D I P L O M A M U N K A

Madarassy Mónika Éva

Debrecen

2009

Debreceni Egyetem

Informatika Kar

Programozói versenyek hálózati ellenőrző és kiértékelő szoftverének fejlesztése

Témavezető:

Kósa Márk Szabolcs
egyetemi tanársegéd

Készítette:

Madarassy Mónika Éva
informatikatanár-matematika

Debrecen
2009

1. Tartalomjegyzék

2. Bevezető

Néhány éve versenyzőként és szervezőként is foglalkozom programozói versenyekkel. Rendszeresen részt vettem az ACM programozói verseny helyi fordulóin, illetve csapatommal az egyik évben sikerült továbbjutnunk a közép-európai döntőbe. Emellett a házi programozói versenyek szervezésében is segítettem. Így volt alkalmam tapasztalatokat gyűjteni, és eközben fogalmazódott meg bennem, hogy hasznos lenne egy olyan szoftver, mely interneten keresztül lehetővé teszi különböző programozói versenyek lebonyolítását.

Manapság a számítógépek nagymértékben elterjedtek, szinte minden háztartásban akad belőlük néhány. Az emberek egyre több idő töltenek a számítógépük előtt, és főleg a fiatalok egyre jobban érdeklődnek iránta, szívesen választanak olyan pályát, ahol számítógéppel kell foglalkozniuk. Ezek a fiatalok korán megismerkednek a programozással, megtanulnak valamilyen programozási nyelven programozni, és középiskolás, egyetemista korukban igénylik, hogy tudásukat összemérhessék. A programozói versenyek éppen ezt a célt szolgálják. Ahhoz, hogy egy ilyen versenyt minden résztvevő számára élvezhetően le lehessen bonyolítani megfelelő szoftverre van szükség.

Még ma is vannak olyan programozói versenyek, melyek során a versenyzők egy adott mappába mentik az általuk megírt forráskódokat, majd a verseny végeztével a szervezők ezeket a gépeket végigjárva valamilyen adathordozóra másolják az előbbi mappákat és otthon kiértékelik. Ebben az esetben a versenyzők az általuk megírt program helyességéről csak napokkal később kapnak visszajelzést.

Az általam elkészített szoftver ezt a problémát hívatott áthidalni. A szoftver segítségével a versenyzők a verseny időtartama alatt interneten keresztül beküldhetik a forráskódokat, amely a szerver gépen lefordításra kerül, majd a megadott tesztesetekre lefut, és azonnal megállapítja az eredményt. Így a versenyzők a beküldés után pár perccel már információt kapnak a forráskódjuk helyességéről. Ez az információ nagyban könnyítheti a versenyzők munkáját, hiszen azonnal tudni fogják, hogy tökéletes-e az általuk készített megoldás vagy még javítaniuk kell rajta.

A szoftver lehetőséget biztosít arra is, hogy a beküldött forráskódok kiértékelése emberi felügyelet mellett történjen, tehát egy úgynevezett versenybíró végezze el. De az is lehetséges, hogy automatikus kiértékelést állítsunk be a verseny egészére, vagy egy idejére. Ez azt

jelenti, hogy nem igényli, hogy egy versenyszervező a verseny teljes ideje alatt a számítógép előtt üljön.

Mindezek után úgy vélem, hogy egy ilyen szoftver hasznos, és mind a versenyzők, mind a szervezők számára megkönnyíti a versenyt. Az általam elkészített szoftverhez hasonlóak már léteznek, de azok általában ACM jellegű versenyeken használhatók. Ez azt jelenti, hogy ezek a szoftverek a versenyzők pontszámát egyféleképpen tudják meghatározni, mégpedig a beküldési idő figyelembe vételével. Az általam készített alkalmazás más pontozási módra is lehetőséget biztosít, és a versenyt adminisztráló személy dönti el, hogy melyiket választja. Így ez az alkalmazás tágabb területen, más jellegű versenyeken is alkalmazható.

3. Programozói versenyekről

Hazánkban többféle programozói versenyt szoktak meghirdetni. A legkisebb kategória, amikor egy oktatási intézmény a saját tanulói számára hirdet versenyt. Ilyen például a Debreceni Egyetemen évente megrendezésre kerülő házi programozói verseny, melynek célja a hallgatók programozás iránti érdeklődésének felkeltése.

Vannak nagyobb, regionális illetve országos szintű versenyek is, ahol több oktatási intézmény tanulói, esetleg több fordulóban méri össze tudásukat. Középiskolások között a legismertebb ilyen verseny az országos szintű Nemes Tihamér programozói verseny. A verseny 1985 óta kerül megrendezésre. Az eltelt évek alatt ez a verseny sokban változott, és mára már három fordulóból álló, három korcsoport számára meghirdetett versenyként vált ismertté.

A három korcsoport 5.-8. osztályos, 9.-10. osztályos, 11.-13. osztályos tanulókat jelent. A különböző korosztályok számára különböző típusú és nehézségű feladatok kerülnek kitűzésre. A típusok között például szerepel szövegszerkesztés, táblázatkezelés, adatbázis kezelés, programozás is. A III. korcsoport egyben az OKTV programozás kategóriája is. A programozás kategóriában elvárt programozási nyelv a Pascal nyelv.

A verseny iskolai fordulóját minden jelentkező iskola saját tantermében rendezheti meg, de több iskola közösen is megrendezheti. A regionális fordulót az erre vállalkozó oktatási intézmények rendezik meg a saját körzetükhöz tartozó iskolák diákjai számára. Az országos forduló Budapesten kerül megrendezésre az Eötvös Loránd Tudományegyetem számítógéptermeiben.

Sajnos ez a verseny mind a mai napig úgy zajlik, hogy a versenyzők az általuk használt számítógép egy megadott mappájába dolgoznak, és a verseny végén az illetékes szervező minden gépet végigjárva valamilyen adathordozóra másolja a fájlokat, amelyeket eljuttatnak az illetékes javítóknak, és a későbbiek során egyesével kiértékelésre kerülnek. Ez szerintem mind a versenyzők, mind a verseny szervezői számára nehézkes és időigényes. A versenyzők az eredményről általában hetekkel később értesülnek.

Az ACM (Association for Computing Machinery) az egyik legrangosabb informatikai társaság, minden évben meghirdeti az ACM nemzetközi programozási versenyt. A versenyen háromfős csapatok vehetnek részt. A verseny során 5 óra alatt csapatonként egyetlen számítógép segítségével körülbelül 6-8 feladatot kell megoldaniuk. A verseny győztese az a csapat, amely a rendelkezésre álló idő alatt a legtöbb feladatot oldja meg. Holtverseny esetén a részidők döntenek. A feladatok egyrészt alapos algoritmuselméleti ismereteket, problémamegoldó készséget, másrészt gyors és hibamentes programozást igényelnek.

Ezen a versenyen évek óta magyar tanulók is részt vesznek. A verseny szintén több fordulós. Lehetséges helyi forduló megrendezése is, amelyet az oktatási intézmény a saját hallgatói között hirdet meg. Ezt a Debreceni Egyetem minden évben meg is teszi, ezzel is lehetőséget adva a gyakorlásra az érdeklődő hallgatók számára. Ez az a forduló, ahol az oktatási intézménynek magának kell gondoskodni a verseny lebonyolításához szükséges tetszőleges szoftverről. Az országos forduló esetén a szervező egyetem gondoskodik a szoftverről.

4. Használt eszközök

Egy ilyen szoftver megíráshoz szükségem volt valamilyen adatbázis kezelőre, valamint egy olyan fejlesztői környezetre, ahonnan az adatbázis adatait elérhetem, módosíthatom, és létrehozhatok egy webes felületet a felhasználók számára.

A szoftvert Windows operációs rendszer alatt fejlesztettem, és Microsoft technológiákat használtam. A forráskódot Microsoft Visual Studio 2008-ban készítettem C# nyelven. Az adatbázis elérésére a Visual Studioba beépített LINQ-t, pontosabban a LINQ to SQL-t használtam. Az adatbázis elkészítéséhez pedig a Microsoft SQL Server Management Studio Express 2005 (SSMSE) szoftvert használtam, ami egy ingyenesen letölthető relációs adatbázis kezelő szoftver, amellyel a Visual Studio hatékonyan tud együttműködni. Az elkészített web-

lap nyilvánossá tételéhez használt web szerver pedig a Windows operációs rendszerbe beépített Internet Information Services (IIS) Manager volt, de más web szerverrel is megoldható.

Szükséges még valamilyen böngésző, amely megjeleníti a felhasználói felületet. A web böngészők a web szerverekkel http protokollon keresztül kommunikálnak. A http segítségével a böngészők adatokat küldhetnek a web szervernek, valamint weblapokat tölthetnek le róluk. Többféle grafikus böngésző létezik, de én a leggyakrabban használt böngészőkkel (Microsoft Internet Explorer, Mozilla Firefox) végeztem a fejlesztést.

5. Szoftver specifikáció

Ebben a részben a szoftver működését és a működésre vonatkozó megszorításokat definiálom. Meghatározom az elkészítendő szoftver főbb funkcióit és működési momentumait.

Ebben a szakaszban fontos mindent jól átgondolni, hiszen minél pontosabban fogalmazzuk meg a szoftverrel szemben támasztott követelményeket, annál könnyebb megalkotni a megfelelő szoftvert. Ezek után a tesztelés, kipróbálás során előfordulhat olyan, hogy további fejlesztésre lesz szükség.

A szoftverrel szemben jelen esetben három felhasználói csoport támaszthat igényeket. Ilyen felhasználói csoportok a versenyzők, a bíró és az adminisztrátorok. Mindhárom csoport szemszögéből meg kell vizsgálni a követelményeket, majd összegezni, rendszerezni.

5.1. Versenyzők szemszögéből

Egy programozói versenyen a versenyzőknek általában az a fontos, hogy az általuk készített forráskódokat egyszerűen el tudják juttatni a bírókhoz, és minél hamarabb értesüljenek az eredményről. Tehát a szoftvernek rendelkeznie kell egy olyan könnyen kezelhető felülettel, amelyen a versenyző beküldheti a forráskódjait. Hasznos a felület egy jól észrevehető részén azonnali jelzést, figyelmeztetést feltüntetni a versenyző számára, amikor a feladatát kiértékelték.

Verseny közben fontos nyomon követni a még hátralévő időt, ezért szükség van egy visszaszámlálóra, ami automatikusan frissül, és a versenyből még hátralévő időt mutatja. Célszerű ezt is könnyen észrevehető helyre elhelyezni.

A versenyzők és a bírók általában nem ugyanazon a helyen tartózkodnak (például másik teremben, másik épületben), ezért lehetőséget kell biztosítani valamiféle kommunikációra.

Erre talán a legegyszerűbb megoldás egy beépített levelezőrendszer megalkotása. A levelezőrendszer kivitelezése közben szem előtt kell tartani, hogy a versenyzők csak a bírákkal tudjanak kommunikálni, egymással ne. Illetve a bírák számára lehetőséget kell adni körlevél küldésére is, hiszen a verseny közben számos olyan dolog merülhet fel, amelyről minden versenyzőt értesíteni kell. A levelezés akkor működik hatékonyan, ha a címzett hamar észreveszi, hogy üzenetet kapott. Ennek érdekében az érkező levelekről célszerű figyelmeztető jelzést feltüntetni a webes felület valamely szembetűnő részén.

Ezek mellett a versenyzőknek lehet olyan igénye, hogy a megoldandó feladatsort ne csak papíron olvashassák, hanem a felületről is letölthető legyen.

5.2. Bírák szemszögéből

A bírák feladata a beérkező forráskódok kiértékelése és a versenyzők kérdéseire válaszok küldése. Válaszok küldését az előbb leírt levelezőrendszer teszi lehetővé. A forráskódok kiértékelésénél az első fontos dolog, hogy a bíró észrevegye, hogy új feladatot küldtek be. Ennek a megoldása a már említett figyelmeztetések feltüntetésével megoldható. Fontos, hogy a feladatok kiértékelése automatizálható és manuálisan megvalósítható is legyen. Előbbire akkor van szükség, ha a bíró valami miatt nem tud számítógép közelben lenni, utóbbira pedig akkor, ha a bíró nyomon szeretné követni a kiértékelés folyamatát. A bíró számára lehetőséget kell biztosítani a kiértékelő rendszer által megállapított eredmény felüldefiniálására, megváltoztatására.

Mivel a kiértékelés során adódhatnak előre nem látott hibák, ezért a bírák számára meg kell engedni, hogy bármikor meg tudják nézni a versenyzők által beküldött kódokat, a kódokból készített programok kimeneteit az egyes tesztesetekre. Ez nem csak hibák esetén hasznos, hanem akkor is, ha valamelyik versenyző nem ért egyet az eredménnyel, szeretné megnézni a hiba okát.

5.3. Adminisztrátorok szemszögéből

Az adminisztrátor felelős a verseny megfelelő működéséért. Első lépésként tehát rendelkeznie kell megfelelő jogosultsággal, hogy egy új versenyt tudjon létrehozni, illetve már meglévő verseny beállításait tudja módosítani. Módosítható beállításnak kell lennie a verseny kezdési és befejezési idejének, a verseny típusának, illetve a büntetőpontnak. Kényelmi szem-

pont, hogy a verseny kezdési és befejezési ideje percre pontosan beállítható legyen már előre. Ez azért hasznos dolog, mert így a versenyzőkkel közölhetjük, hogy pontosan mikor kezdődik a verseny, de az adminisztrátornak abban az időpontban nem szükséges számítógép előtt ülnie, a verseny magától is elindul. A befejezési idő előre beállítása hasonló okokból előnyös. Előfordulhat olyan, hogy a versenyt valamilyen okból kifolyólag le kell állítani, szüneteltetni kell. Ezt is mind lehetővé kell tenni, illetve ilyen esetben a verseny folytatására is szükség lehet.

A versenyhez felhasználók is szükségesek, így az adminisztrátor feladata új felhasználók generálása, a felhasználók versenyhez kapcsolása, esetleg a jelszó és a felhasználói név átállítása. További feladat, amely szintén az adminisztrátor hatáskörébe tartozik a használható programozási nyelvek megadása, és a beérkezett forráskódokra adható válaszok meghatározása.

Ezeken kívül a feladatokról is gondoskodni kell. Az adminisztrátor számára engedélyezni kell új feladatokat létrehozását, a már létrehozott feladatokhoz tesztesetek kapcsolását, a feladatok és a tesztesetek szerkesztését. Mivel több versenyen is szerepelhet ugyanaz a feladat, ezért az adminisztrátornak azt is meg kell határoznia, hogy az egyes feladatok mely versenyekhez kapcsolódnak.

Mindezek mellett szükség lehet a törlésre is. Ez alatt azt értem, hogy ha az adminisztrátornak megengedjük verseny létrehozását, akkor a verseny törlését is lehetővé kell tennünk számára. Hasonlóan felhasználó, illetve feladat, teszteset törlésére is lehetőséget kell biztosítani.

5.4. Közös igények

Minden felhasználó számára elérhetővé kell tenni az eredményjelző táblát, amely a verseny jelenlegi állását mutatja.

Fontosak a biztonsági szempontok, jelszavak. Mivel a rendszer nem regisztráció útján működik, hanem az adminisztrátor hozhat létre új felhasználót, ezért minden felhasználónak meg kell engedni a jelszóváltoztatást, illetve egyes esetekben a felhasználónév váltását is. A versenyzők felhasználónév cseréje verseny közben zavaró lehet, hiszen ilyenkor az eredményjelző táblán is megváltozna a nevük, és túl sok változás már nehezen követhető a többi felhasználó számára, éppen ezért a felhasználónév megváltoztatását korlátozni kell.

6. Általános felépítés

6.1. Adatbázis tervezése

6.1.1. Microsoft SQL Server Management Studio Express 2005

A Microsoft SQL Server Management Studio Express (SSMSE) egy ingyenesen letölthető szoftver, amely könnyen használható grafikus eszközkészletet biztosít az SQL Server 2005 Express Edition és az SQL Server 2005 Express Edition with Advanced Services számára. Az SSMSE az SQL Server 2005 bármely más verziójában készített relációs adatbázis kezelésére is képes. Lehetővé teszi SQL Server 2005-beli adatbázisok, tárolt eljárások létrehozását, lekérdezések és nézetek létrehozását és futatását, adatbázisról backup fájl készítését, valamint adatbázis backup fájlból való helyreállítását, létrehozását.

6.2. Adatbázis felépítése

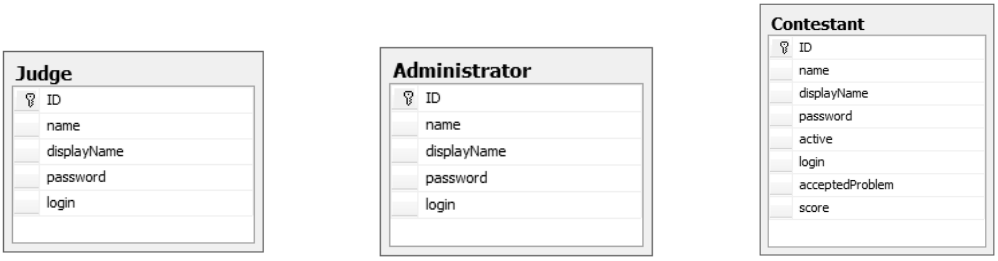
Első lépésként meg kellett határoznom, hogy milyen adatok adatbázisbeli tárolására lesz szükség, ezek az adatok hogyan fognak kapcsolódni. Az összegyűjtött információk táblákra bontásánál igyekezetem elkerülni a redundáns adattárolást, a jól strukturáltságra, a későbbi adatbázis műveletek megkönnyítésére törekedtem.

Minden tábla esetében az elsődleges kulcsot egy számláló valósítja meg, ez minden esetben az ID mező. A táblák nagy része valamilyen módon kapcsolódik egymáshoz, így a kapcsolótáblák szerepe jelentős. Néhány kivételtől eltekintve a kapcsolótáblákban csak a megfelelő táblák elsődleges kulcsait tárolom.

