obchodě. Všechny stránky aplikace budou mít jednotný design.

Jsou dva typy zákazníků, kteří se mohou v obchodu pohybovat: Anonymní a Přihlášení. Liší se v tom, jaké akce mohou provádět. Základní rozdíl ale spočívá v tom, že přihlášení zákazníci mají v databázi uvedenou svojí doručovací adresu, mohou tedy vytvářet objednávky. Anonymní zákazníci mohou pouze prohlížet produkty a přidávat je do košíku. Podrobněji viz UseCase diagram.

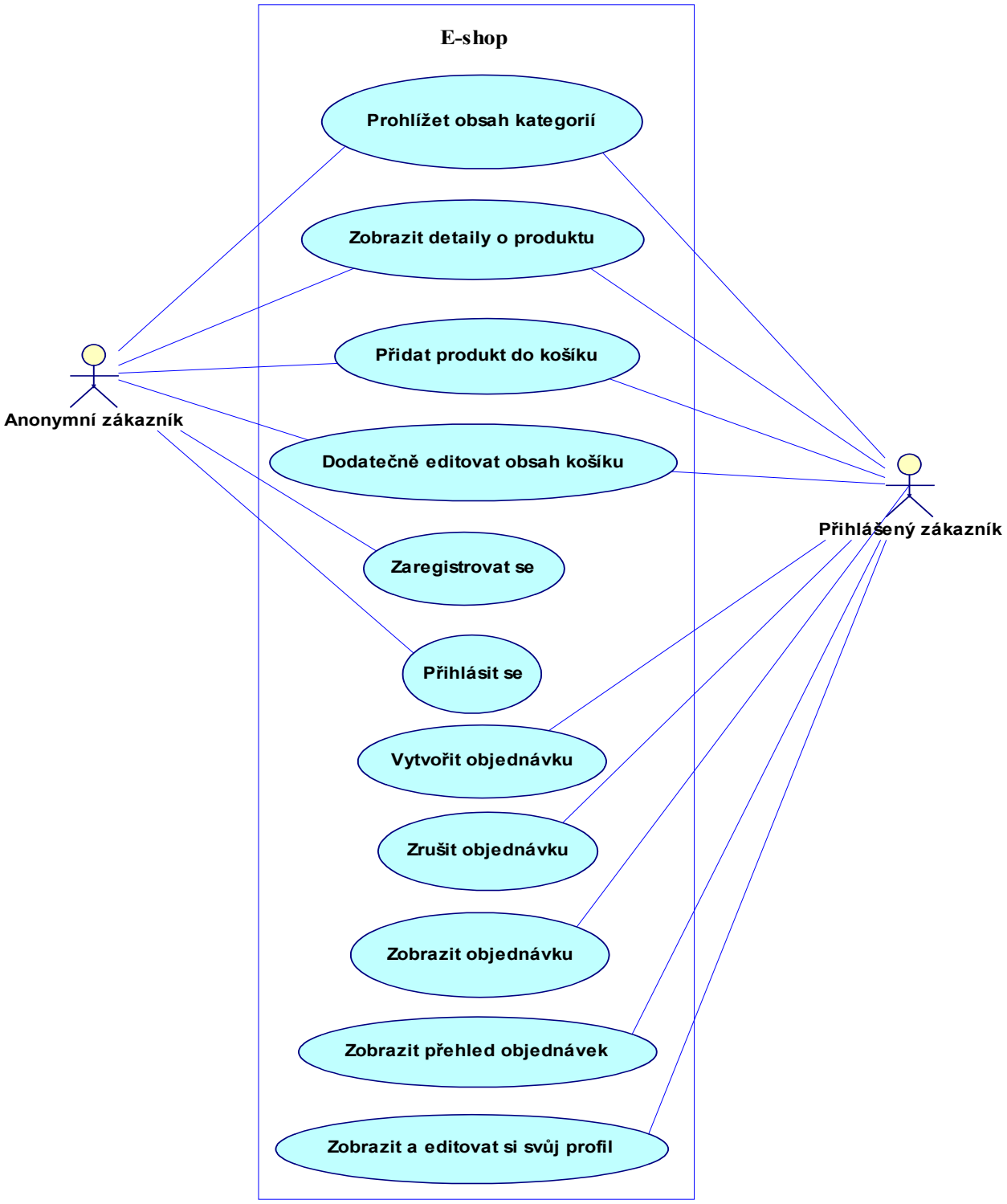
Anonymní zákazník se stane přihlašeným pokud při přihlašování projde autentizací. Přihlásit se můžou pouze zákazníci, kteří v minulosti prošli registrací. Během registrace uživatel do systému a zadá svou doručovací adresu.

Jsou dva typy objednávek, přesněji řečeno dva typy stavů, ve kterých se objedávka může nacházet. První z těchto stavů signalizuje, že objednávka ještě nebyla vyřizena a ještě je

zpracovávána, tj. zboží ještě nebylo zákazníkovi odesláno. Takovouto objednávku může zákazník, který jí objednal, kdykoliv zrušit. Druhý typ objednávek zahrnuje objednávky, které již byly vyřízeny, tj. zboží již bylo odesláno. Tyto objednávky zákazník již zrušit nemůže.

Platby jsou prováděny na dobírku.

Jaké akce lze provádět a která ze dvou skupin zákazníků je může provádět je zřejmé z následujícího Use Case diagramu.



Obrázek 1 – Use Case diagram

Prohlížet obsah kategorií

K prohlížení obsahu kategorií slouží strom kategorií.
Zákazník může kliknout na libovolnou kategorii a zobrazit si tak produkty, které se v ní nacházejí. Ve stromu se zároveň zobrazí přímé podkategorie vybrané kategorie. Vybraná kategorie je zvýrazněna.
Zobrazí se nejen produkty nacházející se v dané kategorii, ale i ve všech jejích podkategoriích.
U každého produktu se zobrazí jeho název, cena a zmenšený obrázek a tlačítka pro přídání do košíku a zobrazení detailních informací o produktu.
Produkty se nebudou zobrazovat všechny najednou, ale ve stránkách maximálně po devíti.
Zákazník si bude moci mezi stránkami listovat.
Bude se zobrazovat počet všech stránek a na které se zákazník zrovna nachází.

Zobrazit detaily o produktu

Zákazník si může u každého produktu zobrazit detailní informace o něm.
Zobrazí se mu název, cena, výrobce, detailní popis , obrázek, tlačítko pro přidání do košíku a pole pro zadání počtu kusů.

Přidat produkt do košíku

Zákazník má možnost přidat produkt do košíku když si prohlíží obsah kategorií nebo když si prohlíží detailní informace o produktu.
Zakazník může do košíku opakovaným klikáním přidat více kusů daného produktu.
Při prohlížení detailních informací je k dispozici textové pole pro zadání přesného počtu kusů.

Dodatečně editovat obsah košíku

Zákazník si může zobrazit obsah svého košíku a případně editovat jeho obsah.
U každého produktu může změnit jeho množství nebo produkt z košíku úplně odstranit.
K dispozici je též tlačítko pro vysypání celého košíku najednou.
Košík bude u každého produktu zobrazovat identifikátor, název, množství a cenu.
Zobrazovat se bude i celková cena všech produktů v košíku dohromady.
U každého produktu v košíku si zákazník bude moci zobrazit detailní informace o něm.

Vytvořit objednávku

Pokud má uživatel v košíku alespoň jeden produkt, tak může vytvořit objednávku.

Zobrazit přehled objednávek

Zákazník bude mít možnost si zobrazit přehled svých objednávek. U každé objednávky se zobrazí její identifikátor, datum kdy byla vytvořena a její stav (zpracovává se, vyřízeno). Dále bude možnost si u každé objednávky zobrazit detailní informace o ní.

Zobrazit objednávku

Zákazník si bude moci zobrazit každou svou objednávku, bez ohledu na její stav. Zobrazí se vždy identifikátor objednávky, datum kdy byla vytvořena, produkty v ní obsažené a celková cena objednávky. U každého produktu v objednávce se zobrazí jeho identifikátor, název a cena.

Zrušit objednávku

Zákazník bude mít možnost zrušit jakoukoliv objednávku, která ještě není vyřízena.

Zaregistrovat se

Pokud se bude chtít uživatel
Během registrace zákazník do systému zadá své identifikační údaje, pomocí těchto údajů se bude provádět jeho identifikace.

Dále musí zadat svou e-mailovou adresu a doručovací adresu. Doručovací adresa se skládá z následujících údajů:

Jméno
Příjmení
Ulice
Město
PSČ

Přihlásit se

Zákazníkovi se zobrazí formulář pro zadání jeho identifikačních a ověřovacích údajů.

Zobrazit a editovat si svůj profil

Zákazník si bude moci zobrazit a případně změnit svůj profil. Profil zákazníka je jeho doručovací adresa a jeho e-mail.

Struktura aplikace

- Kořenový adresář aplikace
  - App_Code
    - CartItem.cs – třída reprezentující produkt
    - ShoppingCart.cs – třída reprezentující nákupní košík
  - App_Themes
    - StyleSheet.css
  - Images – adresář obsahující obrázky zboží
  - NotForAnon – adresář obsahující stránky nepřístupné anonymním zákazníkům
    - CreateOrder.aspx – vytvoření objednávky
    - CreateOrder.aspx.cs
    - DisplayOrders.aspx – zobrazení objednávek zákazníka
    - DisplayOrders.aspx.cs
    - OrderDetails.aspx – zobrazení detailů objednávky
    - OrderDetails.aspx.cs
    - OrderDelete.aspx – zrušení objednávky
    - OrderDelete.aspx.cs
    - Profile.aspx – zobrazení a editace profilu zákazníka
    - Profile.aspx.cs
    - Web.Config – konfigurační soubor
  - CatBrowser.aspx – katalog produktů
  - CatBrowser.aspx
  - ProductDetail.aspx – detaily produktu
  - ProductDetail.aspx.cs
  - Cart.aspx - zobrazení a editace nákupního košíku
  - Cart.aspx
  - Register.aspx – registrace zákazníka
  - Register.aspx.cs
  - Login.aspx – stránka obsahující logovací formulář
  - Login.aspx.cs
  - MasterPage.master – stránka obsahující společné prvky všech stránek
  - MasterPage.master.cs
  - Web.Config – konfigurační soubor
  - Global.asax – soubor obsahující obsluhy událostí, které mohou nastat na kterékoliv stránce v rámci celé aplikace

Soubory .aspx obsahují tzv. markup, což je HTML kód (případně Javascript) plus ASP.NET serverové prvky, což jsou objekty, které ASP.NET většinou vyrenderuje zase jako nějaký HTML kód (plus případně Javascript).

Ke každé .aspx stráce patří jeden soubor s příponou .cs (tzv. code – behind file), který obsahuje kód v jazyce C#, který daná stránka má vykonávat.

Soubory s příponou .cs a soubory MasterPage.master, Web.Config a Global.asax jsou chráněny, ASP.NET odmítne každý požadavek na ně směřující. Uživatelé si je nemohou stáhnout ani zobrazit jejich obsah.

Požadavky na server

Aplikace je vyvinuta pro instalaci na server, kde bude nainstalován software v následujících verzích:

Windows Server 2003
Microsoft SQL Server 2005
.NET Framework 2.0
IIS 6.0 s podporou ASP.NET 2.0

