DIPLOMAMUNKA

Kapusi Viktor

Debrecen

2008

Debreceni Egyetem

Informatikai Kar

Webes alkalmazásfejlesztés

Reservation Manager

Témavezető:	**Készítette:**
Dr. Kuki Attila	**Kapusi Viktor**
Egyetemi adjunktus	**Programtervező matematikus**

Debrecen

2008

Tartalom

BEVEZETÉS **1**

1 RESERVATION MANAGER **2**

1.1 ÁLTALÁNOS LEÍRÁS 2

1.2 ÁLTALÁNOS KÖVETELMÉNYEK 2

1.2.1 Bejelentkezés *2*

2 WEBES ALKALMAZÁSOK **4**

2.1 A WEBES ALKALMAZÁSOK ALAPJAI 4

2.2 ASP.NET KIALAKULÁSA 5

3 ALKALMAZOTT ARCHITEKTÚRA **8**

3.1 MI AZ A TÖBBRÉTEGŰ ALKALMAZÁS? 8

3.1.1 Business Objektumok *9*

3.1.2 Adatkapcsolati réteg (Data Access Layer) *10*

3.1.3 Businees Objektum vagy Típusos DataSet *11*

3.2 RESERVATION MANAGER ARCHITEKTÚRA 11

3.2.1 Alkalmazott technológiák *11*

3.2.2 Adatbázis *11*

3.2.3 Adatkapcsolati réteg (Data Access Layer) *14*

3.2.4 Üzleti logika réteg (Business Logic Layer) *20*

3.2.5 Prezentációs réteg (Presentation Layer) *21*

4 RESERVATION MANAGER ALKALMAZÁS MŰKÖDÉSE **33**

4.1 PORTÁS FELHASZNÁLÓKÖRREL TÖRTÉNŐ MŰKÖDÉS 33

4.1.1 Felhasználó authentikációja *33*

4.1.2 A portás főoldala *34*

4.2 SZÁLLÓDAIGAZGATÓ FELHASZNÁLÓKÖRREL TÖRTÉNŐ MŰKÖDÉS 39

4.3 ADMIN JOGOSULTSÁGGAL TÖRTÉNŐ MŰKÖDÉS 39

4.4 KONZISZTENCIA BIZTOSÍTÁSA 40

5 ÖSSZEGZÉS **41**

IRODALOMJEGYZÉK **43**

FÜGGELÉK **44**

Bevezetés

A diplomamunka célja egy webes alkalmazás, a Reservation Manager fejlesztésének bemutatása, különös figyelmet fordítva az alkalmazott architektúrára.

Az első fejezetben az Reservation Managerrel szembeni igényekről esik szó. A második fejezet egy rövid áttekintést nyújt a webes alkalmazások történetéről, illetve - mivel a Reservation Manager az ASP.NET technológián nyugszik - az ASP.NET kialakulásáról. A harmadik fejezetben az Reservation Manager architektúrájáról lesz szó, mellyel remélem az olyan olvasóknak, akik nem tudják, hogy hogyan kezdjenek egy webes alkalmazás elkészítéséhez, azoknak ezzel segítségére lehetek. Az architektúra ismertetése közben megismerkedhet az olvasó az ASP.NET 2.0-ban alkalmazható eszközökkel is. A negyedik fejezett a Reservation Manager használatát írja le és próbálja ábrákon keresztül szemléltetni. A diplomamunka elolvasását követőleg remélem, kialakul az olvasóban egy átfogó kép arról, hogy mire jók a webes alkalmazások, és hogy hogyan lehet azokat megtervezni és megvalósítani.

1 Reservation Manager

1.1 Általános leírás

Az alkalmazás egy szálloda informatikai hátterét biztosítja. Szobák kiosztását, előfoglalásokat tesz lehetővé.

1.2 Általános követelmények

- Az alkalmazás az ügyfeleket különböztesse meg. Visszatérő ügyfél esetén lehetőség nyílik, a ”kedvenc” szoba felajánlására. Ami annyit jelent, hogy ha az ügyfél szobát akar kivenni, vagy csak telefonon előre lefoglalni, akkor a rendszer az ügyfél adatainak bevitelekor már felajánlja a múltban lefoglalt szobák közül azokat, melyek éppen nem foglaltak.
- Egy előre lefoglalt szoba elfoglalását, a foglalásként megadott időtartam első napján, délig szükséges megtenni.
- Az ügyfél, fekvés, felszereltség, ágyszám, emelet és ár szerint válogathat a szobákból.
- Az ügyfélnek távozáskor kell fizetnie. Fizetéskor bejegyzésre kerül a fizetés módja, illetve az egyéb szolgáltatások díja (ebéd, vacsora) is.
- A rendszer a szállodaigazgató számára biztosít egy felületet, mely megmutatja a szálloda kihasználtságát egy tetszőleges időszakra vonatkozóan. Illetve szobákra, ügyfelekre, szobatípusokra vonatkozólag is lehet statisztikát készíteni.
- A portás csak a jelenleg bentlakó ügyfelek és minden szoba adataira tud rákeresni.
- A szállodában két portaszolgálat működik, a rendszernek gondoskodnia kell önmaga stabilitásáról és a konzisztencia biztosításáról.

1.2.1 Bejelentkezés

Minden egyes felhasználó rendelkezik egy egyedi felhasználónévvel, illetve egy jelszóval. Amikor a felhasználó valamely támogatott böngészőn keresztül (Internet Explorer 7.0, Mozilla Firefox 3.0.1) csatlakozik az alkalmazáshoz, akkor a megjelenő weblapon ezeket az adatokat kell megadni, ha az adatok valós felhasználót takarnak, akkor a felhasználó szerepkörének megfelelő tartalom jelenik meg a böngészőben.

Szerepkörök:

- Admin jogosultságai:
    - Ha szükséges, új szobák felvétele az adatbázisba.
    - Ha egy szoba felszereltsége, vagy esetlegesen fekvése változott, akkor ennek regisztrálása az adatbázisban.
    - Új felhasználók regisztrálása a rendszerben, illetve ha szükséges, létező felhasználok adatainak módosítása.
- Portás jogosultságai:
    - Új foglalás létrehozása, ha az ügyfél még nem része a rendszernek, akkor adatainak felvétele, ellentétben keresés a regisztráltak között.
    - Aktuális foglalások illetve a hozzájuk kapcsolódó ügyfelek adatainak megtekintése.
    - A szálloda szobáinak, illetve azok adatainak megtekintése.
- Szállodaigazgató
    - Adott időintervallumot nézve, a szálloda kihasználtságának megtekintése, illetve nem csak a szálloda, hanem a szobákra és szobatípusokra (felszereltség, fekvés) vonatkozó statisztikák megtekintése is.

2 Webes alkalmazások

2.1 A webes alkalmazások alapjai

Amikor egy weboldal dinamikus tartalmának létrehozásához használunk valamilyen eljárást, vagy valamilyen módon bővítjük egy kiszolgáló szolgáltatásait, lényegében egy webes alkalmazást hozunk létre. Amíg egy weboldal tisztán csak statikus tartalmat jelenít meg, ahol a felhasználó navigálhat, egy webes alkalmazás a felhasználó aktív közreműködését igényelheti.

Az asztali alkalmazások fejlesztésétől eltérően, ahol az alkalmazás részei lokálisan rendelkezésre állnak, a webes alkalmazások fejlesztése során az egyes szoftverösszetevőknek kapcsolat nélküli protokoll segítségével, elosztott hálózatban kell működniük. Az ASP.NET mögöttes technológiái már jó ideje rendelkezésre állnak. Az ASP.NET kissé rejtve, de ugyanakkor megközelíthető módon használja ezeket a technológiákat.

Bár az ASP.NET nagymértékben megkönnyíti a webes alkalmazások fejlesztését, az ASP.NET-es alkalmazás fejlesztése során szilárd elméleti alapokra van szükségünk a rendszer működését illetően.

- HTML-kérések

  A webes böngészők a *HTTP* (HyperText Transfer Protocol) kommunikációs mechanizmus segítségével szólítják meg a web helyeket. A ma ismert web, kutatási projektnek indult a CERN-nél Svájcban. Akkoriban vált a hypertext fogalma egyre népszerűbbé. A *HTTP* a *TCP/IP* felett az alkalmazási réteg része. Eredeti formájában a *HTTP* a hypertext dokumentumok átvitelére szolgált. Vagyis a *HTTP* célja eredetileg a dokumentumok összekapcsolása volt tekintet nélkül bármire, legyenek azok a webes felhasználói felületek, amelyek összekapcsolják a mai modern web helyeket. A *HTTP* korai verziói egyetlen *GET* kérést támogattak, amely visszaadta a megnevezett erőforrást. A későbbiekben a kiszolgáló feladata lett, hogy a fájlt szövegfolyamatként továbbítsa. Miután a válasz megérkezett az ügyfél böngészőjére, a kapcsolat lezárult. A *HTTP* első verziói kizárólag a szövegfolyamok átvitelét támogatták, semmilyen más adatátvitel nem volt lehetséges. A *HTTP* első formális specifikációja az 1.0 verzióban, az 1990-es évek közepén jelent meg. A *HTTP* 1.0 az egyszerű szövegátviteli

protokollon túl összetettebb üzenetküldést tett lehetővé. A *HTTP* különböző (a *MIME* [Multipurpose Internet Mail Extension] által meghatározott) médiákat támogatott. A *HTTP* aktuális verziója az 1.1-es verzió. Kapcsolati protokollként a *HTTP* néhány alapvető parancsra épül (*GET*, *POST*, *HEAD*). A *GET* parancs visszaadja a kérés *URI*-je által meghatározott információkat. A *HAED* parancs kizárólag a kérés *URI*-je által meghatározott fejrészadatokat adja vissza. A *POST* módszer segítségével a kiszolgálóra küldött kérésnek bizonyos „mellékhatásai" lehetnek. A weboldalakkal a kapcsolatot a *GET* parancs segítségével vehetjük fel, és a további kommunikációt *POST* parancsok révén bonyolíthatjuk.

2.2 ASP.NET kialakulása

Körülbelül 1993-ig világszerte igen kevés webkiszolgáló működött. Az első webkiszolgálók nagy része egyetemeken és kutatási központokban üzemelt. Az 1990-es évek elején a web helyek száma drámai növekedésnek indult. Ha az 1990-es évek elején használtuk a webet, biztosan találkoztunk a webhely úttörők *HTML* oldalaival vagy fényképgyűjteményekkel, amelyek *GIF* vagy *JPEG* fájlok hivatkozásait tartalmazták. Akkoriban még nem létezett a Google, a Yahoo és az MSN Search sem. A webhely megtekintésének egyetlen módja az volt, ha ismertük a webhely *URL*-jét (Uniform Resource Locator), vagy valaki más oldalán találtunk hivatkozást az adott webhelyre. Az első webkiszolgálók UNIX gépeken működtek. Feladatuk az volt, hogy betöltsék a *HTML* fájlt és azt visszaküldjék a kérelmező félnek.

A *CGI* (Common Gateway Interface) megjelenésével a böngészők az interaktív webes alkalmazások standard interfészévé léptek elő. Az egyszerű, statikus *HTML* dokumentumokat kiszolgáló webkiszolgáló bizonyos célokra kiválóan alkalmas, az összetettebb alkalmazások megkövetelik a felhasználó és a kiszolgáló közötti kommunikációt.

És a *CGI* itt jut fontos szerephez. A szabványos grafikus vezérlőelemeket megjelenítő *HTML* címkék segítségével a *CGI* alkalmazások a kérésekre dinamikusan válaszolnak. Más szóval, a *CGI* alkalmazások a kérés állapotától és az alkalmazástól függően különböző kimenetet állítanak elő, ezzel is segítik az interaktív alkalmazásokat.

Amikor világossá vált, hogy a web az informatika fontos része, a Microsoft is beszállt az üzletbe. Piacra dobta az *ISAPI*-t (Internet Services Application Programming Interface) és a *HTTP* kéréseket figyelő *IIS*-t (Internet Information Service). Az első UNIX webkiszolgálók

minden új HTTP kérés számára új folyamatot indítottak. Ez rendkívül költséges. A Microsoft webes stratégiája a DLL-eken alapul. Egy *HTTP* kérésre válaszolva sokkal gyorsabb betölteni egy DLL-t, mint elindítani egy teljesen új folyamatot.

Microsoft platformon az *IIS* a 80-as porton figyeli a *HTTP* kéréseket. Az *IIS* néhány kérést közvetlenül kiszolgál, míg más kéréseket az *ISAPI* kiterjesztések DLL-jeihez küldi, és a kéréseket azok hajtják végre. A többi esetben az *IIS* a fájlkiterjesztést leképzi egy specifikus *ISAPI* DLL-re. Sok *ISAPI* DLL előre telepíthető a Windows operációs rendszerrel együtt. Az *IIS* bővíthető, és a különböző kiterjesztéseket bármely *ISAPI* DLL-re leképezhetjük, még saját DLL-jeinkre is. Az *IIS*-sel és az *ISAPI*-val működő webhely készítése során a fejlesztők *ISAPI* DLL-eket alkalmaznak. Ezek a DLL-ek észlelik a kérést, elemeire szedik, és a kérésre válaszolva visszaküldenek valamit (*HTML*) az ügyfélnek.

Noha az *IIS/ISAPI* platform a webes alkalmazások készítésének rugalmas és funkcionálisan gazdag módját biztosítja, ez a platform sem tökéletes. Az *ISAPI* DLL-ek C++-ban készülnek, ezért a DLL-eknek a C++ programozás csapdáit kell kikerülniük.