Mivel a program egyszerre több verseny párhuzamos lebonyolítását teszi lehetővé, ezért az adatbázis szíve a *Contest* tábla, melyben az adott versenyhez szükséges legfontosabb adatok kerülnek tárolásra, valamint az összes többi tábla ezen tábla köré épül. A verseny szempontjából fontos tudnunk a kezdés és a befejezés idejét, a verseny típusát, a zsűrizés módját, és egy nevet, amellyel a versenyt hivatkozhatjuk. A verseny típusa lehet acm jellegű, vagy pontozáson alapuló. Acm jellegű verseny esetén megadhatjuk a rossz megoldásokért járó büntetőpontot is, ami szintén ebben a táblában kerül tárolásra. A zsűrizés történhet manuálisan a bírók által, vagy automatikus zsűrizéssel.

1. ábra - Contest tábla

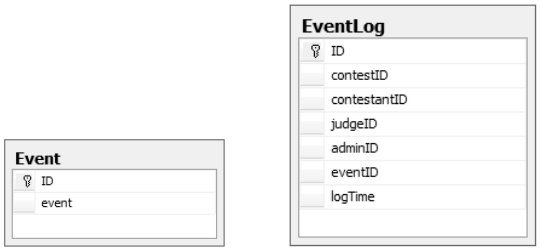
A verseny szempontjából háromféle felhasználót különíthetünk el, amelyeknek más-más szerepük van. Az adatbázisban minden felhasználótípust egy-egy tábla reprezentál: *Administrator*, *Judge*, *Contestant* táblák. A felhasználók egy vagy esetleg több versenyhez kapcsolódhatnak, ezért mindhárom tábla kapcsolatban áll a *Contest* táblával kapcsolótáblák segítségével. A felhasználókról minden esetben ismernünk kell a nevet, a felhasználói nevet, a jelszó. A nevet automatikus generálom, ezzel garantálva egyediségét. A felhasználói nevet a felhasználó megváltoztathatja, de az egyediségre itt is figyelni kell. A jelszó a felhasználó által szintén szabadon változtatható. A felhasználói nevet és a jelszót az adott versenyhez tartozó adminisztrátor is megváltoztathatja. A táblában még szerepel egy bit típusú mező is, ami azt tárolja, hogy a felhasználó jelenleg be van-e jelentkezve. A *Contestant* táblában ezeken kívül még néhány fontos dolgot tárolnunk kell. Ilyen az, hogy az adott versenyzőnek hány pontja van, és hány teljesen hibátlan megoldást küldött be. Az `active` mező bit típusú, és azt tárolja, hogy a versenyző aktív versenyző-e, azaz az eredményjelzőn megjelenjen-e. Erre azért lehet szükség, mert így az adminisztrátor létre tud hozni fiktív versenyzőket, amelyek segítségével futás közben is ellenőrizhető, hogy jól működik-e a rendszer.



2. ábra - A felhasználókat tároló táblák

A verseny szempontjából fontos, hogy aktuálisan ki van bejelentkezve, illetve az is fontos lehet, hogy ki mikor jelentkezett be és ki. Ehhez egy külön táblában (*Event*) tárolom a fontosabb eseményeket: bejelentkezés, kijelentkezés. Egy kapcsolótábla (*EventLog*) köti össze az eseményeket a felhasználókkal és a versennyel. Bár az *EventLog* táblának nem feltétlenül szükséges kapcsolódnia a versenyhez, hiszen a felhasználók már a versenyhez vannak

kötve. Ez a kapcsolat mégis fontos, mert könnyebbé teszi az események versenyenkénti lekérdezését.



3. ábra - Az Event és az EventLog táblák

A felhasználók és az események kapcsolata a versennyel:

4. ábra - A táblák kapcsolata

Az adatbázis másik nagyobb egységét a versenyfeladatokhoz, és a zsűrizéshez kapcsolódó táblák adják. Minden verseny esetében beállíthatjuk, hogy milyen nyelvű forráskódok beküldése lehetséges, illetve, hogy a beküldött feladat milyen minősítést kaphat. A *Language* táblában a programozási nyelvekről tárolunk információkat. Egy-egy programozási nyelvnél fontos a programnyelv neve, illetve hogy milyen kiterjesztésű fájlt lefordítására alkalmas. Emellett fontos, hogy a szerver gépen hol található a fordító, milyen parancs szolgál a fordításra, és mely könyvtárba dolgozhat. A fordítóparancsnál adhatjuk meg a különböző fordítási opciókat is. Ezeket az adatokat az adminisztrátor tetszőlegesen megváltoztathatja, de a változás minden verseny esetén megtörténik.

A válaszok az *Answer* táblában találhatók. Itt az egyedüli fontos információ a válasz leírása, azaz a megfelelő hibaüzenet. Ezek megváltoztatásához senkinek nincs jogosultsága.

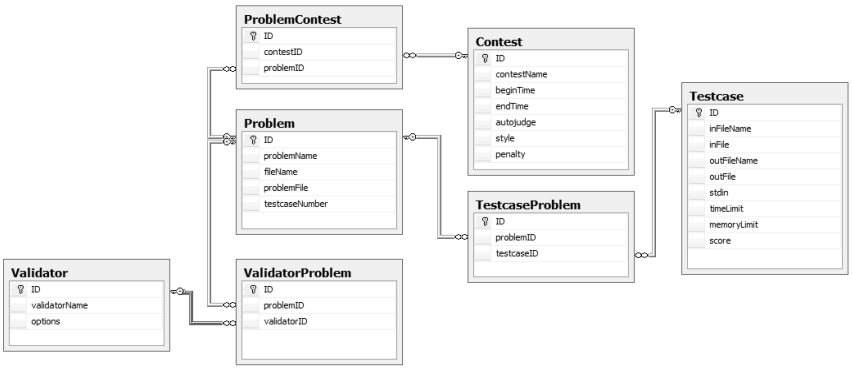
Az adminisztrátor csak annyit tehet, hogy az adott versenyhez kapcsolja a megfelelő válaszokat, tehát csak a kapcsolótábla módosítható. A lehetséges programozási nyelvek és válaszok az adatbázis inicializálásakor kerülnek az adatbázisba, az adminisztrátornak csak ahhoz van joga, hogy az egyes versenyhez kapcsolja azokat. Természetesen egy-egy programozási nyelv, és válasz egyszerre több versenyen is megengedett.

5. ábra - A Language és az Answer táblák

Mivel az adatbázis arra szolgál, hogy programozási versenyek adatait tárolja, így szükség van olyan táblára, amelybe a megoldandó problémák kerülnek. Ez a *Problem* tábla. Az egyes problémákat a versenyekhez kapcsolhatjuk. Megengedett, hogy egy probléma több versenyen is szerepeljen. Emiatt a *Problem* és a *Contest* táblákat össze kell kapcsolni egy kapcsolótábla segítségével. Egy-egy probléma esetén fontos egy név, amellyel a problémára hivatkozhatunk, illetve minden problémához megengedett egy fájl tárolása, amely a probléma leírását tartalmazza. Az adatbázisban maga a fájl, és a fájl neve is tárolódik. Egy problémához több tesztesetet is rendelhetünk, ezért az egyes problémákhoz kapcsolódó tesztesetek száma is bekerül a táblába.

A tesztesetek tárolása külön táblában, a *Testcase* táblában történik. Itt már olyan információkra is szükségünk van, amelyek a beküldött forráskódok zsűrizéséhez elengedhetetlenek. Ilyen információ, hogy a beküldött forráskódból készült program az input adatot a szabványos bemenetről várja, vagy fájlból olvassa. Ennek tárolására egy bit típusú mező, az `stdin` szolgál. Minden teszteset esetén tárolnunk kell az input és az output fájlt, valamint ezek nevét. A *Testcase* táblában fordításhoz kapcsolódó információk is szerepelnek, ilyen a megengedett idő- és memóriakorlát. Mivel pontozáson alapuló versenyt is lebonyolíthatunk, így szükségünk van egy olyan mezőre, amely az egyes tesztesetekért járó pontot tartalmazza. A *Problem* és a *Testcase* táblák egy kapcsolótáblával vannak összekötve.

Miután a versenyzők által beküldött forráskódot lefordítottuk, és az adott tesztesetre lefuttattuk, szükség van az így előállt kimenet, és a példakimenet összehasonlítására. Ezt egy külső program végzi, melyről később lesz szó. Az összehasonlításkor ennek a külső programnak különböző opciókat adhatunk meg. A lehetséges opciókat az adatbázis *Validator* táblája tartalmazza. Az adminisztrátor minden probléma esetén beállíthatja, hogy milyen opcióval történjen a validálás. A *Validator* táblában csak az opciók, és a felhasználó számára érthető rövid leírás kerül tárolásra. A *Validator* tábla szintén egy kapcsolótáblával kötődik a *Problem* táblához.



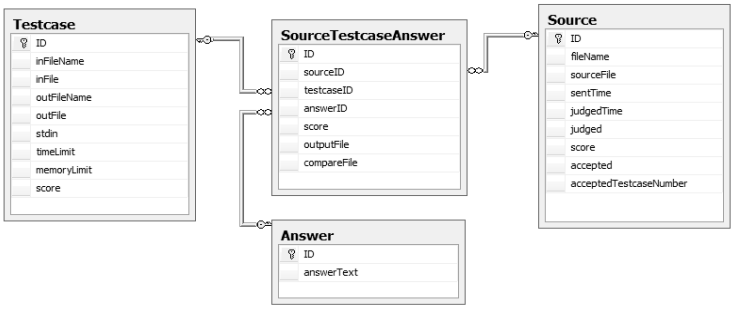
6. ábra - Problem tábla és kapcsolatai

A versenyzők által beküldött forráskódok tárolására a *Source* tábla szolgál. A beküldött fájl mellett a fájl neve és a beküldési idő is bekerül az adatbázisba. A táblában arról is tárolok információt, hogy a beküldött forráskódot kiértékelték-e, és ha igen, akkor ez mikor történt. Ha már ki lett értékelve, akkor az `accepted` mező igaz illetve hamis értékéből kiolvashatjuk, hogy a beküldött forráskód tökéletesen megfelelt-e vagy sem. Attól, hogy egy megoldás nem volt tökéletes, még lehet olyan teszteset, amelyen sikeres volt. Az `acceptedTestcaseNumber` mondja meg, hogy az adott megoldás, hány tesztesetre lett elfogadva. Ezek alapján lesz kiszámolva a `score` mezőbe az adott megoldás pontértéke.

7. ábra - Source tábla

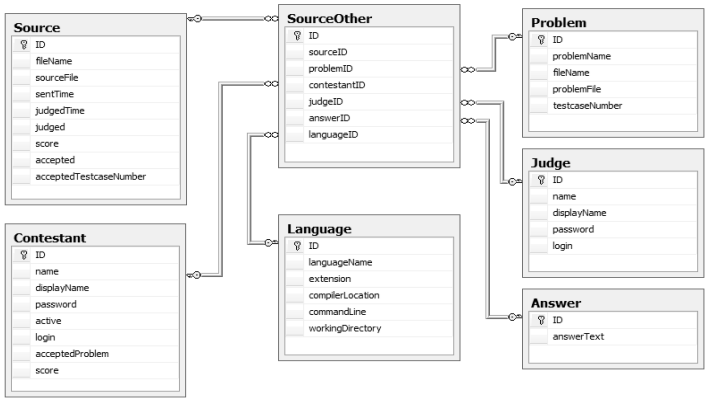
Egyes problémák esetében több teszteset is előfordulhat, ezért szükség van arra, hogy minden teszteset esetén ismerjük a beküldött forráskódból készült program kimenetét, és

eredményét az adott tesztesetre. Erre a *SourceTestcaseAnswer* tábla szolgál, amely a *Source*, a *Testcase* és az *Answer* táblákat köti össze néhány plusz információval kiegészítve az összekapcsolást. Ilyen információ az adott program tesztesetre adott kimenetét tartalmazó fájl, és egy olyan fájl, amelyben ez a kimenet és a példakimenet összehasonlítása szerepel. Ezen fájlok adatbázisbeli tárolása azért fontos, hogy később a verseny adminisztrátorai, és bírái vissza tudják keresni, ellenőrizni tudják. A táblában még szerepel a tesztesetért kapott pont, és a válasz azonosítója, ami az *Answer* táblához kapcsolódik.



8. ábra - SourceTestcaseAnswer

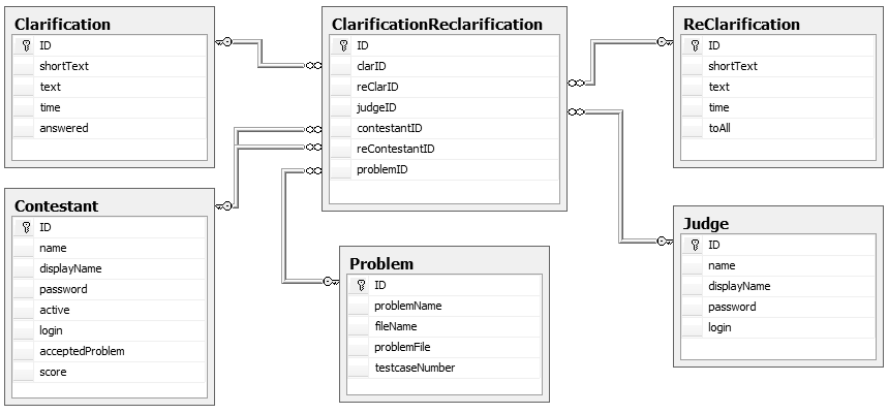
A fentebb már leírt *Source* táblát sok más táblához hozzá kell kapcsolni, erre szolgál a *SourceOther* tábla, mely az egyik legtöbb táblát összekapcsoló kapcsolótábla. Az általa összekapcsolt táblák a *Source*, *Contestant* és a *Judge* táblák, hogy az adott forráskódot mely versenyző küldte be, és melyik bíró zsűrizte. A *Problem*, a *Language* táblák, hogy a forráskódot mely problémára, milyen programozási nyelven küldték be. Illetve az *Answer* tábla, amely a beküldött forráskódra adott végső, összesített választ tartalmazza. Az összesített válasz az első olyan válasz, amely hibaüzenetet tartalmaz. Ha ilyen nincs, akkor a „Yes -–Accepted” válasz.



9. ábra - SourceOther tábla és kapcsolatai

A versenyzők nem csak forráskódokat tudnak beküldeni, hanem egyfajta levelezőrendszeren keresztül a bírókkal is tudnak kommunikálni. A versenyzők által írt kérdések, észrevételek az adatbázis *Clarification* táblájába, míg a bírók válaszai a *ReClarification* táblába kerülnek. Mindkét táblában szerepel az üzenet teljes szövege, valamint egy rövid szöveg, az üzenet első pár karaktere. Ez azért fontos, mert a lekérdezések során sokszor nincs szükség a teljes szövegre, elég csak pár karakter, amely címként funkcionál. Ezen felül a *Clarification* tábla egy `answered` mezővel egészül ki, mely azt tartalmazza, hogy érkezett-e már válasz. A *ReClarification* táblában pedig egy `toAll` mező tárolja, hogy a válasz minden versenyző számára megjelenítésre kerül, vagy csak a kérdést küldő versenyző olvashatja.

A versenyzői kérdéseket és bírói válaszokat össze kell kapcsolnunk, hiszen tudnunk kell, hogy mely kérdéshez, mely válasz tartozik. Ezt a funkciót a *ClarificationReclarification* tábla látja el. Ezen felül ebben a táblában található még külső kulcs a *Problem*, a *Judge* és a *Contestant* táblára is. A *Problem* táblára azért van szükség, hogy meg tudjuk mondani, hogy a kérdés, észrevétel mely probléma kapcsán merült fel a versenyzőkben. A *Judge* táblával való kapcsolat arra szolgál, hogy egyértelműen megállapítható legyen, mely bíró adta a választ. A *Contestant* táblával kétszeres kapcsolat áll fenn, mely versenyző küldte a kérdést, és mely versenyző kapja a választ. Ez utóbbi vagy megegyezik a küldővel, vagy a válasz mindenkinek szól, és ilyenkor egy speciális érték, az „all” nevű versenyző azonosítója lesz. Az „all” nevű versenyző az adatbázis inicializálásakor kerül az adatbázisba és minden versenyhez passzív versenyzőként kapcsolódik, azaz sem az eredményjelzőn, sem máshol nem jelenik meg.



