SZAKDOLGOZAT

Koós Dániel

Debrecen

2009

Debreceni Egyetem
Informatikai Kar

Webes alapú automatikus programtesztelő rendszer fejlesztése

Témavezető:
Kósa Márk
egyetemi tanársegéd

Készítette:
Koós Dániel
mérnök informatikus

Debrecen
2009

Tartalomjegyzék

Köszönetnyilvánítás

Köszönettel tartozom Kósa Márk egyetemi tanársegédnek az egyetemi éveim alatt nyújtott állandó támogatásáért és folyamatos segítőkészségéért. Szakmai irányítása és útmutatásai nélkül dolgozatom nem készülhetett volna el.

Hálával tartozom Espák Miklós egyetemi tanársegédnek is, akinek a segítségével bepillantást nyerhettem a tesztkörnyezet működésébe, és akihez bármilyen kérdéssel és problémával is fordultam, mindig a rendelkezésemre állt.

Végül, de nem utolsó sorban köszönetet mondok azoknak a hallgatóknak, akik visszajelzéseikkel elősegítették az alkalmazásokban rejlő hibák kijavítását.

1. Bevezetés

A Debreceni Egyetem Informatikai Karán több éve sikerrel alkalmaznak automatikus tesztelő szoftvereket a hallgatók tudásának számonkéréséhez. Erre szükség is van, hiszen egyes alapozó tárgyak (például Magas szintű programozási nyelvek 1 és 2) kurzusaira egy félévben több szakról több száz hallgató jelentkezik. Például a 2007/08 tanév 2. félévében a Magas szintű programozási nyelvek 1 tárgyat mintegy négyszáz, gyakorlati aláírással nem rendelkező – mérnök informatikus, gazdasági informatikus és programtervező informatikus – hallgató vette fel, amelyhez 20 gyakorlati csoport társult. A 2009/10 tanév 1. félévében a Magas szintű programozási nyelvek 2 tárgy statisztikai mutatói közel azonosak, így a hallgatók ilyen nagy tömegének számonkérése ma is kihívást jelent. (Voltak félévek, amikor még ennél is több hallgató tudását kellett számon kérniük az oktatóknak.)

Egy félév során a hallgatóknak meg kell írniuk két zárthelyi dolgozatot, és otthon el kell készíteniük három házi feladatsort, amelyet az egyetemi tesztelő rendszeren keresztül kell beadniuk. A beadott munkák kiértékelését a rendszer automatikusan végzi, a másolatokat pedig plágium-ellenőrző szoftver segítségével lehet kiszűrni.

A ma használatos tesztelő rendszernek több elődje is volt. Az első próbálkozás a 2005/06-os tanévben bevezetésre került, Gunda Lénárd által fejlesztett, eredetileg ACM versenyek lebonyolításánál használt PCRM nevű rendszer volt, amely e-mailben várta a megoldásokat, és ez alapján végezte a tesztelést és a kiértékelést. Az ingyenes levelező rendszerek használata és a rendszer zártsága miatt azonban ez a megoldás nem váltotta be a hozzá fűzött reményeket.

Ezután a máig használt rendszer fejlesztéséhez Espák Miklós fogott hozzá. A rendszer tervezésénél az alapvető szempontok a következők voltak: egységesség, nyitottság, hordozhatóság és rugalmasság. A fejlesztett rendszer egységes és nyitott, hiszen a hallgatók és az oktatók ugyanazt a tesztelőt használják a programok ellenőrzésére, és a tesztelés menetét a hallgatók is ismerik. A rendszer segítségével a hallgatók otthon saját tesztkörnyezetet állíthatnak fel, és a nyilvános tesztesetek segítségével tudják ellenőrizni programjaik helyességét. Végezetül, ami talán a legfontosabb, a rendszer rugalmasan módosítható a később felmerülő igényeknek megfelelően (ma már nem csak a Magas szintű programozási

nyelvek 1 és 2 tárgyakhoz, hanem a Mesterséges Intelligencia 1 és Programozás Labor tárgyakhoz kiadott feladatok teszteléséhez is ugyanezt a rendszert vehetik igénybe a hallgatók és oktatók egyaránt).

A rendszert a hallgatók a *hallg.inf.unideb.hu* címen elérhető, GNU/Linux operációs rendszert futtató szerverre SSH klienssel kapcsolódva használhatják – a bejelentkezéshez valamelyik támogatott kurzus felvétele szükséges. Az autentikáció LDAP alapú, a hallgatók a *directory.unideb.hu* címen regisztrált hálózati azonosítójukkal és jelszavukkal jelentkezhetnek be a rendszerbe. Bejelentkezés után bizonyos parancsok kiadásával tesztelhetik és adhatják be feladataikat, illetve kérdezhetik le az adott tárgyhoz tartozó eredményüket.

