Debreceni Egyetem

Informatika Kar

Online jelentkezési rendszer diákköri konferenciához

Témavezető
Dr. Végh János
Egyetemi tanár

Készítette
Kocsis Dániel
Programtervező Matematikus szakos hallgató

Debrecen
2010

Tartalomjegyzék

TARTALOMJEGYZÉK **3**

1. BEVEZETÉS **4**

2. KEZDETI LÉPÉSEK **5**

2.1 AZ EDDIGI GYAKORLAT 5
2.2 KÖVETELMÉNYEK 7
2.3 A RENDSZER HATÁRAI 8
2.4 FOGALMAK ÉS SZEREPKÖRÖK 9
2.5 AZ *OPENCONFERENCE* RENDSZER 10
2.6 KAPCSOLAT A *'KISBAGOLY'* RENDSZERREL 11
2.7 SZOFTVER DESIGN 12

3. A FEJLESZTÉS FŐBB ELEMEI **15**

3.1 AZ ADATBÁZIS 15
3.2 ADATELÉRÉSI OSZTÁLYOK 18
3.3 A WEBES ALKALMAZÁS FELÜLETEI 27
3.4 AUTENTIKÁCIÓ, AUTORIZÁCIÓ 38
3.5 AUTOMATIKUS E-MAILKÜLDÉS 40
3.6 RIPORTÁLÁS, NYOMTATÁSI LISTÁK 42

4.AZ ALKALMAZÁS ÁTTEKINTÉSE **45**

5. ÖSSZEFOGLALÓ **47**

6. KÖSZÖNETNYILVÁNÍTÁS **48**

7. FÜGGELÉK **49**

7.1 A FEJLESZTÉS SORÁN HASZNÁLT ESZKÖZÖK 49
7.2 TELJES ADATBÁZIS DIAGRAM 50
7.3 IRODALOMJEGYZÉK 51
7.4 TELEPÍTÉSI ÉS FEJLESZTÉSI ÚTMUTATÓ 52

1. Bevezetés

Egészen 2008-ig nyúlnak vissza azok a gyökerek, melyek jelentős hatást gyakoroltak rám, és hozzájárultak, hogy elkészülhessen ez a diplomamunka. Ekkor kerültem kapcsolatba egy projekttel, mely két akkori hallgatótársam nevéhez fűződött. Nevezetesen arról volt szó, hogy egy Dr. Végh János által vezetett kurzus keretein belül, ASP .NET alapú web alkalmazást kell fejleszteni az OTDT számára. Akkor még nem sokkal tudtam többet ennél, de már ez is elég volt, hogy felkeltse érdeklődésemet a téma iránt. Szerencsére nem sokkal később, bár nem, mint a kurzus hallgatója, hanem csak, mint lelkes diák, de sikerült bekapcsolódnom a fejlesztésbe. Maga a készülő rendszer nem volt más, mint egy on-line jelentkezési és adminisztrációs szolgáltatásokat biztosító webes alkalmazás. Funkcionalitása kiterjed a hallgatók OTDK-ra történő nevezésére, a pályamunka adatok felvitelére, szekciók, intézmények karbantartására, az országos verseny eredményeinek rögzítésére, valamint különféle program által generált kimutatások és egyéb dokumentumok letöltésére és nyomtatására. A lista persze folyamatosan változott és változik jelenleg is, mivel maga a rendszer jelenleg is fejlesztés alatt áll, és valószínűleg ez még sokáig így lesz.

Ha alaposan szemügyre vesszük az előbbi felsorolást, láthatjuk, hogy a jelentkezések és az eredmények felvitele között egy meglehetősen nagy űr tátong, mely szinte magától értetődővé teszi egy olyan funkcionalitás megvalósításának igényét, amely a jelentkezett pályamunkák elbírálásában segédkezne. Ezen gondolatmenet egyenes következményeként született meg a jelen diplomamunka témája. A megoldandó feladat tehát egy olyan továbbfejlesztése a már létező rendszernek, mely ezek után képes lesz segíteni a bírálati folyamatban résztvevők munkáját, legalább olyan hatékonysággal, mint ahogyan azt a nevezés esetében elértük. Azonban a képhez az is hozzátartozik, hogy az OTDK nem vall egységes irányelvet a bírálati folyamatok lebonyolításának terén (sem). Szekciónként eltérő adminisztrációs eljárások léteznek, eltérő szokások alakultak ki az évek folyamán. Nem lehet tehát egyetlen munkafolyamatot ráerőltetni a különféle szakterületekre, így nem tehetjük meg azt, hogy akár hasra ütéssel, akár kutatómunkával keressünk egy optimális lebonyolítási elvet és implementáljam azt, mint kötelezően használandó eszközt. Éppen ezért nem kívánom, hogy az általam fejlesztett eszköz minden egyes szakterület eltérő követelményének megfeleljen, csupán egy lehetőség, mely egy általánosnak és logikusnak mondható követelményrendszerre épül, és bizonyos kompromisszumokkal jól használható a kívánt célra.

Nem egy alapjaiban új fejlesztés az elbírálói rendszerem, hanem egy modulja egy jelenleg is létező rendszernek, azonban sokszínűségét és funkcionalitását tekintve akár önálló rendszerként is tekinthetnénk rá. A szülő rendszer, mint azt említettem is, elsősorban két hallgatótársam érdeme, nevezetesen Miskolczi Zsolté és Szatmári Andrásé. Kettejük közül Zsolt már végzett a tanulmányaival és Szakdolgozatában a 'Kisbagoly' rendszer általa fejlesztett részét tárgyalta. A jobb érthetőség kedvéért, ahol szükséges, hivatkozni fogok az általa írottakra, bizonyos elemeket átveszek tőle, de ezt az adott helyen jelezni fogom.

2. Kezdeti lépések

2.1 Az eddigi gyakorlat

A következőkben nem egy általános gyakorlatot szeretnék bemutatni, hanem az Informatika szekció bírálati folyamatát. Azért erre esett a választásom, mivel egyrészt természetesen hozzám is közel áll a terület, másrészt mert témavezetőm részt vett az Informatikai Szekció országos versenyének és a debreceni döntőnek is a szervezésében, lebonyolításában, így magától értetődő módon erre a folyamatra van a legnagyobb rálátásom, és itt tudok a legtöbb segítséget kapni a felmerülő kérdések megválaszolásában.

A 2008/2009-ben megrendezett XXIX. OTDK előtt még regisztrációs rendszer sem létezett, csupán szekciónként találhattunk különböző weboldalakat, melyek különféle szolgáltatásokat kínáltak, információkat szolgáltattak az aktuális eseményekről. Eltekintve ezektől, csupán hagyományos postai úton zajlott a hivatalos kommunikáció, amely úgy nézett ki, hogy az OTDT titkárság levelezett az intézményekkel, levelezett a versenyzőkkel, a szekciók felelősei leveleztek az intézményekkel, a bírákkal és még sorolhatnánk. Ennek egyenes következménye, hogy az adminisztráció lassú volt, körülményes, és meglehetősen költséges, ha belegondolunk, nem lehetett ritka, hogy bizonyos dolgok napokat, heteket késhettek pusztán csak azért, mert folyton egymásra várt mindenki. A XXIX. OTDK-ra aztán megszületett a 'KisBagoly' rendszer, mely sok terhet levett a verseny lebonyolításáért felelős OTDT titkárság válláról, annak ellenére is, hogy nem nevezhettük éppenséggel ideálisnak a fejlesztés körülményeit. A bírálat azonban még mindig a hagyományos úton haladt, és jellemző volt mindaz rá, ami a nevezésre is, vagyis lassan és drágán haladt a dolog.

Ez azt jelentette a gyakorlatban, hogy a szekció működéséért felelős személyek felügyelete mellett, különféle táblázatok alapján, megpróbálták beosztani, hogy minden egyes intézmény, mely tegyük fel *n* db pályamunkát nevezett a versenyre, körülbelül *2n* mennyiségű bírálati igényt kapjon. Ez azért volt így, mivel minden egyes pályamunkát alaphelyzetben kettő bírálattal kell ellátni, különböző intézmények bírálóitól. Ha megérkeztek a bírálati igények az intézményekhez, akkor az Intézményi felelős bírálókat keresett hozzá, akik megfelelő rálátással bírtak az adott szakterületre, és ebből következően megfelelő színvonalú bírálattal tudták ellátni azt. Természetesen létezik olyan, hogy valaki összeférhetetlenség vagy egyéb szakmai okok miatt elutasít egy bírálati felkérést. Ilyenkor meg kell oldani az új bíráló felkérését, akár a szekció rendezőjének közbenjárásával. Ha elkészültek a bírálatok a pályamunkákra, fennállhatott az a helyzet, hogy nagyon nagy eltérés mutatkozott a két bírálat között. Ekkor vagy konszenzusra került a sor, vagy egy harmadik döntőbírót kellett felkérni a végleges pontszám kialakítására. Mint láthatjuk, elég sok szereplő vesz részt a munkában, és elég sok olyan pont van, ahol elhalhat az addigi folyamat, és kezdhetnek mindent elölről, ami időbe és pénzbe kerül.

Természetesen a bírálat azt jelenti, hogy a bíráló elolvasva vagy éppenséggel megtanulmányozva a pályamunkát valamiféle értékelést ad róla. Ez konkrétan szöveges és pontszámi értékelésként is megjelenik. Azonban ahhoz, hogy elolvashassa, egy bekötött, kinyomtatott pályamunkát el kell küldeni számára, ami szintén plusz költséget és postai forgalmat jelent. Ha megszületnek a szekciókban a végleges pontszámok, akkor ezekből táblázatokat kell készíteni manuálisan, melyeket a szekciónként megrendezett országos versenyeken a zsűri tagjai számára elérhetővé kell tenni, hogy figyelembe tudják őket venni, mikor megtörténik a pályamunka bemutatása. Természetesen, ahány ember, annyi féle elképzelése van arról, milyen rendszer szerint kell nyilvántartani ezeket az adatokat, így aztán meglehetősen heterogén struktúrájú és formátumú adathalmaz készült el a bírálatok gondozására, ami azt jelentette, hogy közzétenni sem egy egyszerű feladat volt ezeket. Ezek után azonban már nincs más hátra, csak a zsűri döntésének adminisztrációja és a díjak kiosztása.

Végigtekintve a fent leírtakat adódnak a következő észrevételek: a pályamunkák postai úton történő küldözgetése lassú és drága, sok-sok különféle táblázat, melyek csak a felelősök számítógépein léteznek, a határidők betartásának nehézségei, nehézkes kommunikáció, rengeteg hibalehetőség. Amikor tehát megfogalmazzuk a rendszerrel szembeni elvárásokat,

akkor a most felsorolt problémákra igyekszünk megoldásokat keresni, minimalizálva ezek hatását.

2.2 Követelmények

A következőkben leírtak nem tekinthetőek végleges és teljes követelményrendszernek, csupán a Dr. Végh János és köztem lezajlott egyeztetések egyfajta eredményének. A rendszer az itt leírtak alapján készült el, azonban mindvégig ismert volt a tény, hogy ha egyszer ténylegesen használatba kerül a modul, akkor azt majdani felhasználók kérései és véleménye alapján még valószínűleg több helyen is át kell alakítani.

Feltételezzük a következőket a rendszer használata előtt:

- A pályamunkát nevező hallgató a 'Kisbagoly' rendszerben regisztrálta magát a szükséges adatokkal.
- A bírálathoz szükséges pályamunkát és a rezümét elektronikus formában feltöltötte a rendszerbe.
- A jelentkezését a titkárság, valamint a rendező intézmény minden adminisztrációs és formai követelmény teljesülése esetén elfogadta → A hallgató jelentkezettnek tekinthető.

A szekciórendező ezek után minden az általa képviselt szekcióba jelentkezett pályamunkát egy táblázatban tekinthet meg. A táblázatnak a pályamunka által képviselt intézmény szerint rendezve kell lennie. A pályamunkákat egyesével kijelölve és azokhoz lejárati dátumot rendelve el kell tudnia küldeni egy adott intézményhez elbírálást kérve. Erről az eseményről elektronikus értesítést kell kapnia a megfelelő intézmény intézményi titkárának, aki a bírák kiválasztását és hozzárendelését végzi. A szekció rendezőnek a saját felületén látnia kell, épp milyen státuszban van az összes az ő szekciójába nevezett pályamunka.

Miután az intézményi titkár számára el lettek küldve a pályamunkák, szükséges hogy egy saját felületen követhesse azok életútját, és lásson minden számára szükséges információt. Tudnia kell egy előre definiált listából bírálókat felkérni a pályamunka elbírálására. Egy ilyen

módon felkért bírának lehetősége van különböző okokra hivatkozva elutasítani a felkérést, ekkor újbóli felkérésre van szükség, mindaddig, míg a bíráló el nem fogadja azt és le nem zárul ez a lépés. Szükséges, hogy a szekciórendező értesüljön arról, hogy egyes pályamunkák esetén hogyan áll ez a folyamat, hogy bármikor beavatkozhasson.

Ha a bíráló személye eldőlt, akkor számára rendelkezésre kell bocsátani az előzőleg elektronikusan feltöltött pályamunkákat megtekintésre. Mindeközben persze tekintettel kell lenni a szekciórendező által az intézményi elküldés pillanatában definiált határidőre. A bírálat befejeztével egy a rendszerben rendelkezésére bocsátott, több szempontot figyelembe vevő pontozólapot és egy szöveges bírálatot kell kitöltenie és jóváhagynia. Ha a jóváhagyás megtörtént, akkor erről értesíteni kell a rendezőt.

A rendező a saját felületén láthatja a megszületett pontszámokat. Ha a két pontszám túl nagy (pl. több mint 20%-os) eltérést mutat, akkor az ellentét feloldására van szükség, hogy a bírák konszenzusra jussanak. Ha ez nem sikerül a rendezőnek lehetősége van egy harmadik bírát felkérni. Ha befejeződött minden pályamunka esetén a bírálati folyamat és megszületett a két végleges bírálat, akkor lehetőséget kell biztosítani a rendező számára a végleges tagozatba sorolásra. Ekkor a nevezéskor megjelölt tagozat besorolástól eltérő tagozatba kerülhet a pályamunka, illetve teljesen új tagozatok is megjelenhetnek.

Ha megszülettek a végleges tagozatok, akkor lehetőséget kell biztosítani a rendező számára, hogy a zsűri munkáját segítendő dokumentumokat kinyomtathassa rendszer által generált állományok alapján.

2.3 A rendszer határai

Fontosnak tartom minden egyes fejlesztés megkezdése előtt a követelmények mellett tisztázni azt is, hogy mit nem kell tudnia az elkészült programnak. Ezért tehát definiálni kell, hogy pontosan hol kezdődik, és hol végződik a megoldandó feladat. A belépési pont jelen esetben ott található, hogy a szerepköröknek megfelelő felhasználók felvitelre kerültek, a hallgatók beregisztrálták a pályamunkákat és feltöltötték a megfelelő dokumentumokat. Ha ezek a titkárság által elfogadásra kerültek, akkor bírálhatóvá vált a pályamunka.

Az előzetes bírálói folyamat a két végleges pontszám megszületéséig tart. Ezek után a végleges tagozatba sorolás, majd a zsűri munkáját segítendő dokumentumok generálása feladat még a rendszer számára.

2.4 Fogalmak és szerepkörök

A diplomamunka tárgyalása során a következő fogalmakkal találkozhatunk:

OTDK: Országos Tudományos Diákköri Konferencia. Célja, hogy a legkiválóbb egyetemisták és főiskolások tudományos és művészeti eredményeinek bemutatási és értékelési fóruma legyen.

OTDT titkárság: Országos Tudományos Diákköri Tanács titkársága, amely felelős az OTDK lebonyolításáért és az ahhoz szükséges adminisztráció elvégzéséért.

Pályamunka: Az OTDK-ra nevezett tudományos mű. Lehet szöveges, de a szekciót tekintve akár egy szobor vagy festmény, vagy zenedarab is.

Szekció: Az OTDK szekciók szerint épül fel. Egy szekció egy adott tudományterületet reprezentál. Pl.: Informatikai szekció

Tagozat: Egy szekción belüli szűkebb tudományos területet jelöl.

Szerzők: A pályamunka elkészítésében résztvevő hallgatók.