A Microsoft annak érdekében, hogy a web fejlesztést még elérhetőbbé tegye Microsoft platformon, kifejlesztette az *ASP*-t (Actice Server Pages). A klasszikus *ASP* alapja az, hogy egyetlen *ISAPI* DLL, az ASP.DLL értelmezi az ASP kiterjesztéssel rendelkező fájlokat. Az *ASP* fájlok *HTML* kódot, illetve esetenként szkriptkódokat tartalmaznak, amelyet a kiszolgálón kell végrehajtani. Az *ASP ISAPI* DLL szükség szerint végrehajtja a szkriptkódot, és az *ASP*-fájlban található HTML-t visszaküldi az ügyfélnek. A szkriptkód rendszerint *COM* objektumokat hív, és azok végzik el a piszkos munkát, miközben az oldal megjelenését az *ASP* fájl *HTML* kódja határozza meg. Az *ASP*, mivel egy jóval szélesebb körben használt programozási nyelvet (Visual Basic, VBScript) biztosított, rengeteg új programozót vonzott, de mégsem jelentett mindenre megoldást. A klasszikus *ASP*-vel szembeni problémák hívták életre az ASP.NET-et.

A Microsoft .NET keretrendszer a Microsoft platform programozásának teljesen új módját vezeti be. A Microsoft fejlesztőket elsősorban a szál- és a memóriakezelés foglalkoztatja (ami nagyjából az API programozás modellt takarja). Ez a modell, a web fejlesztést is beleértve, a programozás minden területén elterjedt, és nagyon megnehezítette a programozók munkáját. A .NET a kezelt típusok fogalmára épít. A klasszikus Windows-os kódot (és web kódot) készítő fejlesztők az osztályokat C++ vagy Visual Basic segítségével írták. A típusok több

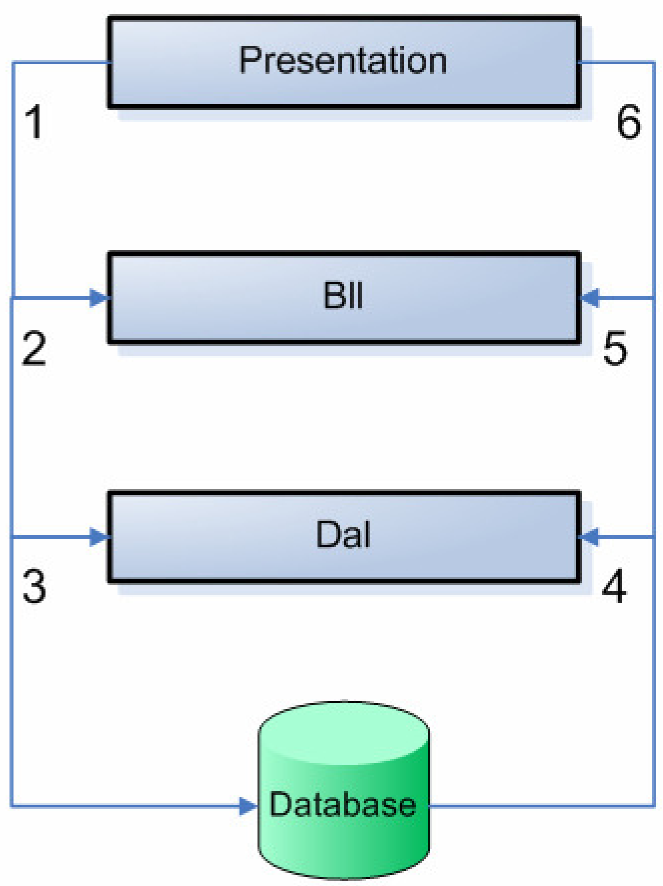
szempontból is hasonlítanak a C++ osztály fogalmához, mivel a típusok is önálló funkcióval bíró állapotegységeket jelentenek. Azonban a hasonlóságnak itt vége is szakad. Míg korábban a fejlesztőre hárult az osztálypéldányok kezelésének feladata, a típusok teljes mértékben a .NET futásidejű szolgáltatásai, azaz a *CLR* (Common Language Runtime) kezeli. Mivel a *CLR* magára vállalja a szál- és memóriakezelés feladatát, a fejlesztők sokkal inkább a konkrét alkalmazásra összpontosíthatnak (és nem kell téves mutatók, memóriatúllépés és megmagyarázhatatlan összeomlások után nyomozniuk). Az ASP.NET bevezeti a futásidejű szolgáltatásokat, valamint a web fejlesztés magas szintű továbbfejlesztését jelentő, kiválóan megtervezett class library-ket. A klasszikus *ASP* beépült az *IIS/ISAPI* architektúrába annak megfontolása nélkül, hogy a korai tervezési döntések a későbbiekben milyen hatással lesznek a fejlesztők munkájára. Mostanra, egy kis idő elteltével, már világosan látszanak az ASP.NET hibái. Az ASP.NET a kezdetektől fogva a *HTTP* kérések kezelésének bővíthető, sokoldalú megoldásának készült. Az ASP.NET befolyásolja az *IIS* működését, mivel az ASP.NET szolgáltatások kéréseit leképezi egy ISAPI DLL-re. Ez a DLL az ASPNET_ISAPI.DLL. A végrehajtás innen az ASP.NET dolgozófolyamatához kerül. Az alapvető kérésfeldolgozást a dolgozófolyamat kezelt típusai végzik. A vezérlés a csatornához csatlakoztatott osztályok között vándorol, néhány osztályt a Microsoft és/vagy harmadik fél biztosít, néhányat pedig a fejlesztő. Az ASP.NET már alapjaiban is a webes alkalmazások készítésének átfogó keretrendszere. A keretrendszer minden része együttműködik a kérések kezelésében. Ezzel szemben a klasszikus ASP.NET szkriptkód strukturálatlan volt, és az alkalmazás kódlogikája is ad hoc jellegű volt. Ekkor érkezett el az ASP.NET 2.0 ideje, mely már sokkal több lehetőséget biztosított.

3 Alkalmazott architektúra

3.1 Mi az a többrétegű alkalmazás?

A rétegelt, vagy más néven a többrétegű alkalmazás, azt a fejlesztés során alkalmazott modellt célozza meg, ahol a megjelenítés, a business (üzleti) logika és az adat, szeparáltan jelennek meg. Ez olyan előnyöket hordozz magában, mint például: az átláthatóbb (tisztább) kód, könnyebb karbantarthatóság, az alkalmazás terhelésének megoszthatósága több szerveren.

A következő ábra egy rétegelt alkalmazás kinézetét szemlélteti:



1.ábra Rétegelt alkalmazás modell

Az ábra azt az általános folyamatot szemlélteti, amelyen keresztül az alkalmazás adatokat reprezentál, melyek egy adatbázisból származnak. Az alkalmazás felhasználója rendszerint a prezentációs (presentation) réteget hívja. A középső réteget az üzleti logika jelenti (Business

Logic Layer), mely hidat képez a prezentáció és a következő réteg, az adatkapcsolati réteg között. Az adatkapcsolati réteg az adatbázishoz intéz kéréseket (select, insert, update, delete).

Ezen modell egyik alapötlete, hogy lehetséges az egyes rétegek cseréje, módosítása, anélkül hogy az hatással lenne a többire, így például egy web site-ot, mint egy a megjelenítési réteget lecserélhetünk egy Windows Form-os alkalmazással.

A 1. ábra által reprezentált folyamat:

1. A prezentációs réteg az üzleti logika rétegtől kér objektumokat.
2. Az üzleti logika réteg végrehajthat néhány ellenőrzést, mint például: az aktuális felhasználónak van e jogosultsága a kéréshez. Majd továbbítja a kérést az adatkapcsolati rétegnek.
3. Az adatkapcsolati réteg kapcsolatot teremt az adatbázissal, és elkéri a megfelelő adatot.
4. A kért adatot reprezentáló rekord, a megtalálását követőleg az adatkapcsolati rétegnek adódik át.
5. Ezek után az adatkapcsolati réteg az adatot egy megfelelő objektumba (Business object) csomagolja és átadja az üzleti logika rétegnek.
6. Majd végül az objektum továbbadódik a prezentációs rétegnek, ahol például egy weboldal formájában jelenik meg.

3.1.1 Business Objektumok

A Business Objektumok olyan egyszerű osztályok példányai, melyek *Object* osztályból származnak és a szükségképpen megfelelő interfészt vagy interfészeket implementálják.

Mindenekelőtt szükséges azon követelmények, adatok meghatározása, melyeket ezek az objektumok fognak magukban hordozni. A következő meghatározandó dolog, hogy az objektumok által végrehajtható műveletek hol tárolódjanak. Az egyik lehetőség, hogy az objektumokban csomagoltan jelennek meg, ekkor „okos” osztályokról beszélünk. A másik lehetőség, hogy csak az adatok tárolódnak az objektumokban („buta” adattároló osztályok), melyek mellett különálló osztályok (manager osztályok) kezelik az adatokkal kapcsolatos interakciókat.

3.1.2 Adatkapcsolati réteg (Data Access Layer)

Az adatkapcsolati réteg megvalósításának az egyik módja, hogy az adat specifikus logikát a prezentációs rétegben helyezzük el. Ez ASP.NET környezetben azt jelentené, hogy ADO.NET kódot írunk ASP kódsorok közé, vagy *SqlDataSource* control-t használunk. Ezáltal szorosan összekapcsoljuk a fent említett két réteget. Ennek elkerülésére a megfelelő szemlélet az, ha az adatkapcsolati logikát elszeparáltan a prezentációs rétegtől, egy külön projectben valósítjuk meg. Ez lesz az adatkapcsolati réteg (Data Access Layer), a továbbiakban *DAL*.

Minden olyan kód mely az adatbázishoz kötődik, mint a kapcsolat létrehozása az adatbázissal, *SELECT*, *INSERT*, *UPDATE* és *DELETE* utasítások stb. a *DAL*-ban helyezendőek el. A *DAL* tipikusan olyan metódusokat tartalmaz, melyek az adatbázis adataihoz férnek hozzá. A lekérdező (*getter*) metódusok hívásukat követőleg kapcsolódnak az adatbázishoz és a megfelelő kérést végre hajtják, majd ennek az eredményével térnek vissza. A visszatérési értékük lehet *DataSet* vagy *DataReader*, melyeket az adatbáziskérés tölt fel. Az ideális eset az, ha ezek típusos objektumok formájában jelennek meg (strongly-typed object), melyek szigorúan fordítási időben definiáltak, még ellenkező esetben a típusnélküli objektumok (lossely-typed object) a futási időig ismeretlenek.

A *DataReader* és a *DataSet* alapból típusnélküli objektumok, mivel a sémájuk az adatbázis lekérdezés által meghatározott oszlopok által definiált. Ahhoz, hogy a megfelelő oszlopot elérhessük a következő szintaktikát kell használnunk: *DataTable.Rows[index][„oszlopnév”]*. Típusos *DataTable* esetén minden oszlopot, implementált tulajdonságként (properties) érhetünk el, így használhatjuk a következő kódot: *DataTable.Rows[index].oszlopnév*.

Ahhoz hogy típusos objektumokkal dolgozhassunk, használhatunk saját business objektumokat, vagy típusos *DataSet*-eket. A Business objektumok, általunk implementált osztályok, melyek tipikusan az adatbázis tábla oszlopait hordozzák magukban tulajdonságok formájában (properties). A típusos *DataSet*-ek olyan osztályok, melyek az adatbázis sémán alapszanak, adattagjai pedig típusosan kötődnek ehhez. A típusos *DataSet* az ADO.NET *DataSet* , *DataTable*, *DataRow* osztályok leszármazottait hordozza magában, illetve *TableAdaptereket*, amelyek a *DataSet DataTable*-inek feltöltésért, illetve az adatbázisba irányuló módosításokért felelős metódusokat tartalmazzák. Amikor egy típusos *DataSet*-et

tervezünk egy *XSD* (XML Schema Definition) fájlt hozunk létre, amely a *DataSet* által meghatározott adatok sémáját tartalmazza.

3.1.3 Business Objektum vagy Típusos DataSet

Ezek után felvetődik a kérdés a fejlesztőben: „business objektum vagy típusos *DataSet*?”. Ez a fejlesztők között vita tárgyát képzi. A típusos *DataSet* könnyen létrehozható a designer-en keresztül (Visual Studio), business entitások adatait szolgáltatja és támogat haladó lehetőséget, mint például: rendezés, filterezés, keresés. Számos, a *DataSet*-et érő támadás alapjai azok a teljesítménybeli problémák, melyek a .NET 1.1 implementációban figyelhetőek meg. Ilyen problémák egyike: alacsony teljesítmény nagyméretű *DataSet*-ekkel történő munka során. Ezek a problémák a .NET 2.0-ban orvosolva lettek. Mint már említettem a *DataSet* business entitásokat reprezentál, illetve kollekciókat ezek tárolására. Így elkerülhető a sok kód sajátkezű megírása. A business objektumok előnye, hogy explicit és teljes kontrolt ad arra, hogy milyen képességekkel rendelkezzenek az objektumok, illetve mi az-az API, amit az objektum magában hordoz, megmutat. Ha egy fejlesztő a tisztán objektum-orientált sémát szereti, akkor otthonosabban és kényelmesebben mozoghat a business objektumok világában. Viszont fontos megjegyezni, hogy minden olyan dolog, amit a fejlesztő meg tud valósítani a saját business objektumaival, azokat véghez tudja vinni *DataSet*-el is. Illetve, ha az elsődleges cél az adatok megjelenítése, és a felhasználó dolgozhat ezekkel az adatokkal, amelyek visszahatnak az adatbázisra, akkor a *DataSet*-tel történő munka gyorsabbnak tekinthető. Viszont ha *DataSet*-et használunk prezentációs vagy az üzleti logika rétegben, akkor ezeket szorosan összekapcsoljuk az adatkapcsolati réteggel, amely természetesen kerülendő. Mindezek után a fejlesztőre bízott, hogy melyik utat választja.