Ezen a parancssoros felhasználói felületen keresztül érik el a hallgatók és a gyakorlatvezetők is a tesztelő környezetet. Azonban amíg a gyakorlatvezetők hatékonyan, gyorsan elvégezhetik a megfelelő módosításokat és egyszerűen előállíthatják a házi feladatokhoz szükséges teszteseteket, addig a hallgatók szempontjából nézve egy idő után bonyolult lehet bizonyos parancsok újbóli begépelése, vagy gyakran több tíz parancs közül a megfelelő előkeresése.

Egyetemi tanulmányaim során magam is használtam a tesztkörnyezetet, és gyakran sok időmet felemésztette a parancsok előkeresése vagy újbóli begépelése, ha módosításokat kellett eszközölnöm a programokban, vagy ha egyszerre több programot is teszteltem.

Az is bonyolulttá tette a rendszer kezelését, hogy a teszteléshez a hallgatóknak egyszerre több alkalmazást kell használniuk. Az állományok szerverre másolásához valamilyen SCP (Secure Copy) kliens (például *WinSCP*), a szerverre kapcsolódáshoz egy SSH (Secure Shell) kliens (például *PuTTY*), a szerkesztéshez pedig valamilyen szövegszerkesztő vagy IDE szükségeltetik. Ezért elhatároztam, hogy készítek egy grafikus felhasználói felületet a már meglévő parancssoros környezet kiegészítésére, amelyben a fentebb említett funkciókat integrálom, és amellyel megkönnyítem a magam és a Linux parancssorától idegenkedő hallgatótársaim helyzetét. Szót tett követett, és 2008 szeptemberére a nyári szakmai ösztöndíjprogram keretein belül elkészítettem az *Automatikus Programtesztelő Kliens* első változatát, amely az egyszerűbb tesztelési funkciókat (tesztelés, beadás, eredmények megtekintése) a felhasználó számára grafikus felületen tette lehetővé. Azonban ebből az alkalmazásból még sok minden hiányzott (ugyanúgy szükség volt egy SCP kliens

használatára is, és az alkalmazás még csak korlátozottan volt konfigurálható). A pozitív fogadtatásnak köszönhetően mára a felhasználói visszajelzések alapján elkészült a dolgozatom tárgyát képező *Automatikus Programtesztelő Kliens* bővített változata, amely a teszteléshez szükséges fájlböngészőt, tesztelő felületet és szerkesztőt/nézőkét egy alkalmazásban ötvözi, és ami a felhasználó igényei szerint konfigurálható.

Azonban itt nem állt meg a fejlesztés. A grafikus felületű tesztelő rendszerben rejlő kiaknázatlan lehetőségeket meglátva 2009 nyarán a kliensprogram mellett elkészítettem az *Automatikus Programtesztelő Portál*t, amely bármilyen segédprogram használata nélkül, egyetlen webböngésző és aktív internetkapcsolat használatával elérhetővé teszi a tesztelő rendszer legfontosabb funkcióit, akárhol is legyen az ember, és bármilyen eszközt is használjon (legyen az asztali számítógép, notebook, netbook, okostelefon vagy PDA).

2. Automatikus Programtesztelő Kliens

Az *Automatikus Programtesztelő Kliens* az egyetem hallgatói szerverén (*hallg.inf.unideb.hu*) futó tesztkörnyezet grafikus kiegészítése, amely a parancssorból futtatható parancsokat és a rendelkezésre álló eszközöket a grafikus felhasználói felület által nyújtott eszközökön keresztül teszi elérhetővé a felhasználó számára. A grafikus felület használatával a több szavas (és olykor több soros) parancsok begépelése egyetlen kattintással helyettesíthető, ami a hallgatók számára egyszerűbbé, és átláthatóbbá teszi a programok tesztelését.