10. ábra - A levelezőrendszer táblái

6.3. Adatbázis inicializálása

Az adatbázis másik számítógépre történő átvitele történhet egy backup fájl segítségével, vagy sql script lefuttatásával. Az adatbázis újonnan történő létrehozásakor egyes táblákba bizonyos rekordokat kötelező elhelyezni. Ilyen rekordok a következők:

Administrator táblába egy adminisztrátor létrehozása, hogy üres adatbázis után legyen valaki, aki be tud lépni a rendszerbe. Ennek az adminisztrátornak „root” lesz a neve.

```
insert into Administrator (name, displayName, password, login) values ('root', 'root', 'root',
'false')
```

Contestant táblába a korábban már említett „all” nevű passzív felhasználó létrehozása, hogy a bírók által küldött körüzenetek tárolása minél könnyebben, egyszerűbben történjen.

```
insert into Contestant (name, displayName, password, login, active, acceptedProblem, score) values
('all', 'all', 'all', 'false', 'false', '0', '0')
```

Ha már vannak felhasználók, szükség van egy versenyre is, hiszen a felhasználók a versenyhez kapcsolódnak, csak akkor tudnak felhasználóként belépni, ha van olyan verseny, amihez kötődnek. Ez a verseny szintén az inicializáláskor kerül az adatbázisba „MainContest” néven. A verseny paraméterei később szabadon megváltoztathatók, vagy akár az egész versenyt figyelmen kívül hagyhatjuk.

```
insert into Contest (contestName, beginTime, endTime, autojudge, style, penalty) values
('MainContest', '2000-01-01 00:00:00.000', '2000-01-01 00:00:00.000', 'false', 'acm', '20')
```

Az *Event* táblába a két lehetséges esemény „Login”, „Logout” szintén ilyenkor kerül.

```
insert into Event (event) values ('Login')
insert into Event (event) values ('Logout')
```

A *Validator* és az *Answer* tábla is az inicializáláskor töltődik fel. Tartalmuk később nem változtatható.

```
insert into Validator (validatorName, options) values ('diff', '')
insert into Validator (validatorName, options) values ('ignore all whitespaces', '--ignore-all-
space')
insert into Validator (validatorName, options) values ('ignore empty lines', '--ignore-blank-
lines')
insert into Validator (validatorName, options) values ('ignore upper/lower case', '--ignore-case')

insert into Answer (answerText) values ('Running...')
insert into Answer (answerText) values ('Yes - Accepted')
insert into Answer (answerText) values ('No - Wrong answer')
insert into Answer (answerText) values ('No - Runtime Error')
insert into Answer (answerText) values ('No - Compile Error')
insert into Answer (answerText) values ('No - Other unknown error')
insert into Answer (answerText) values ('No - Time limit exceeded')
insert into Answer (answerText) values ('No - Memory limit exceeded')
```

A *Language* táblába is bekerülnek a lehetséges programozási nyelvek, amelyek opcionális adatai megváltoztathatók, de újabb programozási nyelv nem adható az adatbázishoz.

```
insert into Language (languageName, extension, compilerLocation, commandLine, workingDirectory)
values ('GNU C', 'c', 'C:\MinGW\bin\', 'gcc -lm {:mainfile} -o {:basename}', 'D:\Suli\Working\')
```

```
insert into Language (languageName, extension, compilerLocation, commandLine, workingDirectory)
values ('GNU C++', 'cpp', 'C:\MinGW\bin\', 'g++ -lm {:mainfile} -o {:basename}',
'D:\Suli\Working\')
insert into Language (languageName, extension, compilerLocation, commandLine, workingDirectory)
values ('Pascal', 'pas', '', '', 'D:\Suli\Working\')
insert into Language (languageName, extension, compilerLocation, commandLine, workingDirectory)
values ('Java', 'java', 'C:\Program Files (x86)\Java\jdk1.6.0_06\bin\', 'javac {:mainfile}',
'D:\Suli\Working\')
```

A *Problem* táblába két rekord kerül. Az egyik a „General”, amelyre azért van szükség, ha a versenyzők nem konkrét feladathoz kapcsolódóan szeretnének kérdezni, észrevételt tenni, hanem valamilyen általános kérdésük van. A „Deleted” rekord pedig arra szolgál, ha egy probléma törlésre kerül, akkor a hozzá kapcsolódó levelezést, és beküldött forráskódokat ne kelljen törölni, mert a későbbiekben esetleg szükség lesz rá.

```
insert into Problem (problemName, testcaseNumber) values ('General', '0')
insert into Problem (problemName, testcaseNumber) values ('Deleted', '0')
```

A *ReClarification* táblába egy „NEW” rekord kerül, amely jelzi, hogy a hozzá kapcsolt *Clarification* táblabeli rekordra még nem érkezett válasz.

```
insert into ReClarification (shortText, text, time, toAll) VALUES ('NEW', 'NEW', '2000-01-01
00:00:00.000', 'false')
```

Ezeken kívül a megfelelő kapcsolótáblákba is bekerülnek a szükséges rekordok.

7. Az adatbázis elérése LINQ segítségével

7.1. LINQ

A fejlesztői világot teljesen áthatja az objektumorientált gondolkodás és tervezés. Ennek ellenére sajnálatos módon még ma is komoly kihívást jelentenek a nem objektumorientált adatforrásokkal való műveletek, mint például XML dokumentumok vagy relációs adatbázisok kezelése. Ezt a problémát igyekszik megoldani és elfedni a LINQ (Language Integrated Query)-technológia.

A LINQ definiálja saját lekérdező operátorainak halmazát, amelyek segítségével lekérdezéseket, projekciókat, szűréseket hajthatunk végre többek között relációs adatbázisokon is. Ahhoz, hogy a lekérdezések bármely adatforráson végrehajthatóak legyenek, az adatokat objektumokká kell „alakítani”, vagyis ha az adatainkat eredetileg nem objektumokként tároltuk, akkor le kell képezni őket azokká. A leképezés során az SQL adatbázis tábláinak és oszlopainak a C#-beli osztályok és tulajdonságok felelnek meg.

A LINQ-nak három fajtája létezik: LINQ to Objects, LINQ to XML (XLINQ) és LINQ to SQL (DLINQ). Én az utóbbit használtam. A LINQ to SQL lehetővé teszi SQL adatbázisokban tárolt adatok elérését, lekérdezését.

7.2. Adatbázis elérése

A Visual Studioban a LINQ to SQL számára egy beépített Object Relation (O/R) Designer található, amely egy grafikus felületet biztosít SQL adatbázisok objektumorientált leképezéséhez. A Designer egy erősen típusos DataContext osztályt generál. A Server Explorer-ben láthatjuk az SQL adatbázisunkat, és onnan bármely táblát áthúzva az O/R Designer-re az előbbi DataContext osztályunkban ez a tábla, mint tulajdonság jelenik meg, valamint minden táblához automatikusan legenerálódik egy parciális osztály a megfelelő attribútumokkal. Ezt nevezzük entity-nek. Bármely entity funkcionalitását kibővíthetjük újabb parciális osztályok létrehozásával.

Első lépésként létrehoztam a saját DataContext osztályomat *CONTESTDataContext* néven, amelybe az adatbázisom összes tábláját, és nézetét áthúztam, azaz legeneráltattam a hozzájuk tartozó parciális osztályokat. Majd minden táblához és nézethez megírtam egy újabb osztályt. Az osztályok nevei annyiban különböznek a megfelelő táblák/nézetek neveitől, hogy egy *D* betűre végződnek. Ezek az osztályok tartalmazzák az olyan fontosabb metódusokat, amelyek az adatbázisbeli adatokon hajtanak végre lekérdezéseket, illetve módosítják az adatbázis tartalmát.

Minden metódus kap egy *CONTESTDataContext* típusú objektumot paraméterül, amely az adatbázissal való kapcsolatot biztosítja. Ennek akkor van szerepe, ha az adott metódust egy tranzakción belül hívom meg, hiszen a DataContext-nek nem szabad megváltoznia a tranzakción belül. Ha viszont nem tranzakció részeként hívom a metódust, akkor ennek a paraméternek `null` értéket adok át. Így a metódusaim először a paraméterül kapott *CONTESTDataContext* értékét vizsgálják, és `null` érték esetén példányosítják.

Minden osztályban szerepelnek a következő metódusok: deleteTáblaNévByID, updateTáblaNévByID, createTáblaNév. Az első metódus a paraméterül kapott ID-val rendelkező rekordot törli a megfelelő táblából. Az update metódus a paraméterül kapott ID-val rendelkező rekordot módosítja a szintén paraméterül kapott objektum alapján. A create metódus pedig létrehoz egy új rekordot a kapott paraméterből. Például az *AnswerD* osztályban szereplő megfelelő metódusok:

```
public static void deleteAnswerByID(int ID, CONTESTDataContext dc)
{
    if (dc == null)
        dc = new CONTESTDataContext();
    try
```

```
        {
            Answer answer = dc.Answers.Single(row => row.ID == ID);
            dc.Answers.DeleteOnSubmit(answer);
            dc.SubmitChanges();
        }
        catch (Exception)
        {
            throw new Exception("Delete error!");
        }
    }

    public static void updateAnswerById(int ID, Answer answer, CONTESTDataContext dc)
    {
        if (dc == null)
            dc = new CONTESTDataContext();
        try
        {
            Answer asw = dc.Answers.Single(row => row.ID == ID);
            asw.answerText = answer.answerText;
            dc.SubmitChanges();
        }
        catch (Exception)
        {
            throw new Exception("Update error!");
        }
    }

    public static void createAnswer(Answer answer, CONTESTDataContext dc)
    {
        if (dc == null)
            dc = new CONTESTDataContext();
        try
        {
            dc.Answers.InsertOnSubmit(answer);
            dc.SubmitChanges();
        }
        catch (Exception)
        {
            throw new Exception("Insert error!");
        }
    }
```

Mindhárom metódus esetén a módosítást végző sorok után szükség van egy dc.SubmitChanges(); utasításra, amely véglegesíti a változást a DataContext-ben, így a legközelebbi, adatbázison végrehajtott művelet már az új adatokkal tud dolgozni.

Szintén minden osztály tartalmaz különböző lekérdező metódusokat. Ezek a lekérdező metódusok több megfelelő rekord esetén egy listával térnek vissza. Mindenhol szerepel az egész tábla tartalmát lekérdező metódus, olyan metódus, amely a táblában szereplő rekordok számát adja vissza, valamint ID alapján egy adott elem lekérése. Szintén az *AnswerD* osztályon bemutatva:

```
    public static List<Answer> getAnswerList()
    {
        CONTESTDataContext dc = new CONTESTDataContext();
        return dc.Answers
            .OrderBy(row => row.answerText)
            .ToList<Answer>();
    }

    public static int getAnswerCount(CONTESTDataContext dc)
    {
```

```
        if (dc == null)
            dc = new CONTESTDataContext();
        return dc.Answers.Count();
    }

    public static Answer getAnswerByID(int ID, CONTESTDataContext dc)
    {
        if (dc == null)
            dc = new CONTESTDataContext();
        Answer answer;
        try
        {
            answer = dc.Answers.Single(row => row.ID == ID);
            return answer;
        }
        catch (Exception)
        {
            throw new Exception("Query error!");
        }
    }
```

Ezeken kívül egyes osztályokban szükség volt más, összetettebb metódusok megírására.

A felhasználókat megvalósító osztályok (*AdministratorD*, *JudgeD*, *ContestantD*) az előbbieken túl csak néhány metódust tartalmaznak. Ilyen például a név, vagy felhasználói név alapján történő lekérdezés. Mivel a név, és a felhasználói név is egyedi, ezért a lekérdezés eredménye egyetlen rekord. Illetve meg kell tudnunk mondani egy adott verseny felhasználóit, valamint a bejelentkezett felhasználókat. Ezek a metódusok a sima lekérdezésen túl egy szűrést tartalmaznak. Visszatérési típusuk szintén egy lista.

A több metódust tartalmazó osztályok a kapcsolótáblákat reprezentáló osztályok. A kapcsolótábláim legtöbbször több-több kapcsolatot valósítanak meg, ezért könnyen elképzelhető, hogy szükségünk van az egyik tulajdonsághoz tartozó rekordok lekérdezésére. Például az *AnswerContest* táblánál maradva, lehet olyan lekérdezésünk, hogy az adott versenyhez kapcsolódó válaszokat szeretnénk tudni, de lehet olyan is, hogy az adott válasz, mely versenyekhez kapcsolódik. Tehát általában ahány külső kulcsot tartalmaz egy kapcsolótábla, annyi különböző lekérdezést írhatunk, amelyek eredménye több-több kapcsolat esetén egy-egy lista lesz.

```
    public static List<AnswerContest> getAnswerContestListByContestID(int ID,
CONTESTDataContext dc)
    {
        if (dc == null)
            dc = new CONTESTDataContext();

        return dc.AnswerContests
            .Where(row => row.contestID == ID)
            .OrderBy(row => row.ID)
            .ToList<AnswerContest>();
    }
```

```
    public static List<AnswerContest> getAnswerContestListByAnswerID(int ID,
CONTESTDataContext dc)
    {
        if (dc == null)
            dc = new CONTESTDataContext();

        return dc.AnswerContests
            .Where(row => row.answerID == ID)
            .OrderBy(row => row.ID)
            .ToList<AnswerContest>();
    }
```

Az adatbázis nézeteit általában arra használom, hogy a felhasználói felületen lévő táblázatokat feltöltsék adattal. Ezért a táblákhoz hasonlóan a nézetekhez is megírtam a megfelelő osztályokat.

8. Felhasználói felület

A felhasználói felület a szoftvernek az egyedüli része, amelyet a felhasználó lát, és használ Éppen ezért fontos, hogy könnyen kezelhető, áttekinthető, viszonylag gyors elérésű legyen, és az aktuális információkat mutassa. Aktuális információkat akkor tudunk megjeleníteni, ha időről időre automatikusan frissül a weblap, viszont a frissítés során esetleg hosszadalmas kódrészletet kellene lefuttatni, sok adatbázis műveletet igényelne, tehát a gyorsaság rovására menne. A megoldás az Ajax technológia, amely lehetővé teszi, hogy ne az egész weblap, hanem annak csak bizonyos részei frissüljenek.

8.1. Ajax

Az Ajax (Asynchronous JavaScript and XML) interaktív web alkalmazások létrehozására szolgáló web fejlesztési technika. A weblap kis mennyiségű adatot cserél a szerverrel a háttérben, így a lapot nem kell újratölteni minden egyes alkalommal, amikor a felhasználó módosít valamit. Ez növeli a honlap interaktivitását, sebességét és használhatóságát.

Az Ajax a következő technikák kombinációja:

- XHTML (vagy HTML) és CSS a tartalom leírására és formázására.
- DOM (Document Object Model) kliens oldali script nyelvekkel kezelve a dinamikus megjelenítés és a már megjelenített információ együttműködésének kialakítására.
- XMLHttpRequest objektum az adatok aszinkron kezelésére a kliens és a web szerver között.

- XML formátumot használnak legtöbbször az adattovábbításra a kliens és a szerver között, bár más formátumok is megfelelnek a célnak, mint a formázott HTML vagy a sima szöveg.

Az Ajax-ot használó oldalak a szervertől az adatokat HTML formázás nélkül kapják, ezért ez által a szerver terhelése és így a válaszideje is csökken. A kisebb mennyiségű adat pedig gyorsabban jut el a hálózaton a szervertől a kliensig, így növelhető az oldalak letöltési sebessége. A szervertől kapott adatokból a HTML kód a böngészőben jön létre javascript segítségével, amely nem lassabb, mintha az erősen leterhelt szerver hozná azt létre. Ennek oka, hogy a kliens oldalon a felhasználók viszonylag gyors személyi számítógépekkel rendelkeznek, amelyek terhelése általában lényegesen alacsonyabb, mint a szerveré. Ráadásul, ha valamelyik kliens gép a lassúsága vagy leterheltsége miatt mégis lassabban hozza létre a HTML kódot, az nem érinti a párhuzamosan jelenlévő többi klienst, amely annál nagyobb előny minél nagyobb a párhuzamosan jelenlevő kliensek száma.

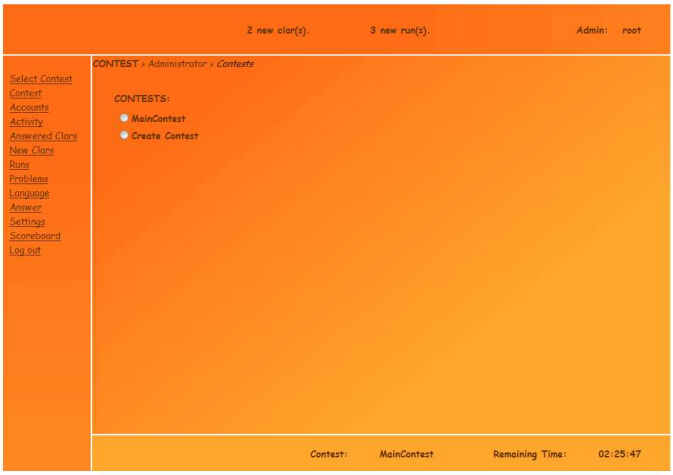
Ajax-ot használva a weblap mögött álló kódban megadhatjuk, hogy az oldal mely részei frissüljenek egy-egy esemény hatására. Például egy linkre kattintva nincs szükség az egész oldal újbóli letöltésére a web szerverről, hanem elég csak a szükséges részeket frissíteni, így a letöltési idő csökken.

Az Ajax technológiát arra használtam, hogy a weblap azon része frissüljön automatikusan, amely a versenyből hátra lévő időt, a beérkezett üzenetek és forráskódok darabszámát jeleníti meg. Hogy ne kelljen minden weblap mögött álló kódba beépítenem ezen információk megjelenítését és frissítését, így azt választottam, hogy master page-et használok.

8.2. Weblapok egységes kinézete

Az ASP.NET master page megengedi, hogy az alkalmazásban lévő weblapok számára azonos kinézetet hozhassunk létre. Ez azt jelenti, hogy elég egyetlen master page-en definiálni azon megjelenítési formákat, kinézetet és alapvető viselkedésmódokat, amelyeket minden weblap számára biztosítani szeretnénk. A master page-en olyan tartalmazó részek is elhelyezhetők, amelyeket majd a weblapok töltenek fel egyedi tartalommal, tehát minden oldal esetén más és más lesz.

Mindhárom felhasználótípusnak más master page-et készítettem, de a köztük lévő eltérés minimális. Külön master page-gel rendelkezik az eredményjelző tábla, és a bejelentkezési oldal is.



11. ábra – Master page az adminisztrátornál

A master page-ek minden esetben négy részből állnak. A baloldalon egy menüsáv látható, amely hivatkozási linkeket tartalmaz a megfelelő oldalakra. A *Scoreboard* link minden master page-en megtalálható, és ez az egyedüli, amely új lapon nyitja meg a megjelenítendő oldalt. A többi link esetében csak a master page középső része fog megváltozni. Ez a középső rész az, amely a beágyazott oldal tényleges tartalmát megjeleníti. A legfelső sor egy SiteMapPath vezérlőelem, amely az oldalak közötti könnyebb navigációt szolgálja. A SiteMapPath jeleníti meg az aktuális oldalig vezető utat, azaz egy elérési utat, és bármely elemére kattintva az így kiválasztott oldal megjelenítődik. Ez az eszköz volt az, amely feleslegessé tette a „Back” gombok elhelyezését az oldalakon.