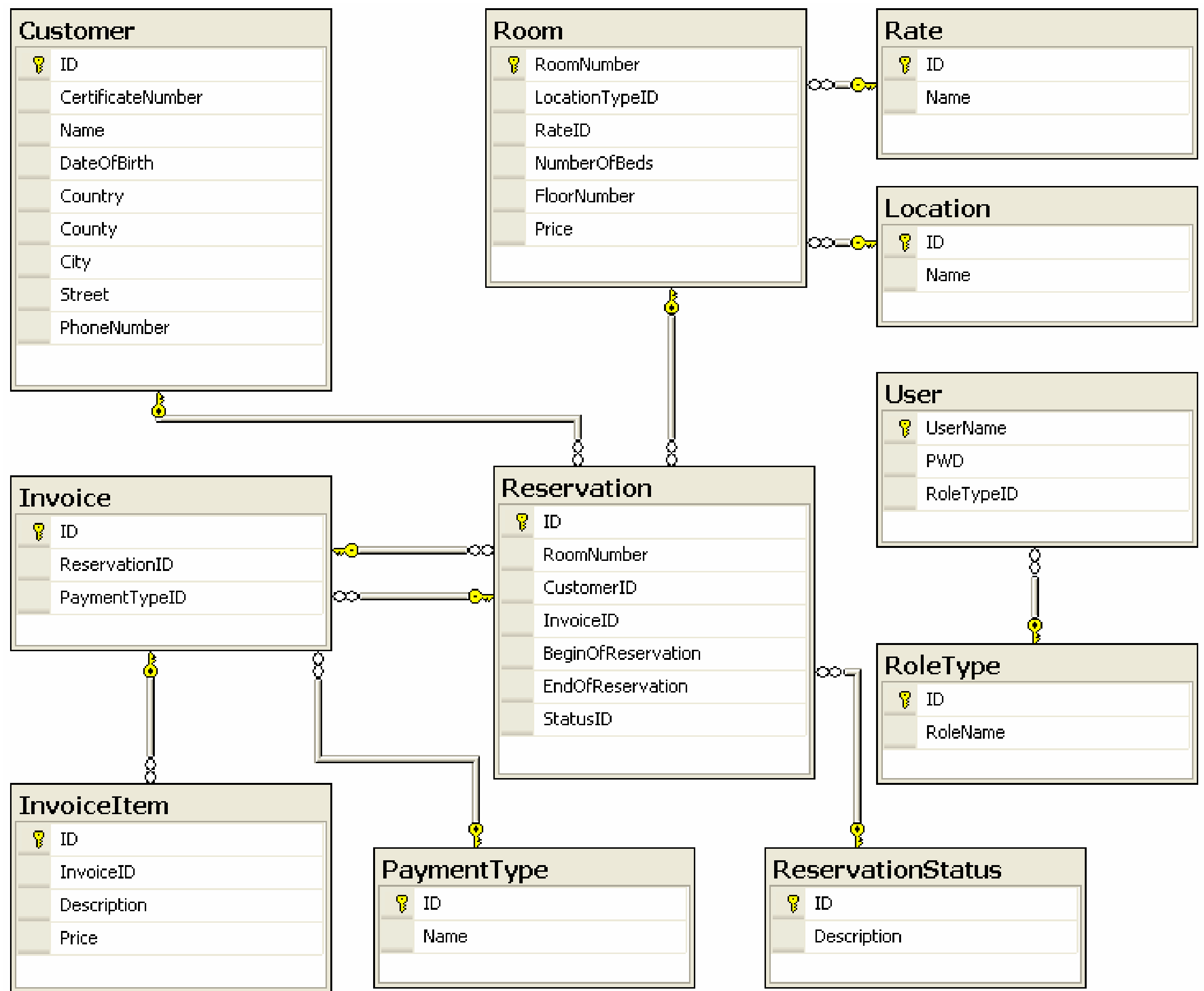
3.2 Reservation Manager architektúra

3.2.1 Alkalmazott technológiák

- Adatbázis: Microsoft SQL Server 2005
- Webes technológia: ASP.NET 2.0

3.2.2 Adatbázis

Az alábbi képen, egy az adatbázisról készült, adatbázis-diagrammot láthatunk.



2.ábra Reservaion Manager adatbázis-diagramm

Nézzük sorban az adatbázis-táblákat:

- Customer:

  A szálloda ügyfeleit reprezentáló tábla. A tábla oszlopai:

  - ID, elsődleges kulcs
  - CertificateNumber, az ügyfél valamely személyiségét igazoló okmányának a száma
  - DateOfBirth, az ügyfél születési ideje
  - Country, az ügyfélhez kapcsolódó ország neve
  - County, *nullable* oszlop, mely az ügyfélhez kapcsolódó megye neve
  - City, az ügyfélhez kapcsolódó város neve
  - Street, az ügyfélhez kapcsolódó utca neve

- PhoneNumber, *nullable* oszlop, az ügyfél telefonszáma

- Location:

  A szálloda szobáinak fekvését reprezentáló tábla

  - ID, elsődleges kulcs
  - Name, a fekvés megnevezése (tengerpartra néző, parkra néző stb.)

- Rate:

  A szálloda szobáinak felszereltségét reprezentáló tábla

  - ID, elsődleges kulcs
  - Name, a felszereltség megnevezése (első osztály, másod osztály, harmad osztály)

- Room:

  A szállodai szobáit reprezentáló tábla

  - RoomNumber, elsődleges kulcs
  - LocationTypeID, külső kulcs a Location táblához
  - RateID, külsőkulcs a Rate táblához
  - NumberOfBeds, ágyak száma a szobában
  - FloorNumber, az emelet száma
  - Price, a szoba egy éjszakára történő foglalásának az ára

- PaymentType:

  A számlafizetés módjait reprezentáló tábla. A tábla oszlopai:

  - ID, elsődleges kulcs
  - Name, a fizetésmód megnevezés (bankkártya, csekk, készpénz)

- Invoice:

  A foglalásokhoz kötődő számlákat reprezentáló tábla. A tábla oszlopai:

  - ID, elsődleges kulcs
  - ReservationID, külsőkulcs a Reservation táblára
  - PaymentTypeID, külsőkulcs a PaymentType táblára

- InvoiceItem:

  Számlákhoz tartozó számlatételeket reprezentáló tábla. Egy számlához minimum egy számlatétel tartozik, mely a szobafoglalás árát tartalmazza (szoba ára * foglaláshossza napokban). A tábla oszlopai:

  - ID, elsődleges kulcs

- InvoiceID, külső kulcs az Invoice táblához
- Description, a számlatétel megnevezése (szoba ára, ebéd ára, stb.)
- Price, a számlatétel értéke

- ReservationStatus:

  A foglalások állapotát reprezentáló tábla. A tábla oszlopai:
  - ID, elsődleges kulcs
  - Description, az állapot megnevezése (waiting, inProgress, paid)
- Reservation:

  A foglalásokat reprezentáló tábla. A tábla oszlopai:
  - ID, elsődleges kulcs
  - RoomNumber, külsőkulcs a Room táblához
  - CustomerID, külsőkulcs a Customer táblához
  - InvoiceID, külsőkulcs az Invoice táblához, *Unique* (egyedi) oszlop
  - BeginOfReservation, a foglalás kezdetének dátuma
  - EndOfReservation, a foglalás végének dátuma
  - StatusID, külsőkulcs a ReservationStatus táblához
- RoleType

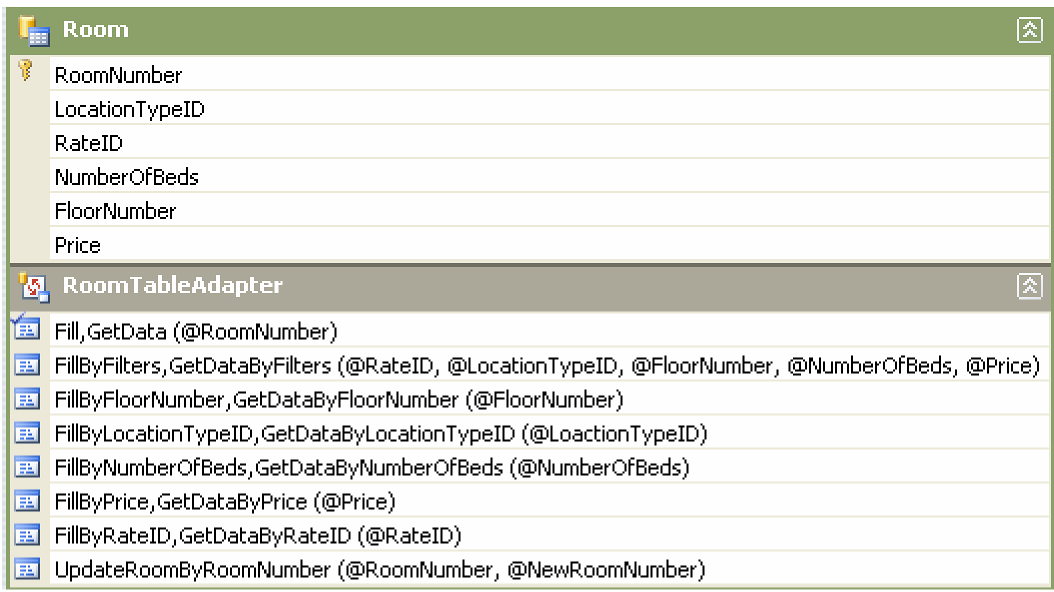
  Az alkalmazás felhasználóinak szerepköreit reprezentáló osztály. A tábla oszlopai:
  - ID, elsődleges kulcs
  - RoleName, a szerepkör megnevezése (Admin, Portás, Igazgató)
- User

  Az alkalmazás felhasználóit reprezentáló osztály. A tábla oszlopai:
  - UserName, felhasználó név, elsődleges kulcs
  - PWD, jelszó
  - RoleTypeID, külső kulcs a User táblához

3.2.3 Adatkapcsolati réteg (Data Access Layer)

Ahogy az előzőekben olvashattuk, az adatkapcsolati réteg az adatbázishoz kapcsolódó kérések irányításáért felelős réteg. Továbbá arról is szó volt, milyen úton implementálhatjuk ezt. Én a típusos *DataSet*-el történő megvalósítást választottam, de ennek ellenére nem vetettem el a business objektumok használatát sem, majd a későbbiekben meglátjuk, hogy miért. A Függelékben 3.ábra szemlélteti az alkalmazás DataSet-jét.

Az adatbázisban megtalálható táblák mindegyikéhez tartozó típusos *DataTable*-t, illetve ezek mindegyikéhez külön-külön kötődő *TableAdapter* tartalmaz a *DatSet*. Nézzük meg közelebbről ezeket. Tekintsük a Room-hoz kapcsolódó *DataSet*-beli adatokat:



4.ábra RoomDataTable és RoomTableAdapter

- A RoomTableAdapter

Ennek az osztálynak egy példánya lesz az, amely lehetőséget ad az adatbázisfelé irányuló, a Room adatbázistáblához kapcsolódó műveletek végrehajtására. A RoomTableAdapter az alábbi publikus metódusokat tartalmazza, melyek hívása, egy az adatbázibeli tárolteljárást hív meg.

  - Get
    - Egy paramétert vár, egy *int* típusú változót.
    - Az *int* típusú változó a SelectRoom tárolt eljárás paramétere lesz, majd a tárolt eljárás hívásának az eredménye egy *RoomDataTable* objektumba lesz becsomagolva, mellyel a metódus visszatér, hívását követőleg.
  - Fill
    - Két paramétert vár, egy *RoomDataTable* típusú objektumot, illetve egy *int* típusú változót.

        - Az *int* típusú változó a SelectRoom tárolt eljárás paramétere lesz, majd a tárolt eljárás hívásának eredménye a paraméterül kapott *RoomDataTable* objektumba lesz becsomagolva.

A többi Fill és Get metódust nem részletezném, ezek annyiban különböznek az előzőekben említettektől, hogy más *select* paramétereket várnak és más tárolt eljárásokat hívnak meg, de működésüknek lényege az ugyanaz.

    - Update
        - Azokat a paramétereket várja, amelyeket az UpdateRoom tárolt eljárás lefuttatásához kell átadnia.
        - A tárolt eljárás meghívását követőleg, a Room adatbázistábla a kívánalmaknak megfelelően módosul.

Több Update metódusa is van az adapternek, ezek csak annyiban különböznek az előzőekben említettől, hogy más paramétereket várnak és más tárolt eljárást hívnak, de ezek is a kívánalmaknak megfelelően módosítják a Room adatbázistáblát.

    - Delete
        - Hívásának eredménye, hogy a paramétere által meghatározott sor törlődik a Room adatbázistáblából.
    - Insert
        - Hívásának eredménye, hogy a paraméterek által meghatározott új sor bekrül a Room adatbázistáblába.

