DIPLOMAMUNKA

Podlovics Tibor

Debrecen

2010

Debreceni Egyetem
Informatikai Kar

Webes projektmenedzselő alkalmazás fejlesztése

Témavezető:
Dr. Kuki Attila
Egyetemi adjunktus

Készítette:
Podlovics Tibor
Programtervező matematikus

Debrecen
2010

Tartalomjegyzék

Ábrák jegyzéke

1. Bevezetés

1.1. A diplomamunkám témája

Dolgozatom témája egy olyan projektek menedzselésére alkalmas webes alkalmazás megtervezése és implementálása, amely kimondottan kis vállalkozások számára készül és nagyban elősegítheti a vállalkozás sikeres működését. A munkám során elkészülő alkalmazást szoftverfejlesztéssel foglalkozó cégek számára terveztem ügyelve arra, hogy igény esetén kevés módosítással más területen működő cégek is hatékonyan használhassák ezt a nagyszerű eszközt.

1.2. A témaválasztás indoklása

Régóta érdekel az internet, a web világa, így mindenképpen valamilyen webes alkalmazást szerettem volna elkészíteni. A mai rohanó világunkban a web térhódítása megkérdőjelezhetetlen. Nemcsak a vállalkozások, hanem a magánemberek igényei is egyre nőnek a mobilitással szemben, az internetezők megszokták a világháló által nyújtott rengeteg kényelmi megoldást (online számlabefizetés, vásárlás stb.), s szinte elvárják, hogy mindig minden a rendelkezésükre álljon. Manapság webes alkalmazások végtelen sorát használják emberek milliói minden egyes nap, gondoljunk csak az elektronikus levelezésre, a pénzügyi tranzakciók lebonyolítására, vagy akár csak egy mozijegy megrendelésére.

Témám pontos kiválasztásában sokat segített az egyetemen hallgatott Informatikai projektmenedzsment kurzus, melynek keretein belül lehetőségem volt megismerkedni azokkal a technikákkal és eszközökkel, amelyek segítségével a vállalatok menedzselik futó projektjeiket, törekedve arra, hogy elkerüljük az idő- és költségkeret túllépését, minimalizálják a kiadásokat és maximalizálják a bevételeket. Megismertem a Microsoft Project szoftver legfontosabb funkcióit, és mindez felkeltette érdeklődésemet a téma iránt. Amennyire lehetőségem volt megismerni az említett szoftvert, azt tapasztaltam, hogy meglehetősen robosztus alkalmazás, nagyon sok funkciót építettek bele, melyek egy nagyvállalat számára kétség kívül hasznosak, de feltételezésem szerint egy kis vállalkozás azok nagy részét nem tudja kihasználni. Ezért döntöttem úgy, hogy készítek egy olyan projektmenedzselő alkalmazást, amely csak a legfontosabb funkciókat tartalmazza, így sokkal könnyebben kezelhető, ezáltal egy kis vállalkozás számára ideális.

1.3. Célkitűzések megfogalmazása

Célom egy olyan webes projektmenedzselő alkalmazás elkészítése, amely átlátható, kezelése könnyen elsajátítható, csak azokat a legfontosabb elemeket tartalmazza, amelyeket egy kisebb cég is ki tud használni, ugyanakkor hatékony segítőtársa lehet a vállalkozás életének. Véleményem szerint a komplex programok nem segítik kellő mértékben egy kis vállalkozás munkáját, hiszen a funkcióknak csak töredékét tudják kihasználni, cserébe viszont egy sokkal bonyolultabban kezelhető, nehezen tanulható alkalmazást kapnak.

Az alkalmazás fejlesztése mellett a legfontosabb célom az, hogy készítése közben gyakorlatot szerezzek, mélyebb ismeretekre tegyek szert a webre történő fejlesztés terén, megismerkedjek a web fejlesztéshez szükséges eszközökkel, elsajátítsam a technológiákat és tapasztalataim révén átfogóbb képet kapjak erről az egyre bővülő világról.

Munkám során törekedtem arra, hogy a gyakorlatba is átültessem a Rendszerfejlesztés technológiája nevű kurzuson hallottakat, és a fejlesztés során végigvezessek egy szoftverfejlesztési életciklust. Tanulmányoztam több folyamatmodellt is és választás végül az evolúciós modellre, azon belül is a feltáró prototípuskészítésre esett, melynek lényege, hogy miután megértettük a követelmények egy részét, specifikáljuk azt, majd implementáljuk, validáljuk és előállítunk belőle egy kezdeti szoftvert. Ezután a folyamatot újra végrehajtjuk mindaddig, amíg az összes követelményt fel nem dolgoztuk. Így egy egyre több funkcióval rendelkező szoftvert kapunk. Azért ezt a modellt választottam, mert a fejlesztés kezdetekor még nem volt minden követelmény teljesen világos, valamint számomra szimpatikus volt, hogy hamar állt működő szoftver a rendelkezésemre.

2. Felhasznált eszközök és technológiák

Webes alkalmazás révén beszélnünk kell szerver- és kliens oldalról. A következőkben szeretnék néhány szót szólni a tervezés és fejlesztés során alkalmazott eszközökről, technológiáról. Fontos szempont volt számomra, hogy a munkám során csak ingyenes eszközöket használjak. Elsőként bemutatom, milyen eszközöket használok, melyek esetemben a XAMPP nevű programcsomag részeként jelennek meg. Ezután az Ajax technológiát és az ezt megvalósító Ext JS javascript függvénykönyvtárat (keretrendszert) tárgyalom.

Mindenekelőtt azonban szükségét érzem, hogy definiáljam a webes alkalmazás fogalmát, hiszen az egész munkám erre épül, valamint szólok néhány szót azok felépítéséről, előnyeiről és hátrányairól.

*„A szoftverfejlesztés világában a **web alkalmazás** (Web application, WebApp) egy program, melyet a weben keresztül érünk el az interneten, vagy intranet hálózaton.”*
(Forrás: http://www.webgo.hu/web-alkalmazas-web-application-webapp)

A web alkalmazások felépítésüket tekintve három rétegre bonthatók. Az első a megjelenítési réteg, amely tulajdonképpen maga a böngésző. A második réteg a web szervereken található üzleti logika, s végül a harmadik rétegbe magát az adatot soroljuk, az őket tároló adatbázisok formájában.

Végül vessünk egy pillantást a web alkalmazások legfontosabb előnyeire és hátrányaira. Egyik nagy előnye, hogy nincs szükség telepítésre és az alkalmazás bárhol elérhető, ahol van internet kapcsolat. Népszerűségének fő oka, hogy az őket használó web böngésző kliensek szinte minden gépen rendelkezésre állnak. Kiemelt fontosságú és az asztali alkalmazásokkal szembeni legnagyobb előnye a rendkívül magas fokú rendelkezésre állás. Továbbá a web alkalmazások költséghatékonyak, kezelésük könnyen elsajátítható, frissítésük legtöbbször automatikusan történik. Végül ide sorolnám azt a megfigyelhető tendenciát, miszerint hatalmas ütemben nő a web alkalmazások népszerűsége, ami önmagában is jelentős fejlesztéseket indukál ezen a területen. Legnagyobb hátránya az alkalmazás függősége az internettől, hiszen web nélkül nincsen webes alkalmazás sem. A másik dolog, amit hátrányként szeretnék megemlíteni, hogy az asztali alkalmazásokhoz képest egy webes alkalmazásnál sokkal nagyobb figyelmet kell szentelni a biztonságnak, ami nyilvánvalóan több erőforrást igényel.

2.1. XAMPP 1.7.3

Munkám során a XAMPP webszerver csomag 1.7.3-as verzióját fogom használni, mely a dolgozat írásakor a legfrissebb elérhető stabil verzió. Néhány szóban bemutatom ezt a könnyen installálható, rendkívül praktikus csomagot. A XAMPP segítségével a fejlesztési fázisban a munkát nem kell mindig feltölteni egy szerverre, hogy tesztelni tudjam, hanem a saját számítógépemen mindezt könnyedén megtehetem, ami nagyban megkönnyíti és gyorsítja a tesztelés folyamatát. A programcsomag nagy előnye, hogy minden lényeges és főként ingyenes kiszolgálót egyesít. Találunk a csomagban többek között web-, adatbázis-, FTP- és levelező kiszolgálót is, s mindenből a legfrissebb verziót tartalmazza, valamint könnyedén telepíthetjük az új verziókat.

A csomag a következőket tartalmazza: Apache, MySQL, PHP + PEAR, Perl, mod_php, mod_perl, mod_ssl, OpenSSL, phpMyAdmin, Webalizer, Mercury Mail Transport System (Win32) és NetWare Systems v3.32, Ming, JpGraph, FileZilla FTP Server, mcrypt, eAccelerator, SQLite, és WEB-DAV + mod_auth_mysql.

Amint láthatjuk, valóban rendkívül széles a rendelkezésünkre álló programok, modulok listája. Ezek közül ebben a projektben nem fogom a teljes skálát kihasználni, amik számomra fontosak: az Apache webszerver, a MySQL adatbázisszerver, a PHP nyelv és a phpMyAdmin az adatbázis kezeléséhez. A következőkben röviden ismertetem ezeket az eszközöket.

2.1.1. PHP 5.3.1

Szerver oldali nyelvnek a PHP 5.3.1-es verziót választottam. Azért erre a verzióra esett a választásom, mert ez a legújabb stabil, tesztelt release. Nézzük meg röviden mi is az a PHP! Eredetileg egy makró készlet volt, mára azonban jócskán túlnőtt eredeti jelentésén, képességei kibővültek és a nyelv népszerűsége ezzel együtt rendkívüli mértékben növekedett. Mára önálló nyelvként működik, hivatalos elnevezése a PHP: Hypertext Preprocessor. Tulajdonképpen egy kiszolgáló oldali parancsnyelvről van szó, melyet jellemzően HTML oldalakon használnak. A kiszolgáló a parancsokat nem küldi el az ügyfélnek, azokat a kiszolgáló oldalán a PHP-értelmező motor dolgozza fel. A PHP-t kifejezetten a világhálóra, a web programozóknak készítették, így szinte minden jellemző problémára megoldást nyújtó függvényeket tartalmaz. Úgy tervezték, hogy modulként futtatható legyen számos kiszolgálói alkalmazással, így itt nem jelentkeznek például a CGI parancsfájlok lassú indulásához hasonló gondok. Az

általam is használt 5-ös verzió számos új szolgáltatást hozott a PHP életébe, melyek közül a teljesség igénye nélkül megemlítek néhányat:

- Beépítették az XML támogatást.
- Tartalmazza az SQLite SQL könyvtárat.
- Támogatottá vált a privát, illetve védett tagfüggvények és tulajdonságok használata az osztályokban.
- Támogatja a statikus függvényeket és tulajdonságokat.

Miért a PHP-t választottam?

- A PHP segítségével *a fejlesztés gyors*, mégsem kell feláldoznunk a stabilitást. Az alkalmazások fejlesztésekkor lehetőség van elválasztani a tervezési, kódolási és összeállítási szakaszt, így hatékony és rugalmas alkalmazásokat fejleszthetünk.
- A PHP *nyílt forráskódú*, amely számos előnnyel szolgál. Azon túl, hogy teljesen ingyenes, ha problémába ütközünk könnyen kapcsolatba léphetünk a többi felhasználóval, ahol számos nagy tapasztalattal rendelkező embertől kaphatunk gyors segítséget. Továbbá a feldolgozóprogram hibáinak javítása felfedezésük után nem sokkal megtörténik, és a felmerült új igények kielégítő szolgáltatások is hamar beépülnek a nyelvbe.
- Fontos szempont a *hordozhatóság*! Alapvetően úgy tervezték, hogy alkalmas legyen számos operációs rendszeren való használatra, együttműködve különböző kiszolgálókkal és adatbázis kezelőkkel.
- A teljesítményt tekintve a hatékony Zend Engine-nek köszönhetően a PHP jól vizsgázik a más parancsnyelvekkel (ASP, Perl, Java Servlet) szemben végzett mérési tesztekben.

2.1.2. Apache 2.2.14

Az Apache egy nyílt forráskódú, robusztus, erőteljes és rugalmas *webszerver*, amely kompatibilis a HTTP/1.1 (RFC2616) protokollal. A webszerver dolga, hogy a hivatkozott URL-t lefordítsa egy fájlnévre, vagy programnévre, amit lefuttat, majd annak eredményét visszaküldje a kliensnek. További nagy előnye, hogy gyakorlatilag platform független, így olyan operációs rendszer alatt használhatjuk, amilyenen csak szeretnénk. Az Apache sok szabványt támogat, melyek nagy része modulok formájában áll rendelkezésre a mag kiegészí-

téseként. Ezek a modulok sok területet lefednek a kiszolgáló oldali programnyelv támogatástól kezdve a hitelesítési sémákig. Statikus és dinamikus weboldalak közzétételére egyaránt használják, de nemcsak weboldalak, hanem egyéb tartalom publikálására is alkalmas, például tetszőleges fájlok megosztására. Sok web alkalmazást az Apache által nyújtott környezethez és szolgáltatásokhoz terveznek. Számos nagy honlap is Apache webszerveren érhető el, például a Google keresőmotor felülete egy módosított Apache-on fut, melyet a Google Web Server (röviden GWS) névre kereszteltek.

A webszerverekkel szemben támasztott legfontosabb követelmények a teljesség igénye nélkül:

- Képes legyen multi-taskingra, azaz feladatok párhuzamos végrehajtására.
- Felhasználók validálás, amennyiben szükséges.
- Magas számítási kapacitás, a lehető legkisebb hardverigény mellett.
- Proxy szerverként is tudjon működni.
- Nagyon fontos a biztonság.

2.1.3. MySQL 5.1.41

A MySQL egy több felhasználós, többszálú, SQL-alapú relációs adatbázis-kezelő szerver. Ez az egyik legelterjedtebb adatbázis-kezelő, a programnyelvek igen széles skálájával használható. Itt is elmondható az a fontos tulajdonság, hogy rengeteg platformon futtatható, gyakorlatilag platform független. A MySQL adatbázisok adminisztrációjára széles körben elterjedt megoldás a PHP nyelven írt, nyílt forráskódú phpMyAdmin, amely szintén része a XAMPP csomagnak és én is ezt fogom használni.

2.1.4. phpMyAdmin 3.2.4

A phpMyAdmin egy nyílt forráskódú eszköz, melyet PHP-ban írtak a MySQL interneten keresztül történő menedzseléséhez. Jelenleg a következő műveletekre képes:

- adatbázis létrehozása és eldobás
- táblák létrehozása, módosítása, eldobása
- mezők hozzáadása, módosítása, törlése
- kulcsok kezelése
- SQL parancsok futtatása

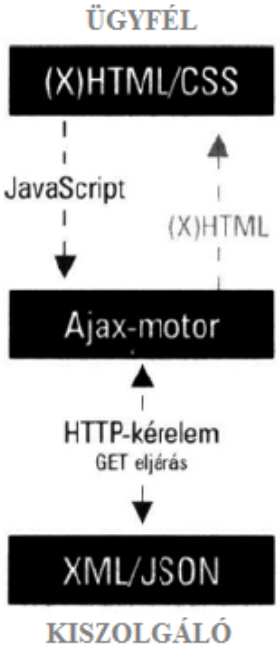
2.2. Ajax

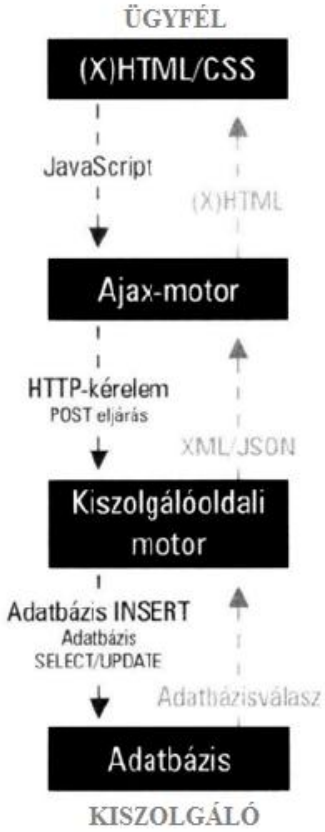
Az AJAX mozaikszó az Asynchronous JavaScript and XML kifejezésből ered. Annak ellenére, hogy csak az utóbbi időben vált igazán népszerűvé, maga a technológia már elég régóta létezik, a Microsoft 1999-ben bocsátotta ki az Internet Explorer 5-tel együtt. Ez az újonnan szerzett népszerűség elsősorban a böngészők által nyújtott támogatásnak köszönhető. Az Ajax központi eleme az XML DOM részét képező XMLHTTPRequest (XHR) objektum. Mivel az XML DOM az Ajax kiemelten fontos része, így mindenképpen szeretném röviden áttekinteni.