Kapcsolattartó: A pályamunka szerzői közül egy hallgató, aki az elektronikus rendszerben regisztrálja és jelentkezteti a pályaművet. A titkárság nem használja, kizárólag a programban jelenik meg.

Szekció felelős: A különféle szekciók vezetéséért és koordinálásáért felelős személyek.

Helyi Felelős: A felsőoktatási intézmény TDT elnöke, aki felel az intézmény és az OTDT titkárság közti kommunikációért és a bírák kiválasztásáért.

Bíráló: Egy adott intézmény oktatója, aki egy adott tudományterület elismert szakértője. Felel az intézményi helyi felelős által számára küldött pályamunkák bírálatáért.

Bírálat: Két részből áll: egy szöveges bírálatból és pontszámokból melyeket különböző szempont szerint kell kiosztani egy 1-10-es skálán.

2.5 Az *OpenConference* rendszer

A nevezett rendszer többször is felmerült beszélgetéseink során, mikor a követelményeket igyekeztünk összeszedni. Az OpenConference, mint ahogyan az a nevéből is sejthető egy nyílt forráskódú rendszer. Elsősorban konferenciák lebonyolítására szánták, és ingyenes, PHP nyelven megírt webes alkalmazásról van szó.

Főbb funkciói:

- Konferencia weboldalának létrehozása
- Előadók megkeresése és felkérése
- Elektronikus elfogadások és előadás abstractok kezelése
- Előadások feltöltése és szerkesztése
- Kereséses megvalósítása a konferencia anyagai között
- Résztvevők regisztrálása
- Előadásokhoz kapcsolódó online társalgások kezelése
- Automatikus üzenetek
- Elektronikus fizetés modul
- és még sok hasznos funkció...

A követelményeket alaposan kiértékelve azonban úgy döntöttem, sokkal egyszerűbb és tisztább megoldás lenne, ha saját magam fejlesztenék ki egy valamilyen szinten már jó alapokkal rendelkező rendszert. Így nem kell szembenézni azzal, hogy egy meglévő alkalmazást kell átalakítani saját igényeinkhez. Nem kell szembenézni az eltérő platformok használatának problémájával, mely az adatbázis rétegtől egészen a prezentációs rétegig terjedne. Lévén hogy az OpenConferene saját adatmodellel rendelkezik eleve nehézkes lett

volna összeházasítani a két adatbázist, nem is beszélve arról, hogy más adatbázis kezelő szoftver fut a kettő alatt, amely még komolyabb kérdéssé tette volna a szinkronizációt. Ennek egyik kellemetlen következménye lenne, hogy a felhasználók nem tudnának könnyedén váltogatni a két rendszer szolgáltatásai között. Természetesen létezik erre megoldás, egy single sign-on szolgáltatás fejlesztésével lehetett volna áthidalni a problémát, ami azonban tekintve a feladat méretét egy túlságosan költséges tényező lett volna. Így tehát úgy döntöttem, inkább a 'Kisbagoly' rendszer alapjain elindulva egy új modult fogok kifejleszteni, ahelyett hogy két meglévő alkalmazást kapcsolnék össze és alakítanék át. Mindazonáltal az OpenConference mint minta, végig jelen volt a fejlesztés során.

2.6 Kapcsolat a *'KisBagoly'* rendszerrel

Az előző pontokból kiindulva már látszanak a fejlesztés főbb irányelvei, ahhoz azonban hogy megértsük a modul felépítését és működését, szükséges néhány elemet bemutatni a szülő rendszerből. Ezen elemek legnagyobb része nem az én kezem munkája. Hallgatótársaim készítették őket a 'KisBagoly' projekt keretein belül. Felsorolásom az eredeti rendszer főbb adattábláit tartalmazza, pár gondolattal jellemezve azok szerepét:

Felhasználói adatok:

táblák: ASPNET_USERS; ASPNET_USERSinROLES; ASPNET_ROLES; USER_DATA

leírás: Az első három tábla az ASP .NET beépített autentikációs szolgáltatásának használata esetén automatikusan legenerálásra kerül. A USER_DATA viszont saját tábla, ebben kerülnek tárolásra az általunk használt felhasználókkal kapcsolatos információk, mint például: intézmény, beosztás, anyja neve, születési adatok…

Pályamunkák:

táblák: Palyamunka,Palyamunka_Hallgatok, PalyamunkaTechnikaiEszkozok

leírás: Az első tábla értelemszerűen a pályamunkák általános adatait tartalmazza, szerzők, cím… A második tábla arra szolgál, hogy az egyes pályamunka esetén a szerzők adatait tárolja. Erre azért van szükség, mert csupán a kapcsolattartó a regisztrált felhasználója a rendszernek, a többiek pedig ilyen formában lesznek elérhetőek. A technikai eszközök pedig

az országos versenyen való előadáshoz szükséges eszközök listáját tartalmazzák pályamunkánként. Pl.: projektor

Országos jelentkezések:

tábla: OrszagosJelentkezesek

leírás: Egy adott pályamunkához tartozó, országos versenyre történő jelentkezéshez szükséges adatokat tárol. Ilyen adatok a szekcióra, tagozatra vonatkozó adatok, az adminisztrációs segédadatok, de itt kerül később rögzítésre a pályamunka helyezése és díjazása is. Ez a fő kapcsolódási pont a két rendszer között.

Intézmények:

*tábla:*Intezmeny

leírás: A magyar felsőoktatási intézmények ide vonatkozó adatai és az intézményi felelős azonosítója, mint külső kulcs.

Szekciók:

*tábla:*OrszagosVerseny, OrszagosVersenyFelelos

leírás: Az első táblában szerepelnek a szekciók adatai, minden egyes szekcióhoz van egy országos verseny felvéve. A második egy kapcsolattábla az OrszagosVerseny és a ASPNET_USERS tábla között.

Tagozatok:

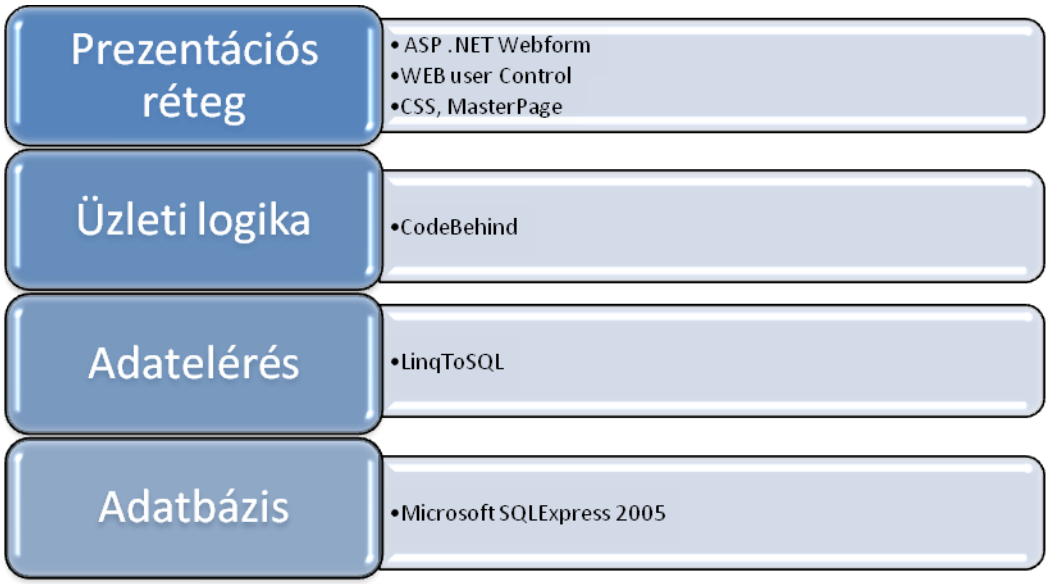
tábla:Tagozat

leírás: A tagozatokkal kapcsolatos adatokat tárolják. Fontos, hogy csak logikai törlés engedélyezett a programban, vagyis csak letiltani és engedélyezni lehet szekciókat.

2.7 Szoftver Design

Az alkalmazás tervezése során a legfőbb szempont az volt, hogy igyekezzek minél jobban alakítható és rugalmas felépítést találni az egyes funkcionalitásokhoz kötődő megoldások helyett. Ismerve az előzményeket erre minden bizonnyal szükség is lesz, de ettől

eltekintve fejleszteni is egyszerűbb egy rugalmas környezetben. Szétnézve a ma divatos modellek között kettőn akadhat meg a figyelmünk, az MVC-n és a hagyományos rétegzett architektúrán. Azonban mikor az ember egy konkrét problémára keresi a megoldást mindig mérlegelni kell, hogy megéri-e az a többletbefektetés, mely egy adott megoldás implementációjával járna, figyelembe véve az általa nyújtott előnyöket. Ahhoz, hogy ezt megtehessünk, ismernünk kell tehát a fent említett két mintát. Az MVC a Model-View-Controller szavak rövidítéséből származik, lényege röviden, hogy a hagyományos rétegzett architektúrát egy picit máshogyan értelmezi. Mégpedig olyan szinten, hogy a végső alkalmazás kitűnően módosítható és könnyen átalakítható legyen. A Model felel az adatok elérésért, a Controller szinten valósul meg az üzleti logika, majd a View nem csinál mást, mint megjeleníti az adott Controller által szolgáltatott adatokat. Tudni kell erről a megoldásról, hogy jelentős mennyiségű forrásállománnyal dolgozik, így meglehetősen sok többletmunkát hárít a programozóra. Pontosan ezt a komplexitást és többletmunkát figyelembe véve választottam egy ennél egyszerűbb, és jelen probléma esetén használhatóbb megoldást. Egy egyszerűbb réteges szerkezetet alakítottam ki, melyet azért ha megfigyelünk, láthatjuk, hogy az előzőeken alapszik, de az üzleti logika és a megjelenítés, hála az ASP .NET környezetnek nem kell, hogy ilyen élesen elváljon egymástól. Ezáltal biztosítva van az, hogy sokkal kevesebb osztályt és objektumot használjak a fejlesztés során.



1. ábra Az alkalmazás réteges szerkezete

Az ábra jól szemlélteti az általam elképzelt és megvalósított felépítést. Megtarthatjuk azt az MVC-ben meglévő előnyt, hogy logikailag elkülönülnek az egyes szintek az alkalmazásban, így sokkal átláthatóbb és jobban szervezett kódot kapunk. Természetesen nem lesz annyira könnyedén módosítható a kód, ha többnyelvűsíteni szeretnénk az alkalmazást, vagy esetleg több hasonló változást szeretnénk bevezetni az oldal több pontján is. A másik nagy előnye a réteges szerkezetnek, hogy a rétegek közvetlenül, csak a megelőző és rákövetkező rétegbeli elemekkel kommunikálnak, így ha netalán szükségessé válna valamely réteg kicserélése, akkor az relatíve egyszerűen végrehajtható lenne. A legtisztábban persze akkor érvényesülne ez az elv, ha interfészeken keresztül valósulna meg a kommunikáció az egyes rétegek között. Jelenleg azonban eltekintettem a kommunikációs interfészek használatától, ugyanis nem éreztem ezt indokoltnak az alkalmazás méretét és komplexitását figyelembe véve.

Az általánosabb magyarázat után rátérnék a konkrét megvalósításra. Az adatbázis réteg egy szabványos MS SQL adatbázison alapszik, melyet Microsoft SQL Express 2005 adatbázis kezelő rendszer futtat. A választás azért éppen erre esett, mivel ingyenes és bizonyos többletszolgáltatásokat és egyszerűsítéseket tartalmaz az ASP .NET technológiával való együttes használathoz. Természetesen miután azonos cég termékeiről van szó, ez valahol el is elvárható. Az adatbázisbeli elemek létrehozása Transact SQL nyelven történik, mely nem sokban tér más SQL nyelvektől, így használata nem jelent nehézségek. A következő szinten az adatelérési réteg szerepel, mely legfontosabb feladata hogy a relációs adatbázisban szabványos úton tárolt adatokat objektumokká alakítsa, és vissza. Ennek mikéntjével később még részletesebben foglalkozom, a lényeg azonban az, hogy e réteg felett már csak objektumokkal dolgozunk és nem relációs adatokkal. Az üzleti logikát az úgynevezett CodeBehind valósítja meg, mely nem más, mint szabványos C# vagy Visual Basic kód. Technikailag ugyanis egy ASP .NET weboldal nem egyetlen fizikai állományból épül fel, hanem több logikailag összetartozó kódállomány van a háttérben, melyek egyike tartalmazza a megjelenítéshez szükséges elemeket, a másik pedig egyéni kódot, eseménykezelőket és más objektumokat az adott nyelven implementálva. Tehát ezen állományok tökéletesen alkalmasak az egyes oldalak háttérében lezajló üzleti folyamatok implementálására. Természetesen a megjelenítéshez más, a webes világban jól ismert technológiákat is használhatunk, mint például a CSS-t, java scriptet és egyéb dolgokat.

3. A fejlesztés főbb elemei

A következőkben a program fejlesztése során felmerült főbb kérdésekkel szeretnék foglalkozni. A sorrend megegyezik az általam követett fejlesztési sorrenddel. Minden esetben részletesem bemutatom az adott lépést, példakódokkal szemléltetem az általam alkalmazott megoldást. Az elkészült szoftverben az itt leírt mintákat használom ismétlődő jelleggel. Azonban ahelyett, hogy ezeket egyesével elemezném, igyekszem kiragadni az általánosabb megoldási mintákat, és azokat részletesen magyarázni.

3.1 Az adatbázis

Mikor elkezdődött a szoftver tervezése, az első dolgom az volt, hogy részletesen kielemezzem a 'Kisbagoly' mögött álló adatbázis minden elemét, kapcsolódási pontokat keresve azzal. Miután végeztem a feladattal, úgy döntöttem, hogy nem készítek új fizikai adatbázist, hanem a meglévőt fogom kibővíteni általam készített új elemekkel. A döntés oka az volt, hogy rengeteg adat van a jelenlegi rendszerben, melyekre szükségem lesz a bírálat folyamán, így ha azonos adatbázissal dolgozom, sokkal egyszerűbb a táblák közti kapcsolatok definiálása külső kulcsok segítségével, valamint nem kell folyton adatot mozgatni a két adatbázis között, ami rengeteg felesleges tranzakciót eredményezett volna, és feleslegesen lelassult volna a folyamat is. A másik nagy előnye, hogy miután már egyszerűen tudtam használni a meglévő adatokat, nem sok új adatra volt szükségem a bírálati folyamatok követéséhez és rögzítéséhez, így minimalizálni tudtam a redundáns adattárolást. Ennek megfelelően csupán 3 új táblát készítettem a bírálat specifikus adatok tárolására. Az alacsony szám megtévesztő lehet, miután ebből az következhetne, hogy csupán egy egyszerű kis alkalmazásról van szó. Ez azonban egyáltalán nem igaz, mivel a felsőbb rétegek nem csupán ezekkel a táblákkal, hanem szinte az adatbázis egészével operálnak. A következőkben ezen táblákat fogom részletezni, és mindenhol külön jelzem a kapcsolódási pontokat melyeken keresztül adatokat érek el a 'Kisbagoly' rendszerből.

Természetesen nem csak táblák, hanem általam készített nézetek is léteznek az adatbázisban, feladatuk főként a nyomtatási listák hátteréül szolgáló adatok összegyűjtése.

Ezeket viszont nem részletezem, miután funkciójuk és számuk gyakran változik, miután a rendszer ezen része még fejlesztés alatt áll jelenleg is. Fontos még megemlíteni, hogy csak gyakorlati okok miatt kerülnek használatra, mivel a később bemutatásra kerülő LinqToSql teljes mértékben alkalmas lenne ezek kiváltására. A következő ábrák az SQL Server Management Studio Express ingyenes alkalmazás segítségével készültek, melyet a fejlesztés során az adatbázissal kapcsolatos adminisztrációs feladatokra használtam. Segítségével szerkeszthetőek akár vizuális, akár manuális úton az egyes elemek. A már elkészült adatbázisból sql kód generálható, illetve az adatbázis archiválása és archív mentésből történő visszaállítása is megoldható.