2.1. Platform, fejlesztői környezet

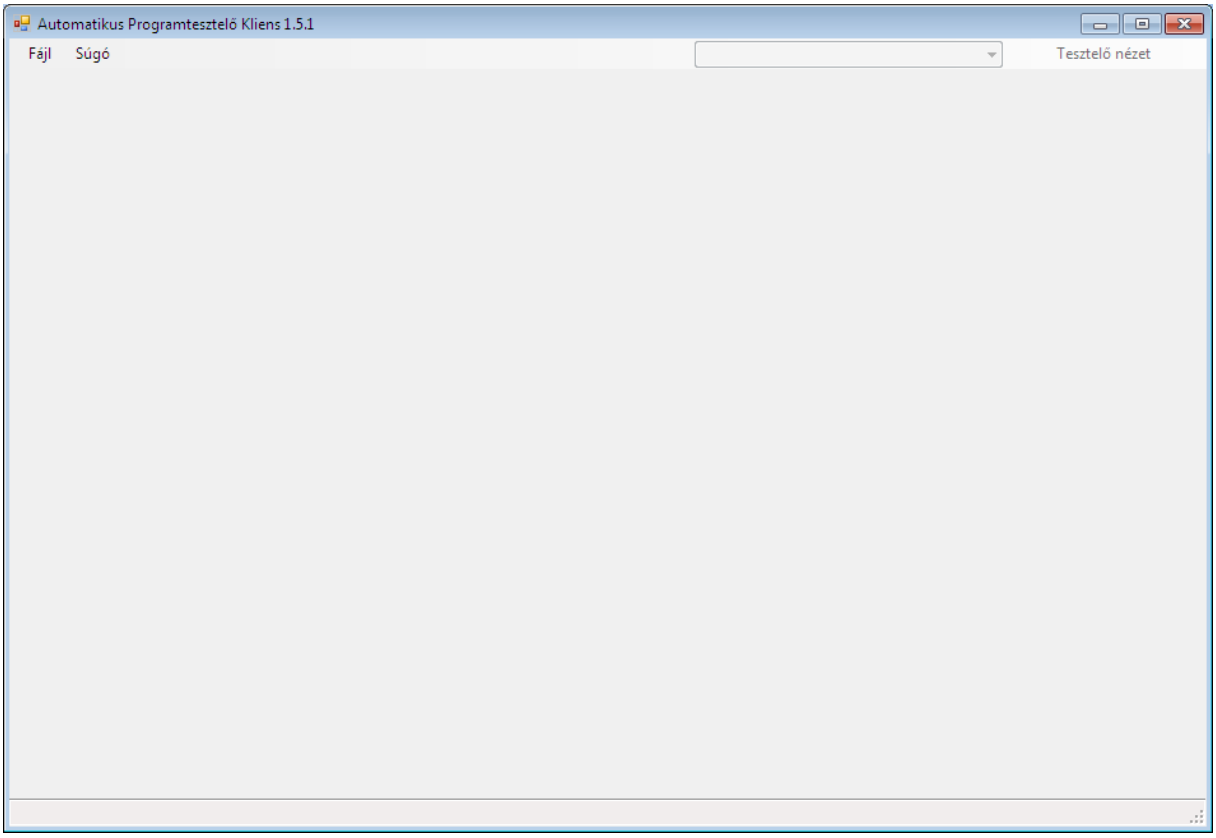
Az alkalmazás fejlesztése Windows platformon, Microsoft Visual Studio 2008 integrált fejlesztői környezet segítségével történt. Az alkalmazás futtatásához a .NET keretrendszer 3.5-ös verziója, a zökkenőmentes munkavégzéshez pedig legalább 1 Mb/s sebességű internetkapcsolat szükséges. Az alkalmazás Windows XP, Vista és 7 operációs rendszereken lett tesztelve, és mindhárom rendszeren teljes funkcionalitással használható.

2.2. A program felhasználói felülete

A program indítása után az 1. ábrán látható felület jelenik meg.

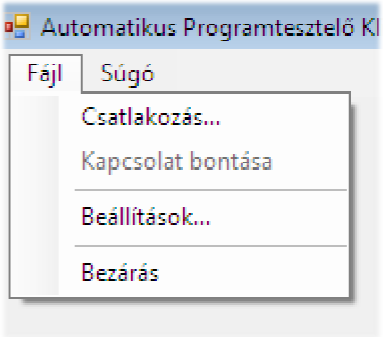
Amíg a felhasználó nem csatlakozik a szerverhez, addig ezzel a felülettel találja magát szemben, és csak a *Fájl* és *Súgó* menüpont tartalma áll a rendelkezésére. A program által nyújtott további funkciók csak a szerverhez történő csatlakozás után válnak elérhetővé.

A felhasználó ugyancsak ezzel a képernyővel találja magát szembe, hogyha kijelentkezik a szerverről, vagy megszakad a kapcsolat, esetleg ha valamilyen belső hiba történik. Utóbbi esetben az állapotsorban egy rövid tájékoztató üzenet is megjelenik, ami segít azonosítani a hiba okát.

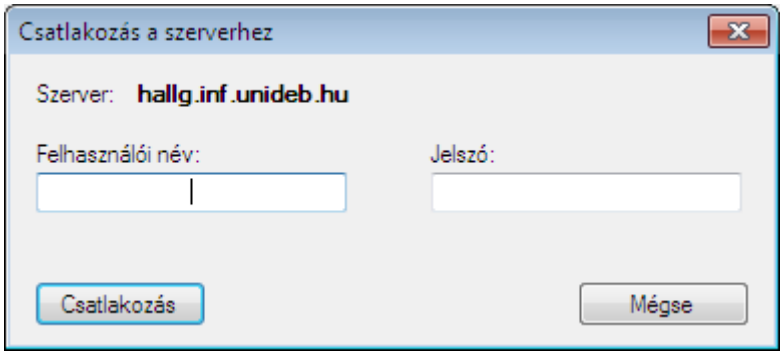


1. ábra: Az alkalmazás nyitó képernyője

A szerverhez történő csatlakozás a *Fájl* menü *Csatlakozás…* menüpontjának meghívásával kezdhető meg. Az itt megjelenő ablakban (3. ábra) kell megadni a hálózati azonosítót és a hozzá tartozó jelszót, majd a *Csatlakozás* gombra kattintva megkezdhető a kapcsolat felvétele a hallgatói szerverrel.