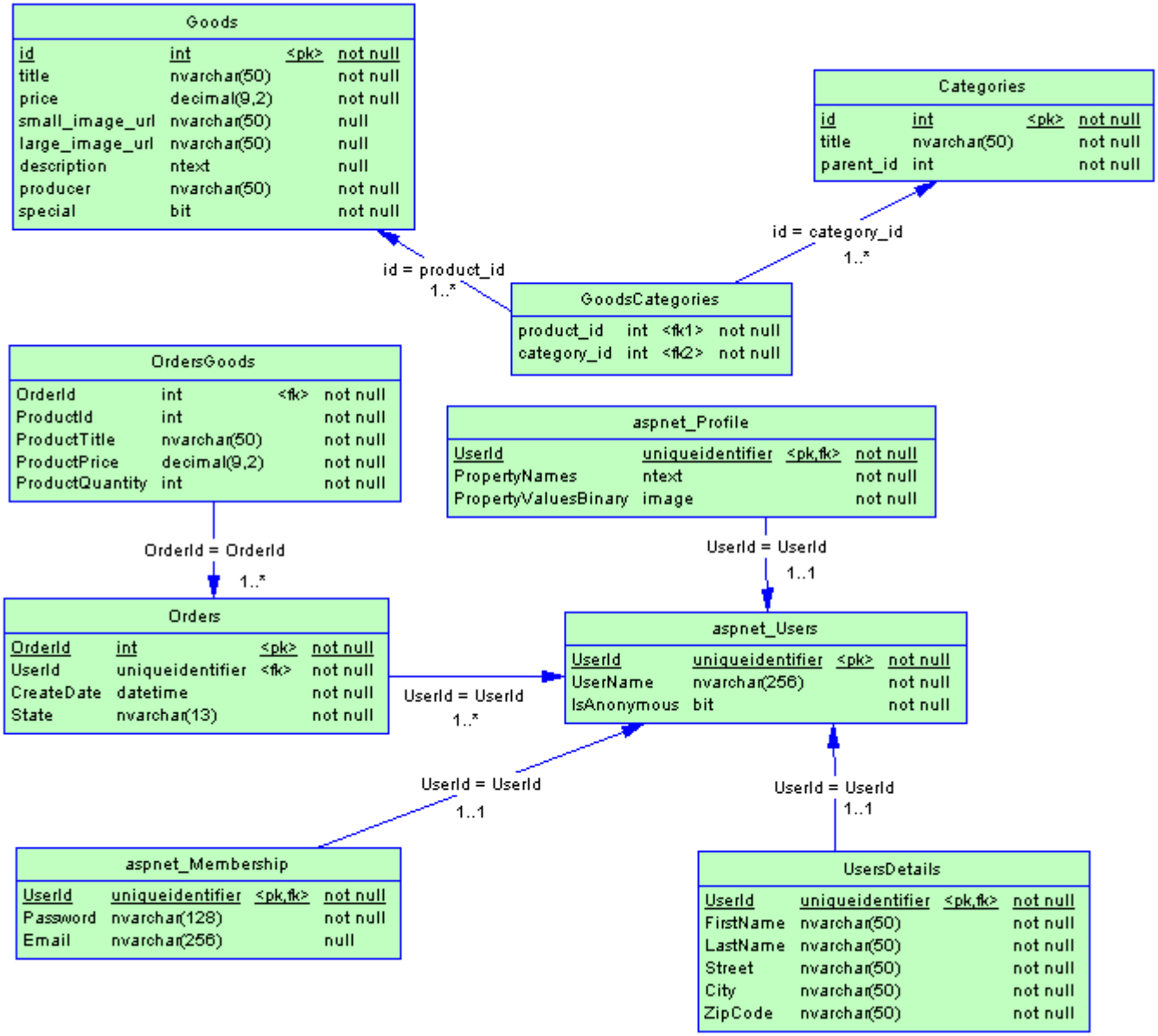
Konfigurace aplikace

V souboru Web.Config je třeba nastavit tzv. connection string, což je řetězec obsahující potřebné údaje (název serveru, název databáze, způsob ověření) pro připojení k databázi, kterou bude aplikace používat. Tento řetezec se nachází v sekci connectionStrings pod jménem DatabaseConnection.

```
<connectionStrings>
<remove name="LocalSqlServer"/>
<add name="DatabaseConnection"
     connectionString="Data Source=.\SQLEXPRESS;Initial
                       Catalog=eshop;Integrated Security=True"
     providerName="System.Data.SqlClient" />
</connectionStrings>
```

Struktura databáze

Na obrázku č. 2 je vidět fyzický model databáze.



Obrázek 2 - Fyzický model databáze

Popis tabulek:

Goods

Obsahje záznany pro každý produkt nabízený v obchodě.

id – identifikátor (primární klíč)
title – název
price – cena

large_image_url – umístění obrázku produktu
small_image_url – umístění zmenšeného obrázku
description – popis produktu
producer – výrobce produktu
special – je produkt zobrazován jako specialní akce nebo novinka?

Categories

Obsahuje potřebné údaje pro každou kategorii.

Id – identifikátor kategorie
Title – název kategorie
Parent_id – identifikátor nadřazené kategorie

Kategorie, které nemají žádnou nadřazenou kategorii, mají parent_id = 0.

GoodsGategories

Vazební tabulka mezi tabulkami Goods a Categories, lze z ní zjistit ve kterých kategoriích se nějaký produkt nachází.

product_id – identifikátor produktu
category_id – identifikátor kategorie

aspnet_Users

Obsahuje záznamy o zákaznících, anonymních i přihlášených. O anonymním zákazníkovi se vytvoří záznam pouze když použije košík. Tato tabulka je využívána systémem Membership. Kromě dalsích udájů tabulka obsahuje:

UserId – identifikátor uživatele (primární klíč)
UserName – přihlašovací jméno
IsAnonymous – je uživatel anonymní nebo zaregistrovaný?

Při přihlašování je ale zákazník identifikován podle zadaného přihlašovacího jména, čemuž odpovídá údaj ve sloupci UserName. K autentizaci slouží údaj ve sloupci Password v tabulce aspnet_Membership.

O anonymním zákazníkovi se vytvoří záznam jen pokud něco přidá do košíku. Anonymní uživatelé mají hodnotu ve sloupci UserName nastavenu na GUID (Globally Unique Identifier), které pro každého vygeneruje ASP.NET (viz kapitola Anonymní uživatelé).

aspnet_Membership

Obsahuje údaje o zaregistrovaných uživatelích.

UserId – identifikátor zákazníka (cizí klíč)
Password – heslo
Email

aspnet_Profile

Tato tabulka je využívána systémem Profiles. Ve sloupci PropertyValuesBinary je serializován objekt `ShoppingCart`, který představuje košík konkrétního uživatele. Identifikátor uživatele je ve sloupci UserId, což je cizí klíč z tabulky aspnet_Users.

UsersDetails

Obsahuje údaje o zaregistrovaných zákaznících.

UserId – identifikátor (cizí klíč)
FirstName – jméno
LastName – příjmení
Street – ulice
City – město
ZipCode - PSČ

Orders

Obsahuje informace o každé objednávce.

OrderId – identifikátor objednávky (primární klíč)
UserId – identifikátor zákazníka (cizí klíč)
CreateDate – datum vytvoření objednávky
State – stav objednávky

Na sloupci State je omezeni (CHECK constraint), záznamy mohou nabývat pouze dvou hodnot: „zpracovává se“ a „vyřízeno“.

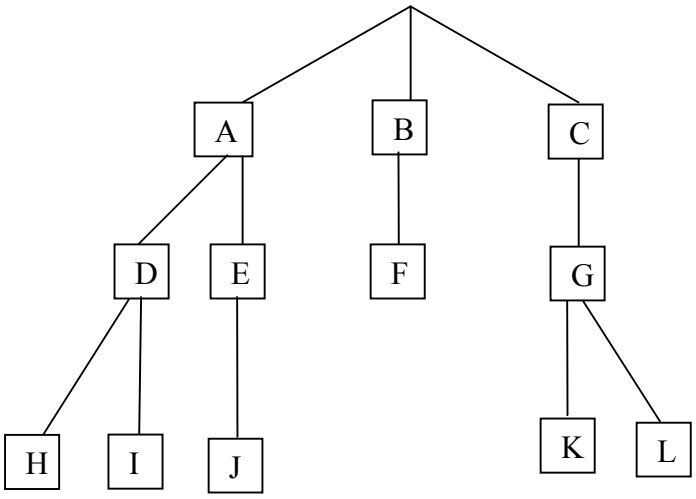
OrdersGoods

Vazební tabulka mezi tabulkami Orders a Goods. Lze z ní zjistit jaké produkty jsou na nějaké objednávce, jaké je jejich množství a jaká byla jejich cena v době vytvoření objednávky.

OrderId – identifikátor objednávky (cizí klíč)
ProductId – identifikátor produktu
ProductTitle – název produktu
ProductQuantity – počet kusů daného produktu na objednávce
ProductPrice – cena produktu v době vytvoření objednávky

Stromová struktura a relační databáze

Jak již bylo řečeno produkty budou rozděleny do kategorií. Každá kategorie může obsahovat podkategorie, čímž vzniká stromová struktura. Stromovou strukturu lze do tabulky uložit tak, že se ke každému záznamu o kategorii uvede odkaz na její nadřazenou kategorii. Na obrázku 3 je vidět příklad stromové struktury a jí odpovídající tabulky TreeStructure. V našem případě se údaj o nadřazené kategorii nachází ve sloupci *parent_id*.



TreeStructure

Id	Title	Parent_id
1	A	0
2	B	0
3	C	0
4	D	1
5	E	1
6	F	2
7	G	3
8	H	4
9	I	4
10	J	5
11	K	7
12	L	7

Obrázek 3 – Stromová struktura a jí odpovídající tabulka

Přímé potomky kategorie, jejíž id = X, lze získat dotazem:

```
SELECT * FROM TreeStructure WHERE Parent_id = X
```

Přímého předka kategorie, jejíž id = X, lze získat dotazem:

```
SELECT * FROM TreeStructure WHERE Id = (SELECT Parent_id FROM TreeStructure
WHERE Id = X)
```

Pro potřeby e-shopu bude pro nějakou kategorii nutné zjistit všechny její předky a všechny její potomky. V obou případech toho lze dosáhnou rekurzivním způsobem. Nějčasteji se tak děje pomocí funkce obsahující ve své definici odkaz na sebe sama. Pro potomky její obecná forma vypadá zhruba následovně:

```
FunkceNajdiPotomky(x)
{
  najdi všechny kategorie, jejichž parent_id = x;
  do a přidej id těchto kategorií;
  pro každou z těchto kategorií vykonej
  {
    do b ulož id kategorie;
    FunkceNajdiPotomky(b);
  }
  vrať a;
}
```

Proměná *a* bude obsahovat id všech potomků kategorie s id = *x*.

Pro předky lze tuto funkci obecně vyjádřit následovně:

```
FunkceNajdiPředky(x)
{
  najdi kategorii, jejíž id = parent_id kategorie, jejíž id = x
  a přidej její id do a;
  do b ulož id nalezené kategorie;
  FunkceNajdiPředky (b);
  vrať a;
}
```

Proměná *a* bude obsahovat id všech předků kategorie s id = *x*.

Konkrétní implementace těchto funkcí v prostředí MS SQL Server by mohla vypadat následovně.

Potomci:

```
CREATE FUNCTION descendants(@x int)
RETURNS @result TABLE (id int)
AS
BEGIN
 DECLARE @q int
 DECLARE @y int
 DECLARE c CURSOR STATIC FOR SELECT id FROM categories WHERE parent_id = @x
 OPEN c
 SET @q = @@CURSOR_ROWS
 WHILE @q <> 0
  BEGIN
   FETCH NEXT FROM c INTO @y
   INSERT @result SELECT @y
   INSERT @result SELECT * FROM descendants(@y)
   SET @q = @q - 1
  END
 CLOSE c
 DEALLOCATE c
 RETURN
END
```

Předci:

```
CREATE FUNCTION ascendants(@x int)
RETURNS @result TABLE (id int)
AS
BEGIN
 DECLARE @y int
 SET @y = (SELECT parent_id FROM categories WHERE id = @x)
 IF @y <> 0
 BEGIN
  INSERT @result SELECT @y
  INSERT @result SELECT * FROM ascendants(@y)
 END
RETURN
END
```

V prostředí Microsoft SQL Serveru lze k účelu vyhledávání v hierarchických strukturách využít též Common Table Expressions (CTE). Jedná se v podstatě o nějak pojmenované SQL dotazy, které mohou být, prostřednictvím svého jména, několikrát po sobě použity v po nich následujícím dotazu. Vytvářejí se pomocí klauzule `WITH`. Mohou obsahovat odkaz na sebe sama a vytvářet tak rekurzivní dotazy.

Tento CTE vrátí předky kategorie, jejíž id = X.

```
WITH Ascendants (id, parent_id)
AS
(
 SELECT id, parent_id FROM Categories WHERE id = X
 UNION ALL
 SELECT a.id, a.parent_id FROM Categories AS a INNER JOIN Ascendants AS b
 ON a.id = b.parent_id
)
SELECT id FROM Ascendants
```

Nejprve je vybrána kategorie, jejíž předci mají být nalezeni. Pomocí jejího id je z tabulky Categories, spojením se sloupcem parent_id, vyfiltrován její první předek. Pomocí id tohoto předka je z tabulky Categories, spojením se sloupcem parent_id, vyfiltrován jeho první předek. Pomocí id tohoto předka je z tabulky Categories, spojením se sloupcem parent_id, vyfiltrován jeho první předek. A tak dále dokud nenastane situace, kdy není vrácen žádný předek. Výsledky jednotlivých dotazů, jsou pospojovány pomocí klauzule `UNION ALL`.