- *RoomDataTable* a *System.Data.DataTable* és a *System.Collections.IEnumerable* osztályokból öröklődő osztály.

```
public partial class RoomDataTable : System.Data.DataTable,
System.Collections.IEnumerable {…}
```

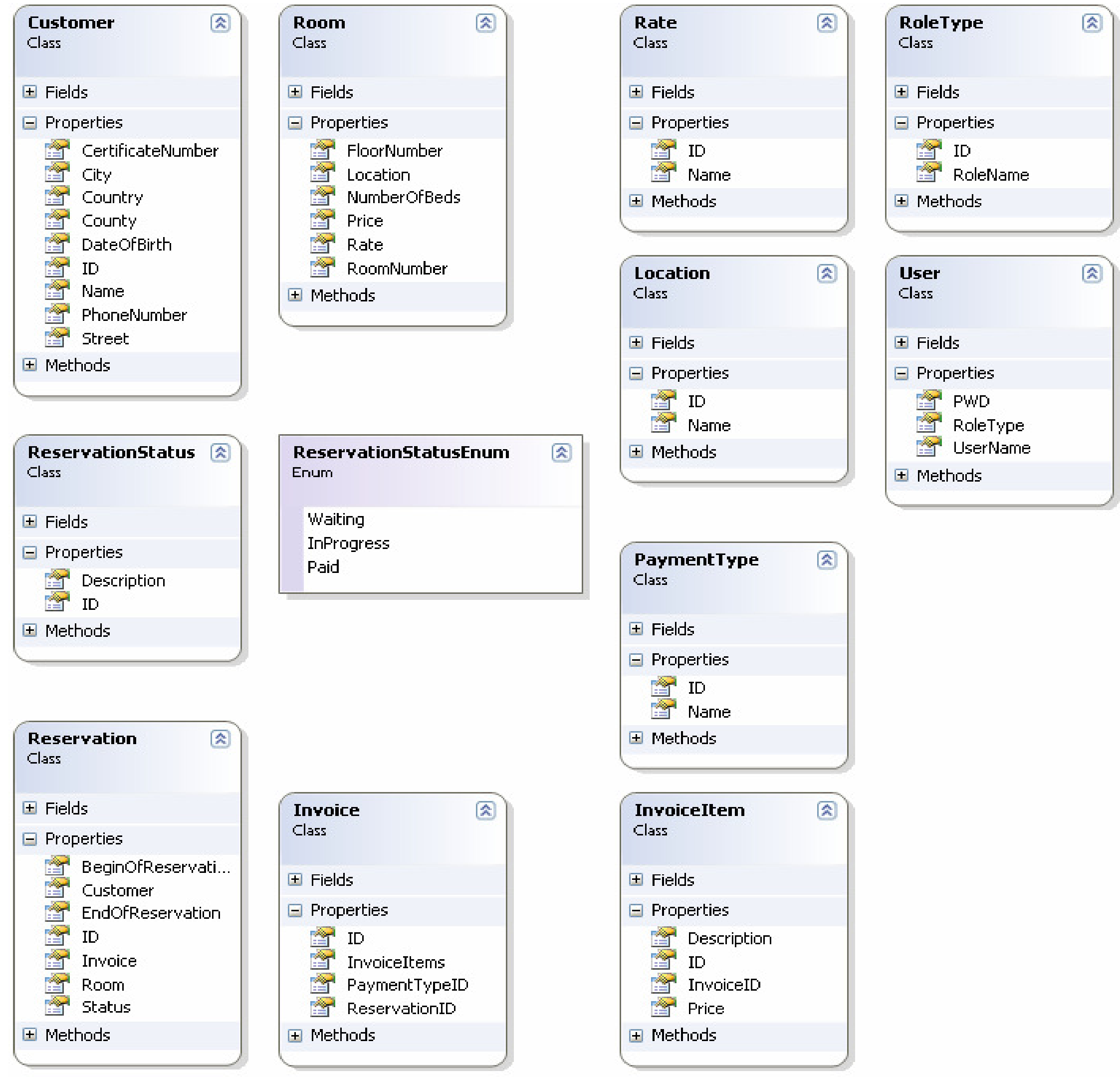
Ezen osztály egy példányának a publikus tulajdonságai (properties) között szerepelnek *System.Data.DataColumn* típusú objektumok, melyek egy az egyben az adatbázistábla oszlopait reprezentálják:

    - RoomNumber
    - LocationTypeID
    - RateID

- NumberOfBeds
- FloorNumber
- Price

Láthatjuk tehát, hogy ezekkel az *TableAdapter*-ekkel és *DataTable*-ökkel, minden, az adatbázishoz kapcsolódó kérés megvalósítottnak tekinthető. Hiszen ha esetlegesen módosítani szeretnénk egy tábla sorát, akkor a megfelelő adapter, megfelelő metódusát hívjuk, a megfelelő paraméterekkel. Hasonlóan történik ez törlésnél, insert-nél és lekérdezésnél is.

A Fill és Get metódusok típusos *DataTable*-ök formájában szolgáltatják a lekérdezés eredményét. Ez abból a szempontból nem jó, hogy ezekkel nem ajánlatos dolgoznunk magasabb rétegekben, hiszen akkor nem tudnánk megőrizni a rétegek szeparáltságát, laza kötöttségét. Ez a magyarázat arra, hogy nem vetettem el a business objektumok használatát, melyeket a „buta" adattároló osztályok használtával valósítottam meg. A következő ábra ezeket szemlélteti:



5.ábra Reservation Manager business objektumok

Ezek az osztályok nagyon egyszerűek, igazából nagyon hasonlóak az adatbázisban található táblákhoz.

Nézzük meg közelebbről a Room osztályt.

Tulajdonságai (properties):

- FloorNumber
- Location
- NumberOfBeds
- Price
- Rate
- RoomNumber

Mint láthajuk, van egy pár különbség a Room adatbázistáblához képest, az egyik ilyen, hogy ebben az osztályban nem a LocationID-t tároljuk hanem egy Location típusú objektumot, amely a szoba fekvésére vonatkozó összes adatot tárolja. Hasonló elmondható a RateID-val kapcsolatban is. Az ilyen megoldás sok esetben tud hasznos lenni, amikor a prezentációs, vagy az üzleti logika rétegben dolgozunk vele. Hiszen ha szükségünk van az előzőekben említett szobatulajdonságokra (bővebb értelemben), akkor nem szükséges az adatkapcsolati réteghez nyúlnunk, hiszen ezeket a tulajdonságokat az objektum, létrejöttekor már tárolja. Már csak annyi a kérdés; milyen úton kapunk ilyen business objektumokat az adatkapcsolati rétegtől?

Az adatkapcsolati rétegben található a DataAccessManager osztály, mely a következőkből áll:

- az összes *TableAdapter* típusnak megfelelő privát, statikus adattag.
- publikus statikus metódusok, melyek a megfelelő *TabelAdapter* metódusokat hívják.

A publikus statikus metódusok lesznek azok, melyek a megfelelő *TableAdapter* metódust hívják. Azokban az esetekben, amikor lekérdezéseket takarnak az adott metódusok, akkor a megfelelő *TableAdpater* metódusa által visszaadott *DataTable*-ők sorait fogják business objektumokba csomagolni.

Nézzünk erre egy pédát, szoba lekérdezése, szobaszám alapján:

```
private static RoomTableAdapter mRoomTableAdapter = new RoomTableAdapter();

public static Room SelectRoom(int roomNumber)
{
      RMDataSet.RoomDataTable userTable = mRoomTableAdapter.GetData
                                                          (
                                                          roomNumber
                                                          );
      Room result = null;
      if (userTable.Rows.Count == 1)
      {
            RMDataSet.RoomRow userRow =
                                 (RMDataSet.RoomRow)userTable.Rows[0];
            RMDataSet.RateDataTable rateTable =
                             mRateTableAdapter.GetData(userRow.RateID);
            RMDataSet.LocationDataTable locationTable =
                  mLocationTableAdapter.GetData(userRow.LocationTypeID);
            if (rateTable.Rows.Count == 1 && locationTable.Rows.Count == 1)
            {
                  RMDataSet.RateRow rateRow =
                                 (RMDataSet.RateRow)rateTable.Rows[0];
                  RMDataSet.LocationRow locationRow =
                             (RMDataSet.LocationRow)locationTable.Rows[0];
                  Rate rate = new Rate(rateRow.ID, rateRow.Name);
                  Location location = new Location
```

```
                                                      (
                                                      locationRow.ID,
                                                      locationRow.Name
                                                      );
                  result = new Room
                        (
                              userRow.RoomNumber, location, rate,
                              userRow.NumberOfBeds, userRow.FloorNumber,
                              userRow.Price, userRow.IsReserved
                        );
            }
      }
      return result;
}
```

Összefogalalva tehát az adatkapcsolati réteg, az adatbázistáblákra épülő *DataTable*-ökből és *TableAdapter*-ekből, illetve az ezekre húzott DataAccessManager-ből áll, ahol az utóbbi, a lekérdezések esetén business objektumokat, illetve bizony esetekben azoknak listáját szolgáltatja.

3.2.4 Üzleti logika réteg (Business Logic Layer)

Ez a réteg lesz az, amely a felhasználói felületről érkező kéréseket továbbítja az adatkapcsolati rétegnek, és ha a kérés adatok lekérdezését jelenti, akkor azokat business objektum(ok) formájában kapja meg és továbbítja a felhasználói felület irányába. A mi esetünkben ez a réteg egyetlen BusinessManager-nek nevezett osztályból áll. Ez az osztály fogja tartalmazni azokat a statikus metódusokat, melyeken keresztül kommunikál a felhasználó felület (felhasználó) és az adatkapcsolati réteg.

Példa egy ilyen metódusra:

```
public static Room SelectRoom(int roomNumber)
{
    return DataAccessManager.SelectRoom(roomNumber);
}
```

Bonyolultabb alkalmazások esetén ezek a metódusok nem ilyen egyszerűek. Az adatkapcsolati rétegbeli metódus hívása estén szokásos ellenőrizni, hogy az adott usernek van e jogosultsága a művelethez. Illetve *cache*-elési folyamatokat is ebben a szekcióban szokták elhelyezni.

3.2.5 Prezentációs réteg (Presentation Layer)

3.2.5.1 Az ASP.NET programozási modellje

3.2.5.1.1 Az ASP.NET weboldalak felépítse

Az ASP.NET weboldalak két részből állnak: egy forráskód és egy *HTML* részből. Fontos, hogy alapvető különbségek vannak a statikus és a dinamikus weboldalak között. A leglényegesebb, hogy a dinamikus weboldalak vegyesen tartalmaznak *HTML* elemeket és kiszolgálóoldali forráskódot. Amikor egy dinamikus weboldalt a felhasználó szeretne megtekinteni, kérésére válaszul végrehajtódik a forráskód, ami előállítja a *HTML*-t, és ezt a dinamikusan létrehozott *HTML*-kódot kapja meg a böngésző (vagy más kliensoldali program).

Mivel a *HTML* (HyperText Markup Language) leíró nyelv sokak számára ismert, az ASP.NET weboldalak ezen részét nem ismertetném. Sokkal érdekesebb a fent említett forráskód rész. Míg a *HTML* rész az ASP.NET oldal külalakját, illetve statikus tartalmát határozza meg, a forráskód az dönti el, mi legyen a dinamikus tartalomban.

Az ASP.NET oldalak forráskód része két helyen lehet:

- külön fájlban – Az ASP.NET oldalak két külön fájlból állhatnak: az *oldalnév.aspx* és az *oldalnév.aspx.cs* (Visual C# programozás nyelv használata esetén) állományokból. Az első tartalmazza a *HTML*-t és a webes control-okat („webes vezérlőket”), míg a második a forráskódot.
- Ugyanannak a fájlnak egy *<script>* blokkjában – A *HTML* és a forráskód rész egyetlen fájlban is lehet, ekkor a forráskód egy kiszolgálóoldali *<script>* blokkba kerül.

Bár mindkét módszer elfogadható, én még is a szeperált változatot ajánlóm, hiszen így a HTML és a forráskód rész tisztábban elkülönül, ráadásul az ASP.NET korábbi változataiban más előnyei is voltak a két külön fájl használatának. Az ASP.NET weboldalak forráskód részét két progarmozási nyelv egyikén írhatjuk meg: Visual Basic vagy Visual C# nyelven.

Az ASP.NET weboldalak forráskód része az obejtumorientált programozás progarmozási modelljén alapszik, mely mint tudjuk az obektumokra épül. Minden ilyen programozási nyelvben az osztályok töltik be a legfontosabb szerepet; ezek írják le absztrakt módon az

obejtumokat. Az osztályok tulajdonságai (properties) írják le az objetumok állapotát, metódusai az obejtumokon végrehajthaó műveleteket, eseményeik pedig olyan műveleteket írnak le, amelyeket az objektumok indíhatnak el. Az objektumok az osztályok példányai, az absztrakció konkrét megvalósulásai.

Az ASP.NET weboldalak létrehozására használt programozási nyelvek fontos része az eseménykezelő. Az esemény, mint már ahogy az a fentiekben elhangzott, olyan művelt, amelyre az objektum használata során kerül sor, az eseménykezelő pedig az a kódrész, amelyet a megfelelő esemény bekövetkezésekor végre kell hajtani. Az ASP.NET weboldalakra gondolhatunk úgy, mint eseményvezérelt programokra. Az ASP.NET oldalak feldolgozásakor többféle esemény is bekövetkezhet.

Az ASP.NET-ben a weboldalak forráskód részében helyetfoglaló utasítások sorban, egymást követően hajtódnak végre. Az esemény vezérelt programozásban ez nincs feltételenül így. Sokszor nem tudhatjuk, milyen sorrendben fognak elindulni az egyes események, ezért a hozzájuk tartozó eseménykezelők hívási sorendje sem ismert. Ezen kívül abban sem lehetünk biztosak, hogy egy adott esemény egyáltalán bekövetkezik e. Amikor egy ASP.NET weboldalt készítünk, számos olyan eseményre feliratkozhatunk és hozhatuk létre hozzá eseménykezelőt, amely az oldal élete során bekövetkezik (Page life cycle). Ilyen esemény például az oldal *Load* eseménye, amelyre minden ASP.NET oldal lekérdezésekor sor kerül. Ha egy kódrészt minden alkalommal végre szeretnénk hajtani, amikor kérés érkezik egy adott oldalra, akkor használhatjuk a *Load* esemény (a feliratkozást követőleg létrejött) eseménykezelőjét.