A MasterPage alsó (Footer), és felső (Header) része az aktuális információkat jeleníti meg, az oldalak közötti váltás során nem változik meg, viszont néhány másodpercenként automatikusan frissül. A Header részben három információ látható. Jobboldalt látszik a bejelentkezett felhasználó titulusa és neve. Az előtte látható két adat opcionális. Ez azt jelenti, hogy csak akkor látszódik, ha tényleges információval rendelkezik, azaz nem nullaértékű. Adminisztrátori és bírói oldalon a clar-ok (clarification) az aktuális versenyre beérkezett és még meg nem válaszolt üzenetek darabszámát mutatja, a run-ok pedig a versenyzők által az adott versenyre beküldött, és még nem kiértékelt forráskódjainak számát jelzi. A versenyzők számára a clar-ok a bíráktól érkezett válaszok darabszámát, a run-ok, pedig a versenyző által beküldött, már kiértékelt forráskódok darabszámát mutatja. Ha a versenyző megtekinti a beér-

kezett üzeneteit, akkor a clar-ok száma nullára csökken, ha pedig az eredményeit megjelenítő oldalra navigál, akkor a run-ok száma csökken nullára. Nulla érték nem kerül megjelenítésre a Header-ben.

A Footer-ben az aktuális versenyből még hátralévő idő látszódik. Mivel az adminisztrátorok, és a bírák egyszerre több versenyen is részt vehetnek, ezért ők a versenyek között válthatnak, tehát kiválaszthatják, hogy melyik verseny adatai kerüljenek megjelenítésre a Header-ben és a Footer-ben. Sőt még azt is megtehetik, hogy egyik versenyről sem kérnek információt. A versenyzők egyszerre csak egy versenyen vehetnek részt, ezért számukra nincs választási lehetőség.

A megjelenített oldalból csak a Header és a Footer rész frissül automatikusan, így csökkentve a végrehajtandó adatbázis műveletek darabszámát. A frissítéshez a korábban ismertetett Ajax technológiát használtam. Ahhoz, hogy a weblap Ajax segítségével kommunikáljon a web szerverrel, szükség van egy ScriptManager vezérlőelemre az oldalon. Ez az elem irányítja és felügyeli a weblap és a web szerver közti információcsomagok áramlását.

A frissítendő adatok egy-egy UpdatePanel-en belül helyezkednek el. Az Update Panel egy Ajax specifikus vezérlőelem, egy konténer elem, amely annyit tesz, hogy nem kerül közvetlenül megjelenítésre, de az általa tartalmazott elemek általa frissíthetők. A frissítést bizonyos gyakorisággal kell elvégezni. Erre szolgál a Timer vezérlőelem, amely megadott időközönként lefuttat bizonyos kliens oldali scripteket, és lehetővé teszi a weblap egyes elemeinek frissülését. Ezen három vezérlőelem segítségével sikerült megoldanom, hogy a fontosabb információk a megfelelő időben frissüljenek. Példaként a versenyből még hátralévő időt megjelenítő kódrészlet a következőképpen néz ki:

```
<head id="Head1" runat="server">
    <script runat="server">
        protected void Page_Load(object sender, EventArgs e)
        {
            contestLabel.Text = contest();
            remTime.Text = getRemTime();
        }
        protected void Timer_tick(object sender, EventArgs e)
        {
            contestLabel.Text = contest();
            remTime.Text = getRemTime();
        }
        private string contest()
        {
            //a verseny nevének lekérdezése
        }
        private string getRemTime()
        {
            //a hátralévő idő lekérdezése
```

```
        }
    </script>
</head>
<body>
    <form id="form1" runat="server">
        <asp:ScriptManager ID="ScriptManager1" runat="server"/>
        <asp:Timer ID="Timer1" runat="server" OnTick="Timer_tick" Interval="3000">
        </asp:Timer>
        <table class="tableStyle" align="center">
            <tr>
                <td class="headerStyle">
                </td>
            </tr>
            <tr>
                <td class="menuStyle">
                </td>
                <td class="bodyStyle">
                </td>
            </tr>
            <tr>
                <td class="footerStyle" align="right" valign="middle">
                   <asp:UpdatePanel ID="UpdatePanel1" runat="server" UpdateMode="Conditional">
                      <Triggers>
                         <asp:AsyncPostBackTrigger ControlID="Timer1" />
                      </Triggers>
                      <ContentTemplate>
                          <asp:Label id="Label2" runat="server" Text="Contest:">
                          </asp:Label>
                                    
                          <asp:Label id="contestLabel" runat="server"></asp:Label>
                                       
                          <asp:Label id="Label1" runat="server" Text="Remaining Time:">
                          </asp:Label>
                                       
                          <asp:Label id="remTime" runat="server">
                          </asp:Label
                      </ContentTemplate>
                   </asp:UpdatePanel>
                </td>
            </tr>
        </table>
    </form>
</body>
```

8.3. Be- és a kiléptetés

A beléptetés a megfelelő login oldalakon történik. Minden felhasználótípusnak külön login oldala van, amelyek abban különböznek, hogy mely adatbázisbeli táblákat használják. A beléptetés folyamata során először a megadott felhasználói név és jelszó páros az adatbázis megfelelő táblájában lévő rekordokkal kerül összehasonlításra. Ha egyetlen rekorddal sincs egyezés, akkor ilyen felhasználónév jelszó párossal nem létezik felhasználó az adott táblában, a beléptetés sikertelen, és erről hibaüzenetet kapunk.

Bejelentkezés szempontjából a második lehetőség, amikor a felhasználónév jelszó párost a felhasználók között megtaláljuk, ilyenkor a beléptetés gond nélkül végrehajtódik.

Minden be- és kilépés esetén egy új rekord kerül az *EventLog* táblába, amely tartalmazza az esemény bekövetkeztének idejét, és egy-egy hivatkozást a versenyre, a felhasználó-

ra, és az *Event* táblában szereplő megfelelő eseményre. Az adminisztrátor az *EventLog* tábla tartalmát az Activity oldalon bármikor megtekintheti, így a be- és kilépések nyomon követhetővé válnak.

A weblapok közti információcserét sessionnel oldottam meg. A sessionben tárolt adatok nem vesznek el a weblapok közti váltások során, épp ellenkezőleg az egész folyamat alatt megmaradnak, és az alkalmazás bármely weblapja elérheti. Sikeres bejelentkezéskor a felhasználói nevet, a jelszót, és a felhasználó adatbázisbeli azonosítóját sessionben tárolom le, kilépéskor pedig törlöm a sessionből. Így minden weblapon ellenőrizni tudom, hogy az aktuális felhasználó jogosult-e az oldal megtekintésére. A session objektumot a forráskódban olyan kulcs-érték párokat tartalmazó kollekcióként kezelhetem, amelyben a kulcs és az érték is string típusú. Ez lehetővé tette, hogy a különböző típusú felhasználók bejelentkezési információit a sessionbe eltérő kulccsal tároljam, azaz egy számítógépen egyszerre minden típusú felhasználóból csak egy lehet bejelentkezve.

Minden weblap betöltődésekor először a megtekintés jogosultsága ellenőrződik a sessionben tárolt adatok alapján. Ha a felhasználónak van joga az oldal megtekintéséhez, akkor a kért weblap töltődik be, ha nincs joga, akkor a megfelelő bejelentkezési oldalra kerül. Az is előfordulhat, hogy a bejelentkezett felhasználónak megváltozik a jelszava, vagy a felhasználói neve, hiszen az illetékes adminisztrátor ezeket az adatokat bárkinél bármikor megváltoztathatja. Ilyenkor a felhasználó továbbra is bejelentkezve marad, csak a kért oldal helyett egy olyan oldal fog betöltődni, amely tájékoztat a változásról, és megjeleníti az új adatokat. Az adminisztrátor számára látható oldalak jogosultságvizsgálata:

```
    protected void Page_Load(object sender, EventArgs e)
    {
        if (Session["admin_id"] == null)
            Response.Redirect("~/_admin/adminLogin.aspx");
        else
        {
            try
            {
                List<Administrator> adminList = AdministratorD
.getAdministratorListWhere(Session["admin_name"].ToString(), null, null, true, null);
                if (adminList.Count == 0 || adminList.Count > 1)
                    Response.Redirect("~/_admin/adminLogin.aspx");
                if (Convert.ToInt32(Session["admin_id"].ToString()) != adminList.First().ID)
                    Response.Redirect("~/_admin/adminLogin.aspx");
                string disp = adminList.First().displayName;
                string pass = adminList.First().password;
                if (!disp.Equals(Session["admin_displayName"].ToString()) ||
!pass.Equals(Session["admin_password"].ToString()))
                {
                    Session["admin_displayName"] = disp;
                    Session["admin_password"] = pass;
                    Response.Redirect("~/_admin/adminChanged.aspx");
```

```
            }
        }
        catch (Exception exc)
        {
            Master.displayErrorMessageOnPopup(exc.Message);
        }
    }
```

8.4. Kivételkezelés

Valamilyen nem várt esemény bekövetkeztekor kivételről beszélünk. Célunk a nem várt események bekövetkeztének minimalizálása, és az előfordulásukkal járó kellemetlenségek csökkentése. A felhasználó számára nem előnyös olyan alkalmazás használata, amely folyton hibaüzeneteket küld, vagy megszakítja a működését, ezért a kivételkezelés az egyik legfontosabb része az alkalmazásnak.

Mivel leginkább az adatbázis tartalmának módosítása, illetve adatok lekérése történik, ezért a legtöbb kivétel az adatbázissal való kapcsolat során keletkezhet. Ilyen adatbázissal kapcsolatos hiba lehet például nem létező rekord módosítása, lekérdezése, törlése, vagy az adott időpillanatban nincs jogunk a táblát módosítani, mert egyik másik felhasználó teszi azt. Ezeknek a hibáknak a kezelése nem csak a felhasználó felé igényel visszajelzést arról, hogy az általa kezdeményezett művelet sikertelen volt, hanem gondoskodni kell az adatbázis konzisztenciájának megőrzéséről is. A konzisztencia akkor borulhat fel, ha több egymástól függő módosító műveletet kell végrehajtanunk. Ilyen esetben vagy az összes műveletet sikeresen végrehajtjuk, vagy ha valamelyik sikertelen, akkor az összest vissza kell vonnunk. Ezt tranzakciók alkalmazásával tehetjük. Minden esetben, amikor egynél több adatbázis módosító műveletet kell végrehajtani, célszerű ezt egy tranzakción belül tenni, és a teljesen sikeres végrehajtás után egy commit utasítással véglegesíteni, vagy hiba esetén rollback utasítással visszaállítani az adatbázist a tranzakció előtti állapotba.

C#-ban a tranzakciókat a System.Transactions névtér TransactionScope osztálya valósítja meg. A tranzakció indítása előtt szükségünk van a megfelelő DataContext osztály egy példányára, amely az adatbázis aktuális tartalmát tárolja. Ezek után példányosítani kell a TransactionScope osztályt, így indítjuk el a tranzakciónkat. Fontos, hogy a tranzakcióhoz kapcsolódó összes utasításnak ugyanazt a DataContext példányt kell használnia. Ahhoz, hogy a változások elérhetőek legyenek a DataContext példányra meg kell hívni a SubmitChanges() metódust, amely csak a DataContextben változtatja meg a tartalmat, az adatbázisban egyelőre még nem. A tranzakció sikeres végrehajtása után a Complete() metódussal véglegesíthetjük az

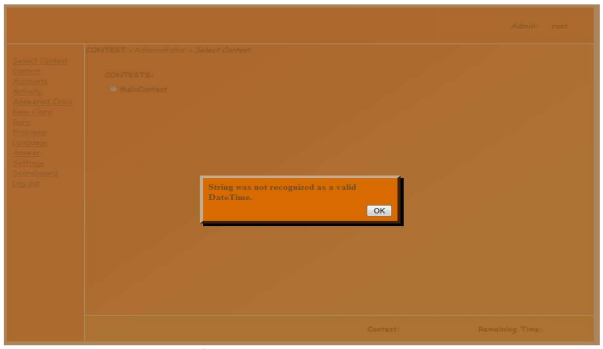
adatbázisban a változásokat. A visszagörgetést nem kell explicit módon feltüntetnünk, a Complete() metódus elhagyásával automatikusan megtörténik.

Egy példa az adminAccounts oldalról, amelyben egy tranzakció keretein belül lekérjük a kívánt versenyzőhöz tartozó rekordot, hamis értékre állítjuk a login mezőjét, azaz kijelentkeztetjük. Kijelentkezés esetén az *EventLog* táblába be kell szúrni a megfelelő sort, amely jelzi, hogy mely versenyző mikor jelentkezett ki. Jelen esetben, ha a kijelentkeztetés valamilyen oknál fogva nem történik meg, akkor az *EventLog* táblába sem kell az új rekordot beilleszteni.

```
    CONTESTDataContext dc = new CONTESTDataContext();
    try {
        using (TransactionScope ts = new TransactionScope())
        {
            Contestant cont = ContestantD.getContestantByID
(Convert.ToInt32(viewGridView.Rows[viewGridView.SelectedIndex].Cells[1].Text), dc);
            cont.login = false;
            ContestantD.updateContestantById(cont.ID, cont, dc);
            EventLog elog = new EventLog();
            elog.contestantID = cont.ID;
            elog.contestID = Convert.ToInt32(Session["admin_contest_id"].ToString());
            elog.eventID = EventD.getEventByEvent("Logout", dc).ID;
            elog.logTime = DateTime.Now;
            EventLogD.createEventLog(elog, dc);
            ts.Complete(); viewGridView.DataBind();
        }
    } catch (Exception exc)
    {
         Master.displayErrorMessageOnPopup(exc.Message);
    }
```

Ebből a kódrészletből látható, hogy egy kivétel bekövetkeztekor a megfelelő master page-en megírt displayErrorMessageOnPopup() metódus hívódik meg a hibaüzenettel felparaméterezve. Ez a metódus láthatóvá teszi a master page-en lévő ModalPopupExtendert, és beállítja, hogy a paraméterül kapott sztringet jelenítse meg. A ModalPopupExtender lehetővé teszi, hogy a weblapon bizonyos információt úgy jelenítsünk meg, hogy közben a lap többi része elérhetetlenné váljon a felhasználó számára, azaz ilyen esetben a lap bármely más részére kattintva semmi sem történik. Általában figyelmeztető- vagy hibaüzenet megjelenítésére használják. Ez az Ajax-ban megtalálható kiegészítőelem vezérlőelemek bármely hierarchiáját képes modális tartalomként megjeleníteni valamilyen háttér előtt. A háttérnek tetszőleges stílus adható, akár teljesen eltakarhatjuk vele a lap többi részét, áttűnését is szabályozhatjuk. A ModalPopupExtender rendelkezik OK illetve Cancel gombbal, melyekre kattintva a moduláris tartalom megjelenítése véget ér, és bizonyos scripteket futtathatók le. Mivel én csak hibaüzenetek megjelenítésére használom, ezért Cancel gombot nem helyeztem el rajta, és scripteket sem futtatok vele.

A ModalPopupExtender számára be kell állítani egy TargetControlID-t, ami annak a vezérlőelemnek az azonosítója, amely aktiválja a megjelenítését. Ez a vezérlőelem általában egy gomb, amely a felületen nem látszik. A PopupControlID annak a vezérlőelemnek az ID-ja, amely mint modális tartalom megjelenítődik. Nálam ez egy Panel, mely egy kétsoros táblázatot tartalmaz. A táblázat első sorában egy UpdatePanelen belül egy címke szerepel. Ez a címke kapja értékül a hibaüzenetet. A második sorban az OK gomb található. Megjelenítéskor tehát a táblázat felső sorában láthatjuk a hibaüzenetet, alatta pedig az OK gombot. A ModalPopupExtender OkControlID tulajdonságának be kell állítani a táblázaton lévő OK gomb azonosítóját, illetve a BackgroundCssClass-nál megadhatjuk a háttér megjelenését tartalmazó stílust.



12. ábra - ModalPopupExtender

A 12. ábrán látható a ModalPopupExtender kinézete. A hátteret barna árnyalatúra állítottam, és az áttűnés 20%. Így a felhasználó még láthatja, a háttérben lévő weblapot, de azt is érzékeltetem, hogy a weblap jelenleg nem elérhető. Az OK gombra kattintva a moduláris tartalom eltűnik, ismét a weblap lesz aktív. Nálam a ModulPopupExtender csak a felhasználó számára jeleníti meg a hibaüzenetet, más kivételkezelési funkciója nincs.

```
    <head id="Head1" runat="server">
        <script runat="server">
           public void displayErrorMessageOnPopup(String errorMessage)
           {
              ModalPopupExtender1.Show();
              lblErrorOnPopup.Text = errorMessage.ToString();
           }
        </script>
    </head>
    <body>
       <form id="form1" runat="server">
          <asp:Button ID="ButtonForPopUp" runat="server" Style="display: none" />
          <asp:Panel ID="popUpPanel" runat="server" Width="300px" CssClass="modalPopup"
Style="display: none">
            <asp:Table ID="tableOnPopup" runat="server" EnableTheming="false" Width="300px" >
              <asp:TableRow>
                <asp:TableCell>
                  <asp:UpdatePanel ID="upPP" runat="server">
                    <ContentTemplate>
                      <asp:Label ID="lblErrorOnPopup" runat="server" EnableViewState="true"
Font-Bold="True" ForeColor="#754E26" Font-Names="MS Comic Sans">
```

```
                    </asp:Label>
                  </ContentTemplate>
                </asp:UpdatePanel>
              </asp:TableCell>
            </asp:TableRow>
            <asp:TableRow>
              <asp:TableCell HorizontalAlign="Right">
                <asp:Button id="btnClosePopup" runat="server" Text="OK"  />
              </asp:TableCell>
            </asp:TableRow>
          </asp:Table>
        </asp:Panel>
        <cc2:ModalPopupExtender ID="ModalPopupExtender1" runat="server"
         OkControlID="btnClosePopup" TargetControlID="ButtonForPopUp"
         PopupControlID="popUpPanel" BackgroundCssClass="modalBackground" DropShadow="True">
        </cc2:ModalPopupExtender>
</body>
```

9. Felépítés

Több felhasználói típus számára készült a szoftver, így a forráskód megírásakor fontosnak tartottam, hogy az egyes felhasználókhoz tartozó kódrészleteket, weblapokat elkülönítsem. Külön mappát hoztam létre az adminisztrátor, a bíró, a versenyző és a mindenki számára elérhető eredményjelző számára. A weblapokat a megfelelő mappákban helyeztem el, így áttekinthetőbbé vált a felépítés.