BiraltPalyamunka

	Column Name	Data Type	Allow Nulls
🔑	Id	int	☐
	Status	int	☑
	OrszagosJelentkezesId	int	☑
	SzuloIntezmenyId	int	☑
	Megjegyzes	varchar(500)	☑
	CreatedDate	datetime	☑
	KovHatarido	datetime	☑
	V_P1	int	☑
	V_P2	int	☑
	V_P3	int	☑
	V_P4	int	☑

2. ábra A BiraltPalyamunka tábla szerkezete

Ezen tábla rekordjai szolgálnak alapul a bírálási folyamat során, mondhatni ez a bírálati alaptáblája. Az `Id` a tábla elsődleges kulcsa, egész típusú azonosító mely értékét automatikusan növeli a rendszer új elem beszúrásakor. Egy 1:1 kapcsolat révén az `OrszagosJelenkezes` táblához kapcsoltam ezt a táblát, ennek megvalósítására szolgál az `OrszagosJelentkezesId`. Az 1:1 kapcsolatok létrehozása ugyan nem túl szerencsés adatbázis tervezési szemszögből nézve, de elengedhetetlen volt ahhoz, hogy a két rendszer függetlenül tudjon létezni, egymás szerkezetének megbolygatása nélkül. A `Status` mező szintén külső kulcsként szolgál, egy később bemutatásra kerülő táblához. Ez jelzi, hogy az adott pályamunka bírálata éppen hol tart a folyamatban. A `SzuloIntezmenyId` azért

került bevezetésre, mert az eredeti adatbázis szerkezete nagyon nehezen támogatta ennek az információnak a kinyerését, és rengeteg tábla érintésével lehetett csak hozzáférni az adathoz, így azt választottam, hogy miután logikailag szorosan kapcsolódik jelen adatokhoz, ezért itt is letárolom. A redundancia ellenére az inkonzisztens állapot kialakulására szerencsére kicsi az esély, miután a folyamat logikája úgy alakul, hogy mire a bírálati folyamat elkezdődik, már biztosan nem módosul a nevező intézmény, maximum a neve, amit viszont külső kulcs révén itt is követhetek. A `CreatedDate` és `KovHatarido` mezők egyaránt dátumokat tárolnak. Az első értelemszerűen a rekord létrehozásának dátuma, a második pedig arra utal, hogy az adott rekordon elvégezendő munkamozzanat meddig tehető meg. Erre azért volt szükség, hogy később ennek alapján automatikus üzeneteket lehessen küldeni a megfelelő személyeknek, arról hogy hamarosan lejár a megszabott határidő. A `V_P1...V_P4` mezők a végleges pontszámokat tartalmazzák, melyek a bírálat eredményeként jönnek létre.

Biralat

Column Name	Data Type	Allow Nulls
Id	int	☐
CreatedDate	datetime	☑
SzovegesBiralat	varchar(500)	☑
Megjegyzes	varchar(500)	☑
P1	real	☑
P2	real	☑
P3	real	☑
P4	real	☑
Status	bit	☑
BiraltPalyamunkaId	int	☑
IntezmenyId	int	☐
BiroId	uniqueidentifier	☑
Veglegesitve	bit	☑

3. ábra A Biralat tábla szerkezete

Ezen tábla rekordjai szolgálnak a bírák által létrehozott bírálatok tárolására. Az `Id` mint ahogyan az előző táblánál is, egész típusú és automatikusan növeli értékét a rendszer. A `BiraltPalyamunkaId` külső kulcsként szolgál a `BiraltPalyamunka` táblához, így tehát minden egyes bírálat alatt álló pályamunkához több bírálat is tartozhat, és tartoznia is kell a sikeres bírálathoz legalább kettőnek. Az `IntezmenyId` itt arra az intézményre utal,

melyben megszületett a bírálat. Tehát az az intézmény, ahová a Szekció rendező bírálatra küldte a pályamunkát. A `BiroId` típusa `uniqidentifier`, ami egy 16 bájtos adattipus, és használatára azért van szükség, mivel az MS SQL ezt használja az ASPNET_USERS tábla rekordjainak azonosítására. A követelményeknél szerepelt, hogy a bírálónak véglegesíteni kell bírálatát ahhoz, hogy azt figyelembe vegyék az értékelésnél, ennek jelzésére szolgál a `Veglegesitve` mező. A `Status` pedig itt arra vonatkozik, hogy a bíráló ha visszautasítja a bírálati felkérést, akkor azt nem kitöröljük a rendszerből, csupán érvénytelenítjük a flag átbillentésével, így később akár statisztikák készítéséhez is felhasználható lesz ez az adat.

BiralatStatus

	Column Name	Data Type	Allow Nulls
🔑	Id	int	☐
	Name	varchar(100)	☑
	Description	varchar(500)	☑
	CreationDate	datetime	☑
▶	Status	bit	☑

4. ábra A BiralatStatus tábla szerkezete

Ez a tábla a bírálatok érvényes státuszainak információit tartalmazza. Egy bírálati folyamatban résztvevő pályamunka életútja folyamán felvehető összes státusza szerepel az adattábla rekordjai között. Azért választottam ezt a megoldást, és nem beépített Status konstansok használatát, hogy a rendszert felügyelő OTDT titkárság flexibilisebbé tudja tenni a rendszer működését az egyes státuszok szerkesztésével. A `Status` bit szintén a logikai törlést szolgálja itt is.

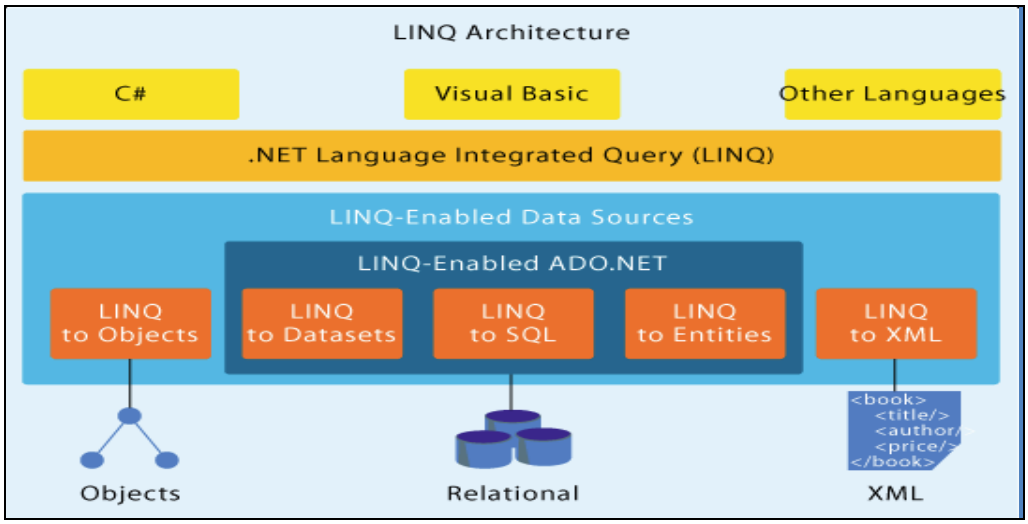
3.2 Adatelérési osztályok

Ahogy az előzőekben a rétegek bemutatásánál utaltam rá, most fogom részletesen bemutatni a rendszer egyik legizgalmasabb elemét, amely a relációs adatok objektumokra való leképezéséért felelős. Ahhoz, hogy lássuk, pontosan miről is van szó, szükséges egy kis háttérismeret a .NET technológia adateléréssel kapcsolatos részéről. Kezdetben az ADO (*ActiveX Data Objects*) a COM (*Component Object Model* - .NET előtti világ) adatelérési

rétege volt. Működésének lényege, hogy egyfajta hidat próbált képezni az alkalmazás és az adatforrás között. Úgy írhattunk programokat, hogy nem is ismertük pontosan milyen adatbázissal dolgozunk. Azonban a .NET Framework megjelenésével jelentősen átalakult ez a terület (is), az új technológia az ADO .NET nevet kapta, megtartva ezzel a tematikusságot, viszont az elnevezésen kívül nem sok köze van egymáshoz a két dolognak. Ez sem új keletű már, így visszatekintve a kezdetekre láthatjuk, hogy az egyes verzióváltásokkor mindig újabb és újabb elemeket csempésztek bele a fejlesztők. Nagyjából két fő változást érdemes kiemelni. Az első a *Data Providerek* használata, mely azt jelentette, hogy ezeken keresztül érhetjük el az adatbázist, a lekérdezett adatokat pedig *DataSet* objektumokban, - a memóriában - XML formában tárolja. Ezt a modellt kapcsolatnélkülinek nevezzük, miután nincs szükség folyamatos kapcsolatra az adatbázissal az adatok eléréséhez. A következő lépcsőfok a LINQ család megjelenése volt a 3.5-ös .NET-ben, mely több részből tevődik össze. A LINQ betűszó jelentése Language Integrated Query, ami annyit tesz, hogy nyelvbe integrált lekérdezés. A technológia lényege, hogy nyelvi eszközökkel tudunk SQL szerű lekérdezéseket írni. A LINQ nagy újítása, hogy nem kötelező relációs adatbázis felett dolgoznunk, ugyanis bármilyen kollekcióra alkalmazhatjuk a lekérdezéseket, amire implementálva van az `IEnumerable<T>` interfész.

A LINQ család tagjai:

1. LINQ To Objects: memóriában tárolt kollekciók kezelésére
2. LINQ To XML: XML forrással dolgozik
3. LINQ To SQL: A Microsoft első ORM(Object Relational Mapping) eszköze. Szerepe, hogy az adatbázis elemeit objektumokként kezelhetjük.
4. LINQ To DataSets: Az előző csak Microsoft SQL-el működik, így ez való a többi adatbázis kezelő fölé. Működési elve, hogy először egy DataSet-be pakoljuk a szükséges adatokat, majd arra lehet LINQ lekérdezéseket írni.



5. ábra A LINQ architektúrája

A jelen alkalmazásban a 3. pontban említett LinqToSql-t használom az adatelérési réteg alapjaként. És ha figyelmes végigolvassuk a felsorolás pontjait, már azt is látjuk, miért éppen a Microsoft SQL Express 2005-öt választottam adatbázis szervernek. Úgy gondolom, ha már eddig végigtaglaltuk a technológiát, érdemes talán erről is egy kicsit bővebben megemlékezni.

A LinqToSql provider nem tesz mást, mint biztosítja a LINQ lekérdezések használatát MS SQL adatbázisok felett. Miután maga az SQL szerver rendelkezik saját lekérdező nyelvvel és egy ehhez kapcsolódó értelmezővel, ezért felesleges lenne egy újabb eszközt írni arra, hogy értelmezze a LINQ lekérdezéseket is. Ehelyett azt a megoldást alkalmazzák, hogy egy speciális konverzió által a LINQ lekérdezéseket áttranszformálják SQL lekérdezésekké. Az egyetlen nehézség csupán az, hogy míg az SQL relációs adatokkal dolgozik, addig a LINQ objektumokon értelmezett. Ebből az alaphelyzetből egyértelműen adódik az, hogy valamiféle leképezést kell megvalósítani az objektumok és a relációs adatok között. Erre jelent megoldást a LinqToSql által definiált Mapping Framework. A leképezés megadásához nem kell mást tenni, mint a megfelelő osztályokat létrehozni. Minden egyes adatbázistáblához pontosan egy ilyen tartozik, mely adattagként tartalmazza a megfelelő oszlopokat, természetesen valamiféle típusegyeztetéssel. A különféle megszorításokat, melyeket egy adott táblára teszünk, pedig az osztály attribútumaihoz elhelyezett megfelelő módosító paraméterekkel tehetjük meg. Ez a rendszer nagyon hasonlatos a JAVA EE által használt annotációkhoz, mellyel a különféle mappelési opciókat jelezhetjük a rendszer számára.

Például:

```
[Table(Name="Lakcim")]
public class Lakcim
{
      [Column(IsPrimaryKey = true)]
      public int LakcimId;

      [Column]
      public int Irsz;
}
```

Természetesen ezek az osztályok automatikusan generálhatók egy már létező adatbázis alapján. Az elkészült kódot ilyenkor persze lehetőség van kézzel is módosítani az igényeinknek megfelelően akár új metódusokat is implementálhatunk, ha szükséges. Egy úgynevezett DataContext objektum lesz az, mely felel majd a generált LinqToSql osztályok kezeléséért, ami azt jelenti, hogy a kód szintjén úgy tekinthetünk rá, mintha ez szimbolizálná az adatbázist. Érdemes megemlíteni, hogy ez szintén egy generált állomány. A Visual Studio 2008 segítségével vizuális eszközöket használva egyszerűen végrehajtható a szükséges beállítások megadása és a kódgenerálás. Csupán egy új .dbml kiterjesztésű elemet kell hozzáadni a projecthez mely két fizikai állományt tartalmaz.

```
Biralat.dbml
        1, Biralat.dbml.layout
        2, Biralat.designer.cs
```

Az első állomány egy szerkesztőben megnyíló felület, melyre tetszőleges adatbázis elemek húzhatóak fel, a második pedig az ezen elemek alapján automatikusan generált osztályokat tartalmazza egyetlen fizikai állományba sűrítve, valamint a DataContext objektumot.

Ha ezzel készen vagyunk, akkor már roppant egyszerű dolgunk van, ugyanis csak azokat a metódusokat és függvényeket kell implementálni, amelyek az alkalmazás egyes elemeihez szükséges adatmozgatási műveleteket biztosítják. Az alkalmazásomban táblánként egy-egy statikus osztályt hoztam létre, melyekben az adott tábla adatain manipuláló metódusok vannak megvalósítva. Akár egyetlen osztályba is beletehettem volna mindet, de így sokkal átláthatóbb a kód. A fejlesztés tipikusan úgy történt, hogy mikor a weboldalak készítésekor szükségem volt valamilyen adatra, akkor írtam hozzá egy új metódust a

megfelelő osztályban. Minden ide kapcsolódó elem a DAO.Biralat névtérben található. A következőkben egyetlen ilyen fent említett osztályt, a BiralatDAO-t fogom bemutatni.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Linq.Dynamic;
using DAO;


namespace DAO.Biralat
{
    public class BiralatDAO
    {

        public static List<Biralat> getBiralatokForBiroByKuldoIntezmeny(Guid
biroID, int iID)
        {
            BiralatDataContext dc = new BiralatDataContext();

            var result = from b in dc.Biralats
                         where b.BiroId == biroID
                         where b.BiraltPalyamunka.SzuloIntezmenyId == iID
                         where b.BiraltPalyamunka.Status == 2
                         where b.Status == true
                         select b;

            return result.OrderBy(x =>
x.BiraltPalyamunka.KovHatarido).ToList();
        }

        public static List<Intezmeny> getKuldoIntezmenyListaForBiro(Guid
biroID)
        {
            BiralatDataContext dc = new BiralatDataContext();

            var result = from b in dc.Biralats
                         where b.BiroId == biroID
                         where b.Status == true
                         where b.BiraltPalyamunka.Status == 2
                         select b.BiraltPalyamunka.Intezmeny;

            return result.OrderBy(x =>
x.intezmenyNev).Distinct().ToList<Intezmeny>();
        }


        public static void updateBírálat(Biralat _b)
        {
            BiralatDataContext dc = new BiralatDataContext();

            Biralat b = dc.Biralats.Where(x => x.Id == _b.Id).First();
            b.BiroId = _b.BiroId;
            b.Megjegyzes = _b.Megjegyzes;
            b.P1 = _b.P1;
            b.P2 = _b.P2;
            b.P3 = _b.P3;
            b.P4 = _b.P4;
            b.SzovegesBiralat = _b.SzovegesBiralat;
```

```
            dc.SubmitChanges();

        }

        public static List<User_Data> getAllBíróByIntézmény(int iId, String
role)
        {
            BiralatDataContext dc = new BiralatDataContext();

            var result = from r in dc.aspnet_Roles
                         from d in dc.User_Datas
                         from uinr in dc.aspnet_UsersInRoles
                         where r.RoleId != null && d.UserId != null
                         where uinr.RoleId == r.RoleId && uinr.UserId ==
d.UserId
                         where r.RoleName == role
                         where d.intezmenyID == iId
                         select d;

            return result.OrderBy(x => x.Name).ToList();
        }

        public static Biralat getBiralatById(int Id)
        {
            BiralatDataContext dc = new BiralatDataContext();

            return dc.Biralats.Where(x => x.Id == Id).First();
        }

        public static List<Biralat> getAllBiralatforPalyamunka(int _pbId)
        {
            BiralatDataContext dc = new BiralatDataContext();

            var q = from p in dc.Biralats
                    where p.BiraltPalyamunkaId == _pbId
                    where p.Status == true  //aktív a status
                    select p;

            return q.ToList<Biralat>();
        }

        public static void  newBiralat(Biralat _pb)
        {
            BiralatDataContext dc = new BiralatDataContext();
            _pb.CreatedDate = DateTime.Now;
            dc.Biralats.InsertOnSubmit(_pb);
            dc.SubmitChanges();
        }

        public static void addNewBiralatForPalyamunka(BiraltPalyamunka _pm,
Biralat _pb)
        {
            BiralatDataContext dc = new BiralatDataContext();