Tento CTE vrátí potomky kategorie, jejíž id = X.

```
WITH Descendants (id)
AS
(
 SELECT id FROM Categories WHERE id = X
 UNION ALL
 SELECT a.id FROM Categories AS a INNER JOIN Descendants AS b ON
 a.parent_id = b.id
)
SELECT id FROM Descendants
```

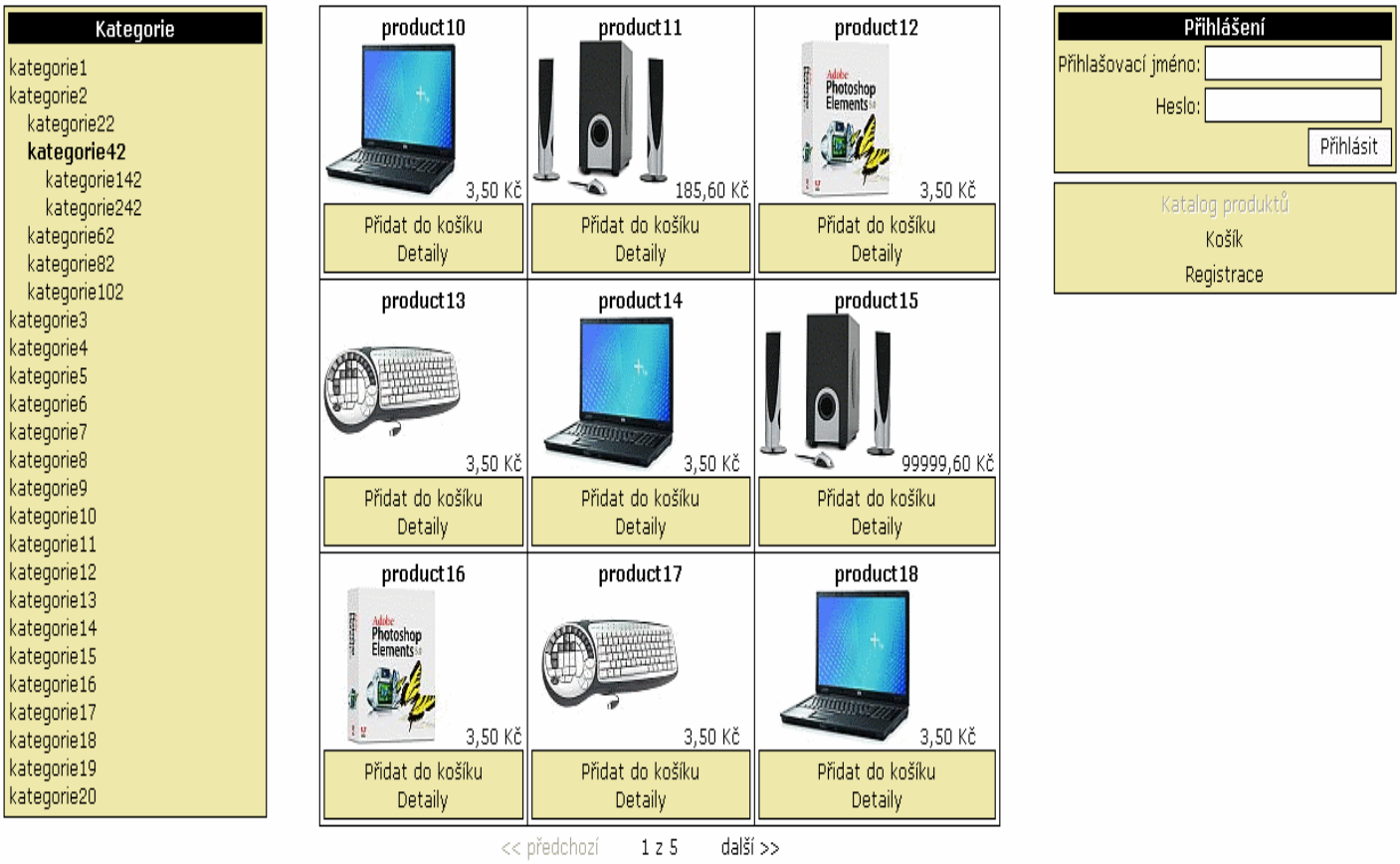
Nejprve vybrána kategorie, jejíž potomci mají být nalezeni. Pomocí jejího parent_id jsou z tabulky Categories, spojením se sloupcem id, vyfiltrováni její potomci. Pomocí parent_id těchto potomků jsou z tabulky Categories, spojením se sloupcem id, vyfiltrováni jejich potomci. Pomocí parent_id těchto potomků jsou z tabulky Categories, spojením se sloupcem

id, vyfiltrováni jejich potomci. A tak dále doku nenastane situace, kdy nejsou vráceni žádní potomci. Výsledky jednotlivých dotazů, jsou opět pospojovány pomocí klauzule `UNION ALL`.

V aplikaci je použito řešení využívající CTE.

Katalog produktů

Katalog produktů je reprezentován souborem CatBrowser.aspx. Obsahuje strom kategorií a tabulku, ve které jsou zobrazeny jednotlivé produkty. Je to zároveň titulní stránka celé aplikace, viz obrázek č. 4.



Obrázek 4 – Katalog produktů

Vytvoření stromu kategorií

ASP.NET 2.0 nabízí dva prvky, které lze využít k zobrazení stromové struktury. Jsou to prvky `TreeView` a `Menu`. Použil jsem prvek `Menu`. Na následujícím obrázku je vidět příklad deklarace tohoto prvku a jemu odpovídající výsledek, jak je zobrazen v prohlížeči.

```
<asp:Menu ID="Menu1" runat="server">
<Items>
<asp:MenuItem Text="Kategorie 1"
              NavigateUrl="CatBrowser.aspx?catid=1">
  <asp:MenuItem Text="Kategorie 1.1"
                NavigateUrl="CatBrowser.aspx?catid=2"></asp:MenuItem>
  <asp:MenuItem Text="Kategorie 1.2"
                NavigateUrl="CatBrowser.aspx?catid=3"></asp:MenuItem>
</asp:MenuItem>
<asp:MenuItem Text="Kategorie 2"
              NavigateUrl="CatBrowser.aspx?catid=4">
  <asp:MenuItem Text="Kategorie 2.1"
                NavigateUrl="CatBrowser.aspx?catid=5"></asp:MenuItem>
  <asp:MenuItem Text="Kategorie 2.2"
                NavigateUrl="CatBrowser.aspx?catid=6"></asp:MenuItem>
</asp:MenuItem>
</Items>
</asp:Menu>
```

Obrázek 5

`Menu` prvek disponuje vlastností `Items`. Je to kolekce objektů typu `MenuItem` představujících jednotlivé položky stromu. Každý `MenuItem` objekt disponuje též vlastností `Items`, která může opět obsahovat objekty typu `MenuItem`.

Celý strom je vygenerováno jako systém vnořených HTML tabulek. Text každé položky je vygenerován jako HTML odkaz, takže na něj lze kliknout. Obsah textu je specifikován ve vlastnosti `Text` objektu `MenuItem`. URL odkazu je specifikováno ve vlastnosti `NavigateUrl` objektu `MenuItem`. Jeho obecná forma vypadá následovně:

CatBrowser.aspx?catid=*CategoryId*

CategoryId je hodnota parametru catid a pro každou položku ve stromu se liší. Je to id odpovídající kategorie. Klikne-li tedy uživatel na nějaký uzel ve stromu, pošle nový HTTP požadavek na stránku CatBrowser.aspx s parametrem catid obsahujícím id vybrané kategorie.

Objekty `MenuItem` ve skutečnosti nebudou takto staticky deklarovány, ale budou vytvářeny dynamicky na základě údajů v databázi.

Při prvním načtení stránky jsou vytvořeny objekty `MenuItem` pouze pro kategorie první úrovně. Za první načtení stránky se považuje HTTP požadavek, který neobsahuje parametr catid. Kategorie první úrovně se získají následujícím SQL dotazem:

```
SELECT id, title FROM categories WHERE parent_id = 0
```

Pro každý záznam je vytvořena instance `MenuItem` a přidána do kolekce `Items` prvku `Menu`. Celý proces je vidět v níže uvedené ukázce kódu. Kvůli přehlednosti jsem vynechal kód zajišťující komunikaci s databází, a uvedl pouze samotný SQL dotaz. `FirstLevelCategories` je objekt typu `SqlDataReader`, pomocí kterého lze procházet řádky výsledku SQL dotazu. Metoda `Read` načte další řádek. Pokud žádný další řádek neexistuje, vrátí false a cyklus skončí. Pomocí metod `GetOrdinal`, `GetString` a `GetInt32` pak lze získat hodnoty v jednotlivých polích (sloupcích) řádku. V tomto případě identifikátor a název kategorie. `menuCategoriesTree` je instance třídy `Menu`.

```
SELECT id, title FROM categories WHERE parent_id = 0

while (FirstLevelCategories.Read())
{
 string CategoryTitle =
 FirstLevelCategories.GetString(FirstLevelCategories.GetOrdinal("title"));
 string CategoryId =
 FirstLevelCategories.GetInt32(FirstLevelCategories.GetOrdinal("id")).ToString();
 MenuItem MenuItem = new MenuItem();
 MenuItem.Text = CategoryTitle;
 MenuItem.NavigateUrl = "eshopurl.aspx?catid=" + CategoryId;
 menuCategoriesTree.Items.Add(MenuItem);
}
```

Pokud parametr catid existuje, má se za to, že uživatel kliknul na nějakou položku ve stromu. Jak již bylo řečeno, pararametr catid obsahuje id této vybrané kategorie. Tato kategorie pak bude ve stromu rozbalena, budou pod ní vidět její podkategorie (přímí potomci). Rozbaleny ve stromu budou ale i předci (pokud nějací jsou) vybrané kategorie (viz obrázek 4). Pro tento účel je vytvořen seznam předků (jejich id) vybrané kategorie, zahrnující i jí samotnou. Je to vlastně cesta k vybrané kategorii. Tento seznam se získá pomocí CTE Ascendants:

```
WITH Descendants (id)
AS
(
 SELECT id FROM Categories WHERE id = X
 UNION ALL
 SELECT a.id FROM Categories AS a INNER JOIN Descendants AS b ON
 a.parent_id = b.id
)
SELECT id FROM Descendants
```

Nejprve se začnou vytvářet položky pro kategorie 1. úrovně. U každé kategorie se kontroluje, jestli její id není obsaženo v seznamu předků. Pokud ano, vyberou se její přímí potomci a pro každého se začne vytvářet odpovídající položka ve stromu. U každého se opět kontroluje, není-li jeho id v seznamu předků. Pokud ano, vyberou se jeho přímí potomci a celý proces se opakuje. Je to opět rekurze.

Celý proces je vidět v následující ukázce kódu. Opět jsem pro přehlednost vynechal kód zajištující komunikaci s databází, SQL dotazy jsou uvedeny přímo. `FirstLevelCategories` a `ChildCategories` jsou opět objekty typu `SqlDataReader`, které umožňují procházet výsledkem SQL dotazů.

```
SELECT id, title FROM categories WHERE parent_id = 0

while (FirstLevelCategories.Read())
{
```

```
 string CategoryTitle = FirstLevelCategories.GetString(1);
 string CategoryId = FirstLevelCategories.GetInt32(0).ToString();
 MenuItem MenuItem = new MenuItem();
 MenuItem.Text = CategoryTitle;
 MenuItem.NavigateUrl = "eshopurl.aspx?catid=" + CategoryId;
 menuCategoriesTree.Items.Add(MenuItem);
 if (CategoryId == ClickedCategoryId)
 {
      MenuItem.Selected = true;
 }
 if (Ascendants.Contains(Convert.ToInt32(CategoryId)))
 {
      CreateCategories(CategoryId, MenuItem);
 }
}

protected void CreateCategories(string ParentCategoryId,  MenuItem
ParentMenuItem)
{
 SELECT id, title FROM categories WHERE parent_id = ParentCategoryId

 while (ChildCategories.Read())
 {
  string CategoryTitle = ChildCategories.GetString(1);
  string CategoryId = ChildCategories.GetInt32(0).ToString();
  MenuItem MenuItem = new MenuItem();
  MenuItem.Text = CategoryTitle;
  MenuItem.NavigateUrl = "eshopurl.aspx?catid=" + CategoryId;
  ParentMenuItem.ChildItems.Add(MenuItem);
  if (Ascendants.Contains(Convert.ToInt32(CategoryId)))
  {
   CreateCategories(CategoryId, MenuItem);
  }
}
```

Zobrazení obsahu kategorie

Kromě stromu kategorií je v katalogu produktů také tabulka, ve které jsou zobrazeny produkty (viz obrázek č. 4). Pokud uživatel klikne na nějakou kategorii ve stromu, zobrazí se mu nejen zboží obsažené přímo v této kategorii, ale i ve všech podkategoriích pod ní, tj. zboží všech potomků této kategorie. Kterou kategorii uživatel vybral se zjistí z parametru catid, který je součástí příchozího HTTP požadavku. Pokud tento parametr neexistuje, zobrazí se produkty, jejichž atribut *special* = *true*, tj. novinky a specialní akce.

Všichni potomci dané kategorie se vyberou pomocí rekurzivniho CTE `Descendants`[1]. Následující dotaz vrátí výsledek tohoto CTE, tj. identifikátory všech potomků dané kategorie (včetně).

```
SELECT id FROM Descendants
```

Spojením výsledku z předchozího dotazu s vazební tabulkou GoodsCategories se vyfiltrují produkty nacházející se v potomcích vybrané kategorie (včetně). Důležitá je zde direktiva `DISTINCT`, jelikož se produkty mohou nacházet v několika kategoriích najednou.

[1] viz kapitola Stromová struktura a relační databáze

```
SELECT DISTINCT product_id FROM (SELECT id FROM Descendants) AS a INNER
JOIN goodscategories AS b ON a.id = b.category_id
```

Výsledek z předchozího dotazu je spojen s tabulkou Goods, čímž se získají podrobnosti o produktech jako je název, cena a url obrázku.

```
SELECT id, title, price, small_image_url FROM (SELECT DISTINCT product_id
FROM (SELECT id FROM Descendants) AS a INNER JOIN goodscategories AS b ON
a.id = b.category_id) AS c INNER JOIN goods AS d ON c.product_id = d.id
```

Stránkování

Produkty se nebudou zobrazovat všechny najednou, ale po stránkách po devíti. Uživatel si mezi stránkami bude moci listovat. Předchozí dotaz se tedy musí opravit tak, aby vracel pouze záznamy o devíti produktech.