Amikor egy ASP.NET weboldal betöltődik, az oldal végig hallad önmaga életciklusán (Page life cycle), mely számos feldolgozási lépést von magaután. Ezek közé tartozik az incilalizálás, állapot visszaállítás és annak karbantartása, eseménykezelők futtatása és a renderelés („viszonzás”). Fontos, hogy a fejlesztő tisztában legyen az oldal életciklusával, hiszen így kódrészletet helyezhet el az alkalmas és a logikának megfelelő életciklus állapotban. Egy control életciklusa az oldalak életciklusán alapszik, de az oldal több eseményt is kivált a tartalmazott control-okon, mint amennyi önmagán az oldalon kiváltódik.

3.2.5.1.2 Az ASP.NET életciklus állapotok

Az életciklus állapotok a következők:

- *Page request*:

  A page request az oldal életciklusának kezdete előtt következik be. Amikor az oldalra kérés érkezik a kliens felöl, az ASP.NET meghatározza, hogy szükséges e az oldalnak újrafordulnia (ezért is van az életciklus kezdete előtt), vagy az oldal egy tárolt (cached) változata lesz a válasz eredménye, az oldal újratöltődése nélkül.

- *Start*:

  Ebben az állapotban, az oldal olyan tulajdonságai mint a *Request* (kérés) és a *Response* (válasz) beállítódnak. Ekkor az oldal meghatározza *IsPostBack* tulajdonsáságnak értékét attól függően, hogy kérés az egy postback, vagy egy új kérés.

- *Page initialization*:

  Az oldal incializálása közben, az oldal tartlmazott control-jai elérhetőek, illetve mindegyikük *UniqueID* (egyedi azonosító) tulajdonsága is értéket kap. Ha az aktuális kérés egy postback kérés, akkor a postback adatok és control-ok tulajdonságai még nem tőltődnek be a *view state*-ből (kliens oldali állapotmentés eszköze).

- *Load*:

  Ezen állapot folyamatán belül, ha az aktuális kérés postback kérés, akkor a control-ok tulajdonságai és állapotai betöltődnek a *view state*-ből.

- *Validation*:

  A Validation (érvényesítés) állapot közben a *Validate* metódusa az összes validator control-nak meghívódik, melynek következtében beállítódnak az *IsValid* tulajdonságai mindenegyes validator controlnak, illetve magának az oldalnak is.

- *Postback* eseménykezelés:

  Ha a kérés postback, mindenegyes eseménykezelő meghívásra kerül.

- *Rendering*:

  Renderelés előtt, az oldal és az összes control mentődik a *view state*-be. Renderelés közben, az oldal mindenegyes tartalmazott control *Render* metódusát meghívja, illetve biztosít egy *text writer*-t, amely a controlok kimenetét fogja tartalmazni és ez az oldal *Response* tulajdonságának, *OutPutStream* tulajdonságába fog beleíródni.

- *Unload* („Ürités”):

  Az Unload állapotba kerülés akkor kezdődik meg, amikor az oldal teljes egészében kirenderelődött. Ekkor, az oldal összes tulajdonsága, az olyanok, mint a *Response* és a *Request* kiürülnek, illetve minden más adat kiürítése (kitisztítása) is végbemegy.

3.2.5.1.3 ASP.NET életciklus események

Mindenegyes életciklus állapoton belül az oldal eseményket vált ki, amelyet a fejlesztő kezelve, saját kódját tudja futtatni. Control-ok eseményei esetén a fejlesztő, vagy az attribútomokon (pl *onclick* attribútom), vagy kódon keresztül tud feliratkozni. Az ASP.NET weboldalak támogatják az automatikus esemény *wire-up mechanizmust*, ami annyit jelent, hogy amikor egy esemény kiváltódik, az ASP.NET megvizsgálja a metódusok specifikációját (nevét és paramétereit) és ha valamelyiket megfelelőnek találja, akkor lefuttatja azt, az adott esemény eseménykezelőjeként. Ha *@Page* direktíva *AutoEventWireUp* tulajdonsága be van állítva, akkor az oldal egyes eseményei automatikusan kezelve leszenek a megfelelő metódusok által.

Az életciklus események a következők:

- *PreInit*:

  A következő esetek valamelyikében érdemes a PreInit eseményre feliratkozni:
  - Az oldal *IsPostback* tulajdonságának vizsgálata, abból a célból, hogy kiderüljön, első alkalommal történik e meg a kérés az adott oldalra (természetesen a kliens szempontjából).
  - Dinamikus controlok (újra)létrehozása.
  - *Master page* dinamikus beállítása.
  - *Theme* tulajdonság dinamikus beállítása.
  - *Profile* tulajdonság olvasása, vagy írása.

  Fontos, hogy ha a kérés az postback kérés, akkor a control-ok értékei még ekkor nem töltődtek be a *view state*-ből. Ha a fejlesztő ebben az állapotban állítja be egy control valamely tulajdonságát, akkor ez az érték valószínűleg felül lesz írva a következő eseményben.

- *Init*:

  Akkor váltódik ki ez az esemény, amikor az összes control inicilaizálva lett. A control-ok tulajdonságainak inicilizálása, vagy olvasása esetén használandó.
- *InitComplete*:

  A Page objektum váltja ki. Olyan műveletek végrehajtása esetén használandó, amelyek megkövetelik, hogy mindenfajta incilizáció befejezett legyen.
- *PreLoad*:

Abban az esetben használandó, ha szükséges olyan műveletek végrehajtása az oldalon, vagy a tartalmazott control-okon, amelyeknek meg kell előznie a *Load* eseményt. Mielőtt a *Page* példány kiváltja az eseményt, betölti a *view state*-et önmaga és minden tartlamazott control-ja számára, majd feldolgoz minden postback adatot, beleértve a *Request* példányt is.

- *Load*:

  Az oldal meghívja önmaga *OnLoad* eseményének metódusát (eseménykezelőjét), majd rekurzívan megtörténik ez minden tartalmazott gyermek control-ra, illetve a gyermek control-ok gyermekére is, mindaddig még az oldal és az összes control be nem töltődik. Ez az esemény akkor használatos, amikor a fejlesztő egy control tulajdonságait szeretné beállítani illetve adatbázis kapcsolatot szeretne létrehozni.

- *Control* események:

  Akkor használatosak ezek az események, amikor control-ok specifikus eseményei kezelendőek, mint például egy *Button control*, *Click* eseménye, vagy egy *TextBox control*, *TextChanged* eseménye. Fontos, hogy egy postback kérés esetén, hogy ha az oldal tartalmaz *validator* control-okat, akkor ellenőrizni kell az *IsValid* tulajdonságát az oldalnak, és minden *validation* control-nak is, mielőtt bármilyen művelet végrehajtódna.

- *LoadComplete*:

  Ez az esemény akkor használatos, ha a fejlesztő olyan műveletet szeretne végrehajtani, amelynek előfeltétele az, hogy minden egyes tartalmazott control illetve maga az oldal is be legyen töltődve.

- *PreRender*:

  Mielőtt ez az esemény kiváltódna, a *Page* objektum meghívja az *EnsureChildControls* metódust az összes tartalmazott control-ra illetve önmagára is. Illetve minden egyes olyan control, amelyekhez adatokat lehet kötni (data bound control) és *DataSourceID* tulajdonsága értékkel rendelkezik meghívja saját *DataBind* metódusát. A PreRender esemény az oldal összes tartalmazott control-ján is kiváltódik. Ez az esemény akkor használatos, ha a fejlesztő az utolsó módosításokat szeretné elvégezni az oldal, vagy a control-ok tartalmán.

- *SaveStateComplete*:

Mielőtt ez az esemény kiváltódna, az oldal és a tartalmazott control-ok mindegyike elmenődik a *view state*-be. Minden az oldalhoz vagy a tartalmazott control-okhoz kötődő változás innenstől fogva nem érvényes, nem mentődik el. Ez az esemény akkor használatos, ha a fejlesztő olyan műveletet szeretne végrehajtani, amelynek előfeltétele, hogy a *view state* elmentődjön, de maga a művelet semmilyen változást nem eredményez az oldalon, illetve a tartalmazott control-okon.

- *Render*:
  Ez nem egy esemény, a kérés feldolgozásának eme állapotában, a *Page* objektum ezt a metódust hívja meg az összes tartalmazott control-on. Minden ASP.NET Web server control rendelkezik a *Render* metódussal, mely a control-ok azon kimenetelét állítja elő, melyek a böngészőnek lesznek válaszul elküldve. Ha a fejlesztő custom control-t készít, akkor tipikusan ezt a metódust definiálja felül (override), hogy előállítsa a control, a kliens programnak megfelelő kimenetelét. Ha a custom control csak a standard ASP.NET Web server control-jait foglalja magában, akkor nem szükséges a Render metódus felüldefiniálása.
- *Unload*:
  Ez az esemény kiváltódik minden egyes tartalmazott control-on illetve az oldalon is. Speciális control-ok esetén ez az esemény az utolsó „kitisztítás” lehetőségét hordozza magában, olyanokat mint például egy adatbázis kapcsolat (connection) lezárása. Maga az oldal szintén az utolsó „kitisztítást” végzi ennek az eseménynek a hatására, fájlokat, adatbázis kapcsolatokat zár le. Az unload állapot végrahajtódása közben az oldal és minden tartalmazott control már ki van rederelve, így ekkor már nincs lehetőség a válasz kimenetelének módosítására, így egy esetleges *Response.Write* metódus hívás az oldalon kivétel kiváltódását eredményezné.

3.2.5.2 ASP.NET 2.0 Client Callback sajátossága

Az alkalmazás jelentős része az ASP.NET ezen lehetőségének implementálásán alpszik, így fontosnak tartom a technológia ismertetését.

Az egyik legjobban figyelmen kivűl hagyott előnye az ASP.NET 2.0-nak, a Client Callback. Ez a fejlesztő számára lehetőséget biztosít, hogy szerveroldali metódusokat hívjon, kliensoldali *JavaScript* kódból, anélkül hogy az oldal teljes tartalmát post-olnia kellene. A *HTTP* protokol állapotmentes természetéből fakadóan, minden alkalommal, amikor a webes

felhasználónak igénye van szerver oldali adatok elérésre, vagy szüksége van olyan kódrészlet végrehajtására, ami a szerver oldalon kell hogy lefusson, első lépésben el kell hogy küldeje (submit) a webolad teljes tartalmát. *Postback* kérés esetén, az eseménykezelők végrehajtják a megfelelő kódrészletet és ezek után szolgáltatják az adatot a felhasználónak, akinek a (kliensoldali programja) bőngészője a weboldal teljes tartalmát úra kirendeli. Ez a fajta eseménykezelő modell a legtöbb esetben teljesen jól működik, de ez az ASP.NET fejlesztőkre néhány olyan megszorítást vet ki, amellyel meg kell tanulniuk együttélni. Ahhoz hogy a weboldal control-jainak állapota (weboldal tartalma) a kliens oldalt megmaradjon, a fejlesztőnek a mindent tároló *view state* infromációival kell dolgoznia (amely jelentősen lelassítja a weboldal tartalmának megérkezését), vagy komplex programozási logikát kell megvalósítania. Másodsorban a weboldal újrarenderelése feldolgozási időt igényel, a szervert terheli. Ez kis méretű weboldalak esetén nem számottevő, de összetettebb oldalak esetén, már a felhasználót is zavarhatja, olyan formában, hogy a böngésző tartalma villog. Ez fakadhat abból, hogy kis sávszélességgel rendelkezik, vagy abból hogy az oldal mérete esetlegesen túlzottan nagy, rosszabb esetben mindkettőből. Ezek miatt, a szerveroldali metódusok klienoldali kódból történő hívása, egy olyan lehetőség, melyet a webes fejlesztőknek még hosszú időre nem szabad elfelejtni. A jó hír az, hogy az ASP.NET rendelkezik ezzel a sajátossággal. Mielőtt az ASP.NET Client Callback sajátosságára terelődne a szó, nézzük mit tehetnek jelenleg a webes fejlesztők a fentemlített problémák legyűrésére. Szerveroldali metódus hívás *JavaScript*-ből egy olyan megoldás, amely használni tudja a Microsoft *XMLHTTP ActiveX* objektumát. Ez az *ActiveX* objektum lehetőséget biztosít, hogy *XML* fájlok utazzanak az Interneten keresztül a *HTTP* prtokollt használva. Ez az objetum használható bármilyen szerverhez (pl.:klasszikus ASP, PHP) címzett *HTTP* kérés indítására, amelynek hatására „nyers” *HTML* adatok érkeznek vissza a klienshez. Mivel az *XMLHTTP* objektum nagyából egy standard *ActiveX* objketum, így megvalósítható úgy, hogy *JavaScriptet* használjon. Nézzünk egy példát, mely a Google kezdő oldalának *HTML* kódját szolgáltatja:

```
function RetrieveGoogleFrontPage()
{
    var XmlHttp = new ActiveXObject("Msxml2.XMLHTTP.4.0");
    XmlHttp.Open("GET", "http://www.google.com", false);
    XmlHttp.Send();
    return XmlHttp.responseText;
}
```

Ahogy a példakódból látszik, az *XMLHTTP* objetum használata meglehetősen egyszerű. Egyszerűen megadható egy *URL*, ahhoz hogy kérést lehessen elindítani egy webszerver felé, melyre a szerver majd válaszol is. Mindez egy *JavaScript* függyvényben elhelyezhető, így az az oldal, amely ezt a kódrészletet tartalmazza, valójában nem fog újrarenderelődni (nem fog postback-elődni) a kérés indítását és feldolgozásást (a választ) követőleg. Vegyük észre azt is, hogy az *XMLHTTP* objetum a kérés teljesítését követőleg a teljes választ, karakterlánc formájában fogja tárolni. Ez azt jelenti, hogy ha egy business objektumot várunk válaszként, akkor szükséges egy olyan speciális weboldal megkonstruálása, amely a business objektumot magába csomagolja.