            _pb.BiraltPalyamunkaId = _pm.Id;

            dc.Biralats.InsertOnSubmit(_pb);
            dc.SubmitChanges();
        }}
```

Először is szeretném tisztázni, mit is jelent a DAO, mint betűszó, ami talán nagyjából el is mondja, mire is való ez az osztály. Data Access Object, vagyis adathozzáférési osztály azt jelenti, hogy a DataContext által létrehozott objektumokat használva, az alkalmazás többi rétegének előkészítem itt a forrásadatokat. Azt is mondhatnánk, hogy egy-egy itt implementált metódus megfelel egy lekérdezésnek logikailag. Ha valahol szükségem lenne egy Listára bizonyos adatokból, ami függ egy-két paramétertől, akkor egyszerűen csak fogom a táblához kapcsolódó osztályt, és beleteszem a megfelelő metódust. A kód elején az úgynevezett `using` direktívák találhatók, melyek hasonlóan más programozási nyelvekhez a más névtérbeli elemek egyszerű használatáért felelnek, hogy ne kelljen teljesen minősített nevekkel dolgozni mindenhol. Az egyik legfontosabb elem jelen példában a Linq névtér behúzása, miután a legtöbb következőekben használt objektum onnan származik. Láthatjuk rögtön az elején, hogy az osztály statikus. Ennek az az oka, hogy nincs szükségem példányszintű dolgokra, csupán egy funkcionalitás gyűjteményt szeretnék létrehozni, mely bármikor felhasználható, és minden egyes hívás eredménye csak a bemenő paraméterektől és a mögöttes adatbázistól függ. Ennek megfelelően természetesen az implementált metódusok is statikusak lesznek. Ha nagyjából áttekintjük a kódot láthatjuk, hogy tipikus CRUD műveletekről van szó, vagy a Create, Read, Update, Delete adatbázis műveletekre hasonlító funkcionalitást valósítanak meg.

Szeretnék pár mondatban foglalkozni, egy számomra fontos és érdekesnek mondható szemléletbeli kérdéssel, nevezetesen azzal, hogy milyen elvet vallottam az egyes adatelérési metódusok fejlesztése során. Teljesen más szemléletmódot követtem, ebben az OO környezetben, mint mondjuk Oracle PL/SQL vagy egyéb környezetekben. Mire is gondolok? Szerencsésnek mondhatom magam, mivel az egyetemi tanulmányaim mellett aktívan dolgozom, és munkám során főként az Oracle cég adatbázissal kapcsolatos technológiáit használom. Itt úgy kell megfogalmazni a problémát, mint ahogy azt adatbázisok órán is tanultunk. Vagyis egy nagyon jól optimalizált lekérdezés összeállítása a cél, ami erőforrás hatékonyan visszaad minden szükséges adatot, amit utána a megjelenítéskor már nem vagy csak kevéssé fogunk átalakítani, tovább szűrni. Ez gyakran nagyon bonyolult lekérdezéseket eredményez, rengeteg táblával és sok-sok optimalizálási és egyben hibázási lehetőséggel. Kissé sarkítva mondhatjuk azt, hogy az üzleti logikát itt a lekérdezés valósítja meg, esetleg egy PL/SQL kód, de valahol a mögött is egy vagy több ilyen lekérdezés húzódik meg. Ezzel szemben itt, nincs semmilyen bonyolult lekérdezés, nincsenek táblákon átívelő JOIN-ok,

nincs semmiféle SQL nyelven megírt elem. Csupán objektumok vannak, azok az objektumok melyeket a LINQ valamint a választott forrásnyelv eszközrendszere határoznak meg csupán, ezáltal sokkal nagyobb szabadsági foka van a programozónak. A használatos elv, pedig az, hogy egyszerű LinqToSql lekérdezéseket használok, amelyek valamiféle alapfokú, kevés számú paramétertől függő szelekciót hajtanak végre, és visszaadnak valamilyen objektumot vagy objektumok egy listáját, általában a tábla objektumok típusával. Akkor mégis, hogy kapcsoljunk össze különféle táblákat? A válasz elég egyszerű és logikus, ha az előbb említett paraméter a másik tábla kulcs értéke, akkor máris sikerült elérnünk az adott tábla összes elemét mely az adott paramétert, mint külső kulcsot tartalmazza. Végezzük ezt akár ciklikusan, és máris összekapcsoltunk két táblát. Így tehát teljes értékűen sikerült kiváltanunk a bonyolult SQL lekérdezéseket, tisztán OO nyelvi környezetet használva. A lekérdezések sebessége természetesen nagyban függ attól, hogyan vannak az indexek elhelyezve az adattáblákon illetve a LinqToSql milyen optimalizációkat hajt végre a lekérdezés generálása során, valamint az adatbázis kezelő milyen végrehajtási terv alapján futtatja azt. Általános érvényű tanács, hogy érdemes és szükséges is a lekérdezések természetéhez kialakítani a tábla indexeit, mellyel nagyon gyorssá lehet tenni az egyes futásokat.

Nézzünk tehát egy példát erre, miről is van szó a gyakorlatban. Vegyük a `getBiralatokForBiroByKuldoIntezmeny` metódust. A metódus nevét mindenhol igyekeztem kellően beszédesre megalkotni: getBírálatok for Bíró byKüldőIntézmény, vagyis az összes bírálatot szeretnénk megtudni, amit egy adott intézmény küldött egy adott elbíráló személynek. Szemléltetve az előzőekben leírtakat, minimum 3 táblát összekapcsoltunk ezzel az egy metódussal, csupán paraméterek használatával. Ha majd később a felületre is rátérünk még logikusabb lesz miért is így dolgozunk. Nézzük a metódus törzsét, sorról sorra.

```
        public static List<Biralat> getBiralatokForBiroByKuldoIntezmeny(Guid
biroID, int iID)
        {
            BiralatDataContext dc = new BiralatDataContext();

            var result = from b in dc.Biralats
                         where b.BiroId == biroID
                         where b.BiraltPalyamunka.SzuloIntezmenyId == iID
                         where b.BiraltPalyamunka.Status == 2
                         where b.Status == true
                         select b;

            return result.OrderBy(x =>
x.BiraltPalyamunka.KovHatarido).ToList();
        }
```

A statikus osztályok használatával, elkerülhetetlen, hogy minden egyes metódusban újra deklaráljuk és példányosítsuk az adatbázisunk eléréséért felelős BiralatDataContext nevű objektumot. Ez nem más, mint az előzőekben is emlegetett, automatikusan generált DataContext objektum, melyet mindig a dbml állomány alapján nevez el a Visual Studio. Mondhatjuk azt, hogy ez az objektum reprezentálja most számunkra az adatbázist, és minden, amit ott tárolunk ezen keresztül érhető el, mint objektum. A következős sorban a lekérdezést írjuk le, de mielőtt rátérnék arra, pár szót arról, hogy mit takar a következő kifejezés:

```
var változónév = érték;
```

A C# 3.0 lehetővé teszi, hogy bizonyos esetekben úgy hozzunk létre változót, hogy nem adunk neki explicit módon típust, hanem a fordítóra bízzuk a típus hozzárendelést, ami azonban abban a pillanatban megtörténik, ahogy értéket adunk neki. (Azonnal meg kell tenni). Ez a hozzárendelt típus nem megváltoztatható, de bizonyos konverziók végrehajthatóak rajta. Tehát a `result` változó típusa a jobb oldalon álló kifejezés visszatérési típusa lesz. Ez a kifejezés pedig nem más, mint a sokat emlegetett Linq kifejezés. Ami úgy néz ki elsőre, mint egy kissé összekavart SQL Select. Előre kerül(nek) a használni kívánt táblaobjektum(ok) neve(i) a `from` kulcsszó után. Ezután jönnek a `where` feltételek, melyek logikai kifejezések, akár `&&`-el összekapcsolva, akár több `where` sor használatával. Végül pedig a `select` kulcsszó, amely megmondja mit is akarunk visszaadni, lehetőség van az egész sorobjektum visszaadására, vagy csak egy attribútuméra. De akár azt is megtehetjük, hogy egy új névtelen típust hozunk itt létre, és azt adjuk vissza.

A `return` utasítás után, pedig a Linq másik alakjával ismerkedhetünk meg a lambda kifejezéssel.

```
x => x.BiraltPalyamunka.KovHatarido
```

Nem mást jelent, mint hogy x értéke legyen a biraltPalyamunka.KovHatarido attribútuma, és mivel egy OrderBy-ba van paraméterként megadva, ezért e szerint lesz rendezve az eredménylista. Lambda kifejezéssel ugyan úgy teljes értékű lekérdezéseket tudunk végrehajtani, mint az előbb leírt módon is. Az hogy éppen melyiket használjuk, helyzete válogatja és persze a programozó tudása, de mindenképpen elmondható, hogy egy borzasztóan kompakt és hatékony eszközrendszer áll rendelkezésre ahhoz, hogy a Linq lekérdezéseket megfelelően használhassuk a C# nyelvben. A `getKuldoIntezmenyListaForBiro` metódus arra szolgál, hogy az összes olyan

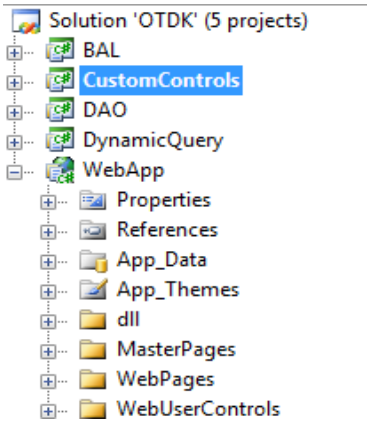
intézmény nevét megtudjuk, mely egy adott bíráló számára bírálati felkérést küldött. Ha valami olyan műveletet hajtunk végre, ami az adatbázisban tárolt adatok megváltozásával jár, akkor egy commitáláshoz hasonló szerepet játszó dolgot kell tennünk. Vagyis a DataContext objektum `submitChanges()` nevű metódusát kell ahhoz meghívnunk, hogy véglegesítődjenek az általunk végrehajtott változtatások. Ennek hiányában csak a memóriában történne meg az összes általunk végrehajtott változás.

3.3 A webes alkalmazás felületei

Mint azt a szoftver design tárgyalásánál is említettem, az üzleti logika és a megjelenítés már nem válik el olyan szervesen, mint az előző réteg. A megjelenített oldalak mögött álló folyamatok mindenütt az oldallal együtt kezelt CodeBehind-ban kerülnek megvalósításra. Ez nem más, mint egy szabványos osztály, annyi különbséggel, hogy itt elérhetjük objektumként mindazon elemeket, melyeket a vele együvé tartozó weboldalon, mint vizuális elemeket létrehoztunk. Ezen elemeknek számos tulajdonságát tudjuk a kódból beállítani, mint attribútumokat, így nagyon kényelmes és hatékony eszközről van szó. Ennek megfelelően a következő példámban is együtt mutatom be a megjelenített felületet és a mögötte dolgozó kódot. Természetesen, miután nem pusztán weboldalak készítéséből áll egy web alkalmazás fejlesztésének folyamata, így aztán itt is lehetőség van arra, hogy a választott forrásnyelven, más, a megjelenítésben nem szereplő osztályokat definiálhassunk. Ez nagyon sokszor fontos lesz, ha olyan funkcionalitást szeretnénk implementálni, mely előkészít valamit a megjelenítés számára. Ezzel tehát elárultam azt, hogy nem volt teljesen igaz, amikor azt állítottam, hogy az üzleti logika eszköze csupán a CodeBehind.

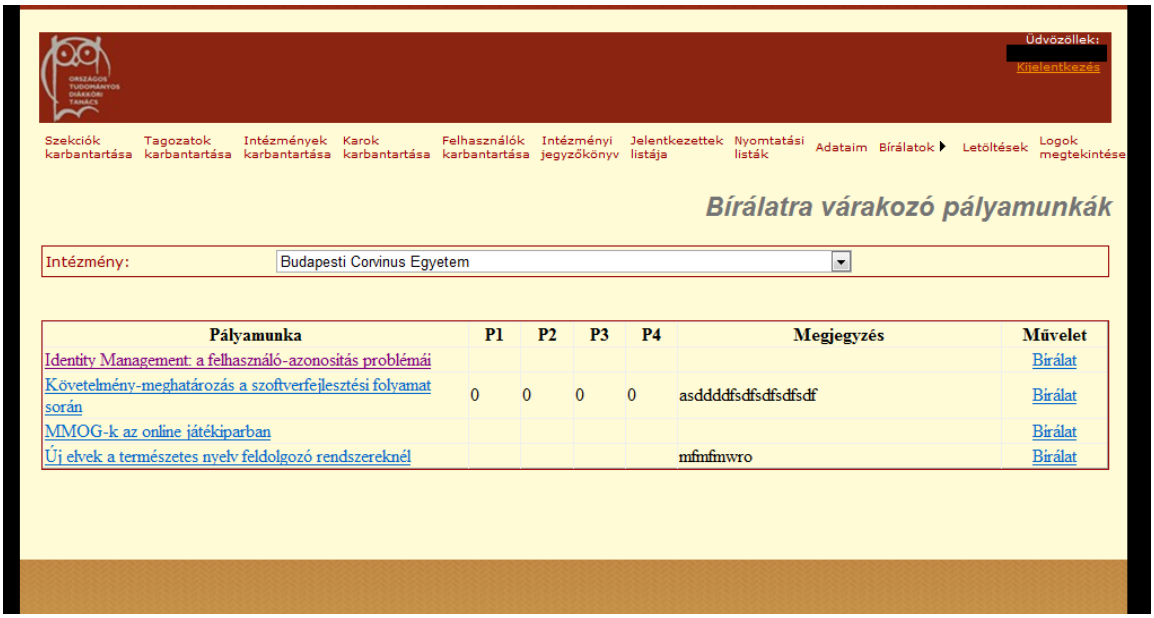
A konkrét megvalósításra rátérve, mint az adatelérési réteg, ez a két réteg is egy új projectben van megvalósítva a solution-on belül. A project pedig nem más, mint egy szabványos ASP .NET web alkalmazás típusú project. Ami azért van kiemelve ennyire, mivel kényelmesebb, ha a VisualStudio-ban ilyet hozunk létre, mert legenerálja a szükséges könyvtárstruktúrát, és elkészít egy pár állományt előre, melyre később még szükségünk lesz. A 6. ábrán láthatjuk is, hogy milyen szerkezetet kell elképzelni. Láthatjuk a DAO és a WebApp projecteket, amikről már volt szó, illetve azt is, hogy a WebApp-on belül milyen struktúrában helyezkednek el az egyes elemek. A References nevű kissé furcsa ikonnal jelzett mappában különféle hivatkozásokat helyezhetünk el külső elemekre, melyeket a projektünk

használni fog. Az AppData tartalmazhatná az adatbázist, de ezt most külön projectbe kiemeltük, így itt nincs szerepe. A többi mappáról pedig még lesz szó a későbbiekben.



6. ábra Az alkalmazás szerkezete

Elsőként hát akkor beszéljünk egy adott oldalról, nézzünk meg miként is épül fel, majd nézzük meg azt is milyen üzleti folyamatok játszódnak le a háttérben. Így egy kicsit eltérünk attól, amilyen sorrendben a rétegek fizikailag egymásra épülnek, de a jobb érthetőség kedvéért nem azt az irányt követjük. Lássuk tehát először, hogy mi is az, ami a böngészőben minket fogad, mikor egy adott oldalra navigálunk. Példánkban azt az oldalt fogom bemutatni, mely a DAO réteg példáját (is) használja, mint adatelérési osztályt.



7. ábra A bírók által látott felület