9.1. Adminisztrátori oldal

Az egész verseny megalkotásáért és problémamentes lebonyolításáért az adminisztrátor a felelős, ő az egyetlen felhasználó, aki szinte minden adathoz hozzáférhet. Ezek alapján nem meglepő, hogy az adminisztrátori oldal a legösszetettebb, ez tartalmazza a legtöbb lekérdezést, a legtöbb táblázatot. Csak a fontosabb, érdekesebb részeket emelném ki.

9.1.1. Verseny beállításai

Az adminisztrátor hozhat létre új versenyt, illetve ő szerkesztheti a már meglévő versenyeket. Fontos kiemelni, hogy az adminisztrátor csak azokat a versenyeket látja, amelyekhez, mint adminisztrátor hozzá van rendelve. Új verseny létrehozásakor az adminisztrátor a verseny jellemzőit beállíthatja, de ezt később bármikor módosíthatja, azzal a megszorítással, hogy folyamatban lévő verseny kezdési ideje és típusa nem állítható át. Verseny szerkesztésénél lehetőség van az adatbázis bizonyos mértékű kiürítésére. Ez alatt azt értem, hogy az adminisztrátor kiválaszthatja, hogy a versenyhez kapcsolódóan milyen tárolt információk maradjanak meg az adatbázisban, és melyek kerüljenek törlésre. Törölni lehet a versenyhez kapcsolódó egész levelezést, a beküldött forráskódokat, illetve a problémákat és a programozási nyel-

veket. Az utóbbi kettő nem törlődik ki fizikailag az adatbázisból, csak az adott versennyel való kapcsolat szűnik meg, azaz a megfelelő kapcsolótábla adott rekordja törlődik.

9.1.2. Felhasználók menedzselése

Az Accounts oldal alatt az adminisztrátornak lehetősége van a felhasználók megtekintésére, és menedzselésére. Láthatja a kiválasztott versenyre aktuálisan bejelentkezett felhasználókat, kijelentkeztetheti őket, illetve szerkesztheti a versenyhez kapcsolódó felhasználók adatait.

9.2. Bírói oldal

9.2.1. Kiértékelés lehetőségei

A bírói oldal legérdekesebb része a beküldött forráskódok kiértékelése. A kiértékelés folyamatának három változata létezik. Az első a hagyományos, manuálisan történő kiértékelés, amikor a bíró értesítést kap az újonnan beküldött forráskódról, majd a kiértékelő oldalon keresztül fordítja, futtatja a kódot, és a felkínált válaszlehetőséget vagy elfogadja, vagy felülbírálja. Ebben az esetben a bíró a kiértékelés folyamatát részletesen látja, az adott választ manuálisan beállíthatja.

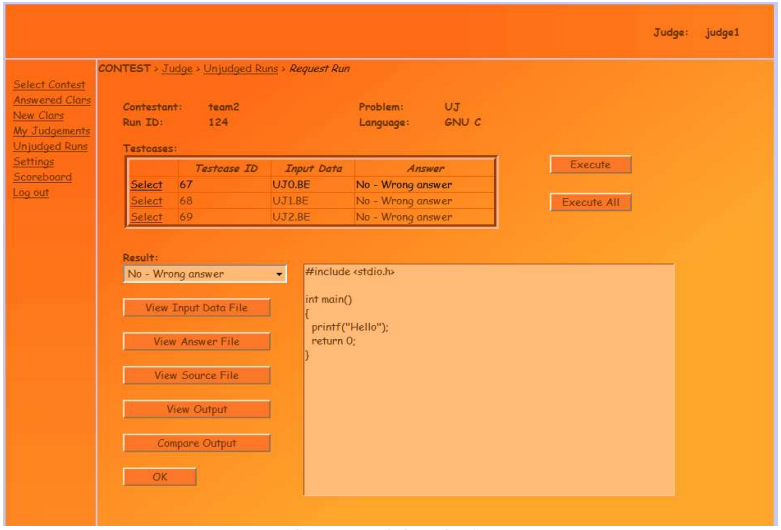
A második lehetőség félig automatikus működésű. A bíró szintén értesülést szerez a forráskód beérkezéséről, de nem kívánja részletesen végignézni a kiértékelés folyamatát, hanem az adott forráskódnál az `autojudge` lehetőséget választja. Ez azt jelenti, hogy a szoftver ezt az egy beküldött kódot automatikusan kiértékeli, a bíró nem avatkozhat közbe, a választ ne bírálhatja felül, csak a végeredményt látja.

A harmadik lehetőség a teljesen automatikus kiértékelés, amikor verseny szinten állítjuk be az `autojudge` tulajdonságot. Ennél az esetnél a kiértékelés minden beküldött forráskód esetén automatikus, így a bíró értesítést sem kap a beérkezett kódokról. Ezt az esetet úgy oldottam meg, hogy a kiértékelést végző kódrészlet a versenyzői oldalon szerepel, és amikor a versenyző beküldi a forráskódot, akkor az nemcsak feltöltődik az adatbázisba, hanem azonnal ki is értékelődik. Ennél a lehetőségnél előfordulhat olyan, hogy valamilyen oknál fogva a kiértékelés nem történik meg sikeresen, ilyenkor a bírói oldalon megjelenik a még ki nem értékelt források között, és a bíró az első két lehetőségből választva elvégezheti a kiértékelést.

Fontosnak tartom kiemelni, hogy függetlenül attól, hogy a kiértékelést mely felhasználói oldalról (versenyző, bíró), mely számítógépről kezdeményezzük, a forráskód minden esetben a szerver gépen fog lefordulni, ott kerül lefuttatásra. Ez azért fontos, mert így garantált, hogy a szoftver fordítás során ugyanazt a fordítót, ugyanazokkal a beállításokkal használja. Az idő- és memórialimit átlépésének figyelése is így válik egységessé, igazságossá, hiszen ebben az esetben nem hivatkozhatunk arra, hogy a beküldött kódok különböző teljesítményű számítógépeken, különböző hardvereken lettek futtatva.

9.2.2. Kiértékelés folyamata

A kiértékelést végző kódrészlet lényegében mindhárom esetben ugyanaz, csak a manuális változatnál több részre van szétszedve. Ezért csak a manuális esetet ismertetem, mert a kisebb egységekre való szétszedés miatt az a legkönnyebben leírható.



13. ábra - A kiérétkelési felület

A kiértékelési felületen kiválaszthatjuk, hogy mely tesztesetre szeretnénk lefuttatni a beküldött kódot, vagy az „Execute All” gombra való kattintással az összes tesztesetre lefuttatódik, és ilyenkor a legördülő listában megjelenik a javasolt válasz. Bármelyik lehetőséget választjuk, először a fordítás történik meg. A fordítást egy új process végzi az adminisztrátor által előre beállított fordítási opciókkal. Sikertelen fordítás esetén a tesztesetre adandó válasz a „No – Compiler error” lesz, és a futtatás, validálás már nem történik meg.

Sikeres fordítás után az adott tesztesetre lefuttatódik a program. Ezek után, ha van beállítva idő- vagy memóriakorlát, akkor azt vizsgálom, hogy a program túllépte-e a korlátot. Ha igen, akkor megfelelően „No – Time limit exceeded”, illetve „No – Memory limit

exceeded” lesz a válasz. Amennyiben a program a megadott időn belül futott le, és nem használt föl a megengedettől több memóriát, megvizsgálom az `exit code`-ot, hogy történt-e valamilyen futási idejű hiba. Ha az `exit code` nem megfelelő, akkor „No – Runtime error” lesz a válasz, egyébként megtörténik a validálás.

A validálást a GNU Diffutils nevű ingyenesen letölthető program végzi. A szoftver `diff` parancsát használom a példa output és az aktuálisan előállított output összehasonlítására. A `diff` parancs különböző opciókkal hívható meg, melyek az összehasonlítás módját befolyásolják. A használatos opciókat az adminisztrátor minden problémára külön-külön tudja beállítani. A `diff` a hasonlítást karakterenként végzi, és teljesen egyező fájlok esetén 0 exit `code`-dal tér vissza. Ez az egyetlen eset, amikor a beküldött forráskód az adott tesztesetre tökéletesen működött, azaz megérdemli a „Yes - Accepted” választ. Ha a két fájl tartalma nem egyezik meg, akkor a „No – Wrong answer” választ kapjuk.

Amennyiben a két fájl összehasonlítása megtörtént, még egyszer meghívom a `diff`-et, ám ezúttal a `side-by-side` opciót is használom. Ezzel az opcióval legenerálható egy olyan fájl, mely tartalmazza az összehasonlítást, azaz a két összehasonlítandó fájlt soronként egymás mellé másolja. Az adatbázisba a beküldött forráskód által előállított output fájl, és ez a fájl kerül majd. Ezt a két fájlt a bíró, és az adminisztrátor bármikor megtekintheti, így ellenőrizhetik a beküldött kódra adott válasz helyességét.

Miután az adott tesztesetre megtörtént a fordítás, futtatás, validálás, megkapjuk a tesztesetre javasolt választ. Amennyiben ez a válasz nem szerepel a versenyen adható válaszok között automatikusan felülíródik a „No - Other unknown error” válasszal. Éppen ezért minden versenyen engedélyezni kell a „Running…”, a „Yes - Accepted” és a „No – Other unknown error” válaszokat. Ez automatikusan megtörténik a verseny létrehozásakor, és később nem módosítható. Mikor a végleges válasz kiválasztásra kerül, megtörténik az adatbázis megfelelő módosítása. Ha csak egyetlen tesztesetre teszteltük a beküldött kódot, akkor egyedül a *SourceTestcaseAnswer* tábla módosítására van szükség, méghozzá a megfelelő rekord `score` és `answerID` mezőit kell frissíteni.

Ha már minden tesztesetre kiértékeltük a beküldött a kódot, akkor a problémára adott választ a kiértékelő felületen a legördülő listából kiválaszthatjuk. Amennyiben az „Execute All” gombbal az összes tesztesetre elvégezzük a kiértékelést, az előbb említett legördülő listá-

ban lesz felajánlva egy lehetséges válasz, amely megegyezik a tesztesetekre kapott első nemleges válasszal. Az OK gombra kattintva az adatbázist a következők szerint módosítjuk:

- A *SourceOther* táblában a megfelelő rekord `answerID`, és `judgeID` mezőit az adott válasz azonosítójával, illetve a kiértékelő bíró azonosítójával írjuk fölül.
- A *Source* táblában az accepted, `acceptedTestcaseNumber`, `judged`, `judgedTime`, `score` mezőket a megfelelő értékekkel írjuk felül. Az `accepted` mező csak akkor lesz igaz értékű, ha a beküldött forráskód minden teszteseten megfelel. Az `acceptedTestcaseNumber` a sikeres tesztesetek számát tartalmazza. A `judged` mező mindenképpen igaz lesz, hiszen ez jelenti azt, hogy a beküldött kódot kiértékeltük. A `judgedTime` pedig a kiértékelés ideje lesz, azaz a jelenlegi idő. A `score` mező értéke pontozáson alapuló verseny esetén a tesztesetekre kapott összpontszám, acm jelleg esetén rossz megoldásnál a büntetés, jó megoldásnál a beküldési idő percben mért értéke lesz.
- A *Contestant* táblában az `acceptedProblem` mező értéke növelődik eggyel, ha a versenyző egy általa még tökéletesen meg nem oldott feladatra ad tökéletes megoldást. Valójában viszont nem növelés történik, hanem lekérdezem az adatbázisból, hogy a versenyző hány problémára küldött be helyes megoldást, és a kapott eredménnyel írom felül ennek a mezőnek az értékét.

A bírói oldalon lévő automata kiértékelő rendszer teljesen ugyanígy működik, míg a versenyzői oldalon történő kiértékelés esetén (amikor az egész verseny automata kiértékeléssel zajlik) a *SourceOther* tábla `judgeID` mezője nem változik, hiszen nem bíró által történik a kiértékelés.

9.3. Versenyzői oldal

A felhasználói felületek közül a legegyszerűbb a versenyzői oldal, hiszen a versenyző rendelkezik a legkevesebb joggal, neki van a legkevesebb lehetősége. A versenyzői oldalon a legérdekesebb weblap az, amelyen keresztül a versenyzők beküldhetik az általuk készített forráskódokat. A beküldés úgy történik, hogy előbb ki kell választaniuk a problémát, majd a programozási nyelvet, ezek után feltölthetik az általuk készített fájlt. Feltölteni csak a verseny ideje alatt lehet, egyéb esetben a sikertelen feltöltésről kapunk egy figyelmeztető üzenetet.

Feltöltés során először ellenőrizni kell, hogy a beküldött fájl kiterjesztése megfelelő-e. Ezt a kiválasztott programozási nyelv alapján tehetjük meg, ugyanis az adminisztrátornak minden programozási nyelvhez meg kell adnia egy kiterjesztést, és csak az ezzel a kiterjesztéssel rendelkező fájlokat lehet feltölteni az adott programozási nyelvhez. A feltöltés annyiból áll, hogy először egy, a szerveren lévő mappába ideiglenesen felmásolódik a fájl, majd az adatbázis *Source* táblájába bekerül egy új rekord, amely tartalmazza a beküldött fájlt, a fájl nevét és a beküldési időt. A *SourceOther* táblába kerülő új rekorddal pedig az újonnan beküldött fájlhoz tartozó rekordot összekapcsoljuk a többi tábla megfelelő rekordjaival, azaz a beküldő versenyzővel, a programozási nyelvvel, a „Running…” válasszal, a problémával. De nem csak a problémához kell hozzákapcsolni, hanem minden tesztesethez is, ami az adott problémához tartozik.

```
    using (TransactionScope ts = new TransactionScope())
    {
       aktSource = new Source();
       aktSource.fileName = filename;
       aktSource.sentTime = date;
       aktSource.sourceFile = file;
       aktSource.score = 0;
       SourceD.createSource(aktSource, dc);
       Answer ans = AnswerD.getAnswerByAnswer("Running...", dc);
       SourceOther so = new SourceOther();
       so.answerID = ans.ID;
       so.contestantID = aktContestant.ID;
       so.problemID = aktProblem.ID;
       so.sourceID = aktSource.ID;
       so.languageID = aktLanguage.ID;
       SourceOtherD.createSourceOther(so, dc);
       List<TestcaseProblem> testcaseList =
TestcaseProblemD.getTestcaseProblemListByProblemID(aktProblem.ID, dc);

       foreach (TestcaseProblem tp in testcaseList)
       {
          SourceTestcaseAnswer sta = new SourceTestcaseAnswer();
          sta.sourceID = aktSource.ID;
          sta.testcaseID = tp.ID;
          sta.answerID = ans.ID;
          sta.score = 0;
          SourceTestcaseAnswerD.createSourceTestcaseAnswer(sta, dc);
       }

       ts.Complete();
    }
```

Ha az egész versenyre be van állítva az automata kiértékelés, akkor az a fájl feltöltése után azonnal megtörténik, sem a bírók, sem az adminisztrátorok nem kapnak értesítést az újonnan beérkező kódról. A kiértékelés hasonlóan történik a bírói oldalon megismerthez, egyetlen különbség, hogy ebben az esetben a kiértékelő bíró azonosítója nem kerül beállításra, hiszen ilyen bíró nem létezik.

9.4. Eredményjelző tábla generálása

Az eredményjelző tábla, azaz *Scoreboard* bármely felhasználó számára bejelentkezés nélkül elérhető. A *Scoreboardon* háromféle eredménykijelzés közül választhatunk, amelyek csak annyiban különböznek, hogy más részletességgel mutatják az eredményeket. Az eredményeket versenyenkénti bontásban láthatjuk az eredményjelzőn.

Mivel mindhárom eredményjelző tábla hasonlóan épül fel, ezért én csak az egyiket ismertetném. Az eredményeket mutató táblázatot manuálisan a forráskódban rakom össze. Először az oszlopokat adom hozzá a táblához, majd feltöltöm a megfelelő sorokkal. Oszlop hozzáadásakor mindenképpen meg kell határoznom az új oszlop nevét, a tartalmazott adatok típusát. Ezen felül beállíthatók bizonyos tulajdonságok, például a ReadOnly, és a Unique tulajdonság.

```
DataTable table = new DataTable("ScoreTable");
DataColumn column;
column = new DataColumn();
column.DataType = System.Type.GetType("System.Int32");
column.ColumnName = "Rank";
column.ReadOnly = true;
column.Unique = false;
table.Columns.Add(column);
```

Az első eredményjelző (Board1) esetén a következő oszlopok kerülnek a táblába:

- Rank – a versenyző helyezése
- Contestant – a versenyző felhasználói neve
- A versenyhez kapcsolt összes probléma nevével szerepel egy-egy oszlop
- Total Sent – a versenyző által összesen beküldött fájlok száma
- Accepted – a versenyző által megoldott problémák száma
- Score/Solved Time – a versenyző pontszáma

Ahhoz, hogy a problémák neveit tartalmazó oszlopokat hozzáadjam a táblához, az adatbázisból le kell kérdeznem a versenyhez tartozó problémákat. Itt figyelnem kellett, hogy a „Deleted”, és a „General” nevű problémák ne kerüljenek be a táblázatba, hiszen ezek minden versenyhez kapcsolódnak, de nem valódi problémák, más funkciójuk van.