Maga az ASP.NET is az *XMLHTTP* objektummal dolgozik, mely a Client Callback működésének alapja, mindez viszont a webes felhasználó, illetve a fejlesztő elől el van rejtve. A Client Callback működéshez igazából két dolog szükséges: az *ICallbackEventHadler* interfész és a *Page.GetCallbackEventReference* metódus. Az architektúra maga, a következő alap lépésekből tevődik össze. A *Page.GeCallbackEventRefernce* metódus egy *JavaScript* kódtöredéket hozz létre, melyet a kliens oldalon kell elhelyezni. Ez a kód részlet fogja tartalmazni azt a *HTTP* kérést, mely használja az *XMLHTTP* objektumot. A kérés majd szerveroldalon lesz kezelve azzal a Web control-lal, amely implementálja az *ICallbackEventHandler* interfészt. Az esetek túlnyomórészében, ez a Web control, maga az oldal lesz, de természtesen Web control is megadható a kérés kezelőjeként. A kérés teljesítését követőleg a válasz egy kliensoldali *JavaScript* függvénynek adódik át, melynek egyedüli feladata hogy reagáljon a kérést követő válaszra.

Nézzünk egy konkrét pédát az alkalmazásból:

A CreateUser.ascx fájl tartalma (része):

```
function onBtnCreateReservationClick()
{
    ...
    if(isValid)
    {
        UseCallbackToReserve
        (
             certificateNumber+":"+
             name+":"+
             dateOfBirth+":"+
             country+":"+
             county+":"+
             city+":"+
```

```
                  street+":"+
                  phoneNumber+":"+
                  beginOfReservation+":"+
                  endOfReservation
            );
        }
    }

    function GetServerDataReservation(serverData, context)
    {
        if(serverData == "True")
        {
            document.getElementById('divMessage').innerHTML = "The
reservation is successfully created!";
            document.getElementById('<%= TableResult.ClientID
%>').style.display = "block";
            document.getElementById('<%=btnPopup.ClientID%>').click();
            message = "";
        }
        else if(serverData == "The room is already reserved for the given
interval. Please choose another room!" || serverData == "Invalid data!")
        {
            message = "Error";
            document.getElementById('divMessage').innerHTML = serverData;
            document.getElementById('<%= TableResult.ClientID
%>').style.display = "block";
            document.getElementById('<%=btnPopup.ClientID%>').click();
        }
    }
```

A CreateUser.ascx.cs fájl tartama (része):

```
public partial class Controls_CreateUser : System.Web.UI.UserControl,
ICallbackEventHandler
{
    private string mCallbackResult = string.Empty;

    protected void Page_Load(object sender, EventArgs e)
    {
        if (
                  !Page.IsCallback &&
                  !Page.ClientScript.IsClientScriptBlockRegistered
                        (
                         "reservation"
                        )
            )
        {
            //Register script block for callback
            string cbReference=Page.ClientScript.GetCallbackEventReference
            (
                  this,
                  "arg",
                  "GetServerDataReservation",
                  "context",
                  True
            );
            string cbScript = "function UseCallbackToReserve(arg,context){"
                                                      + cbReference + ";}";
            Page.ClientScript.RegisterClientScriptBlock
```

```
                (
                        this.GetType(),
                        "reservation",
                        cbScript,
                        true
                );
            }
        }
...
}

public string GetCallbackResult()
{
      return mCallbackResult;
}

public void RaiseCallbackEvent(string eventArgument)
{

//CertificateNumber:Name:DateOfBirth:Country:County:City:Street:PhoneNumber
:BeginOfReservation:EndOfReservation
      try
      {

          //Creating of Reservation object from eventArgument

          if (BusinessManager.InsertReservation(reservation))
          {
              mCallbackResult = "True";
             //Other necessary operations
          }
          else
          {
              mCallbackResult = "The room is already reserved for the given
interval. Please choose another room!";
          }
      }
      catch
      {
          mCallbackResult = "Invalid data!";
      }
}
```

Az első fontos dolog, hogy a *Controls_CreateUser* Web control implementálja az *ICallbackEvent* interfészt. Ennek az interfésznek igazából mindösszesen csak két metódusa van, nevezetesen a *RaiseCallbackEvent* és a *GetCallbackResult*. *RaiseCallbackEvent* lesz az a metódus, amely végrehajtódik a kérés megérkezését követőleg, ez állítja elő a választ karakterlánc (string) formájában, a fenti példa esetében attól függően, hogy a foglalás létrejötte megtörtént e. A *RaiseCallbackEvent* által megkonstruált karakterlánccal fog visszatérni maga a *GetCallbackResult* metódus. Ahhoz, hogy kliens oldalon létrejöjjön a megfelelő kód részlet, meg kell hogy hívodjon a *Page.GetCallbackEventReference* metódus a control *Page_Load* eseményén belül. Ez a metódus fogja létrehozni a megfelelő *JavaScript*

kód részletet, ami amikor meghívodik, kezdeményezni fogja a client callback-et. A *GetCallbackEventReference* metódusnak számos változata van, viszont midegyik rendelkezik azzal a paraméterrel, mely azt határozza meg, hogy melyik Web control fog a kérésre reagálni, melyik fogja kiszolgálni azt. Ebben a példában a *this* lesz első paraméterként átadva, így a kérést *a Controls_CreateUser* példány fogja kezelni. A második paraméter annak a *JavaScript* változónak a neve, amely a kérés paraméterét fogja tartalmazni (*arg*), a példa esetében a létrehozandó foglaláshoz kapcsolódó adatokat. A harmadik paraméter pedig annak a *JavaScript* függvénynek a neve lesz, amely a kérés teljesítését követőleg fog meghívódni, amikor az befejeződött. Ahhoz, hogy a callback-et elindító kód részlet, kliens oldalon megjelenjen, a létrejött *JavaScript* kódot egy kis módosítást kővetőleg kliens oldalon el kell helyezni. Magát a *GetCallbackEventReference* eredményét még egy *UseCallbackToReserve* függvénybe csomagoljuk, amely két paraméter vár, ebből az első a fontos, ez lesz a kérés paramétere, a foglalás adatai.

```
string cbScript = "function UseCallbackToReserve(arg, context){" +
cbReference + ";}";
```

Ezt követően, az így létrejött *JavacScript* kódot regisztrálnunk kell kliens oldalon, ezt a *Page.ClientScript.RegisterClientScriptBlock* metódussal tehetjük meg:

```
Page.ClientScript.RegisterClientScriptBlock(this.GetType(), "reservation",
cbScript, true);
```

Ennek az eredménye a kliens oldalon a következő lesz:

```
<script type="text/javascript">
function UseCallbackToReserve(arg,context)
{
    WebForm_DoCallback
        (
               'ctl00$MainContent$CreateReservation$ctl00'
               ,arg,
               GetServerDataReservation,
               context,
               null,
               true
        );
}
</script>
```

A *_DoCallback* egy ASP.NET 2.0-s beépített *JavaScript* függvény, mely a szerverhez callback kérést indít el. A Függelékben a 6.ábra az események lefutásának sorrendjét illusztráltatja callback *HTTP* kérés esetén.

Amikor a callback elkezdődik - melyet a kezdetek kezdetén a *Create reservation* gomb lenyomása idéz elő (*onBtnCreateReservationClick*) - az *arg* változó tartalmazza azt a paramétert, amelyet a szerveroldalon a *RaiseCallbackEvent* metódus megkap (a foglalás adatai). Fontos, hogy ennek az adatnak *string* típusúnak kell lennie, így ha a fejlesztő komplexebb adatokat szeretne a szerveroldalra eljuttatni, akkor szükséges annak karakterláncra történő átalakítása (serialization). A szerver által szolgáltatott adat szintén *string* típusú a *GetServarDataReservation* függvény első paramétere. A *GetServerDataReservation JavaScript* függvényt a fejlesztőnek kell elhelyezni az *ascx* fájlba. A *GetServarDataReservation* lesz az, amely az emlegetett első paraméterén (*serverData*) keresztül, értesül a foglalás létrejöttének állapotáról, és erről értesíti is a felhasználót.

A *context* változó (paraméter) egy érdekes változó. Habár a *JavaScript* függvényben paraméterként megjelenik, mégsem jut el szerver oldalra, hiszen mint látható a *RaiseCallbackEvent* metódus egyetlen egy paraméterrel, a kérés paraméterével dolgozik. Ellenben a *context* változó értéke lementődik a böngészőben (*cache*) és majd a választ fogadó *JavaScript* függvénynek második paraméterként átadódik. Ez lehetőséget biztosít arra, hogy azonosítsuk a callback hívás környezetét. Képzeljük el azt a helyzetet, amikor számos callback kérés indítására van szükség, amelyek különböző események hatására történnek meg, vagy konkurens callback kérések indítására van szükség, melyek ugyanazon az esemény hatására következnek be. Ezekben az esetekben nem garantált az, hogy a kérés eredményét kezelő *JavaScript* függvény ugyanabban a sorrendben fog meghívódni, mint ahogy a kérések el lettek indítva. Ekkor használható a *context* változó, mellyel egyedi módon megjelölhetőek a callback kérések és ezzel megállapítható, hogy melyik kérés eredménye érkezet vissza kliensoldalra. Fontos továbbá azt is megjegyezni, hogy a callback kérések aszinkron módon futnak le, melynek következtében, ha esetlegesen szerver oldalon a válasz eredményének kiszámítása több másodpercet, esetlegesen több percet is igénybe vesz, a kliensoldali böngésző nem lesz blokkolva. Természetesen, ha webes felhasználó egy másik oldalra megy át, akkor a választ kezelő *JavaScript* függvény nem fog meghívódni, de ettől függetlenül a *RaiseCallbackEvent* metódus meghatározza számításának eredményét.

4 Reservation Manager alkalmazás működése

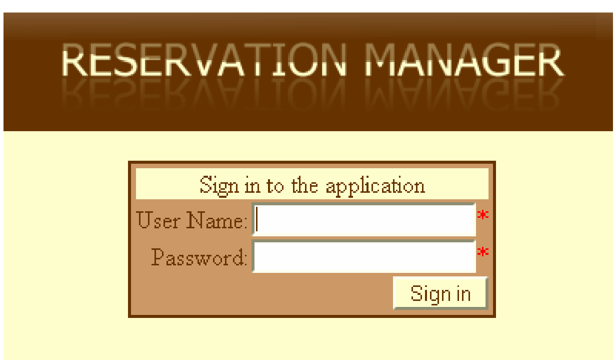
4.1 Portás felhasználókörrel történő működés

4.1.1 Felhasználó authentikációja

Minden felhasználónak, így a portásnak is egy bejelentkező felületen (*Login.aspx*) keresztül kell megadnia felhasználónevét és jelszavát. Csak érvényes adatok megadását követőleg tud a portás, felhasználókörének megfelelő feladatokat ellátni. A rendszer az adatbázisban két portást tart nyílván.

- Az egyik adatai:
  - Felhasználónév: porter01
  - Jelszó: porter01
- A másik adatai:
  - Felhasználónév: porter02
  - Jelszó: porter02

A bejelentkezési felületetet a következő ábra szemlélteti:



7.ábra Reservation Manager bejelentkező felület

A felület rossz adatokat megadása esetén tájékoztatja a felhasználót. Ugyanezzel az oldallal találkozik minden, az alkamazást használni kívánó – legyez az admin, vagy igazgató szerepkörrel rendelkező – felhasználó is.

4.1.2 A portás főoldala

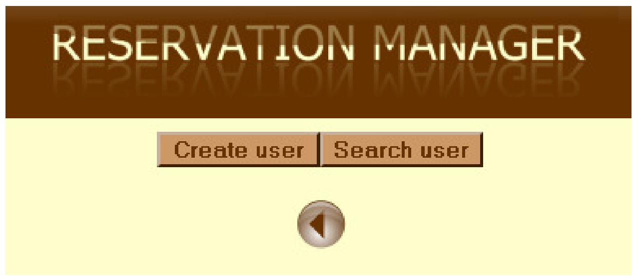
Sikeres authentikációt követőleg a portás a *Porter.aspx* oldalra lesz átirányítva, melyen egy menüből tudja kiválasztani, hogy mit szeretne végrehajtani. A menüt a következő ábra szemléltei:

8.ábra Portás menüje

Három menüpontból lehet választani:

- New reservation:

  Ezt a menüpontot akkor kell, hogy kiválassza a portás, ha új foglalás szeretne létrehozni. Ezt követőleg eldöntheti, hogy már létező ügyfélhez szeretne e felvenni foglalást, vagy egy, a rendszerben még nem létező új ügyfélhez.