MS SQL Server disponuje funkcí `row_number()`, která umí očíslovat řádky ve výsledku nějakého dotazu. Její použití s předcházejícím dotazem bude vypadat následovně[2]:

```
SELECT row_number() OVER (ORDER BY id ASC) AS rownumber, id, title, price,
small_image_url FROM (SELECT DISTINCT product_id FROM (SELECT id FROM
Descendants) AS a INNER JOIN goodscategories AS b ON a.id = b.category_id)
AS c INNER JOIN goods AS d ON c.product_id = d.id
```

Výsledek tohoto dotazu bude stejný jako u předchozího, pouze přibude sloupec `rownumber`, který bude obsahovat pořadové číslo pro každý řádek. Pomocí podmínky `WHERE` pak lze z takto upraveného výsledku dotazu vybrat 9 produktů začínajících na určitém pořadovém čísle. Např. následující dotaz vybere 9 produktů s pořadovým číslem 10 až 18:

```
SELECT id, title, price, small_image_url FROM (SELECT row_number() OVER
(ORDER BY id ASC) AS rownumber, id, title, price, small_image_url FROM
(SELECT DISTINCT product_id FROM (SELECT id FROM Descendants) AS a INNER
JOIN goodscategories AS b ON a.id = b.category_id) AS c INNER JOIN goods AS
d ON c.product_id = d.id) AS e WHERE rownumber >= 10 AND rownumber < 19
```

K zobrazení výsledku je použit datový prvek `DataList`, který data zobrazí v HTML tabulce.

To, která stránka (kterých devět produktů) se má uživateli právě zobrazit, se zjistí z parametru ps (page start), který je součástí příchozího HTTP požadavku. Obsahuje pořadové číslo (údaj ve sloupci `rownumber`) prvního produktu zobrazeného na stránce. Pokud tento parametr neexistuje, je automaticky nastaven na 1. To je případ, když uživatel klikne na nějakou kategorii ve stromu, zobrazí se vždy první stránka, tj. prvních 9 produktů z vybrané kategorie.

Pokud parametr ps existuje, má se za to, že uživatel klikl na některé ze dvou tlačítek „předchozí“ nebo „další“ v dolní části stránky (viz obrázek 4), jimiž si může listovat mezi

[2] MS SQL Server pří použití funkce row_number() vyžaduje, aby řádky ve výsledku dotazu byli seřazeny. Je tomu tak proto, aby byla zajištěna vždy stejná struktura výsledku dotazu a nedošlo při opakovaném použití této funkce k označení stejného produktu rozdílným pořadovým číslem. V tomto případě jsou produkty seřazeny podle id vzestupně, takže se vlastně budou zobrazovat od nejstaršího po nejnověji vložený.

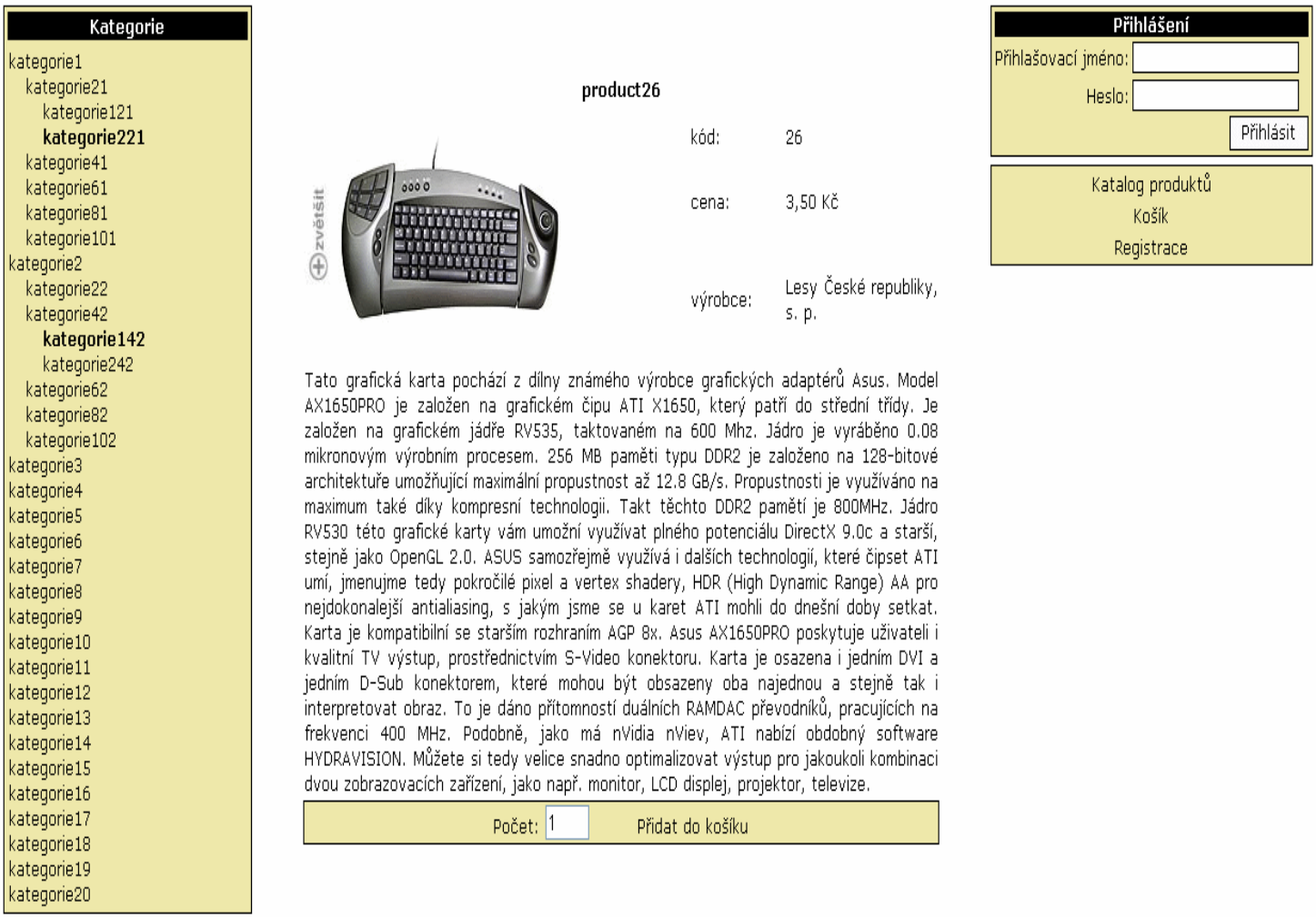
jednotlivými stránkami. Jsou to objekty `HyperLink`, jejichž atribut `NavigateUrl` má následující obecný formát[3]:

categoriesbrowser.aspx?catid=*CategoryId*&ps=*PageStart*

CategoryId je hodnota parametru catid, což je identifikátor právě zobrazované kategorie, pro oba dva odkazy je tato hodnota stejná. *PageStart* je hodnota parametru ps, což je pořadové číslo (údaj ve sloupci `rownumber`) prvního produktu zobrazeného na další nebo předcházející stránce, podle toho o který s odkazů se jedná.

Zobrazení detailů o produktu

Klikne – li uživatel v katalogu produktů (CategoriesBrowser.aspx) na odkaz „Detaily“ u nějakého produktu, je přesměrován na stránku ProductDetail.aspx, kde se mu k tomuto produktu zobrazí detailní informace. Podrobný popis, větší obrázek, výrobce atd. plus tlačítko pro přidání do košíku a textové pole pro zadání množství. Na stránce se opět nachází strom kategorií, kde jsou zvýrazněny všechny kategorie, ve kterých se produkt nachází (viz obrázek č. 6)



Obrázek 6

[3] Objekty `HyperLink` jsou vyrenderovány jako HTML element <a>. Vlastnost `NavigateUrl` odpovídá atributu href.

V HTTP požadavku musí být přítomen parametr *id*, což je identifikátor daného produktu, podle kterého se vyhledávájí v databázi detailní informace. Pokud tento parametr neexistuje, zobrazí se chybová hláška. Potřebné detailní informace se získají následující dotazem:

```
SELECT * FROM goods WHERE id = id
```

Výsledek dotazu je zobrazen pomocí datového prvku `FormView`, který data zobrazí v HTML tabulce.

Vytvoření stromu kategorií je v tomto případě o něco komplikovanější než v katalogu produktů.

Nejprve je vytvořen na zakladě níže uvedeného dotazu seznam všech kategorií (jejich id), ve kterých je daný produkt zařazen.

```
SELECT category_id FROM goodscategories WHERE product_id = id
```

Potom je vytvořen společný seznam předků (opět jejich id) kategorií získaných v předchozím dotazu a to pomocí cyklu, kde pro každou kategorii z předešlého dotazu je proveden následující dotaz, který vrací její předky (včetně).

```
WITH Ascendants (id, parent_id)
AS
(
SELECT id, parent_id FROM categories WHERE id = X
UNION ALL
SELECT a.id, a.parent_id FROM categories AS a INNER JOIN Ascendants AS b ON
a.id = b.parent_id
)
SELECT id FROM Ascendants
```

Postup vytvoření stromu je pak podobný jako v katalogu produktů. U každé kategorie se kontroluje, jestli není v jednom ze svou seznamů. Pokud je uvedena v seznamu předků, tak se rekurzivně začnou vytvářet položky pro její podkategorie (přímé potomky). Pokud je uvedena v seznamu kategorií ve kterých se nachází produkt, tak je její položka ve stromu zvýrazněna. Celý proces je vidět v následující ukázce kódu. `ProductCategories` je seznam všech kategorií (jejich id), ve kterých je daný produkt zařazen. `Ascendants` je seznam předků.

```
SELECT id, title FROM categories WHERE parent_id = 0

while (Reader.Read())
{
 string CategoryTitle = Reader.GetString(1);
 string CategoryId = Reader.GetInt32(0).ToString();
 MenuItem MenuItem = new MenuItem();
 MenuItem.Text = CategoryTitle;
 MenuItem.NavigateUrl = "eshopurl.aspx?catid=" + CategoryId;
 menuCategoriesTree.Items.Add(MenuItem);
 if (Ascendants.Contains(Convert.ToInt32(CategoryId)))
  {
   if (ProductCategories.Contains(Convert.ToInt32(CategoryId)))
    {
     MenuItem.Text = "<b>" + CategoryTitle + "</b>";
```

```
    }
   else
    {
     CreateCategories(CategoryId, MenuItem);
    }
   }
  }

protected void CreateCategories(string ParentCategoryId, MenuItem
ParentMenuItem)
{
 SELECT id, title FROM categories WHERE parent_id = ParentCategoryId

 while (ChildCategories.Read())
 {
  string CategoryTitle = ChildCategories.GetString(1);
  string CategoryId = ChildCategories.GetInt32(0).ToString();
  MenuItem MenuItem = new MenuItem();
  MenuItem.Text = CategoryTitle;
  MenuItem.NavigateUrl = "eshopurl.aspx?catid=" + CategoryId";
  ParentMenuItem.ChildItems.Add(MenuItem);
  if (Ascendants.Contains(Convert.ToInt32(CategoryId)))
  {
   if (ProductCategories.Contains(Convert.ToInt32(CategoryId)))
   {
    MenuItem.Text = "<b>" + CategoryTitle + "</b>";
   }
   else
   {
    CreateCategories(CategoryId, MenuItem);
   }
  }
 }
}
```

Pozn.
Program nepředpokládá, že produkt, umístěný v nějaké kategorii, by byl umístěn zároveň v některém z jejích předků nebo potomků. Je to zbytečné, protože pokud zákazník klikne na nějakou kategorii, tak se mu zobrazí obsah všech jejích potomků, takže produkt je vidět ve všech předcích své kategorie. Kategorie, ve kterých se produkt nachází, se zjistí z tabulky GoodsCategories. Vztah těchto kategorií by tedy neměl být předek – potomek. Nicméně pokud se tak stane a v databázi je uveden produkt v kategoriích, jejichž vztah je předek – potomek, potom je ve stromu, na stránce ProductDetail.aspx, zvýrazněn pouze předek.

Uživatelé

ASP.NET 2.0 nabízí funkčnost Membership. Je to systém tříd, obsahujících kód, který umožňuje vytváření (registraci), správu a ověřování (autentizaci) uživatelů webové aplikace. Základem je tzv. provider, což je potomek abstraktní třídy `MembershipProvider`, který zajiťuje veškerou práci s databází. Provider je dále využíván třídami `Membership` a `MembershipUser`, které poskytují API pro správu uživatelů, tj. disponují metodami a vlastnostmi, jejichž prostřednictvím lze vytvářet, mazat, zablokovat uživatele, změnit heslo, e-mail, zjistit čas posledního přihlášení uživatele atd.[4].