Az oszlopok létrehozása után le kellett generálnom a megfelelő sorokat. Első lépésként lekérdezem a versenyhez tartozó felhasználókat, és minden versenyzőre meghívom `void sumActiveSourceScore(Contestant cont, CONTESTDataContext dc)` a metódust, amely kiszámolja a versenyző pontszámát, és a *Contestant* tábla `score` mezőjét felülírja a meghatározott pont-

számokkal. Utána a korábban lekérdezett versenyzőket tartalmazó listát sorba rendezem a beküldött helyes megoldások száma szerinti csökkenő sorrendbe, és azon belül pontszám szerinti növekvő, illetve csökkenő sorrendbe, attól függően, hogy acm jellegű, vagy pontozáson alapuló versenyről van szó. A sikeres sorba rendezés után minden versenyzőre lekérdezem, hogy problémánként hány fájlt küldött be, és hogy volt-e közte sikeres megoldás. A problémákból képzett oszlopot ezek alapján töltöm ki. Közben megszámolom, hogy a versenyző összesen hány fájlt küldött be, és hány problémát sikerült teljesen jól megoldania. Ezek a számok a „Total Sent”, és az „Accepted” oszlopok adatait adják. A „Score”/”Solved Time” oszlopban a versenyző pontszáma szerepel, amelyet az előbb említett metódus számol ki.

A másik két tábla is hasonlóan épül föl, csak az egyikben szerepel a problémánként elért pontszám is. A *Scoreboardot* tartalmazó weblap az egyedüli, mely nem frissül automatikusan, és amelyet bárki megtekinthet. A kivételkezelése megegyezik a korábban bemutatottal.

10. Tesztelés, kipróbálás

A fejlesztés célja olyan szoftver előállítása volt, amely megfelel a megfogalmazott igényeknek. Fejlesztés során nagy hangsúlyt fektettem a folyamatos tesztelésre. Előbb egyetlen gépen teszteltem az alkalmazást, ez a gép volt a szerver és a kliens is. A későbbiekben több gépet összekötve próbáltam ki a rendszert. Ekkor fedeztem fel azokat a hibákat, amelyek hátterében összetettebb probléma állt. A tesztelés során felmerülő problémákat folyamatosan sikerült kijavítanom.

10.1. Kipróbálás

A 2009. április 19-én megrendezett kari programozói verseny az általam készített alkalmazáson került lebonyolításra. Az előkészületek során számos nehézségbe ütköztünk. Első lépésként a teljes alkalmazást át kellett vinnünk arra a számítógépre, amely a versenyen a szervergép szerepét töltötte be.

Az átvitel során problémát jelentett, hogy a szervergépen Windows XP operációs rendszer futott, amely nem támogatja az IIIS7-et. Ezért a korábbi 5.1-es verziót kellett használnunk web szerverként. Az IIS 5.1-ben viszont néhány beállítást másképp kell elvégezni. Ilyen beállítás a kapcsolatok maximális számának korlátozása, amely az IIS 5.1-ben alapér-

telmezetten néhány főt jelent. A verseny elkezdése után azonnal kiderült, hogy a versenyzők nem tudnak bejelentkezni, és egy script lefuttatásával ezt a számot nagyobbra módosítottuk.

További probléma az adatbázis elérése volt. A fejlesztett alkalmazás Windows autentikációt használ. A szervergépen pedig korábban már feltelepítették a .NET 1.0-as verzióját, amely létrehoz egy rejtett Windows felhasználót ASPNET névvel. Ezért mikor az alkalmazást az IIS-en keresztül elindítottuk, és szerettünk volna csatlakozni az adatbázishoz, hibaüzenetet kaptunk. A hibaüzenetet az okozta, hogy ha létezik az ASPNET felhasználó, akkor az IIS ezt a felhasználót használja az anonymus látogatók számára. Ennek a felhasználónak viszont nem volt joga sem az adatbázis eléréséhez, sem a megadott helyen elhelyezett fájlok módosításához. A probléma megoldását az jelentett, hogy ténylegesen létre kellett hozni egy ASPNET nevű, adminisztrátori jogokkal rendelkező Windows felhasználót, és engedélyezni számára az adatbázis elérését, és a megfelelő fájlok módosítását.

A sikeres beállítások után ellenőriztük, hogy minden teremből elérhető-e a honlap. A másnap reggel elkezdődött a verseny. A verseny elején beérkezett néhány forrásfájlt kénytelenek voltunk kézzel kiértékelni, mert az alkalmazás minden esetben „No – Time out error.” választ adott. Körülbelül 10 perccel későbbre ezt sikerült kijavítanom, és innentől a szoftveren keresztüli kiértékelés is működött. Két feladat volt, amelyet kézzel kellett ellenőriznünk. Ebben a két esetben a tesztesetek olyanok voltak, hogy valamiért nem tudta kezelni az alkalmazás. Ilyen eset minimális számban fordult elő.

A szoftverbe épített levelezőrendszer tökéletesen működött, a versenyzők éltek is ezzel a lehetőséggel, küldtek a feladatokkal kapcsolatos kérdéseket a bíróknak.

Számomra pozitív volt, hogy a versenyzők hamar megtanulták kezelni az alkalmazást, jól boldogultak a használatával. Mindenkinek sikerült beküldeni a forráskódjait, és kérdezni is tudott, amennyiben szeretett volna. Verseny közben az eredményeket nyomon tudták követni, az eredményjelző tábláról le tudták olvasni, hogy mi mit jelent.

10.2. Továbbfejlesztési lehetőségek

Az éles kipróbálás során fogalmazódtak meg távolabbi fejlesztési célok, amelyek megvalósításával még könnyebben kezelhetővé válik a felület. Ilyen fejlesztési cél, hogy a kimeneti fájlok összehasonlításánál ne csak a GNU Diffutilst lehessen használni, hanem az adminisztrátor a felületen keresztül egyéb validátorokat is hozzáadhasson az alkalmazáshoz.

Ennek az igénye többek között azért merült fel, mert előfordulhat, hogy több kimenet is helyes. Ilyen esetben a példa bemeneti fájlból és a versenyző forráskódja által előállt kimeneti fájlból kellene megállapítani, hogy megfelelő-e az előállt kimenet. Erre a GNU Diffutils nem alkalmas.

Másik fejlesztési cél, hogy a felületen megjelenő táblázatokban lehessen szűrést, rendezést végezni. Ezáltal könnyebben lehetne keresni egy beküldött megoldást, vagy egy üzenetet.

A verseny közben olyan igény is felmerült, hogy a bíró ne csak válaszolni tudjon egy kérdésre, hanem lehetősége legyen egy-egy versenyzőnek üzenetet, vagy körlevelet küldeni anélkül, hogy ezt válaszként tenné. Ennek a megvalósítása azért lenne fontos, mert ha a bíró a verseny közben vesz észre hibát valamelyik feladatban, akkor egy ilyen körlevélben minden versenyzőt azonnal tudna értesíteni.

11. Felhasználói dokumentáció

A felhasználói dokumentáció segíti a felhasználókat, hogy könnyebben boldoguljanak az alkalmazás használatával. A felhasználói dokumentációt mindhárom felhasználótípus számára elkészítettem.

11.1. Adminisztrátor

A bejelentkezés az adminLogin.aspx oldalon történik. Bejelentkezés után a baloldali függőleges oszlopban különböző hivatkozásokat láthatunk. A hivatkozások jelentése, használata a következőkben kerül ismertetésre.

11.1.1. Select Contest

Ezen az oldalon lehet kiválasztani, hogy mely versenyről szeretnénk információkat kapni az oldal alján és tetején lévő keskeny sávban. Az oldal alján a verseny neve, és a versenyből még hátralévő idő látható, míg a tetején a versenyre újonnan beérkezett üzenetek és fájlok száma. Újonnan beérkezettnek számít az az üzenet, amely még nincs megválaszolva, illetve az a fájl, amely még nincs kiértékelve. Bármikor megváltoztathatjuk, hogy mely versenyt kívánjuk figyelni.

11.1.2. Contest

Ezen az oldalon van lehetőség új verseny létrehozására („Create Contest”), valamint meglévő verseny szerkesztésére. A bejelentkezett adminisztrátor által látható összes verseny megjelenik az oldalon.

Bármely versenyre kattintva egy új oldal jelenítődik meg a kiválasztott verseny adataival. A verseny bármely adata bármikor szerkeszthető, módosítható. Ez alól két kivétel van. Futó verseny közben nem módosíthatjuk a verseny típusát, hiszen akkor a módosítás előtt és után beérkezett forráskódok másképp lennének pontozva. Nem módosíthatjuk még a verseny kezdési idejét sem, mert ez gyakorlatilag azt jelentené, hogy egy folyamatban lévő versenyről úgy döntünk, hogy az korábban, vagy később kezdődött. Ez a versenyzőkkel szemben szintén nem lenne tisztességes. A befejezési idő szabadon módosítható verseny közben is, hiszen bármikor adódhat olyan esemény, ami miatt a versenyt korábban kell befejezni, vagy esetleg elhúzódik. Az automata kiértékelés és a verseny neve is szabadon változtatható, de az új névnek egyedinek kell lennie. Az oldal alján található néhány jelölőnégyzet, melyek kiválasztásával törölhetők bizonyos versenyre vonatkozó adatok az adatbázisból:

- Clarifications – A versenyhez kapcsolódó összes levelezés törlése.
- Runs – A versenyre beküldött összes forráskód törlése.
- Problems – A versenyhez kapcsoló problémák törlése a versenyből. A problémák nem kerülnek tényleges törlésre, csak a kiválasztott versenyen nem fognak látszani.
- Language – A versenyhez kapcsolódó programozási nyelvek törlése. A programozási nyelvek nem kerülnek tényleges törlésre, csak a kiválasztott versenyen nem fognak látszani.

A „Create Contest” gombra kattintva a lehetséges módosítások megtörténnek. Ha a kezdési idő és a verseny típusa nem módosítható, mégis megváltoztattuk az értéket, akkor erről semmilyen hibaüzenet nem keletkezik, egyszerűen ennek a két értéknek a megváltoztatása nem történik meg. A „Delete Contest” gombra kattintva az egész verseny törlésre kerül. Ez azt jelenti, hogy a versennyel együtt törlődik a hozzá kapcsolódó teljes levelezés, a beküldött forráskódok, a versenyzők, illetve megszűnik a verseny kapcsolata a problémákkal, programozási nyelvekkel, válaszokkal, adminisztrátorokkal, bírókkal.

A Contest oldalon a „Create Contest” választógombra kattintva, új verseny létrehozására válik lehetőség. Új verseny létrehozásakor beállíthatjuk a weblapon látható adatokat, ám ezeket az előbbiekben leírt módon bármikor szabadon megváltoztathatjuk.

11.1.3. Accounts

Az Accounts oldalon a felhasználók megtekintésére, és menedzselésére van lehetőség. A bal felső sarokban elhelyezkedő legördülő lista a versenyenkénti szűrésre szolgál. Az alatta található választógombok segítségével választhatunk, hogy a jelenleg bejelentkezett felhasználókat szeretnénk megtekinteni, vagy az összes felhasználót. A felhasználói típusokat tartalmazó választógombokkal pedig a felhasználók szerinti szűrés oldható meg Minden esetben egy táblázat fog megjelenni, amelyben a kért felhasználók szerepelnek. Ha a bejelentkezett felhasználókat választjuk, akkor a táblázatban lévő megfelelő „Log off” gombra kattintva az adott felhasználót kijelentkeztetjük a versenyből.

Amennyiben szerkeszteni szeretnénk a felhasználókat, a választógombok közül a „Manage Accounts”- ot kell kijelölni. A megjelenő táblázatban a „Delete” gombra kattintva a felhasználó törlése történik meg, míg az „Edit” gombot választva egy új oldal töltődik be, amelyen a kijelölt felhasználó adatait (felhasználói név, jelszó, mely versenyhez kapcsolódik) szerkeszthetjük. Versenyzők esetében további egy adatot módosíthatunk. Ez az adat a „Show on scoreboard”. Ha ez az érték igaz, akkor a versenyző látszódik az eredményjelző táblán, egyébként nem. Versenyző legfeljebb egy versenyhez, adminisztrátor és bíró akárhány versenyhez kapcsolható. A módosítások az „Update” gombra kattintva lépnek érvénybe.

Az Accounts oldalon található még egy gomb, amelyen a „Generate Accounts” felirat szerepel. Erre a gombra kattintva egy új oldal töltődik be, amelyen új felhasználók létrehozása lehetséges. Új felhasználót kétféleképpen hozhatunk létre. Az egyik lehetőség, amikor minden felhasználói típus mellé egy szám kerül, amely megmondja, hány darabot szeretnénk belőle létrehozni, majd a hozzátartozó listában kiválasztjuk, hogy mely versenyhez kapcsolódjanak a létrejövő felhasználók. Ezek után a „Generate” gombra kattintva legenerálódnak a megfelelő felhasználók. A másik lehetőség fájlfeltöltéssel működik. A megfelelő fájlt tallózással megkeressük, majd a „Load” gombra kattintva a fájl feltöltődik, és a benne lévő utasítások végrehajtódnak. Az utasítások kétfélék lehetnek, és a következőképpen kell kinézniük:

- meglévő felhasználó adatainak módosítása: Függőleges vonallal elválasztva meg kell adni a nevet, a felhasználói nevet, a jelszót, és versenyzők esetében egy `true` vagy egy `false` értéket, amely azt jelzi, hogy a versenyző látszódjon-e az eredményjelző táblán. Következő sorban függőleges vonallal elválasztva a versenyek neveit, amelyhez a felhasználó kötődik. Versenyzők esetében csak egy verseny adható meg. A sorok végére a függőleges vonalat nem kell kiírni.

```
név|felhasználói_név|jelszó[|{true|false}]
verseny1|verseny2|verseny3|…
```

- új felhasználó létrehozása: Szerkezetileg hasonlóan néznek ki a sorok, mint a felhasználó módosítása esetén, csak név helyett a „newteam”, „newjudge” vagy „newadmin” szavakat kell használni, attól függően, hogy új versenyzőt, bírót vagy adminisztrátort szeretnénk létrehozni.

```
{newteam|newjudge|newadmin}|felhasználói_név|jelszó[|{true|false}]
verseny1|verseny2|verseny3…
```

11.1.4. Activity

A felhasználók be- és kijelentkezése követhető nyomon ezen az oldalon. A legördülő lista a versenyenkénti szűrést, az alatta található választógombok pedig a felhasználói típusonkénti szűrést teszik lehetővé. A megjelenő táblázatban a bekövetkezett események időben csökkenő sorrendben szerepelnek.

11.1.5. Answered Clars

Az oldalon a már megválaszolt üzenetek tekinthetők meg. A felül található legördülő lista segítségével lehetséges a versenyenkénti szűrés. Az üzenetek ezen az oldalon egy táblázatban szerepelnek, bármelyiket kiválasztva új oldalon részletesen megjelenik a versenyző által küldött eredeti üzenet és a kapott válasz.

11.1.6. New Clars

A még meg nem válaszolt üzenetek tekinthetők meg ezen az oldalon. A felül található legördülő lista segítségével lehetséges a versenyenkénti szűrés. Az üzenetek ezen az oldalon egy táblázatban szerepelnek, bármelyiket kiválasztva új oldalon részletesen megjelenik a versenyző által küldött eredeti üzenet.

11.1.7. Runs

Ezen az oldalon a beküldött forráskódok adatait tartalmazó táblázat található. A táblázat fölött elhelyezkedő legördülő lista a verseny szerinti szűrésre szolgál. A táblázat bármely sorában a „Give/Take” gombra kattintva a forráskódot visszaadhatjuk a bíróknak újbóli kiértékelésre, illetve elvehetjük tőlük. Amikor egy forráskód visszaadódik, az azt jelenti, hogy nem kiértékeltnek számít, azaz nem befolyásolja az eredményjelző tábla adatait, de a forráskódra korábban kapott válasz nem változik meg, ezt csak valamelyik bíró módosíthatja újbóli kiértékeléssel. Amennyiben a forráskódot elvesszük a bíróktól, akkor kiértékelté válik, azaz befolyásolja az eredményjelző adatait, de a forráskódra kapott válasz nem változik meg. Ennek következtében előfordulhat olyan eset, hogy egy forráskód kiértékelt lesz, de a hozzá tartozó válasz még mindig „Running…”, az érte járó pontszám pedig 0. Erre az esetre különösen oda kell figyelni, és ha lehet, el kell kerülni.

A táblázatban a „View” gombra kattintva egy új oldalon megjelenítődik a kiválasztott forráskód tesztesetenkénti eredménye. Ez még mindig táblázatos forma, a táblázat egy-egy sora jeleníti meg, hogy a forráskód az adott tesztesetre hogyan működött. Ettől részletesebb eredményt a „View” gombra kattintva kaphatunk. Ekkor egy új lap töltődik be, amely az adott forráskód adott tesztesetre történő adatait mutatja. Itt a megfelelő gombokat kiválasztva bővebb információhoz jutunk.

- View Input Data File – A tesztesethez tartozó bemeneti fájlt jeleníti meg.
- View Answer File – A tesztesethez tartozó kimeneti fájlt jeleníti meg.
- View Source File – A beküldött forráskódot jeleníti meg.
- View Output – A forráskód által az adott tesztesetre előállított kimeneti fájlt jeleníti meg. Ha valamilyen hiba folytán nem létezik ez a fájl, akkor nem történik változás.
- Compare Output – Az előbb említett két kimeneti fájlt hasonlítja össze egy új weblapon. Ha nem létezik a forráskód által előállított kimeneti fájl, azaz az összehasonlítás nem lehetséges, akkor az új lapon egy üres szövegdoboz fog megjelenni.

11.1.8. Problems

Ezen az oldalon lehetőség van új problémák létrehozására, illetve már meglévők módosítására. Új probléma létrehozásához az „Add” gombra kell kattintanunk. Ekkor egy új weblap fog betöltődni, amelyen megadhatjuk a létrehozandó probléma nevét, a validátor típusát, feltölthetjük a probléma leírását, illetve beállíthatjuk, hogy mely versenyeken legyen látható. A CTRL gomb nyomva tartásával egyszerre több versenyt is kiválaszthatunk. A validátor típusa a következő lehet:

- diff – A hasonlítás karakterenként történik.
- ignore all whitespaces – Hasonlítás közben figyelmen kívül hagyja a szóköz, tabulátor, és soremelés karaktereket.
- ignore empty lines – Hasonlítás közben figyelmen kívül hagyja az üres sorokat.
- ignore upper/lower case – Hasonlítás közben nem tesz különbséget a kis- és nagybetűk között.

A megfelelő beállítások után az „Update” gombra kattintva hozhatjuk létre a problémát.