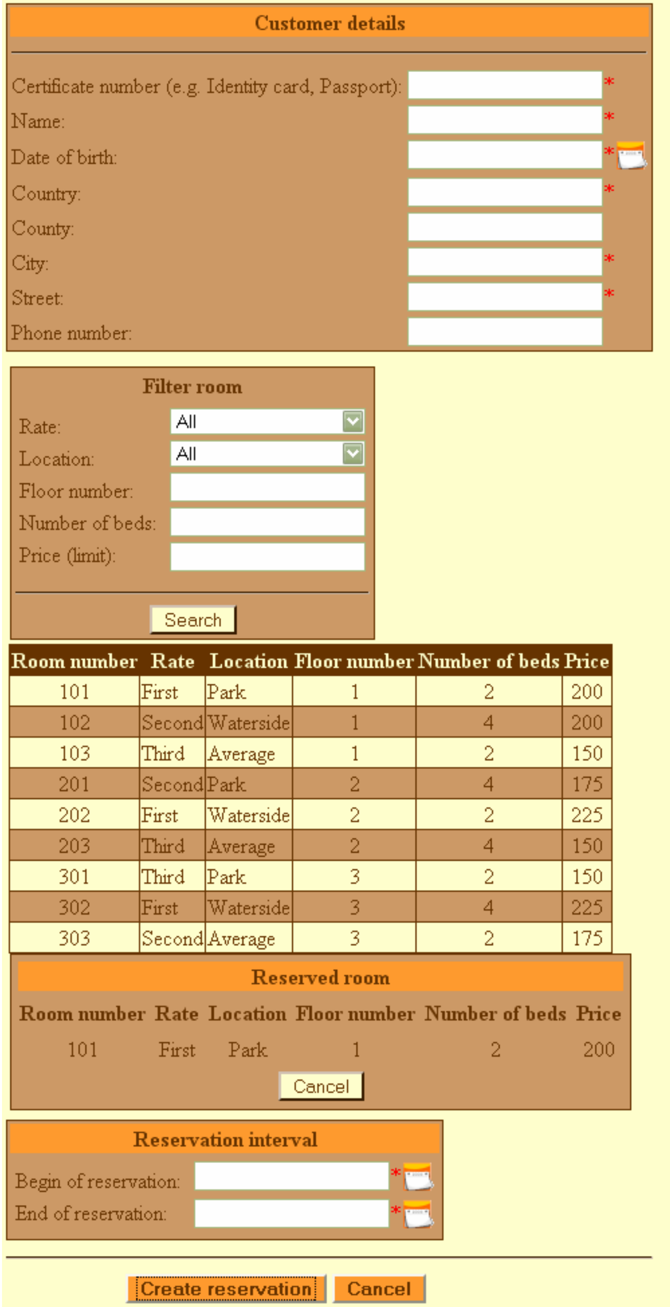


9.ábra Ügyfél választás

  - Új felhasználó és hozzá kapcsolódó foglalás létrehozása:

    Első lépésben az ügyfél adatait szükséges magadni. Majd ezt követően az ügyfél igényeinek megfelelő szűrőfeltételek alapján szoba keresését kell végrehajtani. Öt fajta szűrőfeltétel van: felszereltség, elhelyeszkedés (pl.: parkra néző), emeletszám, ágyszám és ár. Ha nincs megadva szűrőfeltétel, akkor az összes szoba ki lesz listázva. A felület, az esetlegesen rossz szűrük megadásáról értesíti a felhasználót, illetva arról is, ha nincs a feltételeknek megfelelő szoba a szállodában. A szobák kilistázását követőleg, a portásnak a táblázatból ki kell választani az ügyfél számára legmegfelelőbb szobát, a táblázat *Reserve* oszlopában lévő gombra történő kattintással. Ezek után már csak a foglalás idejének kezdetét és végét kell megadnia, és meg is történhet a

foglalás létrehozása, a *Create reservation* gombra történő kattintással. Az előbb említett gomb mindaddig inaktív még a szoba kiválasztása meg nem történt. A oldal bemeneti adai, ellenőrzésre kerülnek az elküldésük előtt, azért hogy kiderüljön, hogy minden szükséges adatt meg van e adva; ha nincs, akkor piros csillag jelnik meg azon beviteli mezők mellett, melyeknek kötelezően adatot kellett volna tartalmaznia.



10.ábra Foglalás létrehozása új ügyfélnek

Az adatok elküldést követőleg, visszajelzést kap portás a művelet végrehajtásának sikerességét illetően.

11.ábra Nem megfelelő adat hibaüzenet

Ez az üzenet (11. ábra), akkor jelenik meg, ha rossz adatok lettek megadva, értem ezalatt például, hogy nem megfelő formátumban lett megadva egy dátum, amely mellesleg csak akkor történhet meg, ha a felhasználó a dátum adatot kézzel beírva adja meg, nem pedig a Calendar popup-ot használva. Ez kerülendő, de nem tilos. Az üzenet megjelenését követőleg, az *OK* gombra történő kattintás után orvosolható a probléma.

12.ábra Foglalt szoba hibaüzent

A következő üzenettel (12. ábra), akkor szembesülhet a felhasználó, ha olyan szobát és foglalási időt adott meg, mely ütközik egy másik, már létező foglalással. Ekkor az *OK* gombra történő kattintást követőleg, a probléma orvosolható.

13.ábra Sikeres foglalás üzenet

Végül ez az üzenet (13. ábra) akkor jelenik meg, ha a foglalás minden hiba és probléma nélkül létrejött. Ekkor az *OK* gombra történő kattintást követőleg, a portás a főoldara lesz átírányítva, ahol a menüpontokból kiválaszhatja, mi az a következő feladat, amit véghez szeretne vinni.

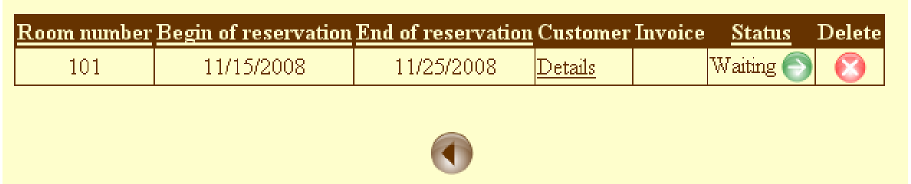
- Létező ügyfélhez történő foglalás létrehozása

  Ez az előzőekben elmondott munkamenettől abban különbözik, hogy itt az ügyfél adatait használva szűrőfeltételül, lehet az ügyfeleket kilistázni. Ekkor a megjelnő ügyfelek küzül – ha többet is vissza adott a keresés - ki kell választani a megfelelőt (a szóban forgót), és majd hozzá lehet megkonstruálni a foglalást, melynek a mechanizmusa a fent tárgyalt módon zajlik le. Viszont annyit még meg kell említeni, hogy a szobák kilistázásakor, azok a szobák fognak a táblázat elején megjelenni, amelyeket esetlegesen már egyszer, vagy akár többször is lefoglalt az adott ügyfél.

14.ábra Ügyfél kereső

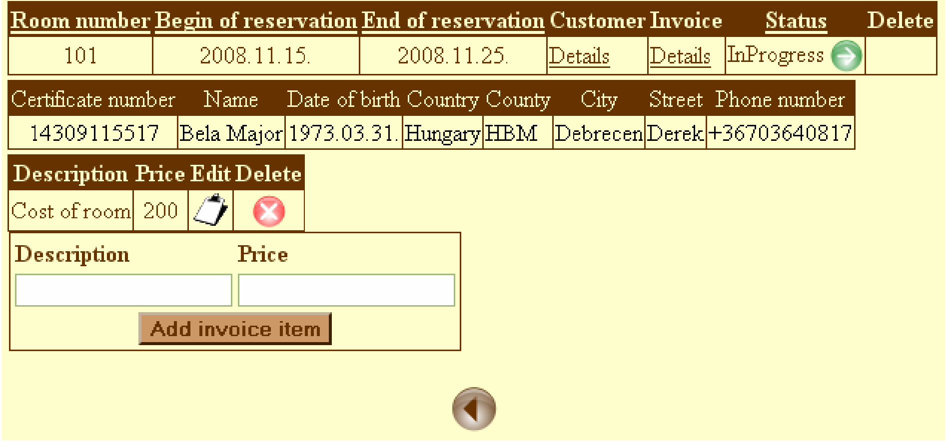
- Current customers

  Ez a menüpont akkor használandó, hogy ha a portás az éppen aktuális foglalásokat szeretné megtekinteni.



15.ábra Aktuálsi foglalások

A táblázat *Customer* oszlopán belül a *Details* linkre kattintva az ügyfél adatai jelennek meg. A *Status* oszlopon belül az adott foglalás állapota változtatható, illetve látható. Amikor egy foglalás létrejön akkor várakozó állapotba kerül (waiting), ha a foglalás kezdetének a napján megérkezik az ügyfél, szobáját átveszi, akkor folyamatban (inProgress) lévő állapotba billenthető át. Amíg várakozó állapotban van a foglalás, addig törölhető. Foglalás törlésére olyan esetekben kerülhet sor pédául, amikor azt lemondták, vagy amikor az ügyfél nem jelenik meg a foglalás kezdetének napján délig. A törlést, a foglalások listáját tartalmazó táblázat *Delete* oszlopában lévő gombbal lehet véhez vinni. Ha a foglalás *inProgress* állapotba kerül, akkor lehetőség van az *Invoice* oszlopon belül a *Details* linkre kattintva a foglaláshoz kapcsolódó számlatételek menedzselésére.



16.ábra A fogalalás számlatételei

Miután a foglalás *inProgress* állapotba került, alapból már egy számlatétel kapcsolódik hozzá, mely a szoba költségét hordozza magában (szoba költség * lefoglalt napok száma). Ez követőleg újabbak is megadhatóak hozzá. Amikor egy foglalásnak vége van, az ügyfél távozik és a számláit kifizette, akkor tehető át a foglalás fizetett (paid) állapotba, ezt követően már semmilyen módosítás nem hajtható végre rajta.

- Rooms

  Ezen menüpont arra használható, hogy a portás megnézze, hogy az adott napon, milyen a szálloda kihasználtsága. A szobák listáját és hozzájuk kapcsolódó adatokat tekintheti meg egy táblázat formájában. A táblázat rendezhető az oszlopok szerint.

Room number	Location type	Rate	Number of beds	Floor number	Price	Status
101	Park	First	2	1	200	The room is reserved on today
102	Waterside	Second	4	1	200	The room is free on today
103	Average	Third	2	1	150	The room is free on today
201	Park	Second	4	2	175	The room is free on today
202	Waterside	First	2	2	225	The room is free on today
203	Average	Third	4	2	150	The room is free on today
301	Park	Third	2	3	150	The room is free on today
302	Waterside	First	4	3	225	The room is free on today
303	Average	Second	2	3	175	The room is free on today

17.ábra Szobák aktuális állapota

4.2 Szállódaigazgató felhasználókörrel történő működés

Mint már említettem, minden felhasználónak be kell jelentkezni így a szállodaigazgatónak is. Az adatbázisaban egyetlen egy szállodaigazgató szerepkörrel bíró felhasználó van regisztrálva, adatai a következők:

- Felhasználónév: director01
- Jelszó: director01

Ehhez a szerepkörhöz egyetlen egy oldal tartozik, ahol egy időintervalumot megadva lehet a szobákat kilistázni, mellyel figyelemmel kísérheti a szállodaigazgató a szálloda kihasználtságát. Az intervallum szerinti keresés eredménye, egy szobák listáját tartalmazó táblázat – hasonló a portás esetében (17. ábra), a Rooms menüpontnál tapasztalhatóhoz - , mely oszlopai szerint rendezhető, ezzel könnyen megállapítható az is, hogy mely szobák a legkeresettebbek.

4.3 Admin jogosultsággal történő működés

Az adatbázisban egyetlen egy admin szerepkörrel bíró felhasználó van regisztrálva, adatai a következők:

- admin01
- admin01

Ehhez a szerepkörhöz egyetlen egy egyszerű oldal tartozik, melyen keresztül az adatbázis adatai módosíthatóak:

- Új szobák felvétele (ritka eset, de sose lehet előre tudni).
- Létező szoba adatainak módosítása.
- Új felhasználók létrehozása.
- Létező felhasználók adatainak módosítása.

4.4 Konzisztencia biztosítása

A rendszerben a konzisztencia biztosítása akkor fontos, amikor új adatok kerülnek be a rendszerbe, és ezeket egyszerre többen is megtehetik (két portás). Két esett van, amely kritikusnak tekinthető és kezelendő.

- Foglalás létrehozása:

  Ebben az esetben a rendszeren belül, egy időpontban csak egy foglalás hozható létre (*lock*). Így ennek következtében biztos, hogy nem áll elő olyan eset, hogy úgy létezik két foglalás ugyanarra a szobára vonatkozóan, hogy időintervallumuknak lenne metszéspontja.
- Foglalás állapotának változtatása:

  Minden állapotváltoztatás előtt az adatbázis tartalma és a felhasználó felület tartalma összehasonlításra kerül, ha ezek megegyezőek akkor megtörténik az állapotváltás. Abban az esetben, amikor az előbb említett adatok különböznek, az azt jelenti, hogy már valaki módosította az adott foglalást, így be kell frissülnie a böngésző tartalmának. Ezt követően a felhasználó eldöntehi, hogy mit szeretne az adott helyzetben tenni. Fontos, hogy a rendszeren belül egyszerre csak egy foglalás állapotváltása zajlik le (*lock*).

A foglalás számlatételeinek módosítása, nem kíván összetettebb munkamenetet, hiszen mindig a legutolsó modosítás a maradandó, és ez bármikot végrehajtható.