[4] Tyto metody ale obsahuje už i třída MembershipProvider a lze je použít na instanci jejího potomka

Výše uvedené třídy jsou dále využívány třídami `CreateUserWizard`, `Login`, `LoginName` a `LoginView` (a dalšími) což jsou serverové prvky, s jejichž pomocí lze rychle vytvořit uživatelské rozhraní.

Objekt `CreateUserWizard` obsahuje definici registračního formuláře, skládající se z HTML elementů a serverových prvků. Pokud formulář obsahuje textová pole pro přihlašovací jméno (`TextBox` s id *UserName*) a heslo (`TextBox` s id *Password*), tak po stisknutí tlačítka jsou tyto údaje předány providerovi, který zajistí uložení údajů v databázi.
Provider umí též pracovat s údaji email, a kontrolní otázka a odpověď v případě zapomenutí hesla. Pole pro tyto údaje jsou nepovinná. Pokud ale ve formuláři jsou, provider je uloží.

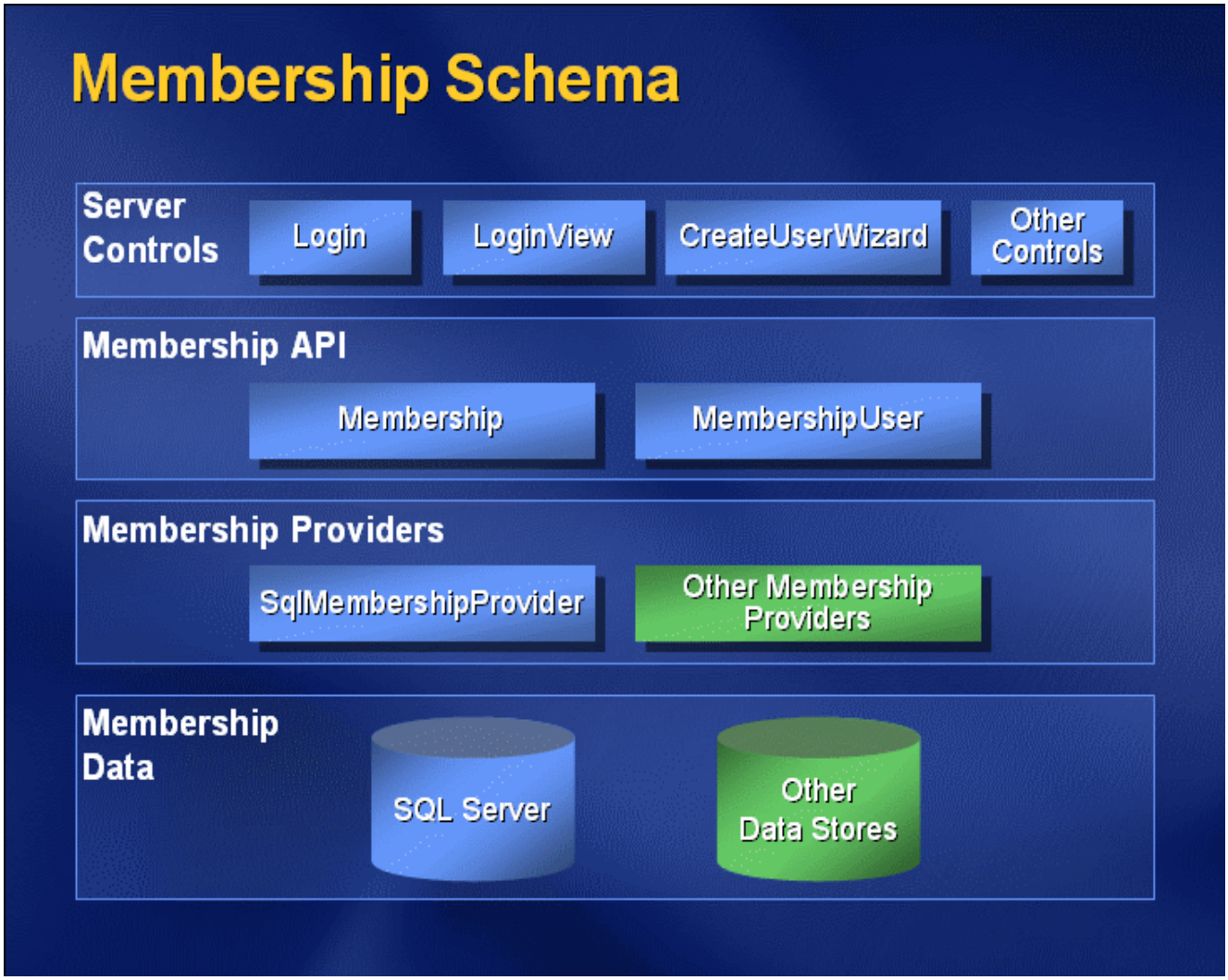
Do formuláře lze samozřejmně zahrnout i pole pro další údaje, vývojař je ale pak musí zpracovat svým vlastním kódem.

Objekt `Login` obsahuje definici přihlašovcího formuláře, která pokud obsahuje textová pole pro přihlašovací jméno (`TextBox` s id *UserName*) a pro heslo (`TextBox` s id *Password*), tak po stisknutí tlačítka jsou tyto údaje předány providerovi, který provede porovnání s obsahem databáze. Prvek může také obsahovat checkBox s id RememberMe,

Objekt `LoginName` je vyrenderován jako HTML element obsahující přihlašovací jméno uživatele.

Objekt `LoginView` obsahuje dvěma vlastnostmi: `AnonymousTemplate` a `LoggedInTemplate`. `AnonymousTemplate` obsahuje markup, který je vyrenderován pro anonymní zákazníky. `LoggedInTemplate` obsahuje markup, který je vyrenderován pouze přihlášeným zákazníkům. Příklad užití tohoto objektu je vidět v kapitole Vytvoření objednávky

Celý systém je přehledně znázorněn na obrázku č. 7.



Obrázek 7 – Membership (převzato z [16])

Jak je vidět, tak implementací providera může být více. Liší se datovým zdrojem, který používají, a implementací vlastností a metod abstraktní třídy `MembershipProvider`. Pokud vyvojář chce, aby jeho provider mohli používat i ostatní Membership třídy, musí svého providera implementovat jako potomka třídy `MembershipProvider`.

ASP.NET již obsahuje jednu takovouto implementaci třídy `MembershipProvider`. Je to třida `SqlMembershipProvider`. Provider tohoto typu spolupracuje s MS SQL Serverem. Obsah databáze, kterou bude používat se vytvoří pomocí utility Aspnet_regsql.exe. Po spuštění a specifikaci cílové dadabáze jsou vytvořeny všechny potřebné tabulky[5] a uložené procedury.

Providera, kterého budou ostatní vrstvy „nad ním“ používat, je potřeba specifikovat a nastavit v souboru Web.Config:

```
<membership>
<providers>
<clear/>
<add
 name="AspNetSqlMembershipProvider" – jméno instance providera
```

[5] Jedná se (mimo jiné) o tabulky aspnet_Users, asp_Membership a aspnet_Profiles, viz kapitola Struktura databáze.

```
 type="System.Web.Security.SqlMembershipProvider" - typ providera
 connectionStringName="DatabaseConnection" - řetězec pro připojení k
                                             databázi[6]
 enablePasswordRetrieval="false" - uživatelé nemůžou získat zpět své heslo
 enablePasswordReset="true" - heslo lze změnit
 requiresQuestionAndAnswer="false" - nevyžadovat kontrolní otázku a odpověď
                                     pro případ zapomenutého hesla
 requiresUniqueEmail="false" - nevyžadovat unikátní e-mailovou adresu pro
                               každého uživatele
 passwordFormat="Hashed" - heslo je uloženo „zahešované"
 maxInvalidPasswordAttempts="3" - max. počet chybných pokusů při zadávání
                                  hesla
 minRequiredPasswordLength="5" - min. délka hesla je 5 znaků
 minRequiredNonalphanumericCharacters="0" - heslo nemusí obsahovat
 passwordAttemptWindow="10"
 passwordStrengthRegularExpression="" />
</providers>
</membership>
```

Výhody používání Membership

Za předpokladu, že si vývojář nemusí napsat svůj provider (tj. používá `SqlMembershipProvider` nebo jiný typ, který před ním někdo vytvořil), tak mu to ušetří spoustu práce. Hlavní výhoda je v možnosti využít prvky `Login`, `LoginName`, `LoginView` a `CreateUserWizard`, které, ve spolupráci s ostatními Membership třídami, zajišťují:

Login – kompletní autentizaci uživatele při přihlašování (v souboru Web.Config lze nastavit maximální počet chybných zadání hesla, účet je příoadně automaticky zablokován)
LoginName – zobrazení uživatelského jména
LoginView – zobrazení někerých prvků stránky pouze přihlášeným živatelům.
CreateUserWizard – vytvoření nového uživatele v databázi (automatická kontrola duplicitního přihlašovacího jména, případně e-mailu; v souboru Web.Config lze nastavit formát hesla a způsob jeho uložení v databázi)

To vše bez jediného řadku kódu, stačí prvky umístit na markup .aspx stránky.

Pokud si vývojář musí vytvořit svého vlastního providera (např. musí použít jiný databázový server než MS SQL Server), tak si sice neušetří moc práce s psaním kódu, ale přináší přináší to s sebou standartní výhody OOP. Providera lze jednoduše používat na více místech v aplikaci i v jiných aplikacích, kód je přehlednější a srozunitelnější, chyby se opravují pouze na jednom místě atd.

Jsou dva druhy uživatelů, kteří se mohou vyskytnout v obchodě: **Anonymní** a **Přihlášení**. Anonymní mohou přidávat zboží do košíku, nemohou ale vytvořit objednávku.

[6] Tento řetězec je uložen na jiném místě v souboru Web.Config, v sekci connectionStrings pod jménem DatabaseConnection, viz kapitola Konfigurace aplikace.

Anonymní uživatelé

Anonymní uživatelé mohou přidávat zboží do košíku, z toho důvodu je potřeba pro ně zajistit jejich identifikaci, aby bylo možno ke každému z nich přiřadit jemu odpovídající košík.

Anonymní identifikace je novinka v ASP.NET 2.0. Způsob jakým se bude realizovat se nastavuje v souboru Web.Config prostřednictvím elementu `<anonymousIdentification>`. Já jsem použil následující nastavení:

```
<anonymousIdentification
  enabled="true" - zapnutí celé funkčnosi anonymní identifikace
  cookieless="UseCookies" - k uchování identifikátoru jsou použity cookies
  cookieName=".ASPXANONYMOUS" - název cookie s identifikátorem
  cookieTimeout="100000" - doba platnosti cookie
  cookiePath="/"
  cookieRequireSSL="false"
  cookieSlidingExpiration="true"
  cookieProtection="Validation" - způsob ochrany cookie
  domain="" />
```

S tímto nastavením bude pro každého nově příchozího zákazníka (tj. s každou novou session) vytvořena cookie s názvem .ASPXANONYMOUS, ve které je uložen jeho identifikátor (GUID). Každý další požadavek pak lze přiřadit konkrétnímu zákazníkovi a provést pak případné změny v databázi a správném místě.

Pro anonymního zákazníka se vytvoří záznam v databázi, pouze když si přidá produkt do košíku. V tabulce aspnet_Users je vytvořen nový záznam, UserId = GUID, IsAnonymous = true. Zároveň v tabulce aspnet_Profile je uložen objekt `ShoppingCart` daného zákazníka[7].

Identifikátor anonymního uživatele se ukládá defaultně jako persistentní cookie, tj. po uzavření prohlížeče cookie zůstává v klientově počítači. Nevýhoda toho je, že pokud uživatel A navštíví obchod, dá věci do košíku, a pak zavře prohlížeč a po něm přijde uživatel B (kt. už v obchodě předtím nakupoval) a přihlásí se, obsah jeho košíku se nastaví na obsah po uživateli A.

Neexistuje možnost, jak explicitně nastavit, aby se pro anonymní identifikátor vytvářela nepersitentní cookie. Lze toho ale dosáhnout následujícím kódem, který je vázán na událost `PreRender` stránky. Musí být umístěn v každé stránce přístupné anonymním klientům:

```
if (Response.Cookies.Count > 1)
{
 HttpCookie c = FormsAuthentication.GetAuthCookie(".ASPXANONYMOUS", false);
 Response.Cookies[".ASPXANONYMOUS"].Expires = c.Expires;
}
```

Pomocí metody `FormsAuthentication.GetAuthCookie` je vytvořena nepersistentí cookie a její vlastností `Expires` je přepsána ta samá vlastnost již vytvořené perzistentní .ASPXANONYMOUS cookie, čímž se z ní stane cookie neperzistentní.