```
<%@ Page Title="Bírálati lista" Language="C#" MasterPageFile="~/MasterPages/Main.Master"
    AutoEventWireup="true" CodeBehind="BirálatokBironként.aspx.cs"
Inherits="WebApp.WebPages.Bírálat.Bíráló.BirálatokBironként" %>

<%@ MasterType VirtualPath="~/MasterPages/Main.Master" %>

<%@ Register assembly="AjaxControlToolkit" namespace="AjaxControlToolkit.HTMLEditor"
tagprefix="cc1" %>

<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" runat="server">
    <h2 class="myH2">
        Bírálatra várakozó pályamunkák</h2>
    <asp:HiddenField ID="hfBiroID" runat="server" />
    <fieldset style="border-color: #990000; border-style: solid; border-width: 1px">
        <table>
            <tr>
                <td class="td elsoOszlop">
                    Intézmény:
                </td>
                <td class="td masodikOszlop">
                    <asp:DropDownList ID="ddlIntezmeny" runat="server" utoPostBack="True"
AppendDataBoundItems="True"
                        Width="510px" AutoPostBack="True" DataSourceID="odsIntezmeny"
DataTextField="intezmenyNev"
                        DataValueField="intezmenyID">
                    </asp:DropDownList>
                    <asp:ObjectDataSource ID="odsIntezmeny" runat="server"
SelectMethod="getKuldoIntezmenyListaForBiro"
                        TypeName="DAO.Biralat.BiralatDAO">
                        <SelectParameters>
                            <asp:ControlParameter ControlID="hfBiroID" DbType="Guid"
Name="biroID" PropertyName="Value" />
                        </SelectParameters>
                    </asp:ObjectDataSource>
                </td>
            </tr>
        </table>
    </fieldset>
    <br />
    <br />
    <asp:Label ID="lblError" runat="server" Text="" Visible="false"
ForeColor="Red"></asp:Label>
    <fieldset style="border-color: #990000; border-style: solid; border-width: 1px">
        <asp:GridView ID="gvBiralatok" runat="server" AutoGenerateColumns="False"
DataSourceID="odsBiralatok" Width="100%">
            <Columns>
                <asp:TemplateField HeaderText="Pályamunka" HeaderStyle-Width="40%">
                    <ItemTemplate>
                        <asp:HyperLink ID="PalyamunkaSzerkURL" Text='<%#
Eval("BiraltPalyamunka.OrszagosJelentkeze.Palyamunka.cim") %>'
                            NavigateUrl='<%#
"~/WebPages/Tester/Palyamunkaszerk.aspx?mode=view&ID=" + DataBinder.Eval(Container.DataItem,
"BiraltPalyamunka.OrszagosJelentkeze.Palyamunka.palyamunkaID") %>'
                            Target="_blank" runat="server" />
                    </ItemTemplate>
                </asp:TemplateField>
                <asp:BoundField DataField="P1" HeaderText="P1" SortExpression="P1" />
                <asp:BoundField DataField="P2" HeaderText="P2" SortExpression="P2" />
                <asp:BoundField DataField="P3" HeaderText="P3" SortExpression="P3" />
                <asp:BoundField DataField="P4" HeaderText="P4" SortExpression="P4" />
                <asp:BoundField DataField="Megjegyzes" HeaderText="Megjegyzés"  ControlStyle-
Width="40%" />
                <asp:TemplateField HeaderText="Művelet" HeaderStyle-Width="10%" ItemStyle-
HorizontalAlign="Center">
                    <ItemTemplate>
                        <asp:LinkButton ID="btnElbírál" runat="server" Text="Bírálat"
CommandArgument='<%# Eval("Id") %>'
                            CommandName="bírál" OnCommand="btnCommandHandler" Visible="true"
/>
                    </ItemTemplate>
                </asp:TemplateField>
            </Columns>
        </asp:GridView>
```

```
        <asp:ObjectDataSource ID="odsBiralatok" runat="server"
SelectMethod="getBirakatokForBiroByKuldoIntezmeny"
            TypeName="DAO.Biralat.BiralatDAO">
            <SelectParameters>
                <asp:ControlParameter ControlID="hfBiroID" DbType="Guid" Name="biroID"
PropertyName="Value" />
                <asp:ControlParameter ControlID="ddlIntezmeny" Name="iID"
PropertyName="SelectedValue"
                    Type="Int32" />
            </SelectParameters>
        </asp:ObjectDataSource>
    </fieldset>



    <%--################################## Update Panel ##################################--%>


    <ajax:ModalPopupExtender ID="mpeBírálat" BackgroundCssClass="modalBackground"
        runat="server" Enabled="true" PopupControlID="pnPopup"
TargetControlID="BTNMODALPOPUPEXTENDER3">
    </ajax:ModalPopupExtender>
    <asp:Button runat="server" ID="BTNMODALPOPUPEXTENDER3" Text="GOMB2" Style="display: none"
/>

    <asp:Panel ID="pnPopup" runat="server" CssClass="modalPopup">
        <h2>Bírálat rögzítése pályamunkához</h2>
        <asp:Label ID="lblPopupError" runat="server" Text="" ForeColor="Red"
Visible="false"></asp:Label>
        <table style="width:600px">
            <tr style="width:30%">
                <td class="td_elsoOszlop" style="width: 180px">
                    Pályamunka címe:</td>
                <td class="td masodikOszlop">
                    <asp:Label ID="lbPalyamunkaCim" runat="server" Text=""></asp:Label>
                </td>
            </tr>
            <tr><td class="td_elsoOszlop" style="width: 180px">Pontszám:</td>
                <td class="td_masodikOszlop">
                    <asp:DropDownList ID="ddlP1" runat="server">
                        <asp:ListItem Selected="True">0</asp:ListItem>
                        <asp:ListItem>1</asp:ListItem>
                        …
                        <asp:ListItem>9</asp:ListItem>
                        <asp:ListItem>10</asp:ListItem>
                    </asp:DropDownList>
                </td>
            </tr>
            <tr>
            <td class="td_masodikOszlop" style="width: 180px">Szöveges értékelés:</td>
            <td class="td_masodikOszlop">
                <asp:TextBox ID="tbSzovegesErtekeles" runat="server" Rows="3" Columns="40"
TextMode="Multiline" /></td>
                </td>
            </tr>
            <tr>
            <td class="td_masodikOszlop" style="width: 180px">Egyéb megjegyzés:</td>
            <td class="td_masodikOszlop">
                <asp:TextBox ID="tbMegjegyzes" runat="server" Rows="3" Columns="40"
TextMode="Multiline" /></td>
            </tr>
        </table>
        <asp:Button ID="btnPopupClose" runat="server" Text="Close"
            onclick="btnPopupClose_Click" />
            <asp:Button ID="btnPopupSubmit" runat="server" Text="Mentés"
            onclick="btnPopupSave_Click" />
        <asp:HiddenField ID="hfBirálatId" runat="server" />
    </asp:Panel>
</asp:Content>
```

Mint láthatjuk az oldalt leíró .aspx fájl kódja egy statikus (X)HTML tagekből és <% %> közé zárt dinamikus elemekből felépülő szövegfájl. A tagek különféle szerveroldali vezérlőket és klasszikus html elemeket reprezentálnak. A dinamikus tartalom, pedig bármilyen vezérlési szerkezetet és adatkötést tartalmazhat, hasonlóan a PHP és egyéb más nyelvekhez. Kezdetben míg csak ASP és nem ASP .NET technológiáról beszéltünk, nem is létezett más, csupán a statikus vezérlők és a dinamikus úgynevezett inline kódok keverékéből előálló kód, mely szerveroldalon futott le, és egy HTML kódot állított elő a kliens gépén. Azonban ma már a CodeBehind használatával többnyire kiküszöbelhető és nem is ajánlott teljesítménybeli okok miatt ezt a modellt használni. Egyetlen kivétel talán a különféle vezérlőkhöz történő adatkötés, melyre ma is ezt a módszert használjuk.

Mielőtt részletesen belemennénk a továbbiak tárgyalásába, szeretném a sokszor használt controlokat vagy másnéven vezérlőket egy kissé jobban definiálni. Az ASP .NET szerver oldali vezérlők olyan elemek, melyeket a szerver ismer, felismer, és azokat programozottan használni tudjuk ezáltal a CodeBehind-ban. Három csoportba lehet őket osztani: HTML server control, Web Server control, Validation control. Az első csoportba hagyományos HTML nyelvben ismert elemek tartoznak, amik attól lesznek server controlok, hogy a runat=”server” attribútumot tartalmazzák, valamint egy egyedi azonosítót mellyel később tudunk rá hivatkozni. A második kategória tagjai sokkal komplexebb elemek, melyek nem feltétlen jelennek meg vizuálisan is az oldalon. Általában javascriptre és HTML-re fordulnak le a kliens gépén. A 3. kategóriába pedig olyan elemek tartozank, melyek feladata valamilyen adatbevitel helyességének ellenőrzése.

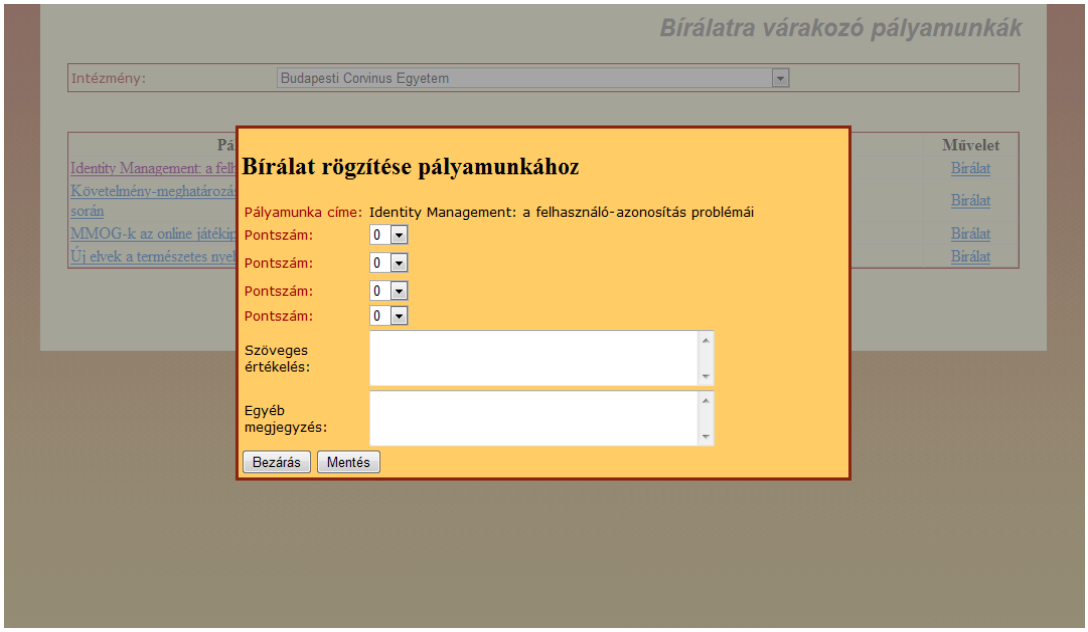
Ezek után a kódot tekintve egyből látjuk, hogy az első sorokban valamiféle dinamikus tartalom szerepel. Az oldal általánosságára vonatkozó beállításokat tehetjük meg a Page direktívában, olyanokat mint az oldal címe, a CodeBehind állomány neve, annak forrásnyelve, és más általános beállítás. A következő elem a MasterPage-re vonatkozó beállításokat tartalmazza, azonban itt egy apró kis kitérőt is szeretnék tenni erre a témára. Webes berkekben ismert probléma, hogy egy komplexebb webalkalmazás vagy weboldal fejlesztésekor általában sok olyan rész van, amely mindig ismétlődik, ezért tehát nem lenne szerencsés mindig újra és újra megírni azt, főként a redundáns tárolással és kezeléssel jelentkező problémák miatt. Nevezzük ezt az ismétlődő részt egy keretnek, ez tartalmazza például az oldal menüjét, fejlécét, láblécét stb… viszonylag ritkán változó dolgokat tehát. Erre született meg válaszként az ASP .NET-ben a MasterPage fogalma. Ez azt jelenti a

gyakorlatban, hogy létrehozunk egy egyébként szabványos ASP .NET weboldalt, amin kialakítjuk a szükséges külalakot, úgynevezett content place holdereket helyezhetünk el rajta olyan helyeken, ahová később valamilyen dinamikus dolgot akarunk megjeleníteni. Ezek lényegében helykijelölők lesznek, melyek olyan kontrolok, amik később megvalósított egyéb elemeket fogják majd tartalmazni. Mint láthatjuk, a mi oldalunk is egy `<asp:Content>` tag-el kezdődik, vagyis éppen egy ilyen helykijelölőbe „illik". Természetesen a Master oldalnak lehet több ilyen betéte is, vagyis igazából ez csak egy minta, amely tartalmazza a legáltalánosabb kinézetre vonatkozó beállításokat és az oldal csontvázát.

Ezek után, visszatérve az eredeti témához, hagyományosnak nevezhető HTML tagek következnek. Majd pedig két érdekesség, egy DropDownList és egy ObjectDataSource control, mik is ezek? A dropdown nem más, mint egy hagyományos lenyíló lista, amit HTML-ben is egyszerűen el lehet készíteni, amitől érdekes lesz az az, hogy az elemek megadásához az említett ObjectDataSourcot használjuk, természetesen van mód az elemek hagyományos módon történő felsorolással való megadására is. Komolyabb alkalmazásokban azonban ezt ritkán használjuk, mivel többnyire valamilyen szintén tárolt adaton alapszik ez a lista. Az ObjectDataSource nem más, mint egy olyan elem, ami biztosítja számunkra azt, hogy az oldalon elhelyezett egyéb elemek számára adatokat szolgáltassunk, mint forrásadat. Valamilyen előzőleg definiált, objektumlistát visszaadó metódust köt magához, és meg kell adnunk azt is, hogy milyen típusú objektumokat fog szolgáltatni. Ezek után már nincs más dolgunk, csak a forrásunkat felhasználó vezérlőnél, ami jelen esetben az említett lenyíló lista, megadni, hogy a visszatérési objektum mely attribútumai lesznek a megjelenítendő komponensek. Itt kell, hogy visszahivatkozzak arra, amit az adatelérési részben említettem, hogy azért használtunk ott egyszerű lekérdezéseket megvalósító metódusokat, mert adatelérésre többnyire csak akkor van szükség, ha valamilyen oldalelemben szeretnénk megjeleníteni valamilyen adatlistát, ami többnyire egyetlen táblán alapszik csak.

Ugorjunk egy kicsit, és nézzünk meg az adatkötést. Ez alatt azt kell érteni, hogy magán az oldalon szeretnénk megjeleníteni bizonyos objektumok attribútumait, és ehhez egyszerű szöveges mezőket vagy más vizuális elemeket használunk. Ilyenkor felmerül a probléma, hogy szeretnénk ezeket az attribútumokat összekötni azokkal a megfelelő elemekkel. Ehhez nem használhatjuk közvetlen az ObjectDataSourcot, mivel nem az egész objektumot akarjuk egy mezőbe beletenni. Pl.: szövegdobozhoz szeretnénk a hallgató nevét kiírni. Ezt a `<%# %>` közötti Bind() vagy Eval() függvények egyikével tehetjük meg. Ezek nem csinálnak mást,

mint az argumentumban megadott kifejezést kiértékelik, és a kapott értékeket, mint szöveget beleírják az oldal forrásban oda, ahol maguk is szerepelnek. Csak olvasható értékekhez, tehát feliratokhoz az Eval(), írható és olvasható adatokhoz, mint például szövegdobozhoz a Bind() használandó. Érdekessége ezeknek a dinamikus tartalmaknak, ami nagyon sokszor jól jött, hogy lehetséges az, hogy a CodeBehindban elhelyezett metódusokat tudunk itt meghívni. Ekkor akár a `Bind('IntezmenyID')` is lehet a paraméter, ami arra jó, hogy akár egy lista kiírása közben, dinamikusan tudjuk ezt kicserélni a megfelelő tábla megfelelő értékére, vagy bármi más transzformációt végrehajthatunk.



8. ábra Felugró ablak

Következő érdekesség az UpdatePanelek használata. Az UpdatePanel nem más, mint egy felugró ablakszerűen megjelenő kis weboldal részlet. Azért csak olyan mintha, és nem az, mivel valójában ez nem egy igazi ablak, hanem csak egy AJAX segítségével dinamikusan előhozható és eltüntethető oldalrészlet. Ahhoz, hogy AJAX-ot használhassunk az oldalon, szükség van egy ScriptManager vezérlő használatára, ezek után már bárhol elhelyezhetjük a megfelelő vezérlőket. Ezen felül még szükséges az AjaxControlToolkit nevű dll-t is a projecthez adni, mely rengeteg hasznos előre elkészített vezérlőt tartalmaz. Az AJAX-ról röviden annyit, hogy segítségével lehetőségünk van arra, hogy az oldal újratöltése nélkül végezhessünk el bizonyos folyamatokat, akár oldalrészletek frissítését is. Tehát aszinkron módon dolgozhassunk az oldalon. Egy UpdatePanel-t definiálni ezért értelemszerűen azon az

oldalon kell, ahol szeretnék ha „felugrana”. A láthatóságát a CodeBehind-ban lehet szabályozni a `show()` és `hide()` metódusok segítségével. Fontos azonban, ahhoz hogy működjön, szükség van egy gombra amihez hozzáköthetjük a vezérlést, még ha nem is akarjuk ezt a gombot megjeleníteni, akkor is bele kell tennünk oldalunk forrásába, legfeljebb a láthatóságát rejtettre változtatjuk. Ezek után már nincs más dolgunk, mint hogy egy `<asp:panel>` vezérlőben definiáljuk a szükséges tartalmat, mintha csak egy új oldalt készítenénk. Fontos, hogy a megfelelő elemek azonosítói megegyezzenek, miután ezáltal tudjuk definiálni, mely elemek mely másik elemmel együttműködve fejtsék ki hatásukat. Természetesen minden egyes oldalelem, amit a panelben elhelyezünk, miután fizikailag is ugyanazon .aspx állományban találhatóak, mint az oldal többi része, így a manipulációjuk is ugyan abban a CodeBehind fájlban történik.

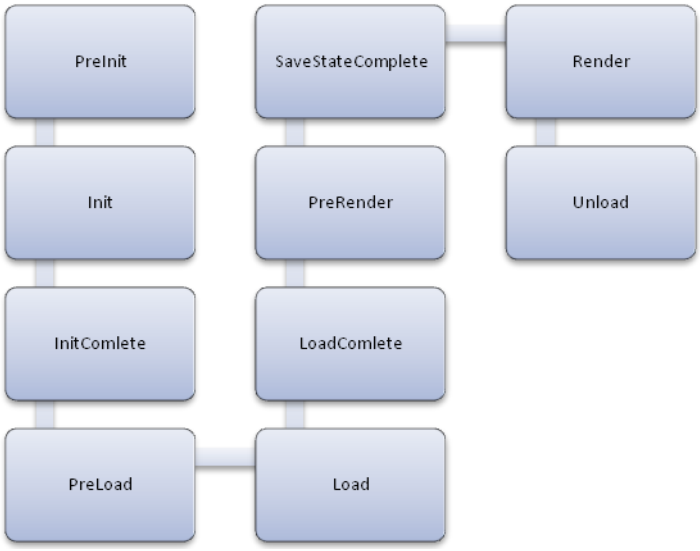
Végül, de nem utolsó sorban, a szintén sok helyen alkalmazott, úgynevezett Web User Controlokról ejtenék szót. Nem másról van szó, mint az újrafelhasználás eszközeiről, hatásuk akkor igazán áldásos mikor több helyen is ugyan azokat a vezérlőket kellene együtt használni, például azonos adatok megjelenítésére, bekérésére. Akkor is jól jöhet azonban, ha szeretnénk kódunkat kicsit átláthatóbbá tenni, például ListView-k, GridView-k esetén. Mi is hát ez? Nem más, mint egy egyszerű vezérlőből álló új weboldal, mely ugyan úgy két fizikai állományból áll, tehát van CodeBehind-ja, csak éppen a kiterjesztése ascx. Azonban a lényeg, hogy egy tetszőleges ASP oldalon ugyan úgy felhasználható, mint bármely más szerver oldali vezérlő. Így tehát nem kell mást tennünk, mint létrehozni egy új web user controlt, létrehozni a kinézetét, definiálni a viselkedését, hozzákötni az adatot, majd a felhasználási helyén beregisztrálni, mint egy új vezérlőt. Ezek után pedig csak a `<tagPrefix:TagName Attribute>` alakban felhasználni azt.

Most, hogy kitárgyaltuk az oldal kinézetének felépítését, jöjjön hát a már sokat emlegetett CodeBehind, mely megvalósítja az üzleti logikáját az alkalmazásomnak. Folytatva a hagyományok, az előbb bemutatott oldal mögött álló kódot fogom szemléltetési eszközként használni. A CodeBehind, mint már említettem egy egyszerű, szabványos C# vagy Visual Basic nyelven megírt osztályt tartalmazó állomány. Az, hogy mely nyelvet használja, attól függ, hogy projektünk készítésekor melyik alapnyelvet választottuk ki, az én esetember azért a C# lett a kiválasztott, mivel JAVA nyelvet tanultam az egyetemen, amihez ez sokkal közelebb áll és könnyebben megtanulható volt, mint a Visual Basic.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.Security;

namespace WebApp.WebPages.Bírálat.Bíráló
{
    public partial class BirálatokBironként : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            if (!User.Identity.IsAuthenticated)
                Response.Redirect("~/Login.aspx");
            if (User.IsInRole("OTDK bíráló")
               || User.IsInRole("OTDT titkárság")
               && !Page.IsPostBack)
            {
                hfBiroID.Value =
Membership.GetUser(User.Identity.Name).ProviderUserKey.ToString();
            }
        }

        protected void btnCommandHandler(object sender, CommandEventArgs e)
{
            if (e.CommandName.Equals("bírál"))
            {
                DAO.Biralat.Biralat b =
DAO.Biralat.BiralatDAO.getBiralatById(int.Parse(e.CommandArgument.ToString(
)));
                hfBírálatId.Value = b.Id.ToString();

                lbPalyamunkaCim.Text =
b.BiraltPalyamunka.OrszagosJelentkeze.Palyamunka.cim;
            }
            mpeBírálat.Show();
        }

        protected void btnPopupClose_Click(object sender, EventArgs e)
        {
            mpeBírálat.Hide();
            clearPopup();
        }

        protected void btnPopupSave_Click(object sender, EventArgs e)
        {
            DAO.Biralat.Biralat b =
DAO.Biralat.BiralatDAO.getBiralatById(int.Parse(hfBírálatId.Value));
            b.P1 = ddlP1.SelectedIndex;
            b.P2 = ddlP2.SelectedIndex;
            b.P3 = ddlP3.SelectedIndex;
            b.P4 = ddlP4.SelectedIndex;
            b.SzovegesBiralat = tbSzovegesErtekeles.Text;
            b.Megjegyzes = tbMegjegyzes.Text;
            DAO.Biralat.BiralatDAO.updateBírálat(b);
            DAO.Biralat.BiralatraVaroDAO.updateStatus(b.BiraltPalyamunka,
"Elbírálva");
```

```
            Master.displayMessageOnPopup("Bírálat sikeresen rögzítésre
került!");
            mpeBírálat.Hide();
            clearPopup();
            gvBiralatok.DataBind();


        }

        public void clearPopup()
        {
            lbPalyamunkaCim.Text = "";
            ddlP1.ClearSelection();
            ddlP2.ClearSelection();
            ddlP3.ClearSelection();
            ddlP4.ClearSelection();
            ddlP1.DataBind();
            ddlP2.DataBind();
            ddlP3.DataBind();
            ddlP4.DataBind();
            tbSzovegesErtekeles.Text = "";
            tbMegjegyzes.Text = "";
        }
    }
}
```