5 Összegzés

Az olvasottakból látható, hogy egy webes alkalmazás elkészítésekor nagyon sok dologra - legfőbb képpen az architektúrára - kell odafigyelni, amelynek jelentősége az alkalmazás bonyolultságával egyre fontosabbá vállik. Az én meglátásom szerint egy alkalmazás élete nem a megvalósítás kezdetével, hanem a megrendelő igényének felmerülésével indul el. Sose szabad szemelől téveszteni azt, hogy megrendelőnk nincs teljesen tisztában azzal, hogy mit is szeretne; benne megfogalmazódik az igény egy esetleges probléma megkönnyítésére, vagy annak orvoslására. Így amikor egy megrendelést kapunk, első dolgunk a probléma lehető leggondosabban történő körbejárása kell hogy legyen. Természetesen, ez bizonyos esetekben csak a megrendelővel történehet meg (igények összegyűjtése). Ekkor kerül előtérbe a jó kommunikáció, amelyhez robusztusabb alkalmazások esetén megfelelő embereket lehet és kell alkalmaznunk (Business Analist). Az említett kommunikációnak állandóan fent kell állnia köztünk és a megrendelő között. Ezzel biztosított az, hogy mindig a jó úton follyon a fejlesztés. Mindig törekedjünk az igények lehető legjobban történő kielégítésére, de e mellett mindig a lehető legtisztább megoldást válasszuk és ha esetlegesen kell, akkor merjünk nemet mondani megrendelőnknek (részletkérdések esetén). Ne felejtsük el, hogy a web bár egyre több és több lehetőséget nyújt a felhasználók számára, mindent megvalósítani vele sem lehet.

Egy jól működő alkalmazás lelke az architektúrában keresendő. Nagyon sok tervezési séma létezik, ezek küzül nekünk kell kiválasztani, hogy melyik az, amely számunkra a legérthetőbb és a legkézenfekvőbb. Én személy szerint a többrétegű alkalmazás mellé teszem le voksom, de ezzel nem azt állítom, hogy makulátlan tervezési modell. Biztos vannak olyanok, akik számára ez kevésbé emészthető. Valószínű, hogy az évek múltával már én is valami újabb tervezést választok majd egy alkalmazás elkészítésekor. Hiszen az informatika és web is egy állandóan fejlődő és állandóan megújúlni képes „egyed". Nagyon sokszor meglepődök, hogy mennyi új technológia létezik, pedig én legfőbbképpnen csak a webes technológiákat próbálom figyelemmel kísérni. Az új technológia új megoldásokat, új pattern-eket von magaután. Így az, hogy milyen modellt választunk egy szubjektív dolog. Mivel a legtöbb alkalmazás fejlesztése nem ott ér véget, hogy azt megrendelőnknek átadjuk, (hiszen a support-ra is gondloni kell), a legfőbb cél, hogy alkalmazásunk bővítése, módosítása a lehető legegyszerűbben történjen meg. Én ezt szemelőtt tartva választottam a három rétegű architektúrát. Ezzel a modellel nagyon jól elkülöníthetőek a funkcionálisan különböző

rétegek, melynek következtében sokkal átláthatóbb és így később könnyebben is módosítható a rendszer.

A modell választását követően a technológia kiválasztás kell, hogy megtörténjen. Ebben az esetben is dúskálhat a fejlesztő a lehetőségek között. Én személy szerint nem mondom azt, hogy van olyan technológia, amely a legjobbnak mondható. Mindegyiknek vannak buktatói és olykor nehezen emészthető sajátosságai. Az hogy én az ASP.NET –et válaszottam a sor eredményei is, hiszen apróbb munkáim során ezt kellett használnom. Régebben - bár csak felszinesen - foglalkoztam a J2EE lehetőségeivel is. Láttam benne is sok érdekes dolgot. Amit észrevettem, hogy nincs új a nap alatt. Nagyon sok közös vonás és elgondolás lelhető fel például az ASP.NET és Java Server Faces között is. Az természetesen megint más, hogy milyen a megvalósítás. Hozzám az ASP.NET áll a legközelebb, de nem vettem el más technológiák megismerését sem.

A Reservation Manager alkalmazás a célul kitűzött feladatokat képes ellátni, de a fejlesztés közben újabb és újabb ötletek jutottak eszembe, melyeket egy következő verzióba már beletenném. Ezek közül a legfontosabb az lenne, hogy a rendszer ne csak egy szálloda adatait, hanem többjét is kezelje, hiszen ekkor lenne igazából a webes technológia teljesen kihasznált. Meglátásom szerint, azzal hogy csak egy szálloda adatai kezeli a rendszer egy vastag kliens is készíthető lett volna. Viszont ha több szálloda használná a rendszert, akkor valódi értelmet kapna a web, hiszen egy ország bármely szállodájának csak egy számítógéppel és egy böngészővel kellene rendelkeznie, nem kell minden kliens gépén alkalmazást telepíteni (vastag kliens esetén). Természetesen egy szálloda mellett is vannak előnyei az alkalmazásnak, mint például az, hogy tetszőleges operációs rendszeren keresztül elérhető (csak egy böngésző kell, hogy fusson rajta). Ami viszont biztos, hogy a fent említett módosítás megvalósítása tisztán és könnyen megvalósítható lenne, hála a rétegelt architektúrának. Manapság egyre több és több böngésző száll versenybe a piacon, ami jót tesz a piacnak, de a fejlesztők lelkivilágának nem. A probléma ott keresendő, hogy a böngészők megjelenítése nem szabványszerű. Így nem garantálható hogy kinézetügyileg a program minden böngészőben megfelelően jelenik meg. Személy szerint én két böngészőre figyeltem oda (Internet Explorer 7.0, Mozilla Firefox 3.0), és előre is elnézését kérem azoknak akik nem ezen böngészők valamelyikét használják. Remélem a jövőben már lesz, egy szabvány, amelyet minden valamirevaló böngészőnek be kell tartania.

Irodalomjegyzék

[1] George Shepherd, *Microsoft ASP.NET 2.0*, 2006

[2] Juval Löwey, *.NET komponensek*, 2004

[3] Nicholas C., Jeremy McPeak, Joe Fawcett, *Professzionális Ajax*, 2007

[4] Stephen Walther, ASP.NET 3.5 Unleashed, 2008

[5] Damon Armstrong, Pro ASP.NET 2.0 Website Programming, 2005

[6] Internetes oldalak:

http://www.dotnetjunkies.ddj.com/Article/E80EC96F-1C32-4855-85AE-9E30EECF13D7.dcik

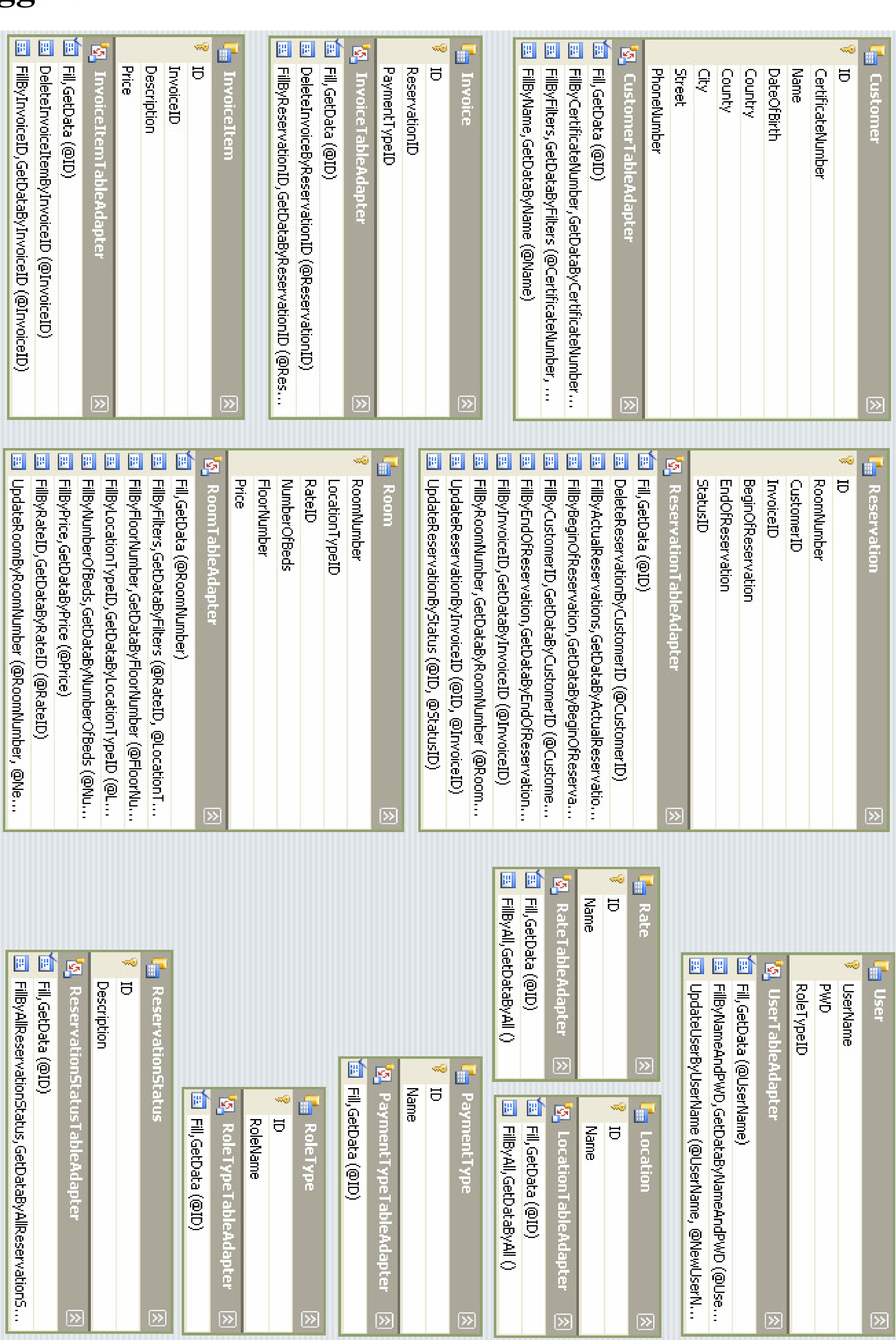
http://msdn.microsoft.com/en-us/library/ms178472.aspx

http://imar.spaanjaars.com/QuickDocId.aspx?quickdoc=416

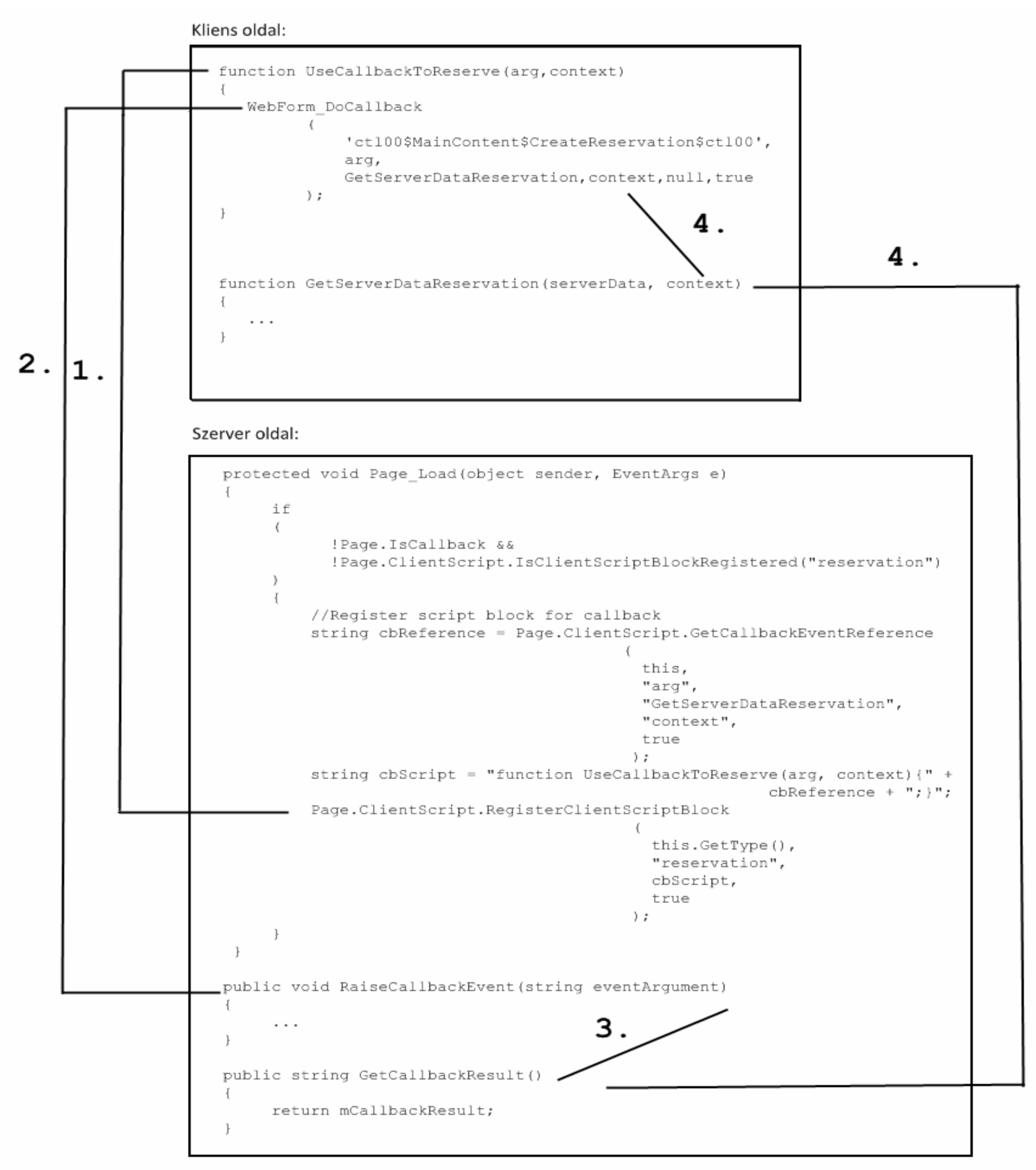
http://www.asp.net/learn/master-pages/tutorial-01-vb.aspx

http://www.asp.net/ajax/ajaxcontroltoolkit/samples/

Függelék



3.ábra Reservation Manager DataSet



6.ábra Callback HTTP kérés menete foglalás létrehozásánál