[7] podobněji viz kapitola Košík

Přihlášení uživatelé

Uživatel se stane přihlášeným pokud při přihlašování úspěšně projde autentizací. U přihlášených uživatelů je potřeba, stejně jako u anonymních, provádět identifikaci jejich HTTP požadavků, aby pak bylo možné provést případné změny v databázi na správném místě. Jakým způsobem se tak bude dít se nastaví opět v souboru Web.Config:

```
<configuration>
 <appSettings/>
  <connectionStrings/>
   <system.web>
    <authentication mode="Forms">
     <forms
      cookieless="UseCookies" - k uchování identifikátoru jsou použity
                                cookies
      name=".ASPXAUTH" - název cookie s identifikátorem
      loginUrl="Login.aspx"> - stránka, na kterou je uživatel přesměrován,
                               pokud cookie neexistuje, ale je vyžadována[8]
     </forms>
  </authentication>
 </system.web>
</configuration>
```

Po kladném ověření přihlašovacích údajů, je vytvořena cookie .ASPXAUTH obsahující identifikátor. Každý požadavek pak lze jednoznačně přiřadit konrétnímu zákazníkovi.

Aby se uživatel mohl přihlásit musí nejprve projít registrací.

Registrace uživatelů

Zákazníci se mohou zaregistrovat na stránce Register.aspx. Registrační formulář je vytvořen pomocí prvku `CreateUserWizard`. Tento prvek zajistí, ve spolupráci s ostatními Membership třídami, vytvoření nového uživatele v databázi, konkrétně jeho přihlašovacího jména, hesla a e-mailu. Ostatní údaje je třeba uložit „klasickým způsobem“.

Registrační formulář obsahuje pole pro vyplnění těchto položek:

Přihlašovací jméno
Heslo
Heslo znovu
E-mail
Jméno
Příjmení
Ulice
Obec
PSČ

Ke každému poli, které vytvořeno pomocí prvku `TextBox`, je přiřazen `RequiredFieldValidator`, který kontroluje jestli pole bylo vyplněno. Všechna pole jsou povinná.

[8] viz kapitola Autorizace

Shoda údajů v polích Heslo a Heslo znovu je zajištena prvkem `CompareValidator`. Kontrolu duplicitních přihlašovacích jmen a kontrolu správného formátu hesla (minimálně 5 znaků) zajišťuje systém Mebership.

Přihlašovací jméno, Heslo a E-mail zpracovává automaticky systém Membership. Přihlašovací jméno je uloženo v tabulce aspnet_Users. Heslo a E-mail jsou uloženy do tabulky aspnet_Membership.

Ostatní údaje jsou uloženy „klasickým způsobem“ do tabulky UsersDetails. Stane se tak prostřednictvím níže uvedeného SQL příkazu.

```
INSERT INTO UsersDetails (UserId, FirstName, LastName, Street, City, ZipCode) VALUES (@UserId, @FirstName, @LastName, @Street, @City, @ZipCode)
```

Místa vyznačená kurzívou jsou parametry, jejichž konkrétní hodnoty jsou načteny z uživatelem vyplněného registračního formuláře. Vyjímkou je parametr *@UserId*, což je cizí klíč (v tabulce UsersDetails) odkazující na providerem nově vytvořeného užívatele (do tabulky aspnet_Users) a jeho hodnota se získá následujícím kódem:

```
object UserId = Membership.GetUser(UserName).ProviderUserKey;
```

Příkaz je poslán na databázový server při události `CreatedUser` prvku `CreateUserWizard`, která nastane bezprostředně po tom, co `SqlMembershipProvider` úspěšně vytvořil nového uživatele.

Přihlašování

Zákazníci, kteří prošli registrací se mohou příhlásit. Identifikace zákazníka probíhá na základě jeho přihlašovacího jména a autentizace na základě hesla. Přihlásit se lze na kterékoliv ze stránek přístupné anonymním uživatelům, protože každá obsahuje k tomu určený formulář (je součástí MasterPage.master[9]). Ten je součástí prvku `Login`. Obsahuje textová pole (`TextBox`) pro zadávání uživatelského jména a hesla. Po stisknutí tlačítka „Přihlásit“ systém Membership zajistí oveření zadaných údajů a, v případě kladného výsledku, je vytvořena identifikační cookie s názvem .ASPXAUTH. Ta obsahuje identifikátor podle kterého lze všechny následující akce (HTTP požadavky) přiřadit konkrétnímu zákazníkovi a provést pak případné změny v databázi na správném místě.

Registrační formulář neobsahuje prvek `CheckBox` s ID = RememberMe, takže cookie .ASPXAUTH je vždy nepersitentní, tj. po uzavření prohlížeče nezůstává uložena v klientově počítači.

Autorizace

Stránky, které jsou přístupné pouze přihlášeným uživatelům jsou umístěny v podadresáři NotForAnon. Jedná se o stránky:

CreateOrder.aspx
DisplayOrders.aspx

[9] Viz kapitola Jednotný layout aplikace

OrderDetails.aspx
Profile.aspx

Tento adresář má svůj vlastní Web.Config soubor[10], který přepisuje ten, který je umístěn v kořenovém adresáři aplikace. Obsahuje následující deklaraci, která zajišťuje, že anonymní uživatelé (jejichž HTTP požadavky neobsahují .ASPXAUTH cookie) budou odmítnuti.

```
<configuration>
    <appSettings/>
    <connectionStrings/>
    <system.web>
      <authorization>
        <deny users="?"/>
      </authorization>
    </system.web>
</configuration>
```

Pokud nepřihlášený uživatel pošle požadavek na některou z výše uvedených stránek, je automaticky přesměrován na stránku Login.aspx[11].

Košík

Košík je vytvořen pomocí funkčnosti Profiles. Tato funkčnost doplňuje Membership tím, že umožňuje uživatelům webové aplikace nadefinovat, vedle standartních vlastností (přihlašovací jméno, heslo, e-mail), které jsou uloženy v tabulkách aspnet_Users a aspnet_Membership, i další vlastnosti, které se ukládájí do tabulky aspnet_Profile. Práci s databází má na starosti opět provider, tentokrát potomek abstraktní třídy `ProfileProvider`.

Vlastnosti uživatelů je třeba specifikovat v souboru Web.Config. Např. definice vlastnosti ShoppongCart (nákupní košík) vypadá takto: web.config obsahuje též specifikaci providera, který je požit pro ukládání vlastností do databáze.

```
<configuration>
 <system.web>
  <profile>
   <providers>
    <clear/>
     <add  name="AspNetSqlProfileProvider" - jméno instance providera
           connectionStringName="DatabaseConnection" - řetězec pro připojení
                                                       k databázi[12]
         type="System.Web.Profile.SqlProfileProvider" /> - typ providera
   </providers>
   <properties>
    <add name="ShoppingCart"          - název vlastnosti
         type="ShoppingCart"          - typ vlastnosti
         serializeAs="Binary"         - způsob uložení v databázi
         allowAnonymous="true" />     - vlastnost mají i anonymní uživatelé
   </properties>
  </profile>
 </system.web>
```

[10] viz kapitola Struktura aplikace
[11] viz začátek kapitoly Přihlášení uživatelé
[12] Tento řetězec je uložen v sekci connectionStrings v souboru Web.Config pod jménem DatabaseConnection

```
</configuration>
```

Provider je typu `SqlProfileProvider`. Tato třída je, podobně jako `SqlMembershipProvider`, již „hotová“ součástí v ASP.NET. Spolupracuje s MS SQL serverem, se stejnou databází jejíž obsah je vytvořen pomocí utility Aspnet_regsql.exe.

Vlastnost je typu `ShoppingCart`, jehož definice vypadá následovně:

```
public class ShoppingCart
{
  private Dictionary<string, CartItem> cartItems = new Dictionary<string,
CartItem>();

  // vsechny produkty v kosiku
  public Dictionary<string,CartItem>.ValueCollection CartItems
  {
    get { return cartItems.Values; }
  }

  // celkova cena vsech produktu v kosiku
  public decimal Total
  {
    get
    {
      decimal sum = 0;
      foreach (CartItem item in cartItems.Values)
        sum += item.Price * item.Quantity;
      return sum;
    }
  }

  // pridej do kosiku novy produkt
  public void AddItem(string id, string name, decimal price, int quantity)
  {
    if (cartItems.ContainsKey(id)) //je produkt s danym id jiz v kosiku?
    {
      cartItems[id].Quantity += quantity;
    }
    else
    {
      cartItems.Add(id, new CartItem(id, name, price, quantity));
    }
  }

  //odstran produkt z kosiku
  public void RemoveItem(string id)
  {
    CartItem item = (CartItem)cartItems[id];
    if (item == null)
    {
      return;
    }
    cartItems.Remove(id);
  }

  //zmen mnozstvi nejakeho produktu v kosiku
  public void ItemQuantityChange(string id, int quantity)
  {
      if (quantity == 0)
      {
```

```
            cartItems.Remove(id);
        }
        else
        {
            if (quantity < 0)
            {
                return;
            }
            else
            {
                CartItem item = cartItems[id];
                item.Quantity = quantity;
            }
        }
    }

    //vyprazdni kosik
    public void Empty()
    {
      cartItems.Clear();
    }

}
```

Jak je vidět, tak instance tohoto typu, která představuje uživatelův košík, disponuje vlastnostmi a metodami pro přidávaní produktů, změny množství produktu, zjištění celkové ceny apod. Důležitá je vlastnost `cartItems`, která představuje produkty nacházející se v košíku. Je to kolekce objektů typu `CartItem`.

Definice typu `CartItem` vypadá následovně:

```
public class CartItem
{
  private string id;
  private string title;
  private decimal price;
  private int quantity;

  public string Id
  {
    get { return id; }
  }

  public string Title
  {
    get { return title; }
  }

  public decimal Price
  {
    get { return price; }
  }

  public int Quantity
  {
    get { return quantity; }
    set { quantity = value; }
  }

  public CartItem(string Id, string Title, decimal Price, int Quantity)
```

```
  {
    id = Id;
    title = Title;
    price = Price;
    quantity = Quantity;
  }

}
```

Obě tyto třídy jsou uloženy v adresáři App_Code, v souborech CartItem.cs a ShoppingCart.cs, čímž jsou přístupné pro všechny stránky v aplikaci.

V kódu lze k Profile vlastnostem přistupovat skrze vlastnost `Profile` stránky[13], která obsahuje referenci na instanci třídy `ProfileCommon`. To je automaticky generovaná třída[14], která má pro každou z vlastností nadefinovaných v souboru Web.Config vytvořen selektor a modifikátor a zajišťuje skrze providera načtení nebo uložení vlastností v databázi (v tabulce aspnet_Profile). Zápisem `Profile.ShoppingCart` tedy lze přímo přistupovat k instantci třídy `ShoppingCart`, tj. ke košíku daného uživatele.

Pozn.:
Profiles by se mohli využít i pro ukládání ostatních údajů o zákaznících, které jsou uloženy v tabulce UsersDetails. Stačilo by je nadefinovat v souboru Web.Config:

```
<profile>
 <properties>
  <add name="FirstName" type="string"/>
  <add name="LastName" type="string"/>
  <add name="Street" type="string"/>
  ...
 </properties>
</profile>
```

Nevýhodou Profiles ale je, že vlastnosti jsou při ukládání do databáze serializovány, všechny jsou zaznamenávány do jednoho sloupce a nelze je tedy vyhledávat, zobrazovat a upravovat pomocí klasických SQL příkazů s podmínkou `WHERE`, případně spojovat s dalšími tabulkami.

Funkce košíku

Produkt lze vložit do košíku na stránkách ProductDetail.aspx a CatBrowser.aspx. Po kliknutí na „Přidat do košíku“ je vykonán kód, který zjistí potřené informace o produktu, který má být do košíku přidán, konkrétně id, název a cenu produktu. Děje se tak vyhledáním konkrétních hodnot v serverovém prvku `DataList`, který slouží k zobrazení produktů vybrané kategorie (a jejích podkategorií) na stránce CatBrowser.aspx nebo v severovém prvku `FormView`, který slouží k zobrazení detailních informací o produktu na stránce ProductDetail.aspx. Tyto informace jsou uloženy do proměných `id`, `title` a `price`. Potom je zavolána metoda `AddItem` objektu `ShoppingCart`:

```
Profile.ShoppingCart.AddItem(id, title, price);
```

[13] Je to vlastnost vždy konkrétní stránky, jakožto potomka třídy `Page`. Třída `Page` tuto vlastnost neobsahuje.
[14] Lze jí nalézt v adresáři Temporary ASP.NET Files

Metoda `AddItem` vytvoří instanci typu `CartItem` a zařadí jí do kolekce `cartItems` košíku. Uložení celého košíku (instance třídy `ShoppingCart`) do databáze do tabulky aspnet_Profile provede automaticky provider.

Na stránce Cart.aspx, která zobrazuje obsah košíku, může zákazník měnit množství jednotlivých produktů v košíku (metoda `ItemQuantityChange`), případně je úplně z košíku odstranit (metoda `RemoveItem`) nebo vyprázdnit úplně celý košík (metoda `Empty`).

Košík je vytvořen pomocí datového prvku `Repeater`, který zobrazí košík jako HTML tabulku. Jako datový zdroj tohoto prvku je použita vlastnost `CartItems` objektu `ShoppingCart` (`repCart` je jméno instance `Repeater`).

```
repCart.DataSource = Profile.ShoppingCart.CartItems;
repCart.DataBind();
```

Přenesení obsahu košíku anonymního uživatele do košíku přihlášeného uživatele

Pokud anonymní (ale zaregistrovaný) uživatel, naplní svůj košík a potom se přihlásí, je potřeba zajistit, aby se obsah jeho anonymního košíku přemístil do košíku, který požívá jako přihlášený uživatel (každý z těchto dvou košíků je veden v tabulce aspnet_Profile jako jiný záznam).