Maga az osztály a `partial` kulcsszóval kezdődik, mely arra szolgál, hogy olyan osztályokat tudjunk létrehozni, amelyek akár fizikailag is külön állományokban helyezkednek el, de egyébként összetartoznak. Főként a generált felületeknél alkalmazzák, mint ahogyan itt is így történt. Így jelezzük, hogy igazából ez a két állomány egy egységet alkot. Osztályunk alaphelyzetben ezen kívül még a `System.Web.UI.Page-t` származtatja, mint ősosztályt. Első metódusunk a visszatérési érték nélküli Page_Load, mely abból a szempontból érdekes, hogy általa kitérhetünk az oldal életciklus fogalomra, mely nem jelent mást, mint azokat a lépéseket, melyek végrehajtódnak minden egyes oldal generálása során, míg végül a böngészőben megjelenik a kész weboldal. Ezeket a szakaszokat mutatja a 9. ábra. Minden egyes ilyen szakaszhoz tartozhat egy megfelelő metódus, melyben elvégezhetjük az adott lépésben szükséges műveleteket. Mint látjuk, most csak a Load eseménnyel foglalkozunk, melyről azt érdemes tudni, hogy ekkor töltődnek be az oldal vezérlői. Nem csinálunk mást, mint megállapítjuk, hogy a felhasználó autentikálva van-e, tehát nem csak beírta valahogy az URL-t a böngésző címsorába, hanem szabványos jutott el ide, ha nincs akkor elküldjük őt bejelentkezni. Ha pedig be van jelentkezve, megnézzük, hogy a megfelelő szerepkörök egyikének tagja-e, és ha igen beírjuk az ID-ját egy rejtett mezőbe, miután erre később még szükségünk lesz.



9. ábra Az ASP .NET oldal életciklus fázisai

A következő metódusunk a `btnCommandHandler`, amely arra szolgál, hogy az oldalon elhelyezett btnElbírál nevű gomb által kiváltott eseményeket lekezelje. Annyiban különleges ez a megoldás, hogy ha nem hagyományos eseményről hanem Commandról van szó, akkor lehetőség van egy úgynevezett Commandargument-ben értéket átadni az eseménykezelőnek. Erre azért volt szükség, mivel itt a táblázat minden sorában szerepel egy gomb, és ahhoz, hogy azonosítani lehessen melyik sorból érkezett a kérés, a sor által reprezentált adatrekord azonosító értékét is átadom, mint argumentumot. Majd azt eltéve egy újabb rejtett mezőbe meghívom a felugró ablak `show()` metódusát, így mikor az ablak feljön, már előre fel van töltve a sorhoz kötődő adatokkal. Egyetlen érdekes dolog van még itt, mégpedig a `btnPopupSave_Click` metódus, melyben frissítek egy adatot `Biralat` táblában. Ez egy nagyon jó példa arra, hogyan is kell használni az adatelérési osztályokat kód szinten. A BiralatDAO-ban definiált getBiralatByID metódus segítségével „elkérem” az adatbázisból a megfelelő sorobjektumot, majd frissítem a megfelelő mezőket egyszerű értékadással, amelyekben az érték a weboldalon megjelenő mezőkből származik, és végül magát az objektumot is elmentem az adatbázisba. egy másik általam implementált metódus segítségével.

Ezen a nagyon egyszerű példán keresztül láthattuk tehát, hogy milyen könnyen lehetséges egészen bonyolultnak tűnő funkcionalitások megvalósítása különféle webes felületeken keresztül. És azt is láthattuk, milyen egyszerű is összekapcsolni a böngészőben megjelenő felületet az adatbázisban tárolt adatokkal, amely talán legtöbbször a legnagyobb gondot jelenti a különféle technológiákban. A következőkben a program fejlesztése során

tapasztalt egyéb érdekes és hasznos lehetőségekről, megoldásokról szeretnék írni, olyan céllal, hogy megmutassam, milyen egyéb fontos elemeket tartalmaz még a programom a látható weboldalakon kívül.

3.4 Autentikáció, autorizáció

Mint az előző példában is láthattuk az autentikáció és az autorizáció kódszinten is megjelenik az alkalmazásban. Ha belegondolunk egy több felhasználós rendszer esetében minimális követelmény lehet az, hogy képes legyen azonosítani a rendszer a regisztrált felhasználóit. Azonban itt jóval többről van szó, nem csupán az azonosítás, vagyis az autentikáció a fontos, hanem az is, hogy személytől, illetve inkább szerepkörtől függően korlátozhassuk a rendszer egyes elemeihez való hozzáférést. Ez már csak azért is fontos, mert bizonyos adatokat nem szeretnénk, ha mindenki látna, illetve nem ajánlatos az sem, hogy egyes adminisztrációs folyamatokba betekinthessen akár a hallgatók is.

Az ASP .NET szerencsére egy nagyon fejlett beépített eszközrendszert biztosít a probléma megoldására. Alkalmazás szinten definiálhatjuk a fejlesztés elején, hogy szeretnénk-e a beépített eszközrendszert használni. Ahhoz hogy ezt megtehessük, mindenképpen szükséges egy élő adatbázis kapcsolat, mivel a már korábban is említett rendszerszintű felhasználókat kezelő táblákat létre kell hozni valahol. Természetesen ezek csak alapvető adatokat tartalmaznak, mint e-mail cím, jelszó, felhasználói név. Ha egyéb adatok is kellenek, egyedi megoldásokkal bővíthető a rendszer. Mindezen beállítások egy konfigurációs állományban is megadhatóak, oda ahová a Visual Studio is begenerálja a különféle varázslókban megadott beállításokat. Ez az állomány a `web.config`, mely az alkalmazás gyökerében található, mint legmagasabb szintű konfigurációs állomány. Az itt megadott beállítások szűkíthetőek természetesen, az oldal szerkezetét alkotó mappákban újabb `web.config` állományok létrehozásával. Általános szabály volt a fejlesztés alatt, hogy a különböző jogosultsággal rendelkező felhasználók részére fejlesztett oldalak külön mappákba kerültek, és ezen mappákban egy ilyen állomány segítségével specifikáltam, kik azok akik megtekinthetik az adott oldalakat és kik nem.

Egy ilyen állomány például a következő is:

```
<?xml version="1.0"?>
<configuration>
    <appSettings/>
    <connectionStrings/>
    <system.web>
        <authorization>
            <allow roles="Admin" />
            <deny users="*" />
        </authorization>
    </system.web>
</configuration>
```

Láthatjuk, hogy teljesen szabványos XML-ről van szó, ahol az érdekes rész az authorization csomópont alatt kezdődik. Itt adhatjuk meg, kik láthatják, és kik nem láthatják a hatáskörbe lévő oldalakat. Lehetőség van felhasználónként és szerepkörönként is szűrni. A * a bárkit jelenti természetesen. Tehát a fenti példa azt jelenti, hogy csak az Admin felhasználók tekinthetik meg az oldalt, mindenki más kizárva. Fontos, hogy azt is adjuk meg ki nem, nem csak azt, hogy ki igen! Természetesen nem csak konfigurációs szinten, hanem kódszinten is van lehetőség az autorizáció kezelésére. Erre olyankor lehet szükség, ha weboldalanként szeretnénk bizonyos felhasználó-függő funkcionalitást megvalósítani, például más és más adatokkal feltölteni egy listát annak függvényében, hogy ki nézi azt meg. Vagy bizonyos funkcionalitásokat letiltani egyes szerepkörök képviselőinek, amire sok helyen van példa az alkalmazásomban is. Ilyen felhasználási eset lehet, hogy ha egy hallgatónak nem szeretnénk megengedni, hogy lássa, kik bírálták el pályamunkáját, viszont azt szeretnénk, ha látná, milyen pontszámokat kapott, ekkor nem kell két külön oldalt készíteni, csupán az egyes elemek láthatóságát aszerint szabályozni, hogy éppen ki az aktuális felhasználó.