Az XML dokumentum-objektummodell az XML dokumentumok elérésének és módosításának szabványos módját határozza meg. Hozzáférést nyújt az XML, illetve az XHTML dokumentumok szerkezetét alkotó elemekhez, teljes elérést nyújtva a JavaScript számára. A hozzáférést a JavaScript DOM-kezeléssel foglalkozó belső objektumhalmaza teszi lehetővé. A modell elengedhetetlen az XHR kérelmek létrehozásakor a kiszolgáló oldalról érkező válaszok feldolgozásához. Ez az XHR az Ajax-modell veleje, ami nélkül a modell nem létezne, s amely lehetővé teszi, hogy a böngésző frissítése nélkül indítsunk HTTP-kérelmeket. Az XHR használatakor nem kell a kiszolgálóra várni, hogy minden kérelemre új oldallal válaszoljon, és lehetővé válik, hogy a felhasználók továbbra is kapcsolatban maradjanak az oldallal úgy, hogy a kérelmek küldése a háttérben folyik. Mindez kulcsfontosságú a felhasználói élmény megőrzéséhez.

Nem minden web alkalmazásban van szükség az Ajaxra, de egy adott alkalmazás számos eleme jobbá tehető az Ajax előnyeinek felhasználásával. Remekül alkalmazható akkor is, amikor kiszolgáló oldali kapcsolatot akarunk létrehozni és adatbázis műveleteket akarunk végezni a böngésző frissítése nélkül. Az Ajax éppen emiatt olyan hatékony, hiszen lehetővé teszi, hogy adatcserét folytassunk a kiszolgálóval, http-állapotkódokat fogadjunk, adatbázisba mentsünk, illetve meghatározzuk, hogy mi jelenjen meg a felhasználó számára, anélkül, hogy egyszer is frissíteni kellene az oldalt.

A háttérben történő adatkezeléstől eltekintve az XHR objektum GET és POST eljárása ugyanúgy működik, mint egy hagyományos HTTP-kérelemnél. A POST és a GET eljárás alkalmazásával egyaránt adatkérelmet küldhetünk a kiszolgálónak, amelyre valamilyen szabványos formátumban kapunk választ, melyek leggyakrabban XML, JSON és szöveges formátumban érkeznek. A következő két ábra segítségével megpróbálom szemléltetni, hogyan történik egy kérés küldése az XHR-nek.





1. ábra: Szabványos Ajax-modell folyamat *2. ábra:* Az adatbázis-megfelelő XHR POST

Munkám elkészítésekor sokat gondolkodtam azon, hogy melyik adatcsere-formátumot használjam, hiszen mind az XML-nek, mind a JSON-nak megvannak az előnyei és a hátrányai is. Végül az XML mellett döntöttem, mert ez szabványos formátum és minden nyelv támogatja. Továbbá fontos szerepet játszott az is, hogy a JSON nem kezeli tökéletesen az ékezetes karaktereket. Röviden tekintsük át ezt a két formátumot, hogy lássuk miről is van szó.

2.2.1. XML és JSON

Az Ajax válaszok különböző formátumokban érkezhetnek, melyek közül a leggyakoribb a JSON (JavaScript Object Notation, JavaScript objektumjelölés), a legszélesebb körben elfogadott pedig az XML (Extensible Markup Language, bővíthető jelölőnyelv). Az XML-t szívesen választják XHR-ként, mert minden programozási nyelv képes használni, továbbá mind kiszolgáló, mind ügyféloldalon támogatott. *„Az XML lényegében egy olyan egyedi címke alapú szerkezet, melyet a fejlesztő határoz meg.”* Az XML ügyféloldal és kiszolgálóoldal között történő átadásával a különböző nyelvek egyszerűen kommunikálhatnak egymással. Fontos megjegyezni, hogy az XML gyakorlatilag bármilyen adatszerkezetet képes megvalósítani.

A JSON felépítését tekintve úgy néz ki, hogy nevek, illetve címkék értékekhez rendelődnek egy tömbszerkezetben, vagy vesszőkkel tagolt listában. Nyelvtana rendkívül egyszerű és igazolhatóan könnyebben kezelhető, mint az XML-é, továbbá fontos előnye, hogy a JSON elemzést támogatja a JavaScript `eval` tagfüggvénye, ami nagyon megkönnyíti az elemzést. Ugyanakkor az `eval` használata veszélyeket is rejt magában, így jobban oda kell figyelni a biztonságra. A JSON fájlok szerkezete megfelel a JavaScript objektumokénak abból a szempontból, hogy egy fájl több objektumból, tömbből, karakterláncból, számból és logikai értékből állhat. Nagy mennyiségű adat kezelésénél az XML az ismétlődések miatt meglehetősen nehézkes, ilyenkor a JSON sokkal jobb választás lehet, mivel végeredményként kisebb méretű adatszerkezetet kapunk. Nem szabad azonban megfeledkeznünk arról, hogy a JSON ügyfél oldali feldolgozása lassabb, valamint, hogy a JSON nem szabványos adatcsere-formátum. Láthatjuk, hogy meglehetősen problémafüggő, melyik formátumot választjuk.

2.3. Ext JS 3.2

Az Ext JS egy rendkívül hatékony és erőteljes JavaScript keretrendszer, mely megvalósítja a fentebb tárgyalt Ajax technológiát. Egy viszonylag új keretrendszerről van szó, első verzióját 2007. április 1.-jén adták ki, valamivel több, mint három évvel ezelőtt. Azóta rengeteget fejlődött, s épp a közelmúltban (2010. április 7) adták ki a 3.2-es végleges verziót. A fejlesztést én még a 3.1.1-es verzióval kezdtem, mert akkor az volt az elérhető legfrissebb stabil release, majd áttértem a 3.2-es végleges verzióra. Tapasztalataim szerint Magyarországon még nem annyira széles körben elterjedt, mint például a JQuery, vagy hasonló könyvtárak, de világszerte rengeteg fejlesztő használja és hazánkban is egyre többen kezdik megismerni ezt a remek eszközt. Magyar nyelvű irodalmat egyáltalán nem találtam a témáról, így angol szakirodalmakra támaszkodva szeretném egy kicsit jobban bemutatni az Ext-et. A következőkben összefoglalom mi is tulajdonképpen az Ext, majd a dolgozat további részében az alkalmazás fejlesztésének leírásakor külön figyelmet fogok szentelni ennek a résznek, bemutatok néhány általam fontosnak tartott kódrészletet is.

Az Ext függvénykönyvtár eredetileg a YUI (Yahoo User Interface) kiegészítéseként jött létre, hogy biztosítsa azokat a dolgokat, amelyek a YUI-ban hiányosak, mint például egy könnyen kezelhető API és olyan widgetek, melyek a valós életben jól használhatóak. Nem is olyan régen történt, hogy fejlesztők egy csapata összefogott és az alap YUI kiegészítőből létrehozták az egyik legerőteljesebb kliens oldali alkalmazásfejlesztő könyvtárat. Az Ext egy

könnyen használható és rendkívül gazdag, már-már asztali alkalmazás szintű felhasználói felületet produkál, ami segíti a web fejlesztőket abban, hogy magára a funkcionalitásra koncentrálhassanak a technikai részletek helyett.

Az Ext nem csak újabb JavaScript függvénykönyvtár, hanem sokkal több annál. Képes együttműködni más JavaScript könyvtárakkal (Scriptaculous, Prototype, JQuery, YUI) úgynevezett adapterek használatával. Tipikusan olyan webes alkalmazások esetén előnyös a használata, amikor az adott alkalmazás és a felhasználó között magas szintű interaktivitás jellemző. Néhány fontos tulajdonsága, amely nagymértékben megkönnyíti fejlesztést:

- Egyszerűen használható, böngésző független és nagyon kifinomult widgeteket biztosít a fejlesztő számára, mint például ablakok, formok, gridek.
- Az EventManager segítségével kapcsolatban van a felhasználóval és a böngészővel egyaránt, reagálva a különböző eseményekre (billentyű leütés, egér kattintás stb).
- A szerverrel a háttérben kommunikál Ajax kérések segítségével, így nincs szükség az oldal újratöltésére.

Fontos, hogy mielőtt használni tudnánk az Ext könyvtárat, hivatkoznunk kell néhány alapvető fájlt. Ezt a HTML oldalunk HEAD részében tehetjük meg, a következőképpen:

```
<html>
    <head>
        <title>Fontos fájlok hivatkozása</title>
        <link rel="stylesheet" type="text/css"
              href="../extjs/resources/css/ext-all.css" />
        <script src="../extjs/adapter/ext/ext-base.js"></script>
        <script src="../extjs/ext-all-debug.js"></script>
    </head>
    <body> </body>
/html>
```

A fájlokat a fenti sorrendben kell hivatkoznunk. Elsőként az `ext-all.css`-t, ami a fő CSS fájl. Ez a fájl tartalmazza a widgetek kinézetét. Ezt a fájlt nem szabad szerkeszteni. Ezután hivatkozhatunk egy külső JavaScript könyvtárat, ha szükségünk van rá, majd az `ext-base.js`-t, ami az Ext magja. Ezt tudjuk megváltoztatni, ha egy másik könyvtárat szeretnénk használni. Ezt követi az elsődleges Ext fájl, az `ext-all.js`, vagy az `ext-all-debug.js`. Ezekben található az összes widget. Utóbbi a fejlesztéshez szükséges.

Most már használhatjuk az Ext által kínált lehetőségek végtelen tárházát. Az alkalmazás fejlesztése során használt elemeket az adott rész tárgyalásánál fogom bemutatni.

3. Az alkalmazás tervezése

Ebben a fejezetben az alkalmazás tervezését, annak fázisait ismertetem. Először is fel kell tárnunk a követelményeket, utána azok alapján összegzem a rendszer funkcióit, majd megtervezem az egyes csoportokhoz tartozó felhasználói felületeket, melyek megvalósításának technikai hátteréről az implementációs részben fogok bővebben foglalkozni. A tervezési fázis utolsó lépéseként elkészítem az alkalmazáshoz tartozó adatbázis struktúráját.

3.1. A követelmények feltárása

Ebben a fázisban történik meg a végfelhasználók által a kifejlesztendő alkalmazással szemben támasztott követelményeinek feltárása, majd azok elemzése. Ezek a követelmények beszélt nyelven vannak megfogalmazva, s legtöbbször ezt egészítjük ki táblázatokkal, diagramokkal. Az alkalmazás tervezésének kezdeti lépése, hogy találkozunk a megrendelővel és megbeszéljük, milyen elvárásai vannak a programmal kapcsolatban, milyen funkciók jelenjenek meg benne. Nagyon fontos a végfelhasználókkal, a megrendelővel folyamatosan tartani a kapcsolatot a fejlesztés teljes ideje alatt, mert a felhasználói követelmények gyakran változnak, melyről mielőbb értesülünk, annál hamarabb tudunk rá reagálni és annál több nem várt kellemetlenségtől kíméljük meg magunkat.

Jelen esetben a megrendelő egy kis szoftverfejlesztő cég tulajdonosa, aki a munka hatékonyságának növelése érdekében szeretne egy projektek menedzselésére alkalmas webes alkalmazást. Fontos elvárás, hogy az elkészülő alkalmazás könnyen kezelhető legyen, csak azokat a szükséges funkciókat tartalmazza, melyre a vállalkozásnak szüksége van, ugyanakkor könnyedén lehessen új funkciókkal bővíteni a rendszert a későbbiekben. A jelenleg szükséges funkciók a következők:

- A felhasználók beléptetése a rendszerbe, ellenőrzés és sikeres belépés esetén a jogosultságának megfelelő felület betöltése az adott alkalmazott számára elérhető funkciókkal.
- Jogosultságkezelés. Ezen belül lehessen jogcsoportokat létrehozni, törölni. A csoportokhoz jogokat és felhasználókat hozzáadni és törölni.
- Projektek kezelése. Aktuális projektek adatainak megtekintése, új projekt hozzáadása, meglévő módosítása. Alkalmazottak projektekhez rendelése. Minden projektet lehessen egy felületen, hatékonyan kezelni.

- Felhasználók kezelése. Alkalmazottak adatainak megtekintése, módosítása. Új alkalmazott hozzáadása, meglévő törlése. Fontos lenne olyan felületet készíteni, amelyen lehetősége van az arra jogosultaknak látni minden alkalmazottat, ugyanakkor könnyedén meg lehessen nézni egy, vagy több ember részletes adatlapját is.
- Cégen belüli levelezési lehetőség megteremtése a hatékonyabb munkavégzés érdekében, hiszen így könnyebben lehet megbeszéléseket, találkozókat szervezni, vagy egyszerűen csak kérdezni valamit.

A jogosultságkezelés teljesen dinamikusan fog történni, amelynek nagy előnye, hogy bármilyen cég a saját belső szabályozása alapján testre szabhatja azt, így minden igényt kielégít. A következőkben megfogalmazok egy ajánlást a jogosultságkezelésre, de még egyszer szeretném kihangsúlyozni, hogy ettől tetszés szerint el lehet térni.

Az általam javasoltak alapján négy felhasználói csoport különül el a rendszerben, melyek tagjai más-más jogosultságokkal rendelkeznek, ezáltal más és más felületet látnak, különböző funkciókat érhetnek el. Az említett csoportokba tartozó végfelhasználókat a szakma kulcsfiguráknak is nevezi. A négy csoport a következő:

- Adminisztrátor
- Vezetőségi tag
- Projekt vezető
- Fejlesztő

Vegyük sorra, hogy az egyes csoportokba tartozó felhasználóknak milyen elvárásai vannak az alkalmazással szemben. Az elvárások felsorolását a legkevesebb jogosultsággal rendelkező csoporttal kezdem és haladok a legszélesebb hatáskörű csoport felé, mert az általam javasolt lehetséges struktúrában a magasabb szintű csoportokban lévő felhasználók rendelkeznek a náluk alacsonyabb szinten lévő csoportok jogaival is.

Programozó követelményei:

- A rendszerbe való belépés után tudja megtekinteni a saját adatlapját, valamint tudjon módosítani azon bizonyos információkat, mint például lakcím, elérhetőségek, jelszó.
- Ki tudja listázni a futó projekteket, de azokon ne tudjon módosításokat végrehajtani.
- Tudjon üzenetet küldeni és fogadni.

Projekt vezető követelményei:

- Ki tudja listázni az összes alkalmazottat.

- Az általa felügyelt projektek bizonyos adatait tudja módosítani, például az egyes projektek állapotát tudja megváltoztatni.

Vezetőségi tag elvárásai:

- Új alkalmazott felvitele, valamint meglévő alkalmazott törlése.
- Projekt adatainak módosítása, új projekt felvitele a rendszerbe.
- Alkalmazottakat tudjon projektekhez rendelni.

Adminisztrátor elvárásai:

- Teljes jogosultságkezelés, tehát tudjon új jogosultsági csoportot létrehozni, a csoportokhoz jogokat és alkalmazottakat hozzárendelni.

3.2. Az alkalmazás funkciói

A készülő alkalmazás funkcióinak összegyűjtése, sok szempontból hasznos lépés. Nagy segítséget jelent a fejlesztő számára, így számomra is az implementáció során. Ezen kívül a megrendelővel folytatott folyamatos egyeztetések során is hasznos lehet, hiszen segítségével csökkenthetjük a félreértésekből adódó funkcionális hibákat.

Bejelentkezés:

Bejelentkezés során a felhasználó megadja a belépéshez szükséges nevét és jelszavát, majd a „Belépés” gomb megnyomásával bejelentkezik a rendszerbe, amennyiben az adatok helyesek és létezik ilyen felhasználó a nyilvántartásban. Belépés után minden felhasználó a jogosultságának megfelelő felületet látja, a számára elérhető funkciókkal.

Kilépés:

Egy bejelentkezett felhasználó kilép az alkalmazásból.

Alkalmazott hozzáadása:

A megfelelő jogokkal rendelkező felhasználó új alkalmazottat vihet fel a rendszerbe megadva annak minden szükséges adatát. A jogosulatlan felhasználók számára ez a lehetőség nem érhető el.

Alkalmazott törlése:

A megfelelő jogokkal rendelkező felhasználó törölhet egy meglévő felhasználót a rendszerből. A jogosulatlan felhasználók számára ez a lehetőség nem érhető el.

Alkalmazott adatlapjának megtekintése:

A felhasználók megtekinthetik saját, vagy akár mások adatlapját is.

Projekt hozzáadása:

A megfelelő jogokkal rendelkező felhasználó új projektet vihet fel a rendszerbe. A jogosulatlan felhasználók számára ez a lehetőség nem érhető el.

Projekt törlése:

A megfelelő jogokkal rendelkező felhasználó törölhet egy meglévő projektet a rendszerből. A jogosulatlan felhasználók számára ez a lehetőség nem érhető el.

Projekt adatainak megtekintése:

A kiválasztott projekttel kapcsolatos adatok megjelenítése.

Jogok megjelenítendő nevének módosítása:

Az adminisztrátor megváltoztathatja egy jog megjelenő nevét.

Jogcsoport hozzáadása:

Az adminisztrátor új jogosultsági csoportot hozhat létre annak nevének megadásával.

Jogcsoport törlése:

Az adminisztrátor törölhet egy meglévő jogosultsági csoportot.