Stane se tak při události `MigrateAnonymous` třídy `ProfileModule`. Událost `MigrateAnonymous` nastane, když se anonymní uživatel přihlásí. Je to globální událost, která může teoreticky nastat na jakékoliv stránce v aplikaci, takže její obsluha je v souboru Global.asax. Kód, který provede celou operaci je uveden níže.

```
ProfileCommon anonProfile = Profile.GetProfile(e.AnonymousID);
//pokud anonymni kosik neco obsahuje, tak se bude migrovat, jinak ne
if (anonProfile.ShoppingCart.CartItems.Count > 0)
{
   Profile.ShoppingCart = anonProfile.ShoppingCart;
}
//vymaze anonymni profil (tj. kosik) z tabulky aspnet_Profile
ProfileManager.DeleteProfile(e.AnonymousID);
//vymaze anonymni cookie
AnonymousIdentificationModule.ClearAnonymousIdentifier();
//vymaze anonymniho uzivatele z tabulky aspnet_Users
Membership.DeleteUser(e.AnonymousID, true);
```

Objednávky

Vytvoření objednávky

Pokud je zákazník přihlášen, může vytvářet objednávky. Stačí naplnit košík a na stránce Cart.aspx, kliknout na odkaz „Vytvořit objednávku“. Zákazník je přesměrován na stránku CreateOrder.aspx, která obsahuje kód, který objednávku vytvoří.

Nejprve se zkontroluje, jestli košík zákazníka vůbec obsahuje nějaké produkty, aby bylo z čeho objednávku vytvářet. Počet produktů v košíku se zjistí skrze vlastnost `CartItems` objektu `ShoppingCart`, která obsahuje kolekci objektů `CartItem`. Tato kolekce[15] disponuje vlastností `Count`, která vrací počet `CartItem` objektů v ní obsažených.

```
int CartItemsNumber = Profile.ShoppingCart.CartItems.Count
```

Pokud je tedy počet produktů v košíku větší než 0, tak je následujícími SQL příkazy na databazovém serveru spustěna uložená procedura, která vytvoří záznam o nové objednávce v tabulce Orders a vrátí identifikátor této obejdnávky.

```
DECLARE @NewOrderId int
EXEC CreateOrder  @UserId, @NewOrderId OUT
SELECT @NewOrderId
```

Parametr *`@UserId`* je identifikátor zákazníka, hodnota ve sloupci UserId v tabulce aspnet_Users. Získá se prostřednictvím třídy `Membership`:

```
object UserId = Membership.GetUser().ProviderUserKey;
```

Statická metoda `GetUser()` vrací objekt reprezentující zákazníka, který poslal požadavek na stánku. Vlastnost `ProviderUseKey` tohoto objektu odsahuje identifikátor (UserId z tabulky aspnet_Users) zákazníka.

Definice uložené procedury:

```
CREATE PROCEDURE CreateOrder
@UserId uniqueidentifier,
@NewOrderId int OUT
AS
INSERT INTO Orders (UserId, CreateDate, State) VALUES (@UserId, GETDATE(),
'zpracovává se')
SET @NewOrderId  = @@IDENTITY
```

Potom je pro každý produkt v košíku zákazníka proveden níže uvedený SQL příkaz, který vytoří záznam(y) v tabulce OrdersGoods. Produkty v košíku jsou reprezentovány vlastností `CartItems` objektu `ShoppingCart`, která obsahuje kolekci objektů `CartItem`. Touto kolekcí lze procházet pomocí cyklu `foreach`.

```
INSERT INTO OrdersGoods (OrderId, ProductId, ProductQuantity) VALUES
(@NewOrderId, @ProductId, @ProductQuantity)
```

[15] objekt typu `Dictionary<string, CartItem>.ValueCollection`

Parametr *@NewOrderId* je identifikátor nově vytvořené objednávky v tabulce Orders.

Hodnoty paramterů *@ProductId* a *@ProductQuantity* se získají z vlastností `Id` a `Quantity` objektu `CartItem`, který představuje produkt v košíku.

Všechny výše uvedené SQL příkazy jsou provedeny jako transakce. K tomuto účelu slouží třída `SqlTransaction`. Změny jsou v databázi trvale provedeny až po zavolání metody `Commit`.

Objednávky mohou vytvářet pouze přihlášení zákazníci, takže odkaz „Vytvořit objednávku“ by se neměl zobrazovat anonymním zákazníkům. Toho lze docílit využitím objektu `LoginView`.

```
<asp:LoginView ID="LoginView1" runat="server">
<LoggedInTemplate>
  <asp:HyperLink ID="hypCreateOrder"
                 runat="server"
                 NavigateUrl="~/NotForAnon/CreateOrder.aspx"
                 Text="Vytvořit objednávku">
  </asp:HyperLink>
</LoggedInTemplate>
<AnonymousTemplate>
  <asp:Label runat="server"
             Text="Vytvářet objednávky můžete pouze pokud jste přihlášen.">
  </asp:Label>
</AnonymousTemplate>
</asp:LoginView>
```

Zobrazení objednávek

Na stránce DisplayOrders.aspx si zákazník může zobrazit přehled všech svých objednávek. V databázi je proveden následující SQL dotaz, který vrátí potřebné údaje o všech objednávkách zákazníka:

```
SELECT a.OrderId, a.CreateDate, SUM(ProductPrice*ProductQuantity) AS Total,
a.State   FROM Orders AS a INNER JOIN OrdersGoods AS b ON a.OrderId =
b.OrderId  WHERE  a.UserId  =  @UserId  GROUP  BY  a.UserId,  a.OrderId,
a.CreateDate, a.State
```

Parametr *@UserId* je identifikátor zákazníka a získá se prostřednictvím třídy `Membership`:

```
object UserId = Membership.GetUser().ProviderUserKey;
```

Výsledek dotazu je zobrazen datovým prvkem `GridView`.

Detaily objednávky

Detaily konkrétní objednávky si zákázník může zobrazit na stránce OrderDetails.aspx, na kterou se lze dostat ze stránky DisplayOrders.aspx kliknutím na číslo objednávky. Detaily se získají spojením tabulek Orders a OrdersGoods níže uvedeným SQL dotazem, který má dva parametry:

@UserId – identifikátor uživatele

@OrderId – identifikátor objednávky, který přijde jako parametr HTTP požadavku

```
SELECT Orders.OrderId, Orders.UserId, OrdersGoods.ProductId,
OrdersGoods.ProductTitle, OrdersGoods.ProductQuantity,
OrdersGoods.ProductPrice, FROM Orders AS Orders INNER JOIN OrdersGoods AS
OrdersGoods ON Orders.OrderId = OrdersGoods.OrderId WHERE Orders.UserId =
@UserId AND Orders.OrderId = @OrderId
```

Parametr *@UserId* se získá pomocí třídy `Membership`:

```
object UserId = Membership.GetUser().ProviderUserKey;
```

Na objednávce je také zobrazeno datum, kdy byla vytvořena a její celková cena. Tyto údaje se získají následujícím SQL dotazem:

```
SELECT CreateDate, SUM(ProductPrice*ProductQuantity) AS Total, State FROM
Orders AS a INNER JOIN OrdersGoods AS b ON a.OrderId = b.OrderId WHERE
a.UserId = @UserId AND a.OrderId = @OrderId GROUP BY CreateDate, State
```

Na základě údaje ve sloupci `State` se rozhodne, jestli bude zobrazen odkaz na zrušení objednávky. Objednávku lze zrušit jen když se nachází ve stavu „zpracovává se“.

K zobrazení výsledku obou dotazů je použit datový prvek typu `Repeater`, který zobrazí data v HTML tabulce.

Zrušení objednávky

Zobrazenou objednávku (stránka OrderDetails.aspx) lze zrušit kliknutím na tlačítko „Zrušit objednávku“. Zákazník je potom přesměrován na stránku OrderDelete.aspx obsahující kód, který objednávku vymaže z databáze. V databázi je zavolána uložená procedura `DeleteOrder`, která nejprve zkontroluje, zda daná objednávka již není vyřízena a pokud není, tak se vykonají SQL příkazy, které vymažou odpovídající záznamy v tabulkách Ordersproducts a Orders. Príkaz pro spuštění procedury vypadá následovně:

```
DECLARE @Result bit
EXEC DeleteOrder @OrderId, @UserId, @Result OUT
SELECT @Result AS Result
```

Parametr *@OrderId* je identifikátor objednávky, jeho hodnota příjde jako parametr HTTP požadavku.

Parametr *@UserId* (identifikátor uživatele) se získá opět prostřednictvím třídy `Membership`.

Proměnná `@Result` obsahuje výsledek celé operace, tj. jestli objednávka byla smázána (1) nebo nebyla (0).

Definice procedury:

```
CREATE PROC DeleteOrder
@OrderId int,
@UserId uniqueidentifier,
@Result bit OUT
AS
```

```
BEGIN
DECLARE @IsCarriedOut nvarchar(13)
SET @IsCarriedOut = (SELECT State FROM Orders WHERE OrderId = @OrderId AND
UserId = @UserId)
IF (@IsCarriedOut = 'zpracovává se')
 BEGIN
 DELETE OrdersGoods WHERE OrderId = @OrderId
 DELETE Orders WHERE OrderId = @OrderId AND UserId = @UserId
 SET @Result = 1
 END
ELSE
 BEGIN
 SET @Result = 0
 END
END
```

Profil zákazníka

Zákazník si může zobrazit a editovat svůj profil na stránce Profile.aspx. Údaje o zákazníkovi jsou (kromě e-mailu) uloženy v tabulce UsersDetails a zjistí se následujím SQL dotazem:

```
SELECT * FROM UsersDetails WHERE UserId = @UserId
```

Parametr *@UserId* (identifikátor uživatele) se opět získá skrze třídu `Membership`:

```
object UserId = Membership.GetUser().ProviderUserKey;
```

E-mail zákazníka je zprácováván systémem Membership. Je uložen v tabulce aspnet_Membership a získá se, podobně jako identifikátor zákazníka, prostřednictvím třídy `Membership` a její metody `GetUser()`:

```
string Email = Membership.GetUser().Email;
```

Jednotlivé údaje jsou zobrazeny v textových polích (`TextBox`), takže uživatel je může editovat. Po kliknutí na odkaz „Uložit změny“, se provede následují SLQ příkaz, který uloží změny v tabulce UsersDetails.

```
UPDATE UsersDetails SET FirstName = @FirstName, LastName = @LastName,
Street = @Street, City = @City, ZipCode = @ZipCode WHERE UserId = @UserId
```

E-mail zákazníka se zaktualizuje prostřednictvím třídy `Memebership` následovně:

```
MembershipUser CurrentUser = Membership.GetUser();
CurrentUser.Email = Email;
Membership.UpdateUser(CurrentUser);
```