Miután már vannak felhasználókat tároló táblák és van autorizáció, arról is beszélni kell, hogyan fognak ezek a felhasználók a rendszerbe belépni, illetve hogyan zajlik az új felhasználók regisztrálása. A következőkben leírtak a rendszer jelenlegi állapotát tükrözik, de elképzelhető, hogy lesznek ezen a téren jelentős követelmény béli változásokat követő átalakítások. Tehát a felhasználók beléptetése egy beépített úgynevezett LoginControl használatával zajlik, melynek használata nagyon kényelmes a programozó számára, ugyanis az előzőekben elkészített felhasználói adatbázisra és a megfelelő beállításokra támaszkodva képes lekezelni a felhasználó beléptetetését és a felhasználói munkamenet létrehozását. Innentől kezdve egy másik control, az úgynevezett LoginView szolgál arra, hogy megjelenítsük a belépett felhasználó nevét, illetve ha még nem lépett be a felhasználó itt

tudjuk azt is jelezni számára, ezen kívül a kijelentkeztetést is általa valósul meg. Magára a felhasználói regisztrációra is létezik beépített eszköz, de ezt inkább saját űrlappal oldottuk meg, mely minden tárolt adatot bekér a jelszót leszámítva, ugyanis azt a rendszer generálja, majd küldi ki e-mailbe a megadott címre. Azért írtam, hogy oldottuk és nem oldottam, mivel mint említettem a rendszer a felhasználókat a KisBagoly rendszerből veszi át a felhasználók kényelme és az egyszerűség végett.

3.5 Automatikus e-mailküldés

A követelmények böngészése közben megtalálhatjuk, hogy a rendszertől azt is elvárnánk, hogy képes legyen bizonyos események bekövetkezte előtt automatikusan értesítést küldeni az érintett felhasználóknak, hogy felkészülhessenek arra. Ilyen esemény lehet az, hogy bizonyos időn belül le fog telni az elbírálásra szánt határidő, de természetesen más hasonló feladatra is tökéletesen alkalmazható ez a megoldás. Kielemezve a helyzetet láthatjuk, hogy amire szükség van informatikai szempontból, az nem más, mint egy olyan szolgáltatás, ami időzítetten és automatikusan képes futni a háttérben, és valamilyen adatbázisban tárolt adatok alapján e-mailt küld bizonyos felhasználóknak, valamilyen tartalommal. Ha már pontosan ismert a követelmény, meg kellet határozni milyen technológiák nyújthatnak segítséget ennek megvalósításában. Maga az ASP .NET nem támogatja azt, hogy időzítetten, felhasználói esemény nélkül induljanak el bizonyos folyamatok, így más megoldást kellett keresnem. A választásom egy önálló kis parancssori alkalmazásra esett, mely nem csinál mást, csupán lefuttat egy lekérdezést az adatbázison és a megfelelő visszatérési adatok alapján e-mailt küld az adott felhasználóknak. Innentől kezdve pedig annyit kell már csak tenni, hogy a szerver számítógép operációs rendszerében beállítom az alkalmazásom időzített futtatását a megfelelő paraméterekkel.

```
private static Boolean TestSendEmail(){
            try
            {
                MailMessage mail = new MailMessage();
                mail.From = new MailAddress("XXXXXXX@xx.com");
                mail.To.Add("#####@xx.com");
                mail.Subject = "Test Email";
                mail.Body = "";
                SmtpClient smtp = new SmtpClient("xxx.xxx.xxx.xxx");
                smtp.Send(mail);
```

```
                return true;
            }
            catch (Exception ex)
            {
                return false;
            }
}
```

A kódban szereplő e-mail címeket biztonsági okok miatt most valótlan adatokra cseréltem, de ettől függetlenül a példa érthető. Egy új `MailMessage` objektumot kell létrehozni, mely magában foglal minden szükség e-maillel kapcsolatos adatot és beállítást, majd ha megfelelően beállítottuk a paramétereket, akkor egy `SmtpClient` objektum segítségével el lehet azt küldeni. Természetesen csatolmányokat is tud kezelni a rendszer, valamint több címzettnek is el lehet küldeni a levelet.

```
private static void Run()
{
   string connectionString = "Data Source=.\\sqlexpress;Initial
Catalog=otdkdb;Integrated Security=True";
   SqlConnection myConnection = new SqlConnection(connectionString);
   myConnection.Open();
   Console.WriteLine("Database Connection has been made.....");

   String query = "SELECT"+
                  " dbo.aspnet_Users.UserName, " +
                  " dbo.BiraltPalyamunka.KovHatarido, " +
                  " dbo.aspnet_Membership.Email as email, " +
                  " dbo.Palyamunka.cim as title " +
                  " FROM" +
                  " dbo.aspnet_Users INNER JOIN" +
                  " dbo.Biralat INNER JOIN" +
                  " dbo.BiraltPalyamunka ON dbo.Biralat.BiraltPalyamunkaId
= dbo.BiraltPalyamunka.Id ON dbo.aspnet_Users.UserId = dbo.Biralat.BiroId
INNER JOIN" +
                  " dbo.aspnet_Membership ON dbo.aspnet_Users.UserId =
dbo.aspnet_Membership.UserId INNER JOIN" +
                  " dbo.OrszagosJelentkezes ON
dbo.BiraltPalyamunka.OrszagosJelentkezesId =
dbo.OrszagosJelentkezes.orszagosJelentkezesID INNER JOIN" +
                  " dbo.Palyamunka ON dbo.OrszagosJelentkezes.palyamunkaID
= dbo.Palyamunka.palyamunkaID";
   SqlCommand sqlComm = new SqlCommand(query, myConnection);
   SqlDataReader r = sqlComm.ExecuteReader();
   while (r.Read())
   {
      String emailto = (String)r["email"];
      String body    = (String)r["title"];
      Console.WriteLine("(" + emailto + ")");
      SendEmail(emailto, "Notification", body);
   }
   r.Close();
}
```

A fenti kódrészlet pedig azért felelős, hogy az adatbázisból kinyerje a megfelelő adatokat, amelyre majd az e-mailküldés épülhet. Itt nem az előzőekben megismert LinqToSql technológiát használom az adatelérésre, hanem egy sokkal egyszerűbb megoldást. Működési elve, hogy egy szabványos connectionString segítségével létrehozom az adatbáziskapcsolatot, majd pedig egy hagyományos T-SQL lekérdezést futtatok az adatbázison, melynek eredménye egy SqlDataReader objektumba kerül. Innen egy ciklus segítségével lehet a visszatérési adathalmazt soronként kiolvasni úgy, hogy a megfelelő értékeket megfelelő típusú változókba mentem. Ezek után pedig nincs más dolgom, csak kiolvasott értékekre soronként meghívni a már megismert e-mail küldő függvényt, ami a megfelelő értékekkel el is küldi az üzenetet. Természetesen lehetett volna itt is rétegzett architektúrát és LinqToSql-t használni, de miután egy nagyon egyszerű alkalmazásról van szó, melytől elvárjuk, hogy gyorsan és kevés erőforrás felhasználássával végezze dolgát, így teljesen felesleges lett volna ennél összetettebb megoldást választani.

3.6 Riportálás, nyomtatási listák

Fontos, hogy egy teljesen web alapú alkalmazás, mely monitoron történő megjelenítésre lett optimalizálva valamilyen kézzel fogható, papír alapú kimenetet is képes legyen produkálni. Ez annál is inkább így van, miután egy alapvetően adminisztrációs célokat szolgáló programról van szó, melytől jogos elvárás, hogy különféle dokumentumokat, kimutatásokat készítsen a rendszerben szereplő adatokról. Mint a bevezetőben is említettem a ’KisBagoly’ rendszer fejlesztésekor az én feladatom volt kitalálni és lefejleszteni a szükséges komponenseket, melyek a hallgatói jelentkezési lapok és a különféle egyéb kimutatások nyomtatásáért feleltek. Természetesen több megoldási lehetősig is rendelkezésünkre áll egy ilyen esetben:

- Professzionális reporting modulok → fizetős és drága alkalmazások
- Egyszerű nyomtatóra optimalizált megjelenítés a weboldalakon → primitív és nehezen optimalizálható A/4-es oldalra
- Microsoft Reporting Services → listákhoz kitűnő, űrlapokhoz alkalmatlan
- Ingyenes eszközök P.: ItextSharp → elsősorban űrlapokhoz jó, kissé bonyolult szerkezet

Látva az előző listát érdemes mérlegelni, milyen az elvárt kimenet és annak megvalósítására mely technológia a leginkább alkalmas. Természetesen a fizetős modulok között találunk teljes funkcionalitást kínáló professzionális eszközöket, melyek képesek lennének minden igényt kielégíteni, de ez egy teljesen pénz nélkül fejlesztett project volt, ahol ez nem jöhetett szóba. Így esett a választásom az utolsó két említett két eszköz kombinációjára, melyekről szeretnék bővebben beszélni a következőkben.

Először is az iTextSharp, ami nem más, mint az ingyenes, PDF generálásra alkalmas JAVA környezetben fejlesztett iText Framework .Net-be átírt változata. Miután a PDF kitűnően használható operációs rendszertől független környezetben, valamint nagyon jól méretezhető és pozícionálható elemekből áll, ezért kitűnő megoldás a különféle űrlapok nyomtatására. A felhasználó csak kattint az oldalon és a program máris legenerálja a szükséges dokumentumot. Természetesen nem előre elkészített állományokkal dolgozunk, ugyanis a legfontosabb az, hogy dinamikusan változzon a tartalma ezeknek a fájloknak. A framework által számunkra felkínált eszközrendszer lehetővé teszi, hogy mintegy HTML szerűen különféle objektumok segítségével leírjuk a megjelenítendő oldalt. Tehát nem valamiféle template állományt töltünk fel adatokkal a megfelelő helyeken, hanem teljesen OO eszközökkel tudjuk leírni a kimeneti állomány kinézetét. A következő rövid kis szemléltető kódrészletből mindez jól megmutatható:

```
Cell cella = null;

            datatable.AddCell(new Paragraph("#", tfejlec));
            datatable.AddCell(new Paragraph("OTDT azonosító", tfejlec));
            cella = new Cell(new Paragraph("Szerző(k)", tfejlec));
            cella.HorizontalAlignment = Rectangle.ALIGN_LEFT;
            datatable.AddCell(cella);
            datatable.AddCell(new Paragraph("Intézmény", tfejlec));
            cella = new Cell(new Paragraph("Dolgozat címe", tfejlec));
            cella.HorizontalAlignment = Rectangle.ALIGN_LEFT;
            datatable.AddCell(cella);
            datatable.AddCell(new Paragraph("OTDT jóváhagyás*", tfejlec));
            datatable.AddCell(new Paragraph("Rendezői jóváhagyás", tfejlec));
            cella = new Cell(new Paragraph("OTDT megjegyzés**", tfejlec));
            cella.HorizontalAlignment = Rectangle.ALIGN_LEFT;
            datatable.AddCell(cella);
            cella = new Cell(new Paragraph("A szakmai bizottság és a rendező
intézmény megjegyzése**", tfejlec));
            cella.HorizontalAlignment = Rectangle.ALIGN_LEFT;
            datatable.AddCell(cella);
                 datatable.AddCell(new      Paragraph("Végleges      jóváhagyás***",
tfejlec));
```

Ez a részlet az országos jelentkezések listájának oldalankénti fejlécét hivatott megjeleníteni.

A másik említett módszer az rdlc riportok létrehozása. Ez egy beépített eszköz a .NET keretrendszerben, mely mind a desktop alkalmazások mind az ASP .NET alkalmazások számára elérhető formában tartalmazza a riport kinézetére és felépítésére vonatkozó információkat. Ahhoz hogy létrehozzuk a kívánt megoldást első lépésben egy úgynevezett DataSet objektumot kell definiálnunk, mely hivatkozásokat tárol a megjeleníteni kívánt nyers adatokra. Ennek magadása úgy történik, hogy kiválasztjuk egy listából a szükséges táblákat, nézeteket, esetleg bizonyos metódusok megírásával szűkítjük az ezekből kinyerhető adatokat, majd ezen metódusok segítségével teremtünk kapcsolatot az adott objektumhoz a későbbiekben. Én az alkalmazásban ezt úgy implementáltam, hogy létrehoztam adatbázis szinten olyan nézeteket, amelyek már a közel végleges formában tartalmazták a riportálásra szánt adatokat, majd ezeket a nézeteket egy az egyben felhasználtam a DataSet-be mint egyetlen adatbázis elemet, így valójában tehát a DataSet csak arra szolgált, hogy egy objektumot építsek az adatbázis elem fölé. Ezek után kell létrehoznunk az .rdlc állományt, melyre két féle képen van lehetőségünk. Vagy kézzel állítjuk össze különböző előre definiált alkotóelemekből, vagy egy varázsló segítségével megcsináljuk a főbb lépéseket, és később kézzel szabjuk személyre az elkészült sémát. A második módszer természetesen sokkal célravezetőbb és gyorsabb, így én is ezt használtam. Az rdlc állomány egyébként nem más, mint egy template fájl, ami leírja, hogy mely DataSet-ből származó, mely adatokat milyen elrendezésben jelenítsük meg. Lehetőség van természetesen az adatok manipulálására beépített függvényszerkesztő segítségével, használhatunk paramétereket a megjelenítendő adatok szűrésre, valamint teljes mértékben testre szabhatjuk a beépített eszközök segítségével a megjelenített kimeneten az adatok elhelyezését. Ha ilyen formán elkészült a leíró állomány, akkor már nincs más dolgunk, mint létrehozni egy weboldalt, ahol elhelyezünk egy ReportViewer controlt, ami felelős lesz azért, hogy megjelenítse az rdlc fájlt az oldalunkon, melyet aztán a felhasználók a beépített lehetőségeknek köszönhetően, többféle általános elterjedt formátumba is kiexportálhatnak, egyetlen gombnyomással. Azonban ezt az állományt nem csak a weboldalunkon mutathatjuk meg a felhasználóknak, hanem létezik egy a reportviewer-hez hasonlatos megoldás a winform-os alkalmazások számára is, így egy elkészült riport újrafelhasználhatóvá válik ilyen értelemben.

4.Az alkalmazás áttekintése

Ebben a fejezetben szeretném végigvezetni az olvasót azon a munkafolyamaton, melyet az általam fejlesztett alkalmazás támogat. Fontos megjegyeznem, hogy ez a folyamat még változhat annak függvényében, hogy a rendszer majdani felhasználói hogyan képzelik azt el, hogyan lenne ez kényelmes és felhasználóbarát számukra.

Bírálatra Váró pályamunkák:

Ez a felület arra szolgál, hogy a szekció felelőse az adott szekcióba jelentkezett, és az adminisztrációs kritériumoknak megfelelő pályamunkákat akár együttesen kijelölve is, bírálatra küldje. A bírálatra küldés úgy történik, hogy kijelöli a megfelelő pályamunkákat és a felugró ablakból hozzájuk rendel két intézményt, melyek részére az elfogadást követően bírálati kérelmekként megjelennek az adott pályamunkák.

Bírálatok karbantartása:

Szintén a szekciófelelős által elérhető oldal. Itt lehet megtekinteni intézményenként, hogy az adott szekcióba nevezett pályamunkák bírálata éppen milyen stádiumban van. Akár vissza lehet vonni az adott dolgozat teljes bírálati folyamatát is, ekkor törlésre kerül a minden bírálati felkérés, ami az adott pályamunkához kapcsolódott.

Bírálatok Megtekintése:

Az egyik leguniverzálisabb eszköz. Több szerepkör felhasználói is megtekinthetik, természetesen csak a számukra elérhető információkat látva. Feladata, hogy egy adott bírálat alatt lévő pályamunkával kapcsolatos minden bírálati információt megjelenítsen. Ha egy szekciófelelős használja az oldalt, akkor lehetősége van megtekinteni azt, hogy az egyes bírálók, hogy állnak a folyamatban, illetve milyen pontszámokat adtak, vagy adtak-e egyáltalán már pontot egy adott dolgozatra. Ha szükséges döntőbírálatot lehet kérni, illetve törölhet egy bírálati felkérést és újat is adhat ki. Ha egy hallgató tekinti meg, akkor névtelenül láthatja az elkészült bírálatokat. Valamint a helyi felelősök és a bírák is megnyithatják, ekkor csak a saját pontszámukat látva.

Bírálatok Intézményenként:

Az adott intézmény helyi felelősének szánt felület, lehetősége van arra, hogy az ő általa képviselt intézménybe érkezett felkéréseket kezelje. Ez azt jelenti, hogy új felkérés esetén bírálót rendelhet hozzá, ha egy bíráló visszautasítja, akkor újat adhat hozzá. Valamint megtekintheti a bírálati adatokat.