A Problems oldalon bármely problémát úgy szerkeszthetünk, hogy a táblázat megfelelő „Select” gombjára kattintunk. Ekkor az új probléma létrehozásakor látott weblaphoz hasonló fog megjelenni. Az összes korábban ismertettet beállítás megváltoztatható. Azonban egy eddig nem szereplő gomb is megjelenik ezen az oldalon „Testcases” felirattal. Ezt a gombot választva szerkeszthetjük a problémához tartozó teszteseteket. A tesztesetek táblázatos formában jelennek meg. A táblázat minden sorában szerepel „Edit”, illetve „Delete” gomb. Az utóbbival a tesztesetet törölhetjük, az előbbivel módosíthatjuk. A táblázat alatt található „Add” gombbal pedig új tesztesetet adhatunk a problémához. A tesztesetek szerkesztése, és létrehozása hasonlóan történik. A megjelenő weblapon beállíthatjuk a teszteset különböző tulajdonságait:

- Time limit – A futási idő maximális értéke másodpercben.
- Memory limit – A felhasznált memória értéke bájtban.
- Score – A teszteset sikeres teljesítésért járó pontszám, ha a verseny pontozáson alapul.
- Input Data File – A teszteset bemeneti adatait tartalmazó fájl.

- Answer File – A teszteset kimeneti adatait tartalmazó fájl.
- A választógombokkal pedig megadhatjuk, hogy honnan olvasson a program. Stdin esetén a bemenet és a kimenet is a szabványos bemenet lesz, azaz tesztelés során az input és az output átirányításra kerül. File esetén a program fájlból olvas, de a szabványos kimenetre ír.

Az „Update” gombbal véglegesíthetjük a változásokat. Ha a tesztesetet nem újonnan hozzuk létre, hanem módosítjuk, akkor csak azokat az adatokat kell átírni, amelyet módosítani szeretnénk, a többi marad a régi.

11.1.9. Language

Ezen az oldalon a programozási nyelvek beállításai adhatók meg. Egy összefoglaló táblázatban az összes lehetséges programozási nyelv megjelenik. Bármelyiket kiválasztva módosíthatjuk a fordítási parancssort (command line), a fordító helyét (compiler location) és azt a könyvtárat, ahová az alkalmazás dolgozhat (working directory). Ebbe a mappába töltődnek le a beküldött forrásfájlok és ide kerül a fordítás, futtatás, validálás eredményeként előálló fájl is. Ezek a fájlok csak ideiglenesek, validálás után automatikusan letörlődnek. Az oldal alsó részén található felsorolásból kiválasztható, hogy mely versenyen lesz elérhető a programozási nyelv. Egyszerre több versenyt is kijelölhetünk. Az „Update” gombra kattintva a változások véglegesítődnek. Fontos, hogy a programozási nyelv neve nem változtatható, és új programozási nyelv sem hozható létre.

11.1.10. Answer

Ezen az oldalon egy táblázatban látható az összes lehetséges válasz. A válasz melletti „Select”-re kattintva egy új oldalra kerülünk, ahol az adott válaszhoz hozzárendelhetjük a versenyeket. Egy válaszhoz több verseny is rendelhető. Az „Update” gombra kattintva a módosítás megtörténik. Fontos, hogy a „Running…”, „Yes - Acceted” és „No – Other unknown error” eredetileg minden versenyhez hozzá van rendelve, és ezek nem módosíthatók.

11.1.11. Settings

A felhasználói név és a jelszó megváltoztatására szolgáló oldal. A felhasználói név és a jelszó megváltoztatása egymástól függetlenül hajtható végre. Megváltoztatáskor az első sorba a régi adatot, a következő kettőbe az újat kell beírni, majd az alatta található gombra kat-

tintva, ha lehetséges, megtörténik a módosítás. A módosítás sikerességéről, illetve sikertelenség esetén a hiba vélhető okáról egy üzenet jelenik meg a megfelelő gomb mellett.

11.1.12. Scoreboard

Itt nézhetjük meg a versenyek jelenlegi állását. A legördülő listában választhatunk a megtekinteni kívánt versenyek között.

11.1.13. Logout

Erre a hivatkozásra kattintva kiléphetünk az alkalmazásból. Kilépést követően a bejelentkezési oldalra kerülünk.

11.2. Bíró

A bejelentkezés a judgeLogin.aspx oldalon történik. Bejelentkezés után a baloldali függőleges oszlopban különböző hivatkozásokat láthatunk. A hivatkozások jelentése, használata a következőkben kerül ismertetésre.

11.2.1. Select Contest

Ezen az oldalon lehet kiválasztani, hogy mely versenyről szeretnénk információkat kapni az oldal alján és tetején lévő keskeny sávban. Az oldal alján a verseny neve, és a versenyből még hátralévő idő látható, míg a tetején a versenyre újonnan beérkezett üzenetek és fájlok száma. Újonnan beérkezettnek számít az az üzenet, amely még nincs megválaszolva, illetve az a fájl, amely még nincs kiértékelve. Bármikor megváltoztathatjuk, hogy mely versenyt kívánjuk figyelni.

11.2.2. Answered Clars

Az oldalon a bejelentkezett bíró által megválaszolt üzenetek tekinthetők meg. A felül található legördülő lista segítségével lehetséges a versenyenkénti szűrés. Az üzenetek ezen az oldalon egy táblázatban szerepelnek, bármelyiket kiválasztva új oldalon részletesen megjelenik a versenyző által küldött eredeti üzenet és a bíró által adott válasz.

11.2.3. New Clars

A még meg nem válaszolt üzenetek tekinthetők meg ezen az oldalon. A felül található legördülő lista segítségével lehetséges a versenyenkénti szűrés. Az üzenetek ezen az oldalon

egy táblázatban szerepelnek, bármelyiket kiválasztva új oldalon részletesen megjelenik a versenyző által küldött eredeti üzenet, és lehetőség nyílik a megválaszolásra. A választ egyszerűen az alul látható üres szövegdobozba kell írni. A „To All” jelölőnégyzet bejelölésével az üzenet minden – az aktuális versenyben részt vevő – versenyző számára elküldésre kerül. Ha ez a négyzet nincs bejelölve, akkor csak az eredeti üzenetet küldő versenyző számára lesz kézbesítve a válasz. Az üzenet küldéséhez a „Send” gombra kell kattintani.

A válasz elküldése után az üzenet megválaszolt lesz, azaz az új üzenetek közül átkerül a már megválaszolt üzenetek közé. Így már csak az a bíró láthatja, aki megválaszolta.

11.2.4. My Judgements

Ezen az oldalon a bíró által kiértékelt forráskódok tekinthetők meg. A felül elhelyezkedő legördülő lista segítségével versenyenkénti bontásban nézhetjük meg a forráskódokat. A forráskódok táblázatos formában láthatók néhány rájuk vonatkozó adattal együtt. Ezek az adatok sorrendben a következők:

- a probléma neve, amire a forráskódot beküldték
- a beküldő versenyző felhasználói neve
- a beküldött fájl neve
- a programozási nyelv, amiben a forráskód készült
- a beküldés ideje
- a forráskódért kapott válasz
- a forráskódért kapott pontszám

Bármely sort kiválasztva, és a „View” gombra kattintva egy új oldalon az adott forráskód tesztesetenkénti eredményei láthatók szintén táblázatos formában. A táblázat egy-egy sora jeleníti meg, hogy a forráskód az adott tesztesetre hogyan működött. Ettől részletesebb eredményt a „View” gombra kattintva kaphatunk. Ekkor egy új lap töltődik be, amely az adott forráskód adott tesztesetre vonatkozó adatait mutatja. Itt a megfelelő gombokat kiválasztva bővebb információhoz jutunk.

- View Input Data File – A tesztesethez tartozó bemeneti fájlt jeleníti meg.
- View Answer File – A tesztesethez tartozó kimeneti fájlt jeleníti meg.
- View Source File – A beküldött forráskódot jeleníti meg.

- View Output – A forráskód által az adott tesztesetre előállított kimeneti fájlt jeleníti meg. Ha valamilyen hiba folytán nem létezik ez a fájl, akkor nem történik változás.
- Compare Output – Az előbb említett két kimeneti fájlt hasonlítja össze egy új weblapon. Ha nem létezik a forráskód által előállított kimeneti fájl, azaz az összehasonlítás nem lehetséges, akkor az új lapon egy üres szövegdoboz fog megjelenni.

11.2.5. Unjudged Runs

Ezen a weblapon a még ki nem értékelt forráskódok találhatók táblázatos formában. A táblázatban néhány adat szerepel a beküldött forráskódról:

- a probléma neve, amire a forráskódot beküldték
- a beküldő versenyző felhasználói neve
- a beküldött fájl azonosítója
- a programozási nyelv, amiben a forráskód készült
- a beküldés ideje

A táblázat fölötti legördülő lista a versenyenkénti szűrést teszi lehetővé. Az alatta található választógombokkal beállítható, hogy a legördülő listában kiválasztott verseny automatikus vagy manuális kiértékelésű legyen.

A táblázatban található „Autojudge” gombot választva a kiválasztott forráskód automatikusan kiértékelődik, átkerül a már kiértékelt megoldások közé. Mivel a bejelentkezett bíró erre az egy fájlra választotta az automata kiértékelést, ez olyan, mintha önmaga végezte volna el, tehát ő lesz az egyetlen bíró, aki a „My Judgements” oldalon láthatja a kiértékelt fájl eredményét.

Ha viszont részletesen is látni szeretnénk a kiértékelés folyamatát, akkor a „Request Run” gombot kell választani a táblázatban. Ezt a lehetőséget választva egy új oldal jelenik meg, amelyen egy táblázatban megtalálható a problémához tartozó összes teszteset. Egy tesztesetet kijelölve és az „Execute” gombra kattintva a beküldött forráskód lefordításra kerül, majd az adott tesztesetre lefuttatódik. A lefutást követően a táblázat „Answer” oszlopában megjelenik a tesztesetre kapott eredmény. Ezzel a lehetőséggel a forráskódot külön-külön minden tesztesetre le tudjuk futtatni. Egy másik megoldás, ha minden tesztesetre egyszerre szeretnénk lefuttatni. Ilyenkor nincs szükség a teszteset kiválasztására, az „Execute All”

gombra kattintva a forráskód a lefordítás után sorban minden tesztesetre lefuttatódik, és a táblázatban minden teszteset mellett megjelenik a rá adott válasz. Valamint a táblázat alatt található legördülő listában a teljes problémára javasolt válasz lesz látható. A legördülő lista más értékét kiválasztva ez bármikor felülbírálható. Az „OK” gombra kattintva a változások véglegesítődnek, azaz valójában ilyenkor rendelődik válasz a forráskódhoz.

A táblázat tetszőleges tesztesetének kiválasztása után a weblapon található további gombok használatával a forráskód adott tesztesetre vonatkozó fájljai – a „My Judgements” oldalhoz hasonlóan – megtekinthetők.

11.2.6. Settings

A felhasználói név és a jelszó megváltoztatására szolgáló oldal. A felhasználói név és a jelszó megváltoztatása egymástól függetlenül hajtható végre. Megváltoztatáskor az első sorba a régi adatot, a következő kettőbe az újat kell beírni, majd az alatta található gombra kattintva, ha lehetséges, megtörténik a módosítás. A módosítás sikerességéről, illetve sikertelenség esetén a hiba vélhető okáról egy üzenet jelenik meg a megfelelő gomb mellett.

11.2.7. Scoreboard

Itt nézhetjük meg a versenyek jelenlegi állását. A legördülő listában választhatunk a megtekinteni kívánt versenyek között.

11.2.8. Logout

Erre a hivatkozásra kattintva kiléphetünk az alkalmazásból. Kilépést követően a bejelentkezési oldalra kerülünk.

11.3. Versenyző

A bejelentkezés a teamLogin.aspx oldalon történik. Bejelentkezés után a főoldal (Main) töltődik be. Ezen az oldalon csak néhány információ látható:

- a verseny neve, amelyen a versenyző részt vesz
- a versenyző által beküldött forrásfájlok száma
- a versenyző által tökéletesen megoldott feladatok száma

Minden oldalon a baloldali függőleges oszlopban különböző hivatkozásokat láthatunk. A hivatkozások jelentése, használata a következőkben kerül ismertetésre.

11.3.1. Submit

Ezen az oldalon van lehetőég a forrásfájlok beküldésére. A felső legördülő listából kiválaszthatjuk a problémát, amelyre be szeretnénk küldeni a fájlt. A kiválasztás után a legördülő lista mellett megjelenik egy hivatkozás a feladat szövegére, amennyibe a feladathoz van leírás feltöltve. Ha nincs ilyen, akkor alul egy figyelmeztetés látható, amely arról tájékoztat, hogy a feladathoz nem létezik leírás.

A második legördülő listában a programozási nyelvet választhatjuk ki. Az alatta található fájlfeltöltő segítségével pedig megkereshetjük és feltölthetjük a forrásfájlt. Fontos, hogy egy problémához egyszerre egy fájl tölthető fel, abban kell lennie a teljes megoldásnak. A tényleges feltöltés a „Send” gombra kattintva történik meg. A feltöltés sikerességéről egy alul megjelenő üzenet tájékoztat. Fájlt csak folyamatban lévő verseny esetén lehet feltölteni.

11.3.2. Runs

A beküldött forráskódok tekinthetők meg. Táblázatos formában néhány információ látható a forrásfájlról:

- a probléma neve, amelyre a fájl beérkezett
- a beküldés ideje
- a beküldött fájl neve
- ki van-e már értékelve
- a kapott válasz

A versenyzők ettől többet nem láthatnak az általuk beküldött forráskód eredményéről.

11.3.3. Clarifications

Ez az oldal a versenyző levelezését jeleníti meg. A versenyző által küldött, és a számára kézbesített levelek egy táblázatban jelennek meg. A táblázat oszlopai sorrendben a következők:

- a probléma neve, amelyre a kérdés vonatkozik
- a kérdés első pár karaktere
- a kérdés küldésének ideje
- érkezett-e már válasz a kérdésre

- ha van válasz, akkor a válasz érkezésének ideje, egyébként üres mező
- ha ebben az oszlopban be van jelölve a négyzet, az azt jelenti, hogy a választ minden a versenyben részt vevő versenyző megkapta, egyébként csak az, aki a kérdést küldte

A táblázat bármely sorát kiválasztva egy új oldalon részletesen megjelenik az adott sorban szereplő kérdés és a rá kapott válasz, amennyiben létezik. Ezen az oldalon a kérdés és a válasz teljes szövege látható.

Ha a táblázat alatti „New Question” gombra kattintunk, új kérdést küldhetünk. A kérdés küldése egy új lapon történik. Egy legördülő listával állíthatjuk be, hogy mely problémához kapcsolódóan szeretnénk kérdezni. Ha a versennyel kapcsolatos általános kérdést kívánunk küldeni, akkor a legördülő listában válasszuk a „General” lehetőséget. Az alul található szövegdobozba írhatjuk a kérdésünk, majd a „Send” gombra kattintva megtörténik a kézbesítés.

11.3.4. Settings

A felhasználói név és a jelszó megváltoztatására szolgáló oldal. A felhasználói név és a jelszó megváltoztatása egymástól függetlenül hajtható végre. Megváltoztatáskor az első sorba a régi adatot, a következő kettőbe az újat kell beírni, majd az alatta található gombra kattintva, ha lehetséges, megtörténik a módosítás. A módosítás sikerességéről, illetve sikertelenség esetén a hiba vélhető okáról egy üzenet jelenik meg a megfelelő gomb mellett.

11.3.5. Scoreboard

Itt nézhetjük meg a versenyek jelenlegi állását. A legördülő listában választhatunk a megtekinteni kívánt versenyek között.

11.3.6. Logout

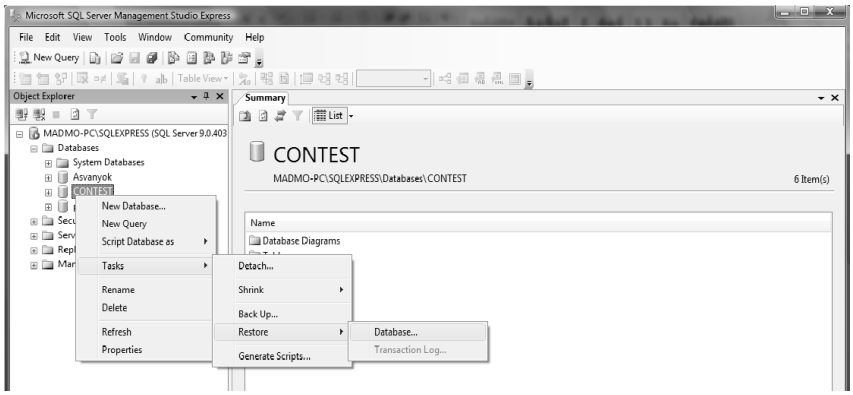
Erre a hivatkozásra kattintva kiléphetünk az alkalmazásból. Kilépést követően a bejelentkezési oldalra kerülünk.

11.4. Telepítés

A telepítés több részből áll, hiszen el kell készíteni az adatbázist, az alkalmazás számára meg kell mondani, hogy hol érheti el az adatbázist, amennyiben nincs feltelepítve a

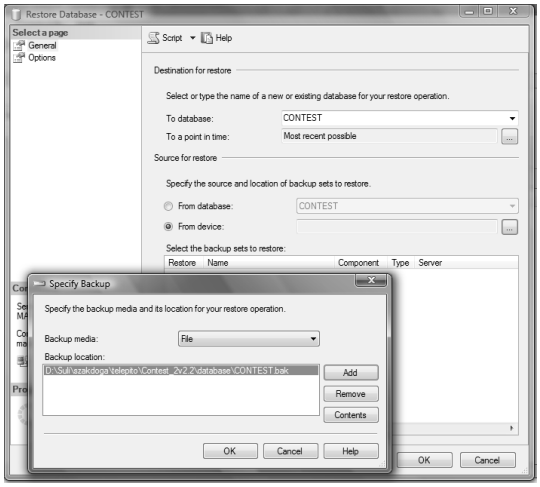
GNU Diffutils, és a használni kívánt fordítók, azokat föl kell telepíteni, valamint be kell állítani a web szervert.