Jogok és felhasználók csoportokhoz rendelése:

Az adminisztrátor jogokat és felhasználókat rendelhet hozzá a meglévő csoportokhoz.

Erőforrások projektekhez rendelése:

Az alkalmazottak és projektek összerendelése. Itt adható meg, hogy ki melyik projekten dolgozzon.

Új üzenet:

Minden felhasználó számára elérhető. Üzenet küldése a kiválasztott munkatársnak.

Bejövő üzenetek:

Minden felhasználó számára elérhető. Megjeleníti az adott felhasználónak érkezett üzeneteket.

Elküldött üzenetek:

Minden felhasználó számára elérhető. Megjeleníti az adott felhasználó által küldött üzeneteket.

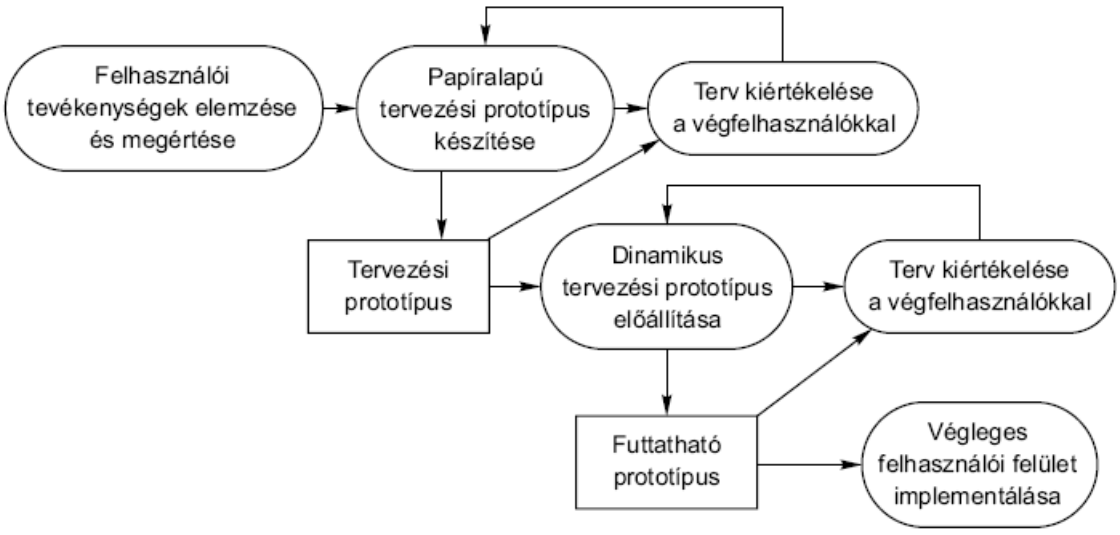
3.3. Felhasználói felületek

A felhasználói felület tervezése előtt meg kell vizsgálni minden egyes végfelhasználói csoportot és felmérni, hogy melyik csoport milyen feladatokat végez, hiszen az adott felhasználónak pontosan olyan felületet kell biztosítani, ami a munkájához szükséges. Ez a vizsgálat

megtörtént a követelmények feltárása során, így a felhasználói felület megtervezését az ott felmért igényekre támaszkodva hajtottam végre. Sokat segített a tervezésben továbbá a rendszer funkcióit összesítő dokumentáció. Nagyon fontos, hogy a felületet a felhasználóval közösen tervezzük, hiszen ő tudja leginkább mire van szüksége és együtt rövid idő alatt el tudunk jutni egy olyan felülethez, amely általa jól használható, és ezáltal esetlegesen tovább pontosíthatjuk az egyes funkciókat is. A felület jóságának mérése metrikák segítségével nehézkes, ezért inkább minőségi jellemzők vannak. Néhány fontos szempont, amelyre kiemelt figyelmet kell fordítani:

- ***Tanulhatóság***. Ez alatt azt az időt értjük, amennyi ahhoz szükséges, hogy a felhasználó olyan szinten megismerje az alkalmazás működését, hogy a napi munkájában fennakadások nélkül tudja használni azt, ezzel profitot termelve a cég számára.
- ***Sebesség***: Itt a felhasználói interakciók és az üzenetek sebességéről beszélünk. Ez nem feltétlenül a felhasználói felületen múlik, sokkal inkább a felület által előidézett funkció sebességén.
- ***Robosztusság***: Mennyire hibatűrő a felhasználói felület.

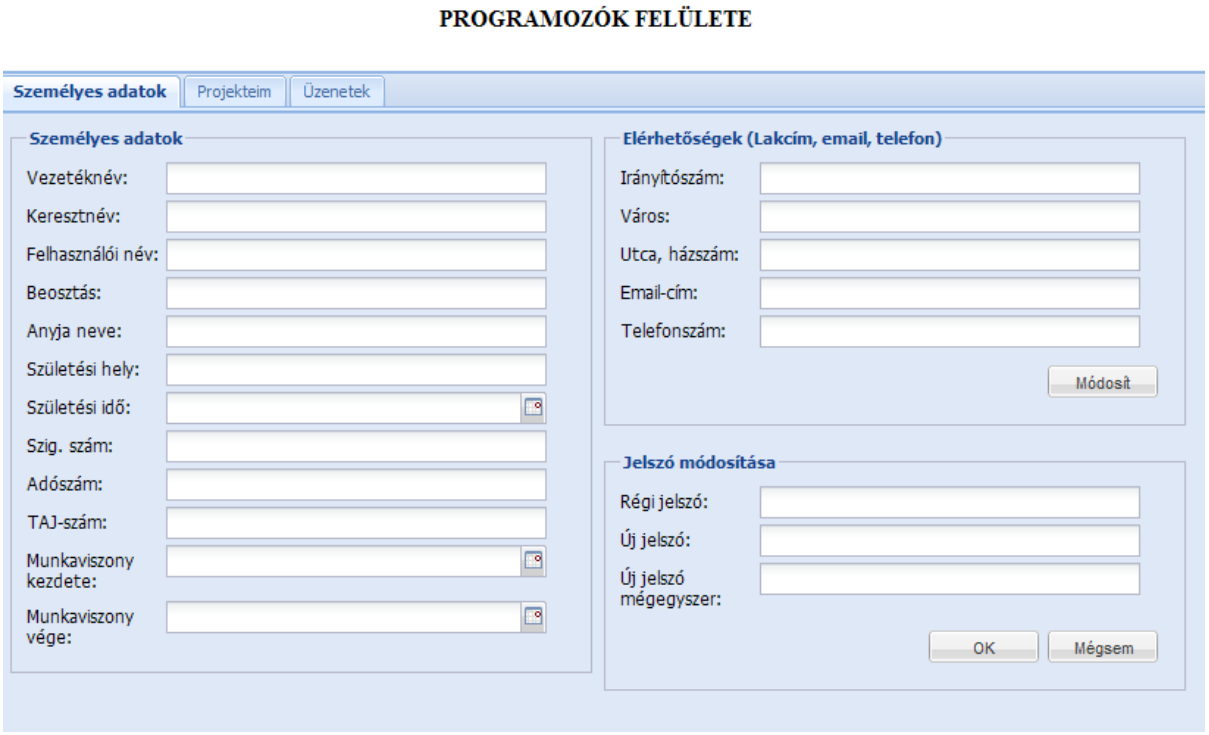
A következő ábra egy felhasználói felület tervezésének menetét szemlélteti.



3. ábra: Felhasználói felület tervezésének lépései

A tervezés során nagy hangsúlyt fektettem arra, hogy a felhasználói felület könnyen kezelhető legyen, minden funkció néhány kattintással elérhetővé váljon. A kinézetnél ügyeltem a színválasztásra, igyekeztem kevés színnel dolgozni és olyat választani, amely a szem számára nem megterhelő több órás használat során sem. A felület első vázlatos terveit papíron készí-

tettem el, majd beleképzeltem magam az adott felhasználó helyébe, hogy megvizsgáljam mennyire jól kezelhető számára az adott elrendezés, miken kell változtatni, míg végül több prototípus elkészítése után kirajzolódott a felület jelenlegi képe. A következő ábrán a programozók felületének tervét láthatjuk.



4. ábra: A programozók felhasználói felületének terve

3.4. Adatbázis tervezése

A tervezés során a feltárt követelményeket szem előtt tartva átgondoltam milyen táblákra van szükség, és az egyes táblákban milyen tulajdonságokat kell tárolnunk.

Felhasználók tábla: Ebben tartjuk nyílván a vállalkozásban dolgozók számunkra fontos adatait, melyek a következők.

- egyedi azonosító
- vezetéknév, keresztnév
- felhasználói név
- jelszó
- beosztás
- lakcím (irányítószám, város, utca, házszám)

- Anyja neve
- születési hely és idő
- személyi igazolvány szám
- adószám
- TAJ-szám
- telefonszám
- email cím
- munkaviszony kezdete
- munkaviszony vége

Projektek tábla: A projektekkel kapcsolatos információk nyilvántartása.

- egyedi azonosító
- név
- típus
- prioritás
- leírás
- a projekt kezdete
- határidő
- a projekt befejezése

Csoportok tábla: A jogosultsági csoportok adatait tartalmazó tábla.

- egyedi azonosító
- csoport neve

Jogok tábla: Az adható jogokról tartalmaz információt.

- egyedi azonosító
- név, amellyel az alkalmazás dolgozik
- megjelenő név, amely a felületen látható a jogok kezelésénél

Helyszínek tábla: A projektekhez tartozó helyszíneket tároló tábla. Segítségével megadhatjuk egy projekthez, hogy az helyileg hol zajlik.

- egyedi azonosító
- helyszín neve

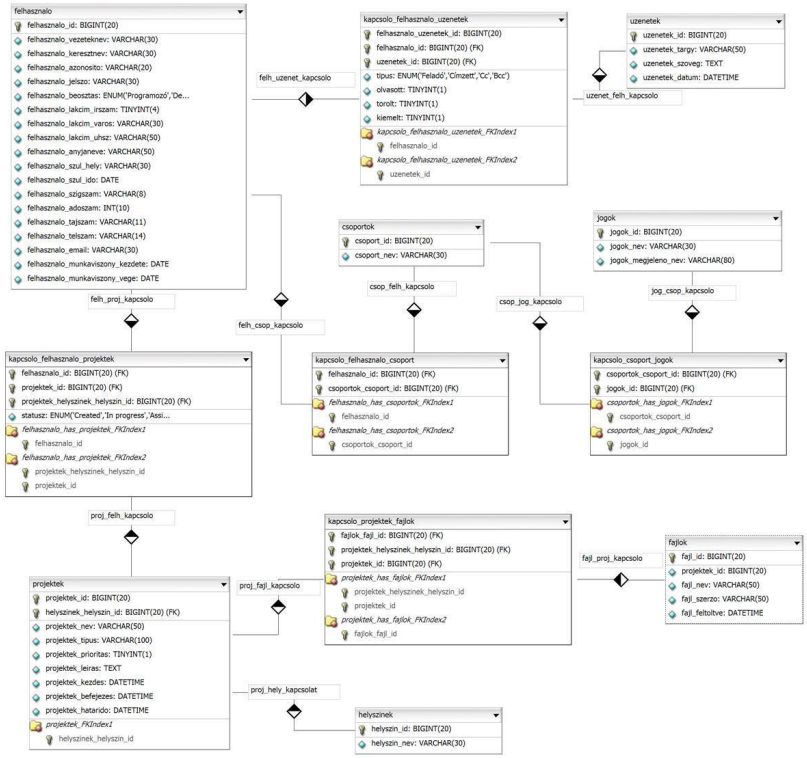
Fájlok tábla: A követelmények között nem szerepelt ugyan a fájl feltöltési lehetőség, de véleményem szerint az egy hasznos funkció, ezért az adatbázist már felkészítem rá, hogy az alkalmazás ilyen irányú fejlesztése gördülékeny végrehajtható legyen.

- egyedi azonosító
- név
- a fájlt feltöltő felhasználó egyedi azonosítója
- a projekt egyedi azonosítója, amelyhez az adott fájl feltöltésre került
- a feltöltés ideje

Üzenetek tábla: Az üzenetek kezeléséhez szükséges információkat tartalmazó tábla.

- egyedi azonosító
- tárgy
- üzenet szövege
- küldés ideje

Az egymással összetartozó táblák közötti kapcsolatot úgynevezett kapcsoló táblák segítségével hoztam létre. Szükséges volt a felhasználók összekapcsolása a projektekkel, a csoportokkal és az üzenetekkel, a projektek összekapcsolása a fájlokkal és a helyszínekkel, valamint a jogosultságkezeléshez a csoportok és jogok tábláinak összekapcsolása. Az elkészült terv alapján a DBDesigner 4 adatbázis tervező szoftver segítségével elkészítettem az adatbázis sémáját, melynek lekicsinyített változatát az 5. ábrán láthatjuk. A teljes méretű kép megtalálható a függelékben.



5. ábra: Az adatbázis sémája

4. Implementáció

Az implementációt több lépcsőben végeztem. Először is létrehoztam az adatbázist a tervek alapján, majd elkészítettem a bejelentkező képernyőt. Ezután elkezdtem kialakítani a felületet az Ext JS segítségével és mindent, amit csak tudtam megvalósítottam az ügyfél oldalon, hogy minél jobban tehermentesítsem a kiszolgálót. Ebben a részben elsősorban az Ext nyújtotta lehetőségeket szeretném bemutatni, így a felhasználói felület kialakítására és a kliens oldalon végrehajtott eseményekre helyezem a hangsúlyt. Természetesen az érdekesebb részeknél a szerver oldali kezelést is ismertetni fogom. Az implementáció leírását az adatbázis létrehozásával kezdem, majd a felhasználói felület jól elkülöníthető komponensei alapján fogom bemutatni.

4.1. Az adatbázis létrehozása

Az adatbázis megtervezése után az implementáció első lépéseként MySQL-ben létrehoztam az adatbázis tábláit. Mivel a tervezés során meglehetősen részletesen bemutattam az adatbázis felépítését, ezért itt csak egy tábla létrehozásának kódját mutatom be. A felhasználókat kezelő tábla létrehozását a következő parancs végzi, mely véleményem szerint nem szorul különösebb magyarázatra.

```
CREATE TABLE IF NOT EXISTS 'felhasznalo' (
      'felhasznalo_id' bigint(20) NOT NULL AUTO_INCREMENT,
      'felhasznalo_vezeteknev' varchar(30) NOT NULL,
      'felhasznalo_keresztnev' varchar(30) NOT NULL,
      'felhasznalo_azonosito' varchar(30) NOT NULL,
      'felhasznalo_jelszo' varchar(30) NOT NULL,
      'felhasznalo_beosztas' enum('Adminisztrator','Vezetosegi
                  tag','Projekt vezeto','Dolgozo') NOT NULL,
      'felhasznalo_irszam' tinyint(4) NOT NULL,
      'felhasznalo_varos' varchar(30) NOT NULL,
      'felhasznalo_utcahsz' varchar(50) NOT NULL,
      'felhasznalo_anyjaneve' varchar(50) NOT NULL,
      'felhasznalo_szulhely' varchar(30) NOT NULL,
      'felhasznalo_szulido' date NOT NULL,
      'felhasznalo_szigszam' varchar(8) NOT NULL,
      'felhasznalo_adoszam' int(10) NOT NULL,
      'felhasznalo_tajszam' varchar(11) NOT NULL,
      'felhasznalo_telszam' varchar(30) NOT NULL,
      'felhasznalo_email' varchar(30) NOT NULL,
      'felhasznalo_munkaviszonykezdete' date NOT NULL,
      'felhasznalo_munkaviszonyvege' date NOT NULL,
      PRIMARY KEY ('felhasznalo_id'));
```

4.2. Bejelentkezés

Az alkalmazás által nyújtott funkciók használatához a felhasználónak először be kell jelentkeznie a rendszerbe, melyet a bejelentkező képernyőn tehet meg. Minden alkalmazott kap egy felhasználói nevet és egy jelszót, amely segítségével belépéskor a rendszer azonosítani tudja őket. Belépés után az alapértelmezésben megkapott jelszó a felhasználó adatlapján megváltoztatható, sőt kifejezetten ajánlott annak cseréje. Amint azt már említettem, a felhasználói felületen mindent az Ext JS segítségével hoztam létre, így a bejelentkező képernyőt is, mely esetemben nem más, mint egy ablakba helyezett form panel. Az Ext rengeteg úgynevezett widgetet biztosít számunkra, melyek rendkívül finoman hangolhatók úgynevezett *konfigurációs objektumok* segítségével, melyekről később részletesebben is beszélni fogok. Mindenekelőtt az `Ext.onReady()` függvényről kell beszélnünk egy kicsit, melynek az a feladata, hogy a paraméterként megkapott függvény végrehajtását várakoztassa addig, amíg minden objektum be nem töltődik. Így nem fogunk olyan problémába ütközni, hogy mivel a JavaScriptek már az oldalunk fej részében betöltődnek, azok esetlegesen olyan objektumra hivatkoznak, amely még nem is létezik, mert még nem töltődött be. Az én alkalmazásomban szinte kizárólag anonim függvényt kap meg, és abban kap helyet minden olyan elem definíciója, amely az adott oldalon megjelenik, de természetesen kaphat paraméterként csak egyetlen függvényhívást is. A bejelentkező oldalam szerkezetileg tehát a következőképpen épül fel:

```
Ext.onReady(function(){
     Ext.BLANK_IMAGE_URL = '/images/s.gif';
     Ext.QuickTips.init();
          //az alapértelmezett felugró üzeneteket inicializálja

     var loginForm = new Ext.form.FormPanel({
          //a formpanel definiálása
     });

     var loginWindow = new Ext.Window({
          //az ablak megadása
     });
     loginWindow.show();

});  //az Ext.onReady() vége
```