Bírálati lista Bíráknak:

Arra szolgál, hogy a rendszer azon felhasználói, akik bírálati joggal rendelkeznek, azok az intézményi felelős által hozzájuk rendelt pályamunkákat listában láthassák. Lehetőség van megtekinteni az adott pályamunka elektronikus verzióját, majd ezt követően egy táblázatban rögzíteni a bírálati pontszámokat és egy szöveges bírálatot. Majd ha ez kész, akkor véglegesíteni azt, melyet a rendszer többi szereplője ezek után fog látni.

Bírálati Státusok karbantartása:

Az OTDT titkárság számára készült, arra szolgál, hogy a bírálatok egyes állapotaihoz rendelhető status értékeket karbantarthassák. Újakat vehessenek fel, törölhessenek a listából. Ezen a statusok fognak később megjelenni az egyes bírálatok esetén, mint aktuális állapotok.

Riportok:

A bírálati folyamat segítésére szolgáló, kinyomtatható listák és dokumentumok. Jelenleg még számuk nem végleges, a végső felhasználás fogja eldönteni. Ami már jelenleg is létezik, és nagy valószínűséggel a későbbiekben is hasznos lesz, az egy olyan riport mely intézményenként tartalmazza a küldött és fogadott bírálati kérelmek számát.

Összességében úgy érzem, sikerült mindazt megvalósítanom, amit az alkalmazás fejlesztése előtt elterveztem. Egy jól használható, a szem számára is kellemes, a felhasználók mindennapjait segítő alkalmazást akartam létrehozni. A projekt világos célja volt az, hogy egyfajta útmutatást adjon arra, ezt így is lehet csinálni. Az, hogy így fogják-e csinálni, már nem a projekt célja, és nem fogom személyes kudarcként megélni, ha csak ez a diplomamunka őrzi majd munkám emlékét. A következő OTDK-t rendező és lebonyolító intézmények és emberek kezében van innentől a döntés, hogy érdemes-e ebbe a rendszerbe még energiát és időt ölni, továbbfejleszteni azt a végső igényekhez.

5. Összefoglaló

Dolgozatom zárásaként szeretném röviden és tömören összefoglalni, hogy pontosan mivel is foglalkoztam a téma kidolgozása során. A kezeim között megszületett rendszer nem más, mint egy ASP .NET technológián alapuló webes alkalmazás, melynek feladata az OTDK versenyek bírálata során végzett adminisztrációs folyamat támogatása. Rétegzett felépítést választottam a program fejlesztéséhez, ami azt jelenti, hogy az adatbázis, az adatelérés az üzleti logika és a megjelenítés nem csak logikailag, hanem fizikailag is elválik egymástól, így biztosítva az átláthatóságot és a rugalmasságot. Az alkalmazás fejlesztés során a Microsoft megfelelő technológiáit használtam fel, minden esetben törekedve a leginkább az igényekhez igazodó változatok kiválasztásához. Ennek megfelelően az adatbázist az SQL Server 2005 Express verziója futtatja, szükséges scriptjeit Transact SQL nyelven (lásd [2]) írtam meg, melyről bővebben olvashatunk a 3.1-es szakaszban. Az adatbázist és a rendszer többi részét egy adatelérési réteg köti össze, mely egy manapság nagyon divatos és fontos technológiát hív segítségül a LinqToSql személyében, erről és ennek használatáról többet a 3.2-es fejezetben olvashatunk. A felhasználókkal történő kapcsolattartás és a követelményekből fakadó a rendszer működésével szemben támasztott elvárások megvalósítása az ASP .NET által kínált szolgáltatások segítségével jött létre, melynek részleteiről a 3.3-as rész nyújt több információt. Mindezen komponenseket a Visual Studio 2008 fejlesztői eszköz segítségével készítettem el.

Ahogy látjuk, alkalmazásom jóval túlmutat egy HTML alapú statikus weboldal fejlesztésén és olyan eszközöket vonultat fel, melyek lehetőséget adnak akár a klasszikus asztali alkalmazások által nyújtott szolgáltatások kiváltására is. A fejlesztés során tudatosan törekedtem arra, hogy ne csak pusztán egy leírását adjam munkámnak, hanem igyekezzek a felhasznált korszerű technológiákat minél érthetőbben és pontosabban összefoglalni. Így munkám akár olyan szempontból is érdekes lehet, ha valaki csak a dinamikus webes alkalmazások fejlesztése iránt érdeklődik. Igyekeztem olyan elemeket kiemelni, melyek érdekesek és tanulságosak lehetnek, melyek elkészítése során akár magam is problémákba ütköztem, és akár mintaként is szolgálhatnak hasonló esetek megoldására.

6. Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani elsősorban Dr. Végh Jánosnak, aki témavezetőm és mentorom volt mindvégig, segített megtalálni kérdéseimre a választ és ajtaja mindig nyitva állt előttem, akár csak egy beszélgetésre is. Elsősorban az Ő érdeme, hogy ez a projekt egyáltalán gondolati szinten megszülethetett, majd onnan tovább haladva eljuthattam idáig. Szeretném megköszönni hallgatótáraimnak, Miskolczi Zsoltnak és Szatmári Andrásnak, hogy esélyt adtak arra, hogy részese legyek mindannak, amiről most itt írtam, és ami még azután következhet. Szeretném megköszönni nekik, hogy türelmesen és megértően segítettek nekem abban, hogy az általuk már megszerzett tudást én is megkaphassam, és hogy együtt léphessek be velük az előttünk megnyílt kapukon. Valamint szeretném megköszönni az OTDT titkárságnak mindazt a segítő munkát és sok köszönetet, mellyel munkánkat elismerték, és amellyel erőt adtak a folytatáshoz. Végül, de nem utolsó sorban pedig családomnak és páromnak, valamint szeretteimnek és mindazoknak, akik kitartottak mellettem ez alatt az öt év alatt, áldozatos munkájukkal és türelmükkel segítettek engem, végigélték velem az összes élményt, és végigküzdették velem az összes nehézséget, amelyeken keresztül mentem az elmúlt évek alatt!

7. Függelék

7.1 A fejlesztés során használt eszközök

- Microsoft Visual Studio 2008 sp1

  http://msdn.microsoft.com/en-us/vstudio/default.aspx

- Microsoft SQL Express 2005

  http://www.microsoft.com/sqlserver/2005/en/us/express.aspx

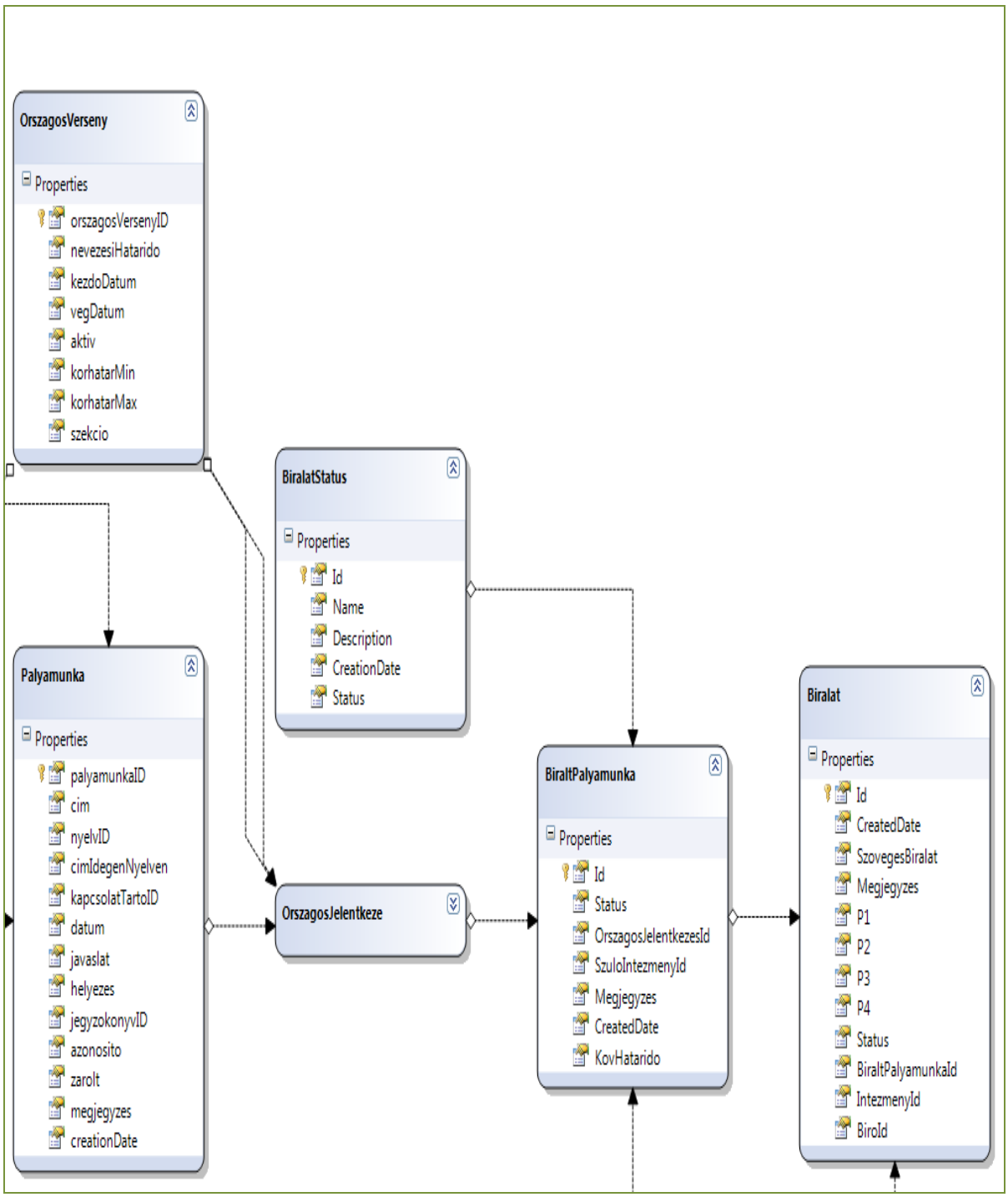
- SQL Server Managemant Studio Express 2005

  http://www.microsoft.com/downloads/details.aspx?FamilyId=C243A5AE-4BD1-4E3D-94B8-5A0F62BF7796&displaylang=en

- iTextSharp

  http://itextsharp.sourceforge.net/

7.2 Teljes adatbázis diagram



7.3 Irodalomjegyzék

[1] Open Conference System: Online dokumentáció
http://pkp.sfu.ca/?q=ocs

[2] Transact SQL on-line referencia
http://msdn.microsoft.com/en-us/library/ms189826(SQL.90).aspx

[3] Linq összefoglalás
http://en.wikipedia.org/wiki/Linq#Architecture_of_LINQ_in_.NET_Framework_3.5

[4] ASP .NET: Wikipedia szócikk
http://en.wikipedia.org/wiki/Aspx#Code-behind_model

[5] Reiter István: C#; elektronikus jegyzet
http://people.inf.elte.hu/reiter_i/csharp.pdf

[6] Trey Nash: C# 2008

[7] Panem Kiadó

7.4 Telepítési és fejlesztési útmutató

Ezt a részt azzal a céllal készítem munkámhoz, hogy ha a következő OTDK rendező úgy gondolja érdemes időt és energiát fektetni a rendszer véglegesítésébe és ezt esetleg saját fejlesztőkkel szeretné megtenni, akkor könnyedén bekapcsolódhassanak a munkába. A főbb fejlesztési irányelveket tartalmazzák az előző fejezetek, így itt pusztán csak a szükséges technológiai lépéseket szeretném felsorolni az irodalmiság igénye nélkül.

Az adatbázis beállítása:

Első és legfontosabb lépés a megfelelő eszközök telepítése után, hogy a szükséges adattáblákat létrehozzuk a rendszer számára. A könyvtárban leadásra kerülő CD-ROM tartalmaz minden szükséges SQL scriptet, ami a táblák létrehozásáért felelős. Természetesen önmagában nem fog működni, miután szükség van a Kisbagoly rendszer tábláira is. Fontos, hogy a rendszer az adatbázist *otdkdb* néven ismeri, így érdemes ezt a nevet használni, ha nem akarunk a más elnevezésből fakadó változtatásokkal foglalkozni. A mellékelt scripteket legegyszerűbben a már említett Microsoft SQL Server Managemant Studio programmal lehet lefuttatni.

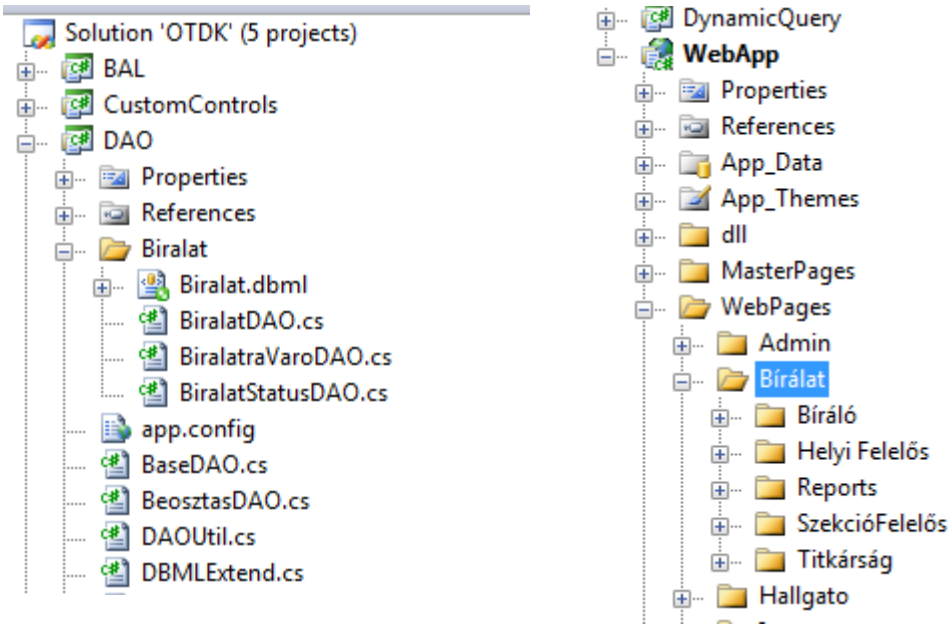
Lépései:

- Az *otdkdb* adatbázis létrehozása a megfelelő állapotban
- Az adatbázis kiválasztása a bal oldali listából
- A mellékelt script megnyitása
- Futtatás
- Győződjünk meg arról, hogy a kívánt táblák létrejöttek

A forrásállományok használata:

Szintén a CD-ROM tartalmazza a dolgozatban már több helyen is idézett forrásállományokat. Mint azt ott is említettem a megjelenő weboldalak fizikailag több állományból épülnek fel, így sokkal több állománnyal kell foglalkoznunk mint a megjelenő oldalak száma. Első lépésben a megadott állományokat a megadott helyekre be kell

másolnunk, majd meg kell nyitnunk a ’Kisbagoly’ rendszer solution-jét Visual Studio-ban. Ha ez megtörtént jobb oldalon a solution explorer ablakban láthatjuk a teljes alkalmazást alkotó elemek hierarchikus listáját, ahol a jobb egérgombbal klikkelve az *add existing item* segítségével logikailag is hozzáadhatjuk a már fizikailag létező elemeket. A következő ábrák jelzik, hogy mely állományokról és milyen szerkezetről is van szó.



10. ábra Az alkalmazás a Solution explorerben

Ha ezzel megvagyunk, akkor szükséges, hogy újra buildeljük az alkalmazást, hogy minden egyes projekt frissüljön az új elemekkel. Több tipikus hiba is ekkor szokott előjönni, mint például a connection string eltéréséből adódó hibaüzenetek. Ebben az esetben legtöbbször az a baj, hogy míg az SQL Express használata esetén így néz ki:

```
      connectionString="DataSource=.\sqlexpress;Initial                Catalog=otdkdb;Integrated
Security=True"
```

Addig a teljes verziós SQL Server esetén:

```
      connectionString="Data Source=.\;Initial Catalog=otdkdb;Integrated Security=True"
```

Ezeket a beállításokat a web.config állomány és a DAO projektben található .dbml állományok tartalmazzák, így fontos, hogy mindkét helyen ellenőrizzük helyességüket. ha ezeket a lépéseket megtettük, akkor elméletileg már több probléma nem lehet a fejlesztés folytatásával, a project pontosan abban az állapotban fog az adott gépen szerepelni, ahogy én azt most abbahagytam.