- Ha még nincs legalább 2.0 verziójú .NET a szervernek szánt gépen, akkor szükséges a feltelepítése. A telepítő CD-n a .NET mappában megtalálható a 32 bites, és a 64 bites operációs rendszerre szánt .NET 2.0, valamint a .NET 3.5 is.
- Ezután gondoskodni kell adatbázis kezelő szoftverről. A telepítő CD-n az SQLServer2005 nevű mappában a Microsoft SQL Server Management Studio Express 2005 (SSMSE) 32 bites és 64 bites telepítője is megtalálható. Amennyiben Windows Vista operációs rendszert használunk a „User Account Control”-t (UAC) a telepítés idejére ki kell kapcsolnunk, mert bekapcsolt állapotba nem engedi feltelepíteni a szoftvert.
- Miután rendelkezünk adatbázis kezelő szoftverrel, elkészíthetjük az adatbázist. Indítsuk el az előző lépésben telepített SSMSE-t, és Windows autentikációval jelentkezzünk be. A baloldalon megjelenő Object Explorerben jobb gombbal a Database elemre kattintva válasszuk a „New Database”lehetőséget. Az új adatbázis neve kötelezően „CONTEST” legyen. A létrejött adatbázisra jobb gombbal kattintva válasszuk Task→Restore→Database lehetőséget.



14. ábra - Task->Restore->Database

- A felbukkanó ablakon válasszuk a „From Device”-t, majd adjuk hozzá a backup fájl elérési útját. A backup fájl a telepítő CD Contest_2v2.2/database mappában található CONTEST.bak fájl. Ekkor elkészül a kezdeti adatbázis.



15. ábra - Backup fájl kiválasztása

- Következő lépésként az IIS-t kell feltelepíteni. Az IIS valójában egy Windows komponens. A „Vezérlőpult/Programok telepítése és törlése” ablak baloldalán elhelyezkedő „Windows összetevők hozzáadása vagy eltávolítása” lehetőséget választva az új ablakban megjelenő listában ki kell választani az IIS-t, majd az OK gombbal menteni a módosításokat.
- A feltelepült IIS-ben létre kell hozni egy új web site-ot, amelynek a virtuális könyvtára a telepítő cd-n található Contest_2v2.2 mappa. Az IIS-en belül egyéb beállításokat is elvégezhetünk.
- Utolsó lépésként a Contest_2v2.2 mappában található web.config fájlban a ConnectionStringet úgy kell módosítani, hogy az aktuális adatbázis elérési útját tartalmazza.

```
<connectionStrings>
  <add name="CONTESTConnectionString"
       connectionString="Data Source=merlin\sqlexpress;Initial Catalog=CONTEST;Integrated Security=SSPI"
       providerName="System.Data.SqlClient" />
</connectionStrings>
```

- Ha még nincs feltelepítve a GNU Diffutils, akkor a telepítő CD-ről a GNU Diffutil mappában található fájllal telepítsük fel.
- A versenyen használni kívánt fordítókat szintén telepítsük fel, és az adminisztrátornak a webes felületen minden fordító elérési útját be kell állítania.

11.5. Szükséges hardver és szoftver

A versenyt futtató szerver gépen lennie kell legalább 2.0-as verziójú .NET-nek, valamint az előző részben leírt szoftvereknek: SSMSE, IIS.

Minden számítógépen, amiről a webes felületet el szeretnénk érni lenni kell valamilyen böngészőnek. Ajánlott böngészők Microsoft Internet Explorer 7+, Mozilla Firefox 3+.

12. Összefoglalás

A középiskolások és egyetemisták között egyre elterjedtebb, hogy különböző versenyeken mérik össze tudásukat. A szervezők egyik feladata, hogy megfelelő versenyzési körülményeket teremtsenek. Ehhez azonban áttekinthető, könnyen kezelhető szoftverre van szükség. Egyetemi éveim alatt én is több programozói versenyen részt vettem mind versenyzőként, mind szervezőként. Így a verseny mindkét oldaláról megtapasztaltam, hogy a jelenleg használt programtesztelő szoftvereknek melyek az erősségeik és a hiányosságaik. Ezek után döntöttem úgy, hogy megpróbálok egy olyan alkalmazást készíteni, amely a megismert erősségeket egyesíti. A szakdolgozat célja egy ilyen alkalmazás elkészítése volt.

Az elkészítendő alkalmazás fő funkciójaként azt tűztem ki, hogy lehetővé tegye a versenyzők számára forrásfájlok beküldését, és a beküldött kódokat automatikusan, vagy a verseny bírói által kiértékelje. A fő funkciót sikerült megvalósítanom, a kipróbálás során működőképes volt. Ezen kívül egyéb kiegészítésekkel is bővítettem.

Ilyen kiegészítés volt a levelezőrendszer, amely lehetővé teszi, hogy a versenyzők a felületen keresztül kérdéseket küldhessenek a bíróknak, akik erre hasonlóképpen válaszolnak. Mindkét oldalon újabb üzenet érkezésére az alkalmazás felhívja a figyelmet, így a felhasználók hamar értesülnek az új információkról.

Szintén kiegészítés, hogy a beküldött forrásfájlokat a bírók és az adminisztrátorok a felületen keresztül bármikor visszanézhetik. Ez akkor fontos, ha valamelyik versenyzőnek kérdése van egy korábban beküldött forráskódjához. Ezt a funkciót az április 19-ei versenyen is használtuk. Volt olyan versenyző, aki a verseny után kíváncsi volt rá, hogy mi a hiba az általa beküldött forrásfájlban, és így egyszerűen visszakereshettük. Nemcsak a forráskódot, de az egyes tesztesetekre adott kimeneteket is láthattuk.

Összességében a kitűzött célt úgy érzem, sikeresen megvalósítottam, az alkalmazás az éles kipróbálás során megállta a helyét. Jövőbeni terveim között szerepel a távolabbi céloknál megfogalmazottak megvalósítása, a szoftver további tökéletesítése. Szeretném, ha az általam készített alkalmazást a jövőben ténylegesen használnák, és ezáltal segítséget nyújthatnék a programozói versenyek szervezőinek.

13. Irodalomjegyzék

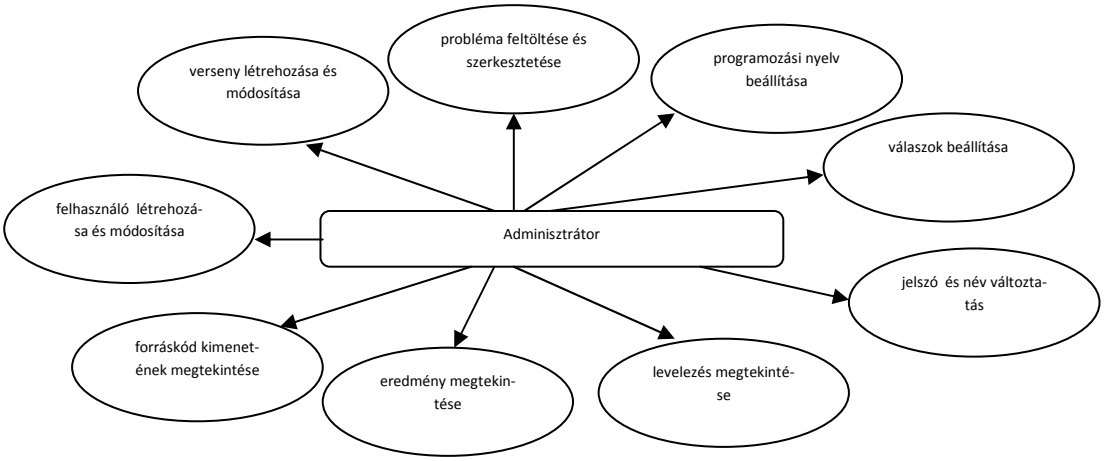
- John Sharp: Microsoft Visual C# 2005 lépésről lépésre, SZAK Kiadó, 2005
- Bill Evjen – Scott Hanselman – Devin Rader: Professional ASP.NET 3.5 In C# and VB, Wiley Publishing, 2008
- Dr. Kovács Emőd – Hernyák Zoltán – Radványi Tibor – Király Roland: A C# programozási nyelv a felsőoktatásban, Programozás tankönyv
- MSDN Library for Visual Studio 2008 - ENU

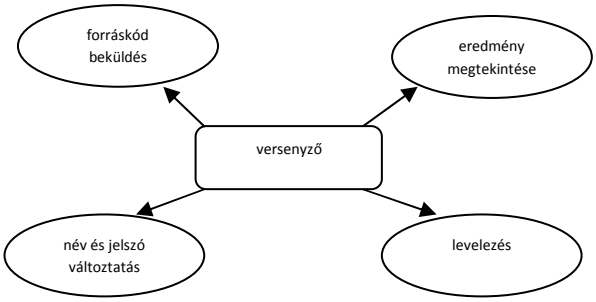
Internetes hivatkozások:

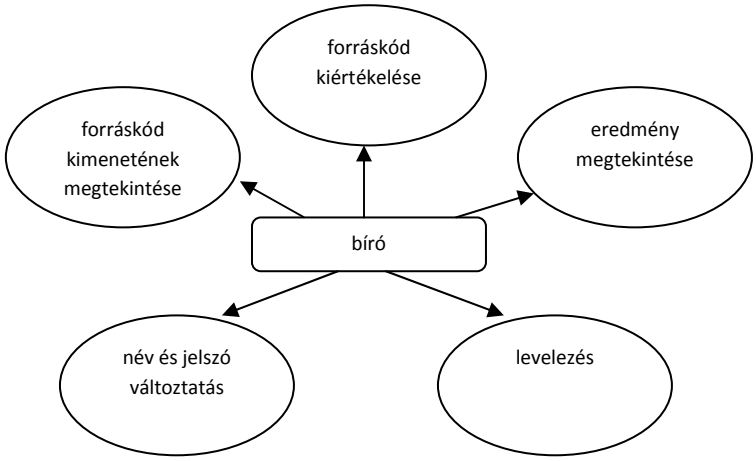
- http://msdn.microsoft.com/en-us/vbasic/bb688084.aspx
- http://msdn.microsoft.com/en-us/library/kx37x362.aspx
- http://www.asp.net/ajax/ajaxcontroltoolkit/Samples/
- http://www.architekturaforum.hu/blogs/inovak/archive/tags/LINQ/default.aspx

14. Függelék

14.1. Felhasználói esetdiagramok

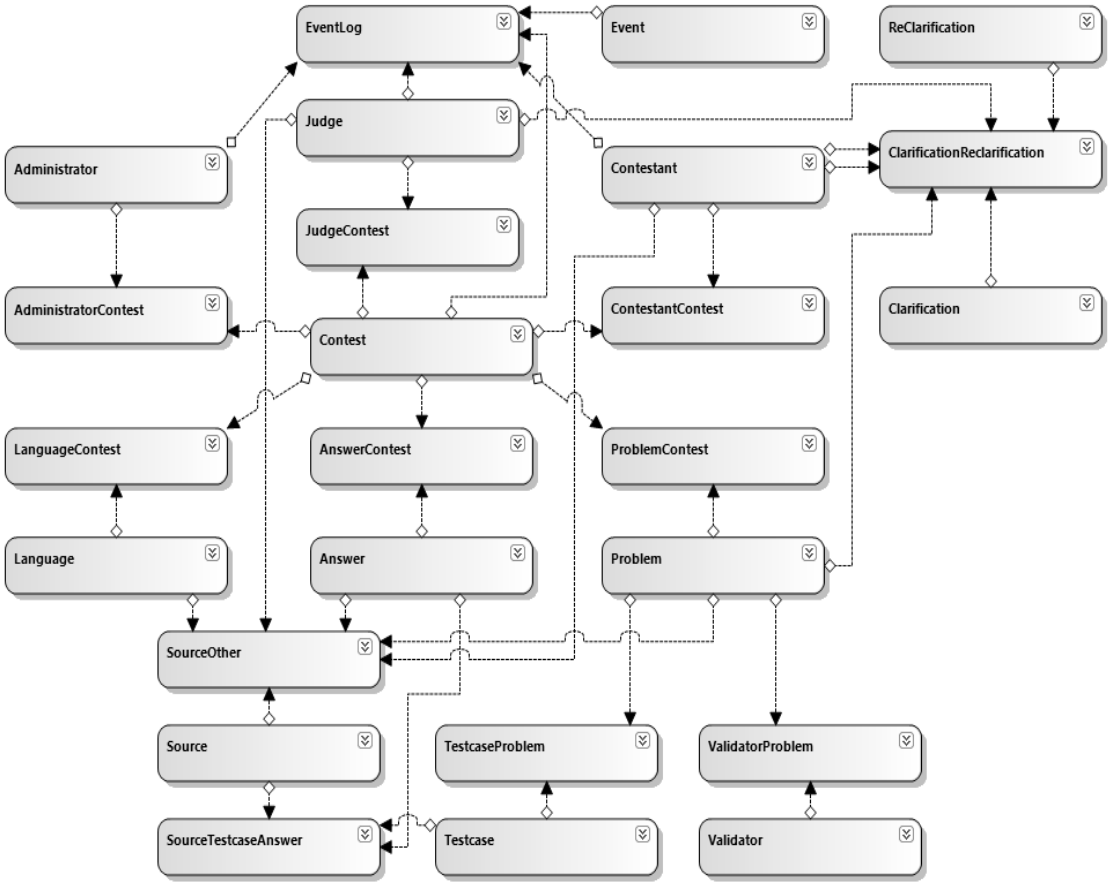




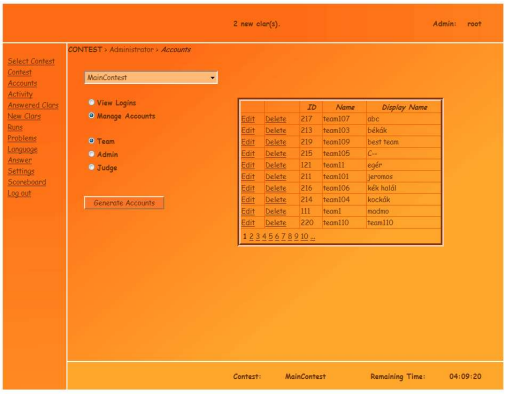


14.2. Adatbázis

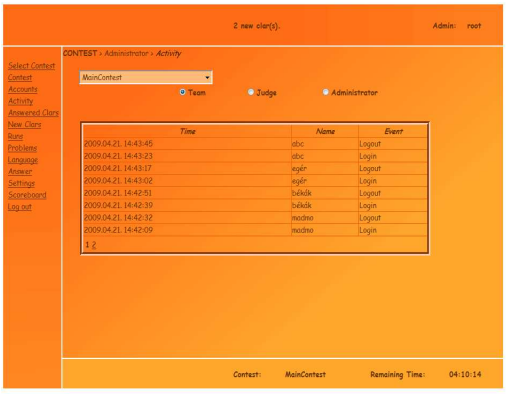
Táblák és kapcsolataik



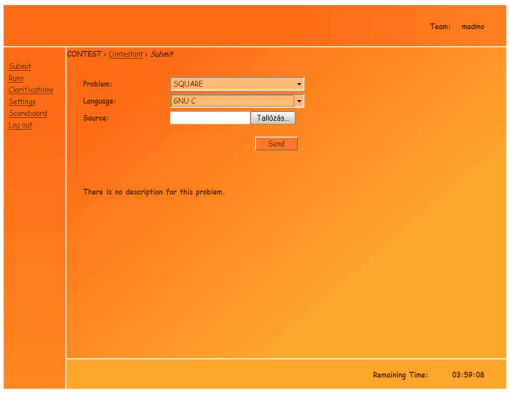
14.3. Képek az alkalmazásról működés közben



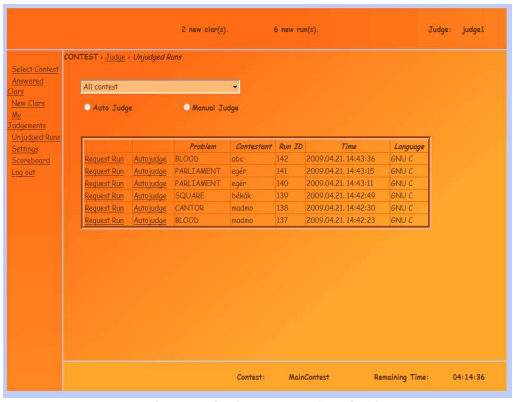
1. kép - Felhasználók menedzselése



2. kép - Bejelentkezések megtekintése



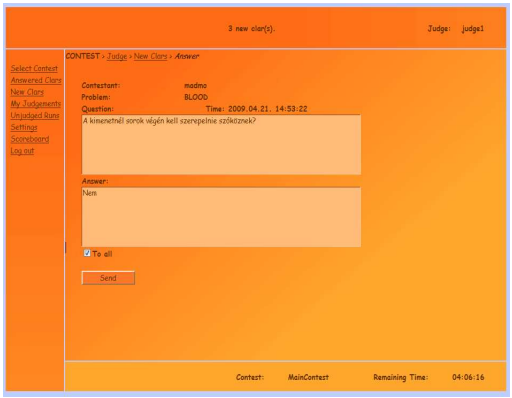
3. kép - Forrásfájl beküldése



4. kép - Kiértékeletlen forrásfájlok

5. kép - Saját eredmények megtekintése

6. kép - Beérkezett forráskódok (admin)



7. kép - Kérdés megválaszolása

8. kép - Eredményjelző tábla

15. Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani mindazoknak, akik lehetővé tették ezen szakdolgozat elkészítését.

Külön köszönetet mondanék témavezetőmnek, Kósa Márknak a nyújtott segítségért, és az ötletekért, amelyeket az alkalmazás megvalósítása közben adott. Illetve azért, hogy lehetővé tette az alkalmazás valódi versenyen való kipróbálását.

Meg szeretném köszönni Pánovics Jánosnak a valódi versenyen való kipróbálás során nyújtott segítségét.