Az `Ext.onReady()` függvényt mindig használnunk kell. Most pedig lássuk, hogyan épül fel a bejelentkezést végző form panel.

```
var loginForm = new Ext.form.FormPanel({
     url: 'check_login.php',
     bodyStyle: 'padding:15px;background:transparent',
     buttonAlign: 'center',
     border: false,
     labelWidth:100,
     items: [{
          xtype: 'textfield',
          id: 'username',
          fieldLabel: 'Felhasználói név',
          allowBlank: false,
          blankText: 'A mező kitöltése kötelező! Kérem adja meg a
                     felhasználói nevét!',
          minLength: 6,
          minLengthText: 'A felhasználói név minimum 6 karakter
                     hosszú kell legyen!',
          maxLength: 32,
          maxLengthText: 'A felhasználói név maximum 32 karakter
                     hosszú lehet!',
          msgTarget: 'side',
          validationEvent: false
     },{
          xtype: 'textfield',
          id: 'password',
          fieldLabel: 'Jelszó',
          inputType: 'password',
          allowBlank: false,
          // ellenőrző kód van, mint a felhasználói névnél
     }],…
```

Először is létrehozunk egy új FormPanel példányt, majd a már említett konfigurációs objektumok segítségével beállítjuk a szükséges paramétereket. Az `url` után megadhatjuk a form feldolgozását végző fájl nevét, amely esetünkben a check_login.php. A `bodyStyle` segítségével tudjuk szabályozni a megjelenítést, a `buttonAlign` pedig azt mondja meg, hogy hová legyenek igazítva a formon megjelenő gombok. A `border` logikai érték segítségével tudjuk szabályozni, hogy legyen-e keret az adott objektum körül, vagy sem. A `labelWidth` pedig a

címkék szélességét határozza meg képpontokban. Ezek a beállítások érvényesek általánosságban a form panelünkre. Természetesen nem csak ezek a tulajdonságok állíthatóak be, a teljes lista megtalálható az Ext JS API dokumentációjában.

Az `items` egy tömb, mely a form panelen található elemeket tartalmazza, jelen esetben két szövegmezőt és egy gombot. A szövegmezők definiálását az `xtype: 'textfield'` sor végezi el, de szeretném megjegyezni, hogy itt példányosítással is létrehozhattuk volna a szövegmezőnket. A hatékonysága miatt választottam ezt a megoldást, de ahhoz, hogy megértsük, vessünk egy pillantást arra, **mi is az `xtype`!?** Az `xtype` egy úgynevezett komponens konténer elem, amely az Ext JS könyvtárhoz kapcsolódik, és segítségével gyorsan tudunk objektumokat definiálni. Az ilyen módon definiált elemek nagy előnye, hogy csak akkor töltődnek be a böngésző memóriájába, amikor ténylegesen használjuk azokat, így azok az objektumok, amelyek nem jelennek meg azonnal a képernyőn, nem foglalják le az értékes memóriát. Ez pedig jelentős mértékben növelheti a rendszerünk átlagos teljesítményét, főleg ha egy nézet sok elemet tartalmaz. Így amikor csak tudtam az objektumokat `xtype` segítségével definiáltam.

Amint láthatjuk ezen elemek is konfigurálhatók, adhatunk nekik saját azonosítót, címkét és számos más tulajdonságot állíthatunk be. Az `allowBlank` segítségével például könnyedén ellenőrizhetjük, hogy az adott mező ki van-e töltve, ami egy nagyon hasznos funkció. Ha ezt az értéket false-ra állítjuk, az azt jelenti, hogy a mező kitöltése kötelező, melynek elmulasztása esetén a rendszer figyelmezteti a felhasználót olyan szöveggel és akkor, amikor csak szeretnénk. Alapértelmezésben azonnal jelzi az Ext, hogy üresen maradt egy olyan mező, amelynek kitöltése kötelező, azonban ez is beállítható a `validationEvent` hamisra állításával. Ebben az esetben a rendszer csak akkor jelzi a hiányosságokat, amikor a gomb megnyomásával elküldjük a felvitt adatokat. Jelen esetben további korlátozásokat is beállítottam a felhasználói név és a jelszó minimális és maximális hosszára, így ha túl rövid, vagy túl hosszú karaktersorozatot gépelünk be, akkor a „Bejelentkezés” gomb megnyomása után erről is informálja a rendszer a felhasználót.

```
buttons: [{
     text: 'Bejelentkezés',
     handler: function(){
          if(loginForm.getForm().isValid()){
               loginForm.getForm().submit({
                    url: 'check_login.php',
```

```
                    waitMsg: 'Kérem várjon...',
                    success: function(loginForm, resp){
                         Ext.Msg.alert('Success', 'Welcome "'+
                                   resp.result.message  +'"  on
                                   the Project manager v1.0');
                    },
                    failure: function(loginForm, resp){
                         Ext.Msg.alert('WARNING',
                                   resp.result.errormsg);
                    }
               });
          }…
```

Gombokat a `buttons` részben tudunk definiálni, ahol az `itmes`-hez hasonlóan egy tömböt kell megadnunk, a tömb elemeit pedig {} párok között kell elhelyezni és több tömbelem esetén azokat egymástól vesszővel kell elválasztani. Nekünk most csak egyetlen gombunk van „Bejelentkezés” felirattal, amelyet a `text` opcióval állítottam be. Ezután található az eseménykezelő, `handler`. Itt található az a kódrészlet, amely lefut a gomb megnyomásakor. Jelen esetben ez egy anonim függvény, amely megnézi, hogy a kitöltött form helyes-e, és ha igen, akkor a `submit` függvény meghívásával elküldi azt. A `submit` függvényen belül mindenképpen reagálnunk két eseményre: sikeres- és sikertelen küldés. Ha a küldés sikeres, akkor végrehajtódik a `success` után megadott rész, sikertelen küldés esetén pedig a `failure` utáni anonim függvény. Ez a form egy ablakban jelenik meg a bejelentkező képernyőn, amelyet a következő néhány soros kód készít el:

```
var loginWindow = new Ext.Window({
     title: 'Üdvözli Önt a Project Manager v1.0',
     layout: 'fit',      //az elrendezés stílusa
     height: 150,        //ablak magasságának megadása
     width: 300,         //ablak szélességének megadása
     closable: false,    //nem zárható be
     resizable: false,   //nem méretezhető
     draggable: false,   //nem mozgatható
     items: [loginForm] //egyetlen eleme a létrehozott form
});
loginWindow.show();      //az ablak megjelenítése
```

Ezzel készen is van a bejelentkező képernyőnk. Hiba esetén a „Bejelentkezés” gombra kattintva a rendszer figyelmezteti a felhasználót, hogy mit rontott el.

6. ábra: Hiba a bejelentkezésnél

Végül, mielőtt továbblépünk, térjünk vissza néhány szó erejéig a fejezet elején említett **konfigurációs objektumra** és vizsgáljuk meg mi is az!? Ez az elsődleges módja annak, hogy az Ext-tel megcsináltassuk mindazt, amit szeretnénk, ez biztosítja a használt függvény különböző opcióinak beállítási lehetőségét. Régen a függvényeket csak előre jól meghatározott argumentumokkal hívhattuk meg, melyeknek kötött volt a sorrendje, így emlékeznünk kellett arra. További probléma ezzel a megoldással, hogy az opcionális argumentumokkal szemben nagyon csekély rugalmasságot mutat. A konfigurációs objektumok használatával sokkal magasabb szintű flexibilitást kapunk, az argumentumok sorrendje többé nem számít. A módszer lehetővé teszi, hogy meghatározatlan számú argumentumot használjunk, olyan keveset, vagy éppen olyan sokat, amennyire az adott esetben szükségünk van. Egy másik fontos előny, hogy a függvény alapvető használhatósága nem fog sérülni, amikor egy későbbi ponton hozzáadunk, vagy éppen elhagyunk néhány argumentumot. Felépítését tekintve a konfigurációs objektum nagyon hasonlít a CSS-re, vagy a JSON-ra. Tulajdonképpen ez is az adat strukturálásának egy lehetséges módja, hogy az könnyebben olvasható legyen a programozási nyelvek, esetünkben a JavaScript számára. Végezetül néhány kulcsfontosságú dolog, melyet a konfigurációs objektumról tudnunk kell, hogy dolgozhassunk vele:

- A rekordok egy kapcsos zárójelpáron belül helyezkednek el, ami szimbolizálja, hogy ezek a rekordok egy objektum részét képezik.
- Minden rekord egy név/érték párból áll, melyeket egymástól kettőspont választ el. Több ilyen pár esetén azokat vesszővel választjuk el egymástól.

```
{
        név1: érték1,
        név2: érték2
```

```
}
```

- Az érték rész bármilyen típusú adatot tartalmazhat, beleértve a logikai típust, tömböt, függvényt is, de lehet akár egy másik objektum.

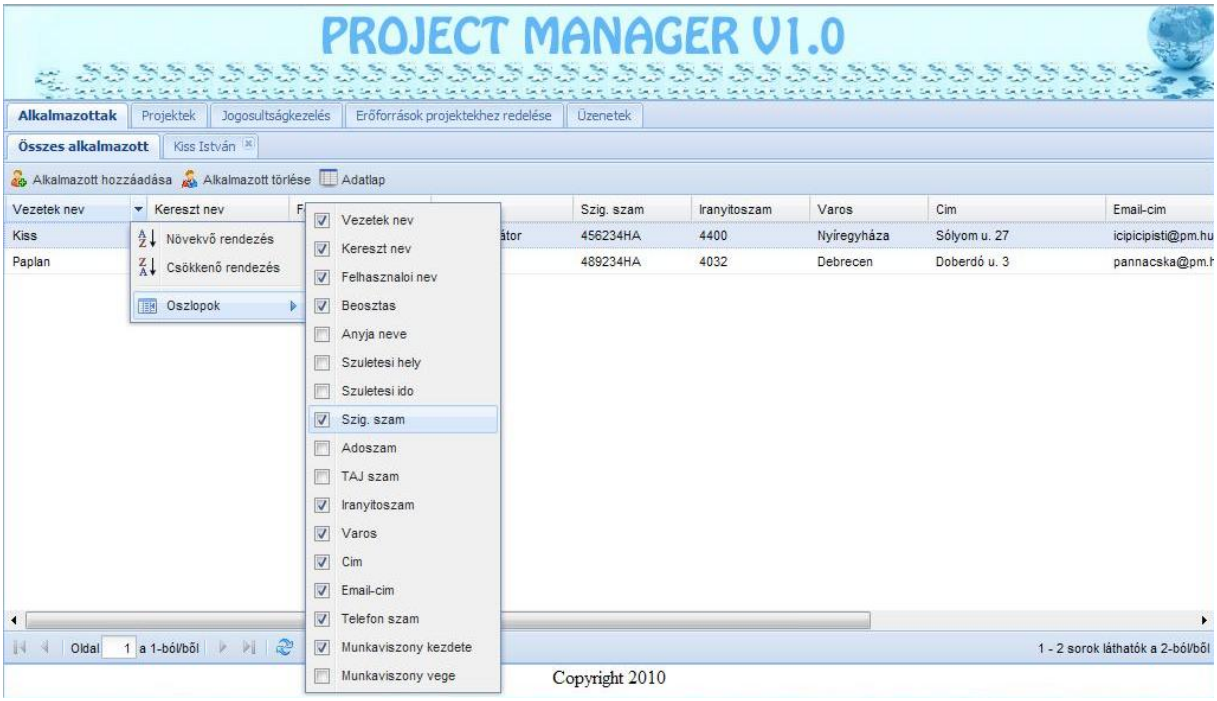
```
{
        név1: true,
        név2: {
                név3: érték3,
                név4: false
        }
}
```

- A tömböt szögletes zárójel definiálja, amely szintén tartalmazhat mindenféle objektumot.

```
{
        név: ['egy', 'kettő', 'három']
}
```

4.3. Alkalmazottak kezelése

Ennél a résznél is fontos szempont volt a könnyű kezelhetőség megvalósítása. A megrendelőnek az volt a kérése a felülettel kapcsolatban, hogy azon látható legyen az összes alkalmazott, és azok minden adata, de legyen lehetősége valahogyan kiválasztani bizonyos embereket, akiknek az adatait külön is megtekintheti. Ezt a komponenst ezért úgy készítettem el, hogy az „Alkalmazottak” fül alatt létrehoztam egy dinamikusan bővíthető úgynevezett tab panelt. A panel első fülén egy szerkeszthető gridben megjelenik az összes alkalmazott és nekik minden tulajdonsága. A tulajdonságokat az egyes oszlopok reprezentálják, az alkalmazottakat pedig az egyes rekordok. Az Ext-nek köszönhetően a gridnek vannak olyan rendkívül hasznos, beépített funkciói, melyek segítségével rendezhetjük a gridünket egy adott oszlop, tulajdonság alapján, vagy a számunkra éppen szükségtelen oszlopok megjelenítését kikapcsolhatjuk a könnyebb áttekinthetőség érdekében, amint azt a x. ábrán láthatjuk. A grid továbbá lapozható, megadhatjuk hány rekordot szeretnénk egy lapon megjeleníteni és miután elértük ezt a limitet, a következő rekord már az új lapon fog szerepelni. A gridben szereplő minden rekord szerkeszthető, mindössze annyit kell tennünk, hogy kétszer rákattintunk. Ha kiválasztunk egy alkalmazottat és az „Adatlap” feliratra kattintunk, akkor egy új, zárható fülön megjelennek az illető adatai. Nagyon egyszerűen tudunk alkalmazottat hozzáadni, illetve törölni is.



7. ábra: Grid oszlopain végezhető műveletek

Nézzük meg, hogy lehet a leírtakat megvalósítani az Ext JS segítségével. A diplomamunka keretein belül sajnos nincs lehetőségem a kódot részleteiben ismertetni, így csak az alapvető technikai megoldásokat szeretném bemutatni. Kezdjük a tab panel létrehozásával, amely rendkívül egyszerű, csupán a következő néhány sorra van szükségünk.

```
        title: 'Alkalmazottak',    //a fülön megjelenő felirat
        xtype: 'tabpanel',         //az objektum típusának megadása
        id: 'alktabID',            //azonosító hozzárendelése
        activeTab: 0,    //melyik fül legyen az aktív alapértelmezésben
        items: [{        //a tabpanel elemei, az egyes fülek megadása
              //editorgrid
        }
  }…
```

Ezáltal létrehoztuk a tab panelünket, most már hozzáadhatjuk az egyes füleket. Esetünkben most kezdetben egyetlen fül lesz, amely az összes alkalmazottat megjelenítő gridet fogja tartalmazni. A tab panel `items` részébe a következő kód kerül:

```
items: {
     title: 'Összes alkalmazott', //a fülön megjelenő felirat
     xtype: 'alk_grid',           //az objektum típusának megadása
     id: 'usergridID'             //azonosító hozzárendelése
}
```

Ebben a részben vizsgáljuk meg kicsit az `xtype: 'alk_grid'` sort. Tulajdonképpen ez a rész hozza létre az alkalmazottakat tartalmazó gridet. Annak érdekében, hogy a kód jobban kezelhető legyen, kiterjesztettem az EditorGridPanel osztályt és beregisztráltam egy új xtypeként. A kiterjesztést végző kódot, valamint az alkalmazottak kezelésével kapcsolatos dolgokat egy külön fájlba (`alkalmazottak.php`) rendeztem össze. Az adatmodelleket a `data_models.php`, az adattárakat pedig a `data_stores.php` fájl tartalmazza az áttekinthetőség és karbantarthatóság miatt. Az említett osztály kiterjesztését a következő kód végzi:

```
Felhasznalok.FelhasznaloGrid = Ext.extend(Ext.grid.EditorGridPanel, {
    initComponent:function() {
        Ext.apply(this, {
            store: Ext.StoreMgr.lookup('store_alk_id'),
            columns: getAlkColumnModel(),
            tbar: [{…}]…
```

A kiterjesztés során beállítjuk a kezdőértékeket. A `store` beállításnál adjuk meg az adattárat, amely az adatokat szolgáltatja a gridünk számára, a `columns` után pedig meghatározom, milyen oszlopmodellt használjon a kiterjesztésem. Jelen esetben azt az adattárat fogjuk használni, amelynek az azonosítója a `store_alk_id`, és ez lesz a kiterjesztett osztály tárolója. Mielőtt azonban ezt használni tudnánk, először is létre kell hoznunk, majd be kell regisztrálnunk a Store manager számára. A tároló létrehozását a `util.php` fájlban található `getXmlStore(id, url, fields)` függvény végzi, amely a következőképpen épül fel:

```
function getXmlStore(p_storeId, p_url, p_fieldsTomb) {
    return new Ext.data.Store({
        storeId: p_storeId,
        proxy: getPostProxy(p_url),
        reader: getXmlReader(p_fieldsTomb),
        autoLoad: v_autoLoad
    });}
```

A paraméterként megkapott adatok alapján létrehoz egy új store példányt, beállítja az azonosítóját, a kiszolgálót, amely az adatokat szolgáltatja, valamint létrehoz egy `XmlReadert` a harmadik paraméterként megkapott modellhez és visszatér ezzel a példánnyal. Az említett függvényt a `data_stores.php` fájlban lévő kód hívja meg az ott megadott paraméterekkel, majd a

létrehozott tárolót beregisztrálja a tároló menedzser számára, így válik számunkra használhatóvá.

```
Ext.StoreMgr.register(
    getXmlStore('store_alk_id', 'alkalmazottak.php', alk_model) );
```

Az oszlopok definícióját az `alkalmazottak.php` fájlban implementált `getAlkColumnModel()` függvény adja vissza. Mivel ez a grid editálható, így minden egyes tulajdonság, oszlop esetén, amelyet szerkeszthetővé szeretnénk tenni, létre kell hoznunk egy megfelelő form mezőt. Ezeknek a mezőknek a létrehozását a `util.php` fájlban található megfelelő függvények végzik, melyek a következők:

- `function getGridEditorTextField(p_maxLength, p_vtype)`
  Az egyes szöveges mezők szerkesztését teszi lehetővé. Paraméterként megadhatjuk a mezőbe írható szöveg maximális hosszát, illetve megadhatunk egy validátort, amely ellenőrzi, hogy a bevitt adat formailag helyes-e. Megadtam két tulajdonságot, melyek közül az egyik kötelezővé teszi a mező kitöltését (`allowBlank`), a másik pedig automatikusan kijelöli a szöveget, ha kétszer rákattintunk (`selectOnFocus`), így az ott lévő tartalom törlésével már nem kell foglalkoznunk, csak be kell gépelnünk, amit szeretnénk.
- `function getGridEditorDateField()`
  A dátumot tartalmazó mezők szerkesztését teszi lehetővé.
- `function getGridEditorNumberField(p_maxLength, p_vtype)`
  A számokat tartalmazó mezőket tudjuk a segítségével editálni. A szöveges mezők szerkesztéséhez hasonlóan itt is megadható maximális méret és ellenőrizhetjük a bevitt adatok szintaktikai helyességét. Itt is beállítottam az említett két tulajdonságot.

Mi az a vtype? Az Ext lehetőséget biztosít a felületen az egyes mezőkbe bevitt adatok szintaktikai helyességének ellenőrzésére. Az úgynevezett vtype-ok segítségével ellenőrizhetjük a felhasználótól érkező bemenetet és hiba esetén tájékoztathatjuk arról a felhasználót. Ezek gyakorlatilag bármilyen bemenet ellenőrzését lehetővé teszik, mivel reguláris kifejezések segítségével működnek. Az Extnek vannak beépített ilyen típusai, mint például email, url, alpha, alphanum stb. Ezeken kívül természetesen lehetőségünk van saját típus definiálására is, amelyhez remek alapot nyújthatnak a beépítettek. Minden vtype definíció rendelkezik névvel, maszkkal, hibaüzenettel és egy függvénnyel a teszteléshez. Esetemben a saját vtype-ok definícióját a `util.php` fájl tartalmazza. Tekintsük meg például a telefonszám formátumának ellenőrzését végző vtype definícióját.

```
p_VTypesArr['phoneVal'] = /^[0-9]{2}[\/][0-9]{2}[-][0-9]{3}[-][0-9]{3,4}$/;
p_VTypesArr['phoneMask'] = /[0-9-\/]/;
p_VTypesArr['phoneText'] = 'Nem megfelelő telefonszám, pl: 06/70-555-1234';
p_VTypesArr['phone']= function(v){
            return Ext.form.VTypes['phoneVal'].test(v);}
```

Az első sor tartalmazza azt a reguláris kifejezést, amely ellenőrzi a bemenetet. A maszknál adjuk meg azokat a karaktereket, amelyeket engedünk begépelni az adott mezőbe, majd megadunk egy hibaüzenetet, amelyet a program feldob, amikor hibát észlel. Végül pedig megadjuk azt a függvényt, amely teszteli a bemenetet. Szeretném megjegyezni, hogy az Ext JS oldalán található fórumban rengeteg megírt vtype-ot lehet találni, melyeket érdemes átnézni és már egy meglévőt az igényeinkre alakítani.

A gridhez létrehoztam egy felső eszköztárat, melyen elhelyeztem az „Alkalmazott hozzáadása”, „Alkalmazott törlése” és „Adatlap” elemeket. Minden esetben, amikor módosítjuk a grid valamelyik rekordját, a változtatások automatikusan bekerülnek az adatbázisba. Vessünk most egy pillantást ezek működésére.

```
tbar: [{
     text: 'Alkalmazott hozzáadása',
     icon: 'images/add16.gif',
     cls: 'x-btn-text-icon',
     handler: function() {
       new Ext.Window({
             id:'alk_form_window',              //azonosító
             title:'Új alkalmazott felvitele',  //ablak címsora
             width:700,height:450,   //szélesség, magasság
             x:320,y:82,             //pozíció megadása
             layout:'fit',           //elrendezés
             closable:true,          //bezárható
             border:false,           //nincsen keret körülötte
             maximizable:true,       //teljes képernyőssé tehető
             items:{xtype: 'alk_form', id: 'alk_form_new'}
                   //saját xtype használata
       }).show();  //megjeleníti az ablakot
  }
```

Az „Alkalmazottak hozzáadása” gombra kattintva megjelenik egy ablak, amely egy form panelt tartalmaz. A form kitöltésével és elmentésével tudunk új alkalmazottat felvenni a rendszerbe, aki az adatait a grid segítségével is könnyedén tudja módosítani, kivéve a jelszót, melynek megváltoztatására

kizárólag az adatlapon van lehetőség. A form saját xtype-ként van beregisztrálva, olyan módszerrel, amelyről már beszéltem. Felugró ablakként megjelenik szinte teljesen ugyanaz a form, mint az egyes alkalmazottak adatlapja, annyi különbséggel, hogy itt nincs lehetőség a jelszó megadására, módosítására. Nézzük most meg, hogyan történik az alkalmazott törlése.

```
text: 'Alkalmazott törlése',
  handler: function() {
    userGrid = Ext.getCmp('usergridID');
    var sm = userGrid.getSelectionModel();
    var selObj = sm.getSelected();

    if (sm.hasSelection()){
      Ext.Msg.show({
        title: 'Alkalmazott törlése',
        buttons: Ext.MessageBox.YESNO,
        msg: 'Valóban törölni szeretné a következő alkalmazottat: ' +
             selObj.data.v_name + ' ' + selObj.data.k_name + '?',
        fn: function(btn){
          if (btn == 'yes'){
            conn_alk.request({
              params: {
                action: 'delete',
                id: selObj.data.id
              },
              success: function(resp,opt) {
                userGrid.getStore().remove(selObj);
              },
              failure: function(resp,opt) {
                Ext.Msg.alert('Error', 'Nem sikerült törölni a követ-
                              kezőt: ' + selObj.data.v_name + ' ' +
                              selObj.data.k_name + ' Ok: ' + resp);
              }
            });
          }
        }
      });
    } else {
    Ext.Msg.alert('Figyelmeztetés',"Előbb ki kell választanod egy sort");
    }…
```

Itt egy anonim függvény található, amely először lekéri az azonosítója alapján a gridpanelt, amelyből törölni szeretnénk, majd azt, hogy a gridünk milyen kiválasztási modellt használ és ezeket elmenti egy-egy változóba. Esetünkben ez egy úgynevezett `RowSelectionModel`, tehát ha rákattintunk a grid egy rekordjára, akkor a rekordot tartalmazó sor lesz kiválasztva. Ezután a `selObj` nevű változóba belekerül az aktuálisan kiválasztott sor. Az ezt követő részben az történik, hogy amennyiben van sorunk kiválasztva, akkor egy felugró üzenetben az alkalmazás megerősítést kér tőlünk, hogy valóban végre szeretnénk-e hajtani az adott alkalmazott törlését, majd ha az „Igen" gombra kattintunk, a rendszer elküld egy megfelelően felparaméterezett kérést. Amennyiben a szerver felől az a válasz érkezik, hogy sikerült a törlés, akkor eltávolítom a gridből a kiválasztott objektumot, ellenkező esetben egy hibaüzenet segítségével tájékoztatom a felhasználót, hogy a törlés sikertelen volt. Ha nem választottunk ki sort, úgy szeretnénk törölni, akkor egy hibaüzenetet kapunk, hogy nincsen sor kiválasztva. Végül vessünk egy pillantást az adatlap kezelésére.

```
text: 'Adatlap',
   handler: function() {
   var sm = Ext.getCmp('usergridID').getSelectionModel();
   var selObj = sm.getSelected();

   if (sm.hasSelection()){
      l_alkTab = Ext.getCmp('alkTabID');
      l_newUserTab = l_alkTab.getItem('azon', selObj.data.id);
      if(!l_newUserTab) {
         l_newUserTab = l_alkTab.add({
            title: selObj.data.v_name + ' ' + selObj.data.k_name,
            id: 'tab_user_id_' + selObj.data.id,
            azon: selObj.data.id,
            closable: true,
            items: {xtype: 'alk_form', id: 'alk_form_id_' + selObj.data.id}
         });
      }
      l_alkTab.activate(l_newUserTab);
   } else {
    Ext.Msg.alert('Figyelmeztetés', "Előbb ki kell választanod egy sort");
   }…
```

Ezen `handler` első néhány sora pontosan ugyanazt a funkcionalitást látja el, mint amit a törlésnél bemutattam. A feltételvizsgálat után, amennyiben van kiválasztott sorunk az Ext segítségével ID alapján lekérjük azt a tab panelünket, amely az összes alkalmazottat megjelenítő gridet is tartalmazza. Az `l_newUserTab`-nak értékül adjuk a kiválasztott objektum azonosítóját és megvizsgáljuk, van-e már

megnyitva ilyen azonosítóval fül. Amennyiben még nincsen, úgy létrehozunk egy zárható tabot, amelyen a kiválasztott alkalmazott teljes neve fog megjelenni, maga a panel pedig az előzetesen xtypeként beregisztrált adatlapját fogja tartalmazni. Abban az esetben, ha már fel volt nyitva ez a tab, a rendszer automatikusan ráugrik arra a fülre, nem nyit fel újat. Ha nem választottunk ki sort, úgy kattintunk rá az Adatlap elemre, akkor egy hibaüzenetet kapunk, hogy nincsen sor kiválasztva. Az alkalmazottak kezelésénél minden esetben látszik az összes felhasználót tartalmazó grid és egy másik fülön az adott user saját adatlapja. Abban az esetben, ha egy felhasználónak nincs jogosultsága megtekinteni az összes alkalmazottat, úgy természetesen csak a saját adatlapját látja. Ezután hozzáadtam egy alsó lapozó eszköztárat, melyet a következő néhány sor készít el.

```
bbar: new Ext.PagingToolbar({
     pageSize: 10,           //elemek száma egy oldalon
     displayInfo: true,      //különböző információk mutatása
     store: this.store       //adattár megadása
})
```

Miután mindent meghatároztam, végeztem az osztály kiterjesztésével, már csak annyit csináltam, hogy az így létrehozott saját osztályomat beregisztráltam egy új xtypeként, a következő parancs segítségével.

```
Ext.reg('alk_grid', Felhasznalok.FelhasznaloGrid);
```

Így ha egy olyan felületre van szükségem, amelyet az eddigiekben ismertettem, nincs más dolgom, csak xtype-ként meg kell adnom a létrehozott nevet (`alk_grid`). Ezen kívül több szempontból is hasznos saját xtype definiálása. Egyrészt a kód újrafelhasználhatósága miatt, hiszen így nem kell több helyen ugyanazt a kódot implementálni, vagy akár csak bemásolni. Az `alk_grid`-et például felhasználtam a jogosultságkezelésnél és az erőforrások projektekhez rendelésénél is. Ezen kívül sokkal egyszerűbb létrehozni az adott felhasználó jogosultságának megfelelően megjelenő felületet is.

4.4. Projektek

A projektek szervezésére és nyomon követésére az előzőekben tárgyalt alkalmazottakéhoz hasonló megoldást készítettem, így ebben a részben csak azokkal a dolgokkal foglalkozom, amelyek a felhasználók kezelésétől eltérnek. A projektek kezelése ebben az esetben is egy szerkeszthető grid segítségével történik, de ez a grid azon túl, hogy szerkeszthető, csoportosítható is. Ez gyakorlatilag mindössze két módosítást igényel a kódban.

Egyrészt az adattároló megadásánál a `getXmlStore(id,url,model)` függvény helyett a `getXmlGroupingStore(id,url,model,groupingField)` függvényt hívom meg. Ez a már ismertetett `getXmlStore(id,url,model)` függvénytől annyiban különbözik, hogy ebben az

esetben beállításra kerül az a mező, amely alapján a gridben megjelenő adatok kategorizálva lesznek.

```
function getXmlGroupingStore(p_storeId, p_url, p_fieldsTomb, p_groupField)
{
     return new Ext.data.GroupingStore({
          storeId: p_storeId,
          proxy: getPostProxy(p_url),
              sortInfo: {
                   field: p_groupField,
                   direction: "ASC"
              },
          groupField: p_groupField,
          reader: getXmlReader(p_fieldsTomb),
          autoLoad: v_autoLoad
     });}
```

Természetesen ez csak egy alapértelmezést ad, mert a kifinomult gridnek köszönhetően az alkalmazás használata során bármelyik tulajdonság alapján elvégezhetjük a csoportosítást, vagy akár ki is kapcsolhatjuk azt.

A másik dolog, hogy az editor grid definiálásakor a `view` konfigurációnál az alkalmazottaknál használt `GridView` helyett most egy új `GroupingView`-t kell példányosítani a következőképpen:

```
view: new Ext.grid.GroupingView().
```

Ezzel a két kis módosítással máris elértük, hogy a projekteket tetszőleges tulajdonságuk alapján csoportosítani tudjuk, így az alapértelmezett prioritás alapján történő csoportosítás segítségével például azonnal láthatjuk melyek a kiemelt fontosságú projektek.

Az egyes projektek adatlapja tartalmazza az olyan hasznos információkat, mint például a határidő, kik dolgoznak az adott projekten, az egyes dolgozók hogy haladnak a feladatukkal, mennyi időt töltenek a projekt fejlesztésével, valamint milyen állapotban van a projekt készültsége. A szoftver evolúciójának egyik iránya ennek a felületnek fejlesztése.

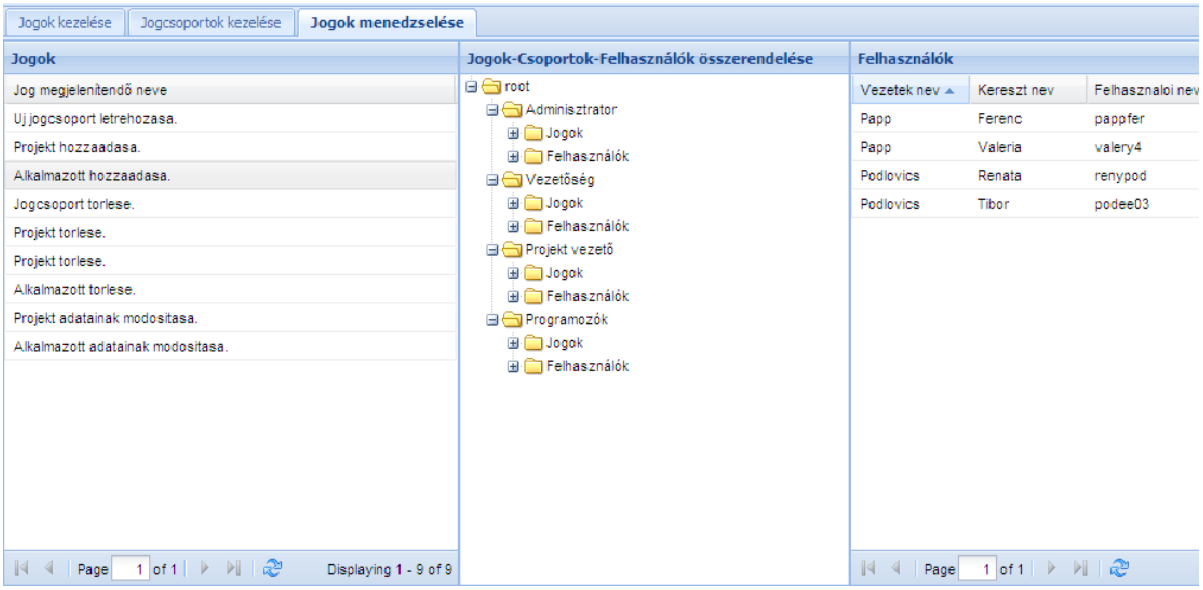
4.5. Jogosultságkezelés

Az alkalmazás implementálása során ez volt számomra a legérdekesebb, és egyben a legnehezebb rész. Sokat gondolkodtam azon, hogyan lehetne hatékonyan megvalósítani a jogosultságkezelést. Végül arra jutottam, hogy a tab panelen három fülre van szükségem.

Az elsőn egy egyszerű editor grid segítségével a jogok kezelése történik. Itt két oszlopban vannak kilistázva a rendszerben kioszthatóó jogok. Az első oszlopban található a jogok azon elnevezése, amelyet az adatbázis is használ, amely nem módosítható a felületről, a másodikban pedig minden joghoz meg tudunk adni egy beszédes megjelenítendő nevet. A többi részhez hasonlóan itt is saját xtype-ként regisztráltam be a gridet, amelyet a jogok menedzselésénél is használok, itt tehát újra előtérbe kerül az újrafelhasználhatóság.

A második fülön a jogosultsági csoportokat tudjuk kezelni. Ezen a részen is egy editor grid található, melybe fel tudunk venni új csoportot és ki tudunk törölni meglévőt. Programozás technikailag erről a két fülről nem fogok beszélni, mert a rajtuk megtalálható elemeket már korábban bemutattam. Az adattárolókat itt is a már említett `data_stores.php` fájl regisztrálja be, valamint a regisztráció előtt meghívja a `util.php` fájl megfelelő függvényét, amely elkészíti a tárolót a megadott paramétereknek megfelelően és egyben hozzá is rendeli a szükséges XmlReadert. Az adatmodelleket a `data_models.php` fájl tartalmazza. A harmadik fül már tartogat számunkra újdonságokat, amelynek megvalósítása a munkám során a legnagyobb kihívást jelentette.

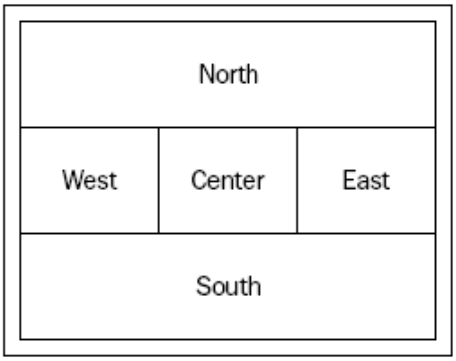
Az utolsó fülön történik a felhasználók és a jogok csoportokhoz rendelése. Hosszas mérlegelés és tervezés után úgy döntöttem, hogy egy három oszlopból álló felületet fogok használni. Az első oszlop tartalmazza a kioszthatóó jogokat, a harmadik oszlop pedig a felhasználókat. Itt újra felhasználtam a megírt saját xtype-okat. A középső oszlopban egy tree panel található, melyet úgy építek fel az adatbázisból, hogy minden egyes felhasználói csoportot reprezentáló mappának van két almappája: „Jogok” és „Felhasználók”. Tehát amikor a csoportok kezelésénél létrehozunk egy új csoportot, akkor az bekerül az adatbázisba és ezáltal automatikusan beépül a fába is az említett két almappával együtt. A megfelelő mappába drag-and-drop módszerrel lehet belehúzni a jogokat és a felhasználókat. Természetesen a „Jogok” mappába csak jogokat, a „Felhasználók” mappába csak felhasználókat tehetünk, amire a rendszer figyel. Mielőtt ennek a résznek a technikai megvalósításáról beszélnék, egy képen tekintsük meg, miről is van szó.



8. ábra: A jogosultságkezelést megvalósító rész

Elsőként vizsgáljuk meg az **elrendezést**. Eddig még nem beszéltem erről az igen fontos részről, ezért most foglalkozunk vele egy kicsit, ugyanis az elrendezés az, ami a grideket és más widgeteket egy igazi web alkalmazássá gyúrja össze. Ahhoz, hogy az alkalmazásunk megjelenése konzisztens maradjon a támogatott böngészőkben, és biztosítani tudjuk a leggyakoribb lehetőségeket, az Ext JS egy nagyon erőteljes elrendezés kezelő rendszerrel (Layout Management System) rendelkezik. A különböző részek menedzselhetőek, mozgathatók, eltüntethetők és adott eseményre reagálva megjelenhetnek ott, és akkor, amikor éppen szükségünk van rá. Az Ext *paneleket* használ, melyek a legtöbb elrendezés alapját képezik. Ezek közül én is használtam már a FormPanelt, a GridPanelt és ebben a részben a TreePanel kerül előtérbe.

Az elrendezés kezelés másik alapvető eleme az úgynevezett *viewport*, ami egy speciális panelszerű komponens, mely a teljes látható területet lefedi és magába foglalja az egész elrendezést. Régiókból épül fel, melyek a következőképpen helyezkednek el:



9. ábra: A viewport régiói

Az én alkalmazásom megjelenése is egy viewporttal van definiálva, amelyben nem használtam fel a keleti és nyugati régiót. Számos elrendezést használhatunk az Ext-ben, amelyek részletes ismertetésére sajnos most nincs lehetőségem, mert az önmagában is egy hatalmas témát ölelne fel, de röviden bemutatok közülük néhányat.

- border layout: Ebben az esetben minden egyes régió egy határoló résszel van elválasztva, amelynek mozgatásával az egyes részek átméretezhetőek.
- card layout: Ide tartozik például az általam használt tab panel is. Azért a kártyáról kapta a nevét, mert úgy viselkedik, mint egy pakli kártya, hiszen mindig pontosan definiált a lapok sorrendje és bármelyik mozgatható a „pakli" tetejére, hogy láthatóvá váljon.
- accordion layout: Nagyon hasznos elrendezés, amely a tab panelekhez hasonlóan működik. Több részből tevődik össze, amely ugyanazt a területet használja és egyszerre csak egy kerül megjelenítésre.
- hbox: A column modellhez hasonló tagolást biztosít.
- vbox: Segítségével hosszanti irányban tagolhatjuk a felhasználható területet.

Dióhéjban ennyit mindenképpen szükségesnek tartottam megemlíteni az elrendezésekkel kapcsolatban, mielőtt megvizsgáljuk, hogyan épül fel a jogok menedzselését végző rész.

Amint azt az x. ábrán már láthattuk, ez a rész két grid panelből és egy tree panelből épül fel. A gridek implementálásával nem kívánok foglalkozni, mert azt már korábban megvizsgáltuk. Vegyük most nagyító alá a tree panelt létrehozó kódot. Először is megadtam a fát betöltő úgynevezett TreeLoader-t, melyet egy változóba el is tároltam. Ez megkapja annak a fájlnak a nevét, amelyből majd be kell tölteni az adatokat a fába.

```
var jogosultsagTreeLoader = new Ext.tree.TreeLoader({
     dataUrl:'jogok_menedzselese.php'
});
```

Ezután egy változóba elmentettem a fa gyökerét, melyet `AsyncTreeNode`-ként hoztam létre. Azért használok aszinkron csomópontot, mert így a fát szépen áganként tudom felépíteni, miközben kérem le az adatokat a szerverről, nem pedig az egészet egyszerre. Az AsyncTreeNode Ajaxot segítségével biztosítja a felhasználókat, hogy ne kelljen túl sokat várakozniuk. Itt szeretném megjegyezni, hogy az alkalmazásomban a `connection.php` fájl tartalmazza az Ajaxos hívásokat, valamint az egyes figyelők `success`, illetve `failure` ágának kezelését. Ezek után lássuk a fát létrehozó kódrészletet.

```
var jogosultsagTree = new Ext.tree.TreePanel({
     requestMethod: 'POST',       //a http kérés módja
     useArrows: true,             //Vista stílusú nyilak használata
     autoScroll: true,
          //görgetősáv automatikus megjelenítése, ha szükséges
     animate: true,               //animáció engedélyezése
     enableDrop: true,            //a fába dobás engedélyezése
     containerScroll: true,       //a tároló görgethető a belehúzáskor
     titleCollapse: true,
          //kinyithatjuk és bezárhatjuk a mappát a nevére kattintássaé
     ddGroup: 'grid2tree',        //drag and drop csoport megadása
     loader: jogosultsagTreeLoader,   //betöltő megadása
     root: jogosultsagRootNode,  //gyökér definiálása
     selModel: new Ext.tree.MultiSelectionModel(),
          /*
           * Megadhatjuk, hogyan tudunk elemet kiválasztani a fában.
           * Ennek a modellnek a segítségével egyszerre több elem
           * is kiválasztható.
           */
     id: 'jogosultsagTree_id',    //azonosító hozzárendelése a fához
     region: 'center',            //elhelyezés a középső régióban
     title: 'Jogok-Csoportok-Felhasználók összerendelése',
     layout: 'fit',               //elrendezés megadása
     width: 300,                  //szélesség megadása pixelben
     split: true                  //méretezhetővé teszi az adott részt
});
```

Ezzel létrehoztam és testre szabtam a tree panelemet. Létrehoztam még egy felugró menüt is a törlés megvalósítására.

```
var contextMenu = new Ext.menu.Menu({
     items: [
          { text: 'Delete', handler: jogosultsagTreeDeleteHandler }
        ]
});
```

Amennyiben szeretnék valahonnan kitörölni egy, vagy több elemet, nincs más dolgunk, mint kijelölni azokat, a jobb egérgombbal egyet kattintani valamelyik kijelölt elemen és a felugró menüből kiválasztani a törlés funkciót. Ekkor lefut a `handler` után megadott eseménykezelő, mely annyit csinál, hogy a kiválasztott csomópontokat belehelyezi egy tömbbe, és amíg a tömb ki nem ürül, minden elemét megvizsgálja, hogy az levél-e. Ha igen, akkor kitörli, ha

nem, akkor pedig megszünteti a kijelölést. Ezáltal a rendszer azt is biztosítja, hogy törölni csak levélelemet tudunk, ami esetünkben jog, vagy felhasználó lehet. A fához hozzáadtam még néhány eseménykezelőt, melyek kódja a függelékben megtalálható.

```
jogosultsagTree.on('contextmenu', jogosultsagTreeContextHandler);
```

Ez a kezelő határozza meg, hogy melyik elemnél kell megjeleníteni a felugró menüt.

```
jogosultsagTree.on('beforenodedrop',jogosultsagTreeBeforeNodeDropHandler);
```

Megvizsgáljuk vannak-e kiválasztott rekordok, és ha igen, akkor létrehozunk egy üres tömböt, melybe az adataink kerülni fognak. Egy for ciklus segítségével végigmegyünk a kiválasztott rekordokon és az aktuálisat a ciklus magjában eltároljuk egy „r" változóban. A következő részben megvizsgáljuk, hogy melyik gridből húztuk át az adatot, hogy a fában létrejövő új csomópontnak megfelelő nevet tudjunk adni. Ezután az elemnek generálunk egy azonosítót és megvizsgáljuk, hogy a cél csomópontnak létezik-e már ilyen azonosítójú gyermeke. Ha létezik, akkor már benne van az adott elem a mappában és a rendszer nem engedi kétszer belehúzni, egyébként pedig beleteszi az elemet a fába egy új levélként.

```
jogosultsagTree.on('nodedragover', jogosultsagTreeNodeDragOverHandler);
```

Ez az eseménykezelő vizsgálja, hogy amikor húzunk egy elemet valamelyik gridből a fába, akkor az szabályos-e. Csak az olyan mozgatást engedélyezi, amikor a jogok gridből a jogok mappába, vagy a felhasználók gridből a felhasználók mappába húzunk át elemet. Az összes többi esetben jelzi is, hogy a műveletet nem lehet végrehajtani. Így az is nyilvánvaló, hogy csak a fába lehet elemet húzni, és ha olyan helyre szeretnék elemet dobni, ahová nem lehet, az a művelet automatikusan sztornózásra kerül.

4.6. Erőforrások projektekhez rendelése

Az erőforrások projektekhez rendelésénél egy hasonló drag-and-drop megoldást alkalmaztam, mint a jogosultságkezelésnél, annyi eltéréssel, hogy itt csak egy gridet kell együtt kezelnem egy fával, így erről a részről nem is szándékozom részleteiben írni. Az erőforrások esetünkben az alkalmazottak. Annyit szeretnék megjegyezni, hogy ebben a részben nem történnek vizsgálatok arra nézve, hogy egy adott alkalmazott ráér-e akkor, amikor az adott projekt fut, a program nem jelzi az ütközéseket és nem kezeli a ledolgozott órákat, fizetéseket. Ezek a funkciók a program további fejlesztése során fognak bekerülni a rendszerbe.

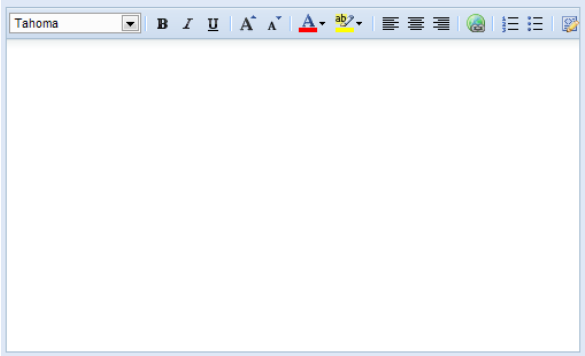
4.7. Üzenetek

Itt egy cégen belüli levelezési lehetőség megvalósítása volt a cél, tehát nem egy e-mail rendszer létrehozása. Esetemben nincs szükség levelező szerverre sem. A munkafolyamat hatékonyabbá tételében nagy szerepe van az üzeneteknek, hiszen alkalmazásukkal gyorsan kérhetünk segítséget, vagy beszélhetünk meg találkozót, csak hogy néhány lehetséges felhasználási területet említsek. A felépítés nagyon egyszerű, a funkciók három fülön vannak elhelyezve.

Az elsőre kattintva tudunk új üzenetet küldeni. Itt egy form panelt hoztam létre, melyen két szöveges mező található a címzett és a tárgy megadására és egy szerkesztő terület, ahol létrehozhatjuk az üzenetünket. Bámulatos milyen könnyedén hozhatunk létre az Ext JS segítségével egy kis szerkesztőt, amely formázási lehetőségeket is biztosít számunkra. Csupán a következő néhány sorra van szükségünk.

```
…{
      x:230, y: 80,
      xtype: 'htmleditor',
      name: 'msg', height: 300
}…
```

Sőt mi több, gyakorlatilag egyetlen sor az, amelyik az 10. ábrán látható szövegszerkesztőnket létrehozza, a többi opció csak a pozíciót, a magasságot és a nevet adja meg.



10. ábra: HTML editor

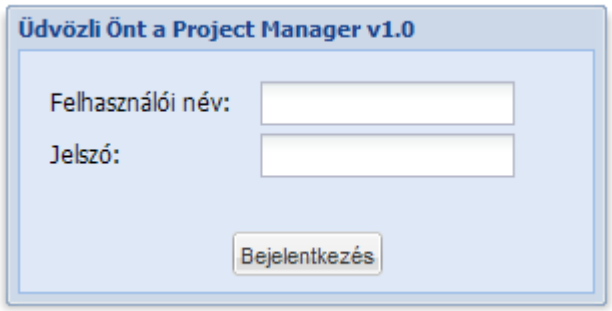
A további két fül tartalmazza a bejövő, valamint az elküldött üzeneteket. A két rész teljesen hasonlóan épül fel, az oldal felső felében egy grid helyezkedik el, amely az üzeneteket tartalmazza, és ha kiválasztunk egy üzenetet, akkor annak szövege az alsó részen megjelenik. Mindkét helyen lehet üzenetet törölni, illetve a bejövő üzeneteknél lehetőség van azonnal válaszolni egy gomb segítségével. A válasz gombra kattintva a rendszer átirányít minket az üzenetek írásához, ahol a címzett és tárgy mezők automatikusan kitöltődnek.

5. Az alkalmazás bemutatása

5.1. Felhasználói dokumentáció

Ebben a fejezetben bemutatom az alkalmazást, annak felépítését és használatát a felhasználó szemszögéből, afféle felhasználói dokumentáció jelleggel, hiszen egy alkalmazás fejlesztésének fontos eleme a dokumentáció. Tekintsük most át, hogy a felhasználó milyen felületekkel találkozik, s az egyes menüpontokban milyen funkciók állnak rendelkezésére.

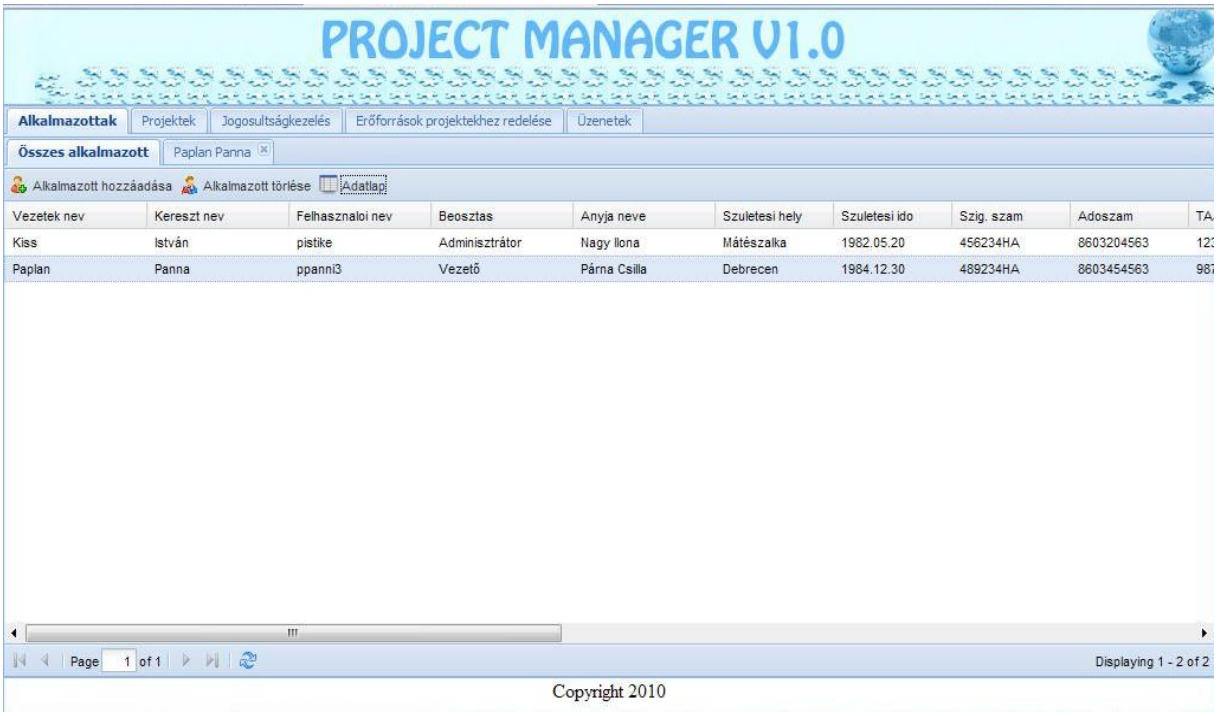
Amikor a felhasználó böngészőjében meglátogatja a web alkalmazásunkat, elsőként a bejelentkező panel jeleneik meg számára, ahol felhasználói nevének, és jelszavának helyes beírása után a „Bejelentkezés” gombra kattintva tud belépni a felületre, amennyiben a beírt felhasználói név, jelszó páros létezik a rendszerben, vagyis az autentikáció sikeres volt. A bejelentkező képernyővel az implementációs részben már foglalkoztunk, ám ott elsősorban fejlesztői oldalról vizsgáltuk. Vessünk egy pillantást rá, hogyan is épül fel ez a panel a következő képernyőkép segítségével.



11. ábra: A bejelentkező panel

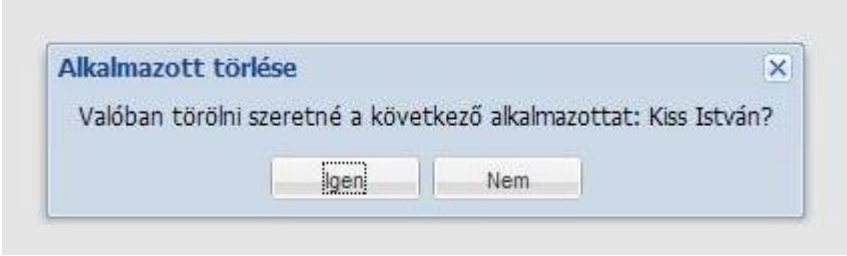
Mivel az alkalmazás egy adott cég dolgozói számára kell, hogy elérhető legyen, így regisztrációra nincs lehetőség, az alkalmazottaknak a rendszer adminisztrátora, vagy a megfelelő jogosultsággal rendelkező személy adhat hozzáférést, azaz egy felhasználói nevet és egy jelszót.

Belépés után a felhasználó a jogosultságának megfelelő elemeket és funkciókat tartalmazó felületet látja. Ebben a leírásban a teljes felületet ismertetni fogom, vagyis egy olyan felhasználóét, akinek mindenhez van jogosultsága. A többi felhasználói felület is ebből származik, csak jogosulatlan elemek nincsenek megjelenítve, azok az adott alkalmazott számára nem elérhetőek. Sikeres belépés után az alábbi felület jelenik meg:



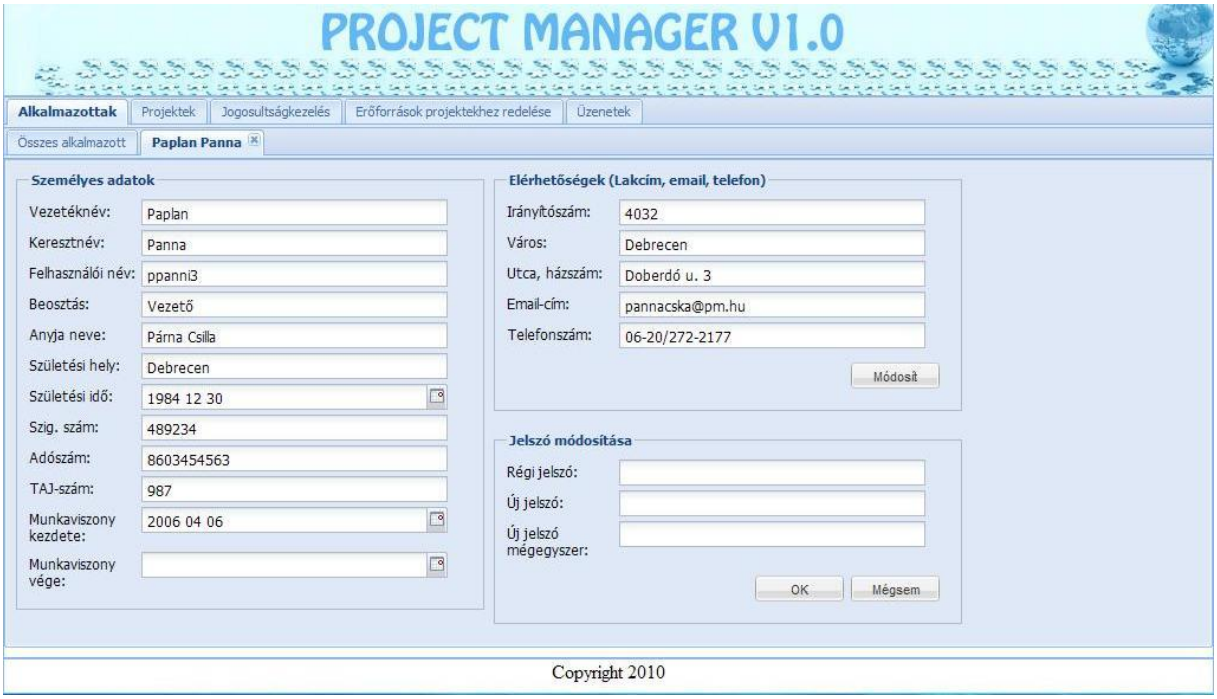
12. ábra: A felhasználói felület

Amint láthatjuk a felület úgynevezett tab panelekből, fülekből épül fel. Alapértelmezésben az „Alkalmazottak” fül aktív, melyen megjelenik az összes alkalmazott. Könnyedén tudunk új személyt felvenni és egy kiválasztottat törölni, vagy annak megtekinteni az adatlapját a megfelelő gombok megnyomásával. Alkalmazott hozzáadása során a táblázat utolsó sorába történik az új felhasználó beszúrása, ahol minden tulajdonságot be kell állítanunk. Törléskor a rendszer egy felugró ablakban megerősítést kér a művelet végrehajtásáról



13. ábra: Felhasználó törlése

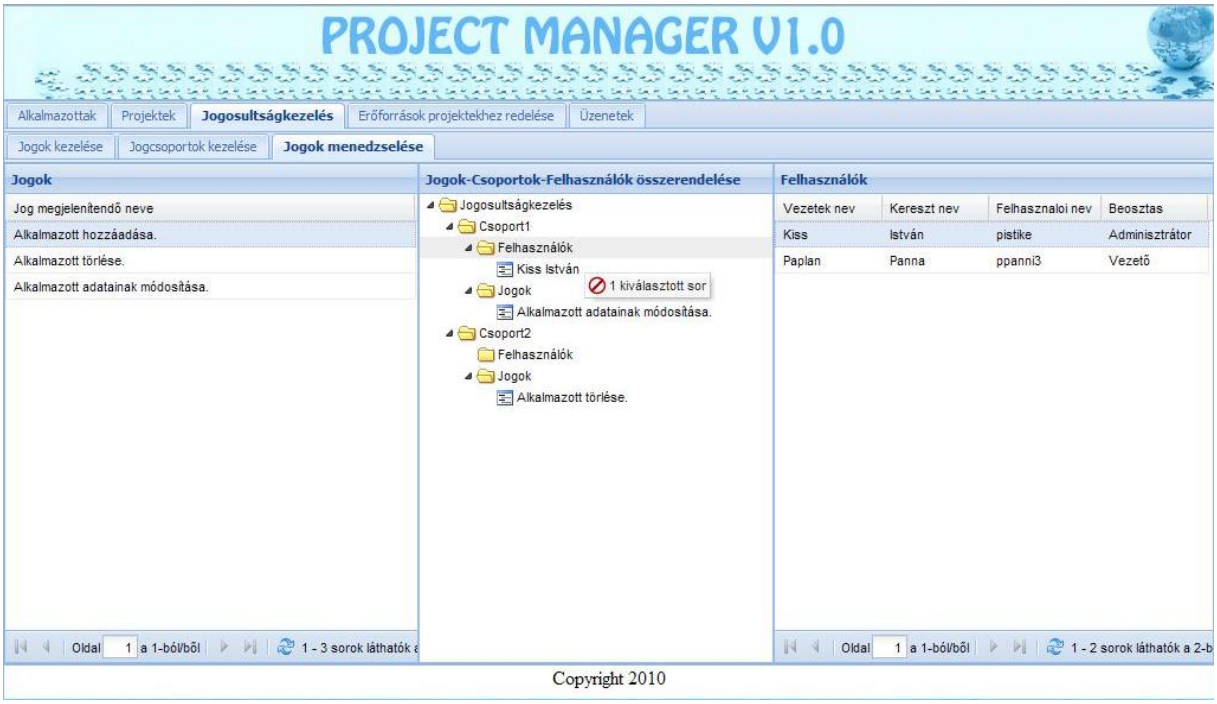
Több felhasználó adatlapja is megnyitható, melyek a megnyitás sorrendjében az „Összes felhasználó” nevű fül után fognak megjelenni dinamikusan egy-egy zárható fülön. Az egyes füleken a kiválasztott alkalmazott neve fog szerepelni, amint azt a x. ábrán láthatjuk. Amint azt már említettem, egy alkalmazott adatlapja csak egyszer nyitható meg, ha már meg van nyitva és még egyszer rákattintunk az „Adatlapra”, akkor a megnyitott fülre ugrik a program.



14. ábra: Alkalmazott adatlapja

A projektek fülre kattintva tekinthetjük meg a cégnél futó projektek adatait egy olyan táblázatban, amelyet bármelyik tulajdonság szerint csoportosíthatunk. alapértelmezésben a prioritás alapján történik meg a kategorizálás. Projektet felvenni a rendszerbe, vagy törölni onnan, az alkalmazottakéhoz hasonló módon tudunk.

A jogosultságkezelés három részből áll: jogok-, jogcsoportok kezelése és maga a jogosultságok menedzselése. A jogok kezelése annyit jelent, hogy a rendszerben kioszthatóó jogoknak megadhatjuk, mi legyen a felületen megjelenő neve. A jogcsoportok kezelésénél lehetőségünk van új csoportot felvenni, vagy egy meglévőt eltávolítani a rendszerből. A harmadik fülön tudjuk megadni, hogy az egyes csoportokba kik tartozzanak, illetve a csoportnak milyen jogosultsága legyen. A felület közepén elhelyezkedő fába tudjuk a két szélső táblázatból behúzni a felhasználókat és jogokat. Minden csoportot reprezentáló mappának van két almappája, melyek közül az egyik a felhasználókat a másik a jogokat tartalmazza. A rendszer ellenőrzi, hogy az egyes mappákba csak a megfelelő elemek kerülhessenek, vagyis a „Jogok” mappába csak jogok, a „Felhasználók” mappába csak felhasználók, továbbá egy csoporton belül, egyik mappába sem húzható bele ugyanaz az elem kétszer, így a program csökkenti a hibaforrások lehetséges számát. Egyszerre több elemet is kijelölhető, így csoportosan is mozgathatunk át jogokat és felhasználókat egyaránt a fába.



15. ábra: Jogosultságkezelés

A fából törölni nagyon egyszerűen tudunk, mindössze ki kell jelölnünk azt az elemet, vagy elemcsoportot, amelyet el szeretnénk távolítani, jobb egérgombbal kattintunk egyet valamelyik kiválasztott elemre, majd a megjelenő menüből kiválasztjuk a törlés lehetőséget. Fontos dolog, hogy csak levélelemet tudunk törölni, mappát nem!

Az erőforrások projektekhez rendelése a jogosultságkezeléshez hasonlóan történik, itt is egy táblázatból húzhatjuk be az erőforrásokat a fa megfelelő projektet reprezentáló mappájába.

Végül a rendszer biztosít egy belső levelezési lehetőséget, ezáltal a találkozók, bemutatók, trainingek szervezése sokkal gyorsabbá válik, valamint könnyedén tud a felhasználó segítséget kérni a többiektől. Lehetőség van új üzenet írására, vagy reagálni egy bejövő üzenetre. A rendszer tárolja a küldött üzeneteket, így az a felhasználó számára bármikor visszakereshető. Ezen felül mind a bejövő, mind a kimenő üzenetek közül van lehetőség az általunk kiválasztott törlésére.

5.2. Néhány szó a biztonságról

Napjainkban az internet és a webes alkalmazások terjedésével egyre nagyobb hangsúlyt kap a biztonság, ezért én is szeretnék néhány szót szólni róla. Az alkalmazás fejlesztésénél igyekeztem odafigyelni a biztonságra.

Először is az alkalmazottak jelszavát az adatbázisban titkosítva tárolom. Ez azért nagyon fontos, hogy amennyiben valaki esetleg hozzáférést szerez az adatbázishoz, akkor sem tud mit kezdeni a jelszavakkal, mert azok titkosítva vannak eltárolva. A MySQL password() függvényét használtam a jelszavak titkosítására, amely az eredeti jelszóból kiszámol egy új sztringet és azzal tér vissza, azt tárolja az adatbázisban. A password() függvény által alkalmazott kódolási technika nem visszafejthető.

Másodszor beszéljünk kicsit az úgynevezett „SQL injekcióról". Ennek a támadási módszernek az a lényege, hogy egy adatbázist használó alkalmazásnak valamilyen módon olyan adatokat juttatunk be, hogy az eredetileg végrehajtandó SQL parancsok helyett az alkalmazás az általunk megadottakat végzi el. Tekintsük például a felhasználó bejelentkezését, ami egy nagyon gyakori problémaforrás. Ebben az esetben egy PHP szkript végzi a bejelentkező panelen megadott felhasználói név és a jelszó ellenőrzését a következő kód szerint:

```
$q = mysql_query(
    "select 0 from users WHERE username = '$username' AND pass = '$pass'"
);
// ha van ilyen felhasználó ilyen jelszóval
if (mysql_num_rows($q)==1)
    exit("sikeres belépés");
else
    exit("sikertelen belépés");
```

Ez a módszer tökéletesen működik mindaddig, amíg a felhasználók a megfelelő adatokat adják meg, és nem trükköznek. Azonban, ha például a felhasználói névnek azt adják meg, hogy *admin*, jelszónak pedig, hogy *'OR' 1*, akkor a PHP által indított mysql query a következőképpen néz ki: `"select 0 from users WHERE username = 'admin' AND pass = '' OR '1'"` Ez a lekérdezés pedig kiválasztja azt a felhasználót, akinek felhasználói neve admin. jelszava pedig üres, vagy 1, ami önálló feltételként jelenik meg, ez pedig azt jelenti, hogy logikailag igaz, tehát ez a feltétel mindig teljesül. Így máris beléptünk a megadott felhasználói névvel. Ezzel a módszerrel szinte bármilyen SQL parancsot lefuttathatunk, ami nagyon veszélyes. Az ilyen jellegű támadások ellen védekezhetünk úgy, hogy például a jelszót kódolva tároljuk. A másik módszer a bemenet szűrése, ha tudjuk milyen típusú adatot várunk, akkor nem szabad bármit beengedni, továbbá nem szabad kiírni semmilyen adatbázisra jellemző információt. Az általam fejlesztett alkalmazásnál figyeltem ennek a biztonsági résnek a védelmére, szűröm a speciális karaktereket, és a jelszavak is titkosítva vannak eltárolva.

Fontosnak tartom itt megjegyezni továbbá, hogy a kliens oldalon történő validálás mellett minden beérkező adatot a szerver oldalon is le kell ellenőriznünk, hiszen alapvető szabály, hogy a felhasználó által küldött bemenetben nem szabad megbízni. Így igaz ugyan, hogy sok dolgot kétszer is leellenőrzünk és feleslegesnek tűnhet a kliens oldalon is validálni, azonban a felhasználói élményt, a használhatóságot nagyban javítja, ha már az adatok bevitelekor figyelmeztetjük a usert, ha valamit nem a megfelelő formátumban adott meg.

Végül az úgynevezett *session lopás*ról szeretnék beszélni. A session-ök, vagy más néven munkamenetek arra valók, hogy egy felhasználót a megfelelő autentikáció után úgymond bent tartson egy webes alkalmazásban. Bejelentkezés után a felhasználóhoz olyan változók rendelődnek és tárolódnak le a szerveren, amelyek nem szabadulnak fel az oldal újratöltéskor, így elérhető az adott domain valamennyi PHP kódjában. A webszerverek a klienst, ami a felhasználó böngészőjét jelenti esetünkben és ezáltal magát a felhasználót is, úgynevezett sütik (cookie) segítségével kötik össze a szerveren ily módon tárolt adatokkal. JavaScript segítségével pedig elérhető a sütik tartalma, mivel azt a böngésző tárolja, ezáltal pedig a PHPSESSID, a session azonosítója is elérhetővé válik. Ha ezt sikerül megszerezni, nincs más dolgunk, mint bejelentkezni az adott rendszerbe, a kapott session azonosítót kicserélni, amire léteznek különböző programok és máris más nevén leszünk bent anélkül, hogy a felhasználói nevét, vagy jelszavát megszereztük volna. Innentől kezdve csak a biztonsági résen, a támadón és annak szakértelmén múlik, mekkora kárt tud okozni. Ez ellen nem védekeztem, és igazából nem is létezik tökéletes megoldás az ilyen jellegű támadások ellen.

6. Üzembe helyezés és tesztelés

Miután elkészült az alkalmazás egy használható verziója, fontos lépés tájékoztatni a felhasználót arról, hogyan tudja használatba venni azt ezért a következő néhány sorban leíromm milyen lépések szükségesek az alkalmazásom üzembe helyezéséhez.

Először is létre kell hozni az adatbázist. Ehhez ki kell választani az adatbázist, majd le kell futtatni az általam mellékelt SQL szkript közül azt, amelyik a táblák létrehozását végzi. Ezen felül mellékelek egy olyan SQL-t is, amely példaadatokkal tölti fel a már meglévő adatbázist. A második lépésben létre kell hozni a rendszerben egy olyan felhasználót, akinek írási jogosultsága van a létrehozott táblákhoz, vagyis van megfelelő jogosultsága. Ezután módosítani kell a programban található `db_conn.php` nevű fájlt, amely következőképpen épül fel:

```
<?php
     $host = 'localhost';     // az adatbázis-szerver host-ja
     $felh = 'root';
            // felhasználó, mely hozzáfér az alkalmazás tábláihoz
     $jelsz = '';             // jelszó
     $db = 'diplomamunka';
            // MySql adatbázis, amely az alkalmazás tábláit tartalmazza
     mysql_connect($host, $felh, $jelsz)
         or die('Nem sikerült kapcsolódni az adatbázis-szerverhez a követ-
         kező paraméterekkel: host=' . $host . ', user=' . $felh);
     mysql_select_db($db)
         or die('Nincs ilyen adatbázis: ' . $db);
?>
```

Be kell tehát állítani az adatbázis-szerver hostját, a létrehozott felhasználót, a jelszót és az adatbázis nevét, amely a táblákat tartalmazza. Ezek után máris használható az alkalmazásom.

Ejtenék még néhány szót a tesztelésről. User tesztet végeztem Windows és Linux környezetben. Mindkét esetben megvizsgáltam az Ext által támogatott böngészők egy részét, hogy valóban teljesül a böngésző függetlenség. Teszteltem az alkalmazást Internet Explorer 7.0.5730.11, Firefox 3.6.3, Opera és Chrome alatt, s a böngészőfüggetlenség valóban teljesült, a megjelenés egységes volt, az implementált funkciók megfelelően működtek. Machintos platformon Safarival nem volt lehetőségem letesztelni az alkalmazást, így erről nem tudok állást foglalni.

7. Összefoglalás

A munkám során arra törekedtem, hogy a dolgozat elején megfogalmazott célokat a lehető legmagasabb színvonalon teljesítsem. A rendszerfejlesztés technológiája nevű kurzuson tanultakat követve igyekeztem végigvezetni egy rendszerfejlesztési életciklust, és az ott szerzett tudást átültetni a gyakorlatba. A saját bőrömön tapasztalhattam, hogy a tervezést milyen körültekintően kell elvégezni, mert annak ellenére, hogy nagy figyelmet fordítottam erre a szakaszra, a fejlesztésnél előjöttek olyan problémák, amelyekre nem számítottam és orvoslásuk bizony sok fejtörést okozott a számomra. A célul kitűzött webes projektmenedzselő alkalmazás első verziója végül elkészült, melyben minden funkció használható. A fejlesztés előrehaladtával arra döbbentem rá, hogy egy szoftver bizony soha nincsen teljesen kész, mindig van továbblépési lehetőség. Az implementáció során számos dolog jutott eszembe, amelyek beépítésével tovább növelhető az alkalmazás használhatósága.

Röviden szeretném ismertetni a szoftver evolúciós lehetőségeit, azokat a továbbfejlesztési irányokat, amelyek eszembe jutottak. Az egyik ötletem egy olyan komponens beépítése a programba, amely lehetővé teszi az egyes projektekről statisztikák, kimutatások készítését, a projekt fázisainak és készültségének grafikus megjelenítését. Az erőforrások projektekhez rendelésénél megvalósítani a már említett hiányosságokat, azaz megvizsgálni, hogy az adott alkalmazott beosztható-e a projektre, nincs-e ütközés más munkával, esetleg szabadsággal. Ütközés esetén a rendszer megoldást javasolna, amely segítségével elkerülhető az összeférhetetlenség. Továbbá hasznos lenne egy olyan modul készítése, amely nyomon követi a felhasználók munkáját. Ez alatt értem például a ledolgozott munkaórák számát, ezek alapján a fizetések megállapítását, esetleges túlórák kezelését. A projektekkel kapcsolatos események és az alkalmazottak tevékenységének logolása is hasznos segítőtárs lehetne egyrészt a nyomon követés, elemzés szempontjából, másrészt az alkalmazottak ösztönzésére is fel lehetne használni ezt a dolgot. Gondolok itt arra, hogy például minden hónapban a legjobban teljesítő dolgozó jutalomban részesül, vagy az, aki egy adott projekten kiemelkedő munkát végez, és még sorolhatnám. Ezeken túl rengeteg kis apróság van, amelyek tovább növelhetnék a rendszer hatékonyságát. Ezek elsősorban a különböző nem kívánatos felhasználói események kiküszöbölésére irányulnak. Például ha a felhasználó módosít valamit az adatlapján és elfelejti elmenteni, akkor a rendszer figyelmeztesse őt az ablak, vagy panel bezárásakor.

A dolgozat készítésével sikerült közelebb kerülnöm a webes alkalmazásfejlesztéshez, betekintést nyertem sok, számomra eddig ismeretlen dologba és rádöbbentem milyen hatalmas a web világa, milyen sok területhez kell érteni annak, aki a webre kíván fejleszteni. Alap szinten megismerkedtem az Ajax technológiával is, annak számos előnyös tulajdonságával, valamint behatóbban tanulmányoztam az Ext JS nevű JavaScript keretrendszert, amely tapasztalataim szerint hazánkban még kevésbé ismert, de a népszerűsége egyre nő, nem véletlenül. Egy rendkívül erőteljes és jól használható eszközt ismerhettem meg, amely nagy segítséget jelent minden web fejlesztő számára a kliens oldal elkészítésében. A dolgozat tárgyalásánál is erre a részre helyeztem a hangsúlyt, megpróbáltam minél szemléletesebben bemutatni azokat a részeket, amelyeket a munkám során felhasználtam. A jövőben szeretném tovább képezni magam ezen a területen és szélesebb körű, mélyebb ismeretekre szert tenni.

Felhasznált irodalom

Könyvek

Chris Bates: *XML Elmélet és gyakorlat*, 2004, Panem Könyvkiadó

Colin Ramsay, Shea Frederick, Steve 'Cutter' Blades: *Learning Ext JS*, 2008, PACKT Publishing

Jesus Garcia: *Ext JS in Action*, 2009, Manning Publications Co.

Jorge Ramon: *Ext JS 3.0 Cookbook*, 2009, PACKT Publishing

Kris Hadlock: *Webalkalmazások fejlesztése Ajax segítségével*, 2007, Kiskapu Kiadó

Matt Zandstra: *Tanuljuk meg a PHP 5 használatát 24 óra alatt*, 2005, Kiskapu Kiadó

Sági Gábor: *Webes adatbázis-kezelés MySQL és PHP használatával*, 2005, BBS-INFO Kiadó

Internetes források

Ext JS API dokumentáció: http://www.extjs.com/deploy/dev/docs/

Ext JS Learning Center: http://www.extjs.com/learn/Main_Page

Ext JS fórum: http://www.extjs.com/forum/

Ext JS 3.2 példák: http://www.extjs.com/deploy/dev/examples/

Saki's Ext example page: http://examples.extjs.eu/

MySQL titkosítás: http://dev.mysql.com/doc/refman/5.1/en/encryption-functions.html

PHP biztonság: http://www.php-blog.hu/php-magyar-kezikonyv/security.database.html

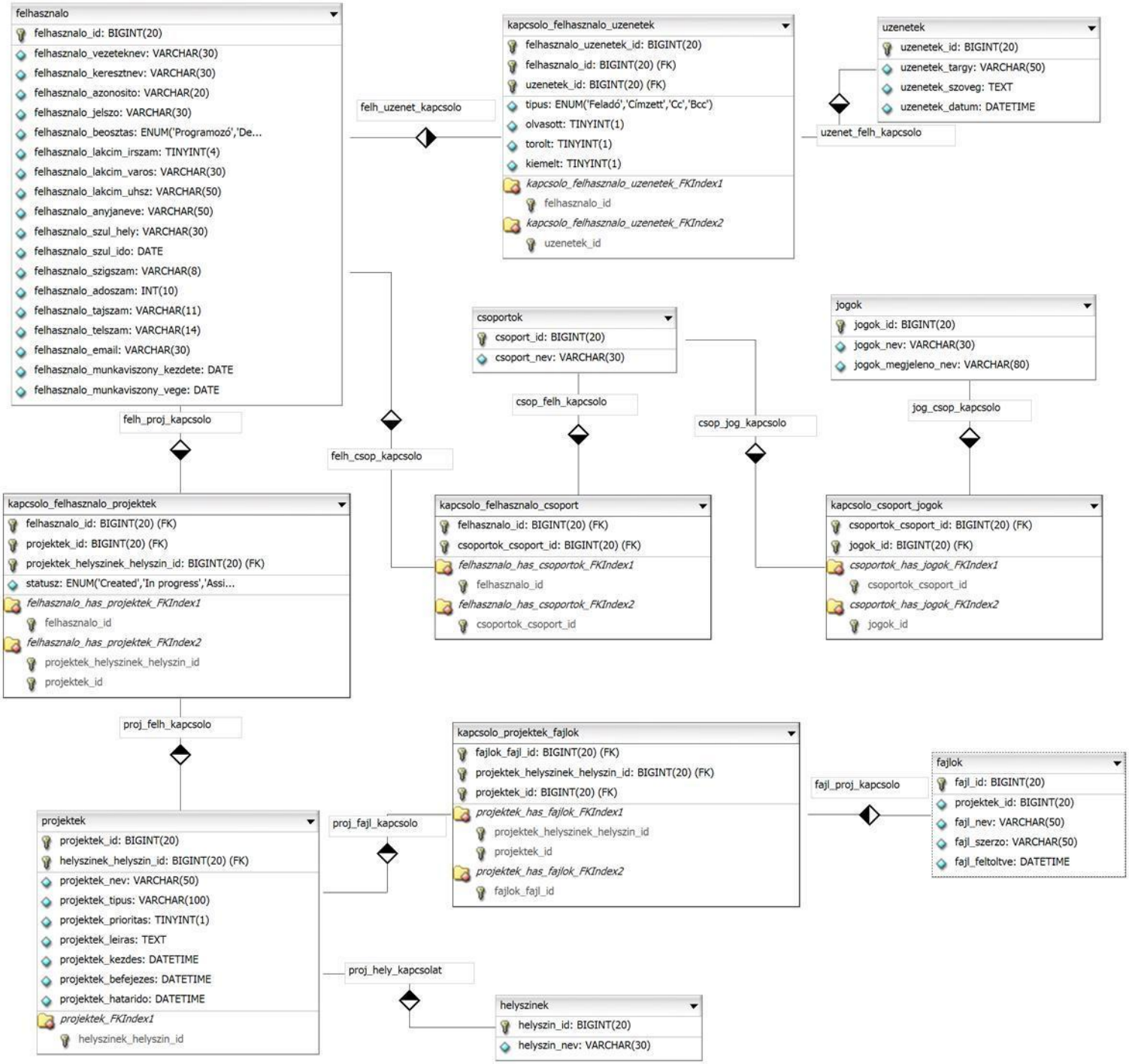
PHP manual: http://www.php.net/manual/en/

XAMPP: http://www.apachefriends.org/en/xampp.html

Web alkalmazás fogalmának meghatározása: http://www.webgo.hu/web-alkalmazas-web-application-webapp

Függelék

Az adatbázis sémája:



A jogosultságkezelésnél használt eseménykezelők implementációja:

jogosultsagTreeDeleteHandler

```
function jogosultsagTreeDeleteHandler() {
     var selectedArray = Ext.getCmp('jogosultsagTree_id').
                                  getSelectionModel().getSelectedNodes();
     while (selectedArray.length) {
          if (!selectedArray[0].leaf) {
               selectedArray[0].unselect();
          } else {
               selectedArray[0].remove(true);
          }
     }
}
```

jogosultsagTreeBeforeNodeDropHandler

```
function jogosultsagTreeBeforeNodeDropHandler(e) {
    parentNode = e.target;
    targetPlace = parentNode.id.substr(0, 3);
    targetId = parentNode.id.substr(4);
    gridId = e.source.grid.id;
    e.cancel = true;

    if(Ext.isArray(e.data.selections)) {
        e.dropNode = [];
        var r;
        for(var i = 0; i < e.data.selections.length; ++i) {
            r = e.data.selections[i]; // kiválasztott rekordok kivétele

            if(gridId == 'jog_gridID') {
                elem_neve = r.get('visible_name');
            } else if(gridId == 'jog_usergridID') {
                elem_neve = r.get('v_name') + ' ' + r.get('k_name');
            }
            elem_id = targetPlace + '_' + r.get('id');
            if ( !parentNode.findChild('azon', elem_id) ) {
                e.cancel = false;
                e.dropNode.push(
                    this.loader.createNode({
                        azon: elem_id,
```

```
                        text: elem_neve,
                        leaf: true
                    })
                );
            }
        }
        return true;
    }
}
```

jogosultsagTreeNodeDragOverHandler

```
function jogosultsagTreeNodeDragOverHandler(e) {
    parentNode = e.target;
    targetPlace = parentNode.id.substr(0, 3);

    gridId = e.source.grid.id;

    if (
            ( e.point != 'append' )
        || ( parentNode.getDepth() != 2 )
        || ( gridId == 'jog_gridID' && targetPlace != 'jog' )
        || ( gridId == 'jog_usergridID' && targetPlace != 'alk' )
    ) {
        e.cancel = true;
        return false;
    }

    e.cancel = false;
    return true;
}
```

Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani mindenkinek, aki segítséget nyújtott számomra a diplomamunkám sikeres elkészítésében.

Mindenekelőtt szeretném megköszönni témavezetőmnek, Dr. Kuki Attila Egyetemi adjunktusnak a dolgozat készítése során tanúsított segítőkész hozzáállását és a munkám előrehaladását segítő építő jellegű észrevételeit.

Továbbá őszinte megbecsülésemet és szívből jövő hálámat szeretném tolmácsolni mindazoknak, akik támogatásukkal, segítségükkel, türelmükkel hozzájárultak ahhoz, hogy dolgozatomat a lehető legoptimálisabb körülmények között készíthessem el. Külön szeretném megköszönni a Kedvesem és Családom bíztatását és türelmét, valamint két barátomnak, Harsányi Gábornak és Papp Ferencnek azt, hogy mindig szívesen álltak rendelkezésemre és segítséget nyújtottak, ha valahol elakadtam, felhívták figyelmem a probléma okára és elmagyarázták annak javítási lehetőségét, ezáltal sok hasznos információra tettem szert.