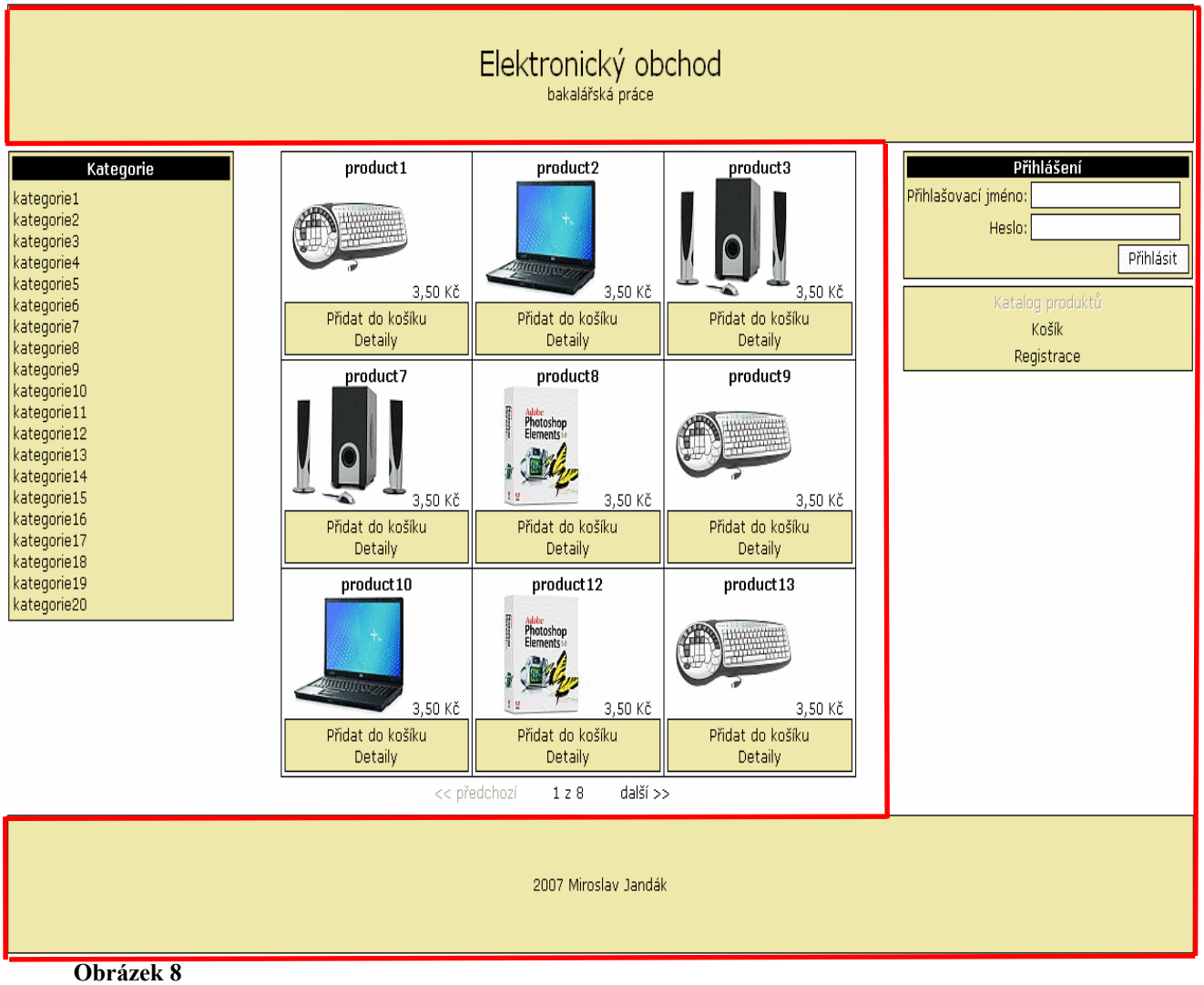
Jednotný layout aplikace

Jednotný vzhled všech stránek je zajištěn využitím tzv. Master Pages, což je nová vlastnost ASP.NET 2.0 umožňující vytvoření jedné obecné šablony, jejíž obsah se pak v konkrétních stránkách mění pouze v určených místech.

Tuto šablonu představuje soubor MasterPage.master, který obsahuje markup (HTML elementy a ASP.NET serverové prvky) společný všem stránkám. Místa, do kterých je „vložen“ markup nějaké .aspx konkrétní stránky, jsou reprezentována objekty `ContentPlaceHolder`. Konkrétní .aspx stránka má svůj markup rozdělen na časti (objekty `Content`), z nichž každá odkazuje na jeden z objektů `ContentPlaceHolder`, ve kterém je zobrazena.

Na obrázku č. 8 je červenou čárou ohraničena část společná všem stránkám aplikace. Jak je vidět, tak tuto část tvoří hlavička, navigační menu a patička. Markup této části se nachazí v souboru MasterPage.master.

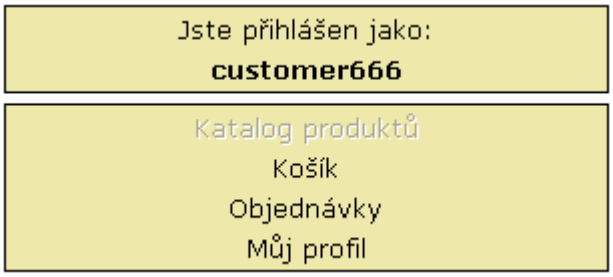
Oblast, která není červenou čárou ohraničena, je v souboru MasterPage.master reprezentována prvkem `ContentPlaceHolder`. Je to část, jejíž obsah se pro každou stráku aplikace mění.



Obrázek 8

Navigační menu se mění podle toho jestli se jedná o anonymního či přihlášeného uživatele, viz obrázky č. 9 a 10. K tomuto účelu je využit objekt `LoginView`.

Obrázek 9 – Navigační menu pro anonymního zákazníka



Obrázek 10 – Navigační menu pro přihlášeného zákazníka

Jednotný vzhled HTML prvků a písma je docílen pomocí kaskádových stylů (CSS) nadefinovaných v souboru \ App_Themes\ StyleSheet.css

Další vývoj

Možnosi aplikace jsou zatím omezené. V následujících odstavcích nastíním, jakým směrem by se měl ubírat další vývoj, aby se z aplikace stal plnohodnotný e-shop.

Administrační rozhraní

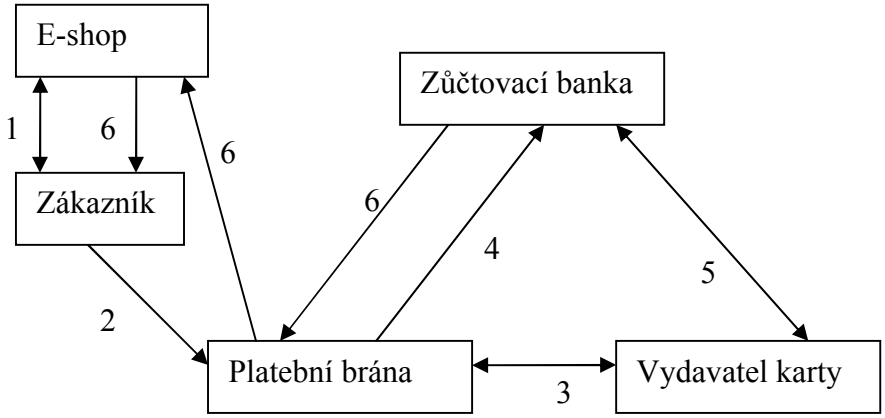
Aplikace zatím předpokládá, že správce obchodu ovládá SQL nebo pro práci s databází použije další aplikace, např. MS Excel. Toto není přípiš standartní postup, administrace je v takovém případě hodně nepohodlná. Drtivá většina internetových obchodů disponuje administračním rozhraním.

Toto rozhraní by mělo umožňovat vytváření a mazání kategorií, přidávání a mazání produktů, změnu vlastností produktu, přehled všech zaslaných objednávek, přehled zákazníků, automatické vymazávání neaktivních uživatelů

Elektronické platby

I když v současné době placení přes Internet není v ČR příliš rozšířeno a převládá placení na dobírku, situace se v tomto směru mění. V zahraničí jsou elektronické platby daleko více rozšířeny, dá se předpokládat, že se tak stane i u nás [19].

V současné době několik českých bank používá 3-D Secure systém, což je v současné době pravděpodobně nejbezpečnější možný způsob platby přes Internet. Tento systém byl vyvinut společnostmi Visa. Jak systém funguje zjednodušeně zobrazuje následující schéma:



Obrázek 11: 3-D Secure

1. po odeslání objednávky je zákazník přesměrován na platební bránu
2. zákazník platební bráně předá údaje o platební kartě, která pomocí jich vyhledá odpovídající banku (vydavatel karty)
3. platební brána pošle požadavek na ověření k vydavateli karty, ten pošle zpět odpověď, vydavatel karty určuje metodu ověření svého klienta
4. pokud je opověď kladná tak platební brána pošle požadavek na provedení transakce zůčtovací bance
5. zůčtovací banka pošle požadavek na povolení transakce vydavateli karty a obdrží odpověď
6. výsledek je poslán a zobrazen zákazníkovi

Pokud obchodník chce využívat tento systém, musí se zaregistrovat u provozovatele platební brány a musí mít účet u banky (zůčtovací banka), která tuto platební bránu využívá.

Veškerá komunika mezi zákazníkem, platební bránou a bankami probíhá prostřednitvím protokolu SSL.

Vydavatel karty zákazníka by se měl také účastnit 3-D Secure systému.

Z vývojářského hlediska je tedy třeba do aplikace doplnit kód, který zákazníka, pokud si vybere, že chce platit kartou, přesměruje na platební bránu, případně další kód zajišťující komunikaci s platební bránou.

V ČR jsou v současné době pravděpodobně nejrozšířenější tyto dvě platební brány:
MUZO – pro obchodníky s účty u KB, ČSOB a eBanky
Platební brána české spořitelny – pro obchodníky s účty u české spořitelny

Další fukčnosti

Vyhledávání – funkce vyhledávání umožňuje zákazníkům rychle vyhledat nějaký konkrétní produkt, pokud například znají jeho přesný název. Vyhledávat se může také v názvech kategorií neho v detalním popisu produktů.

Pomoc v případě zapomenutí hesla – pokud zákazník zapomene své heslo, může se pokusit získat nové pomocí kontrolní otázky a odpovědi, zadal při registraci. Pokud si zákazník vzpomene na odpověď (otázka slouží jako nápověda) je mu do jeho e-amilové schránky zasláno nové heslo. Tento mechanismus ušetří zákazníka nové registrace.

Komentování a hodnocení produktů – uživatelé mohou jednotlivé produkty hodnotit (například pomocí známek 1 - 5), případně k nim psát komentáře týkající se (ne)spokojenosti s produktem. Tímto otevřeným přístupem může obchodník dát najevo, že mu skutečně záleží na spokojenosti zákazníka.

Seznam použitých zdrojů

Odborná literatura:

[1] EVJEN, Bill, HANSELMAN, Scott, MUHAMMAD, Farhan, SIVAKUMAR, Srinivasa, RADER, Devin. ASP.NET 2.0 Programujeme profesionálně. ISBN 80-251-1286-1

[1] MACDONALD, Matthew. ASP.NET 2.0 a C#: tvorba dynamických stránek profesionálně. ISBN: 8086815382

[2] MIHULE ,Tomáš. Internetový obchod. ISBN: 80-86394-97-2

Internetové zdroje:

Portály:

[3] Microsoft Developer Network. *Portál společnosti Microsoft určený pro vývojáře používající její technologie.* www.msdn.com

[4] www.asp.net *Officiální stránky společnosti Microsoft věnované technologii ASP.NET.*

Články a dokumenty:

[5] PUŠ, Petr. Poznáváme C# a Microsoft.NET: off-line verze. http://poznavame-c-msnet.wz.cz/download.php

[6] C# Version 2.0 Language Specification *(Specifikace programovacího jazyka C# verze 2.0)*. http://download.microsoft.com/download/9/8/f/98fdf0c7-2bbd-40d3-9fd1-5a4159fa8044/csharp%202.0%20specification_sept_2005.doc

[7] CROWLEY, James. Tree structures in ASP.NET and SQL Server - Storing Trees in SQL Server *(Stromová struktura ASP.NET a SQL Server – ukládání stromů v SQL Serveru)*. http://www.developerfusion.co.uk/show/4633/2/

[8] HAYDEN, David. Paging Records Using SQL Server 2005 Database - ROW_NUMBER Function *(Stránkování záznamů s využitím SQL Server 2005 funkce ROW_NUMBER)*. http://www.davidhayden.com/blog/dave/archive/2005/12/30/2652.aspx

[9] GUTHRIE, Scott. Efficient Data Paging with the ASP.NET 2.0 DataList Control and ObjectDataSource *(Efektivní stránkování dat pomocí ASP.NET 2.0 prvků DataList a ObjectDataSource)*. http://weblogs.asp.net/scottgu/archive/2006/01/07/434787.aspx

[10] WALTHER, Stephen. Storing User Information with ASP.NET 2.0 Profiles *(Ukládání informací o uživatelích pomocí ASP.NET 2.0 profilů)*. http://msdn2.microsoft.com/en-us/library/ms379605(VS.80).aspx

[11] ONION, Fritz. A New Solution to an Old State Storage Problem. http://msdn.microsoft.com/msdnmag/issues/06/04/ExtremeASPNET/

[12] RUVALCABA, Zak. Building an ASP.NET Shopping Cart Using DataTables *(Sestavení nákupního košíku pomocí ASP.NET prvku DataTable)*. http://www.sitepoint.com/article/net-shopping-cart-datatables

[13] CONNOLLY, Randy. Personalization with Profiles and Web Parts in ASP.NET 2.0 *(Personalizace pomocí profilů a webové části v ASP.NET 2.0)*. http://www.informit.com/articles/article.asp?p=703789&rl=1

[14] VOSS, Terry. Understanding the Profile Object of ASP.NET Version 2.0 *(Objekt Profile v ASP.NET verze 2.0)*. http://aspalliance.com/559_Understanding_the_Profile_Object_of_ASPNET_Version_20_Beta_1

[15] K. SCOTT, Allen. Profiles In ASP.NET 2.0. http://www.odetocode.com/Articles/440.aspx

[16] PROSISE, Jeff. A Look Inside Membership, Role Management, and Profiles in ASP.NET 2.0. http://download.microsoft.com/download/9/d/1/9d1eba25-ea08-4375-a053-705129128f25/WEB321_Prosise.ppt

[17] Verified by Visa System Overview External Version 1.0.2 https://partnernetwork.visa.com/vpn/global/retrieve_document.do?documentRetrievalId=119

[18] Pay MUZO – Základní informace http://nadace.racek.org/priloha/paymuzo.pdf

[19] AMBROŽ, Jan. Platby kartou přes Internet pokulhávají. http://www.lupa.cz/clanky/platby-kartou-pres-internet-pokulhavaji

[20] http://www.eproton.cz/Images/Archive2/1/karty_3dsecure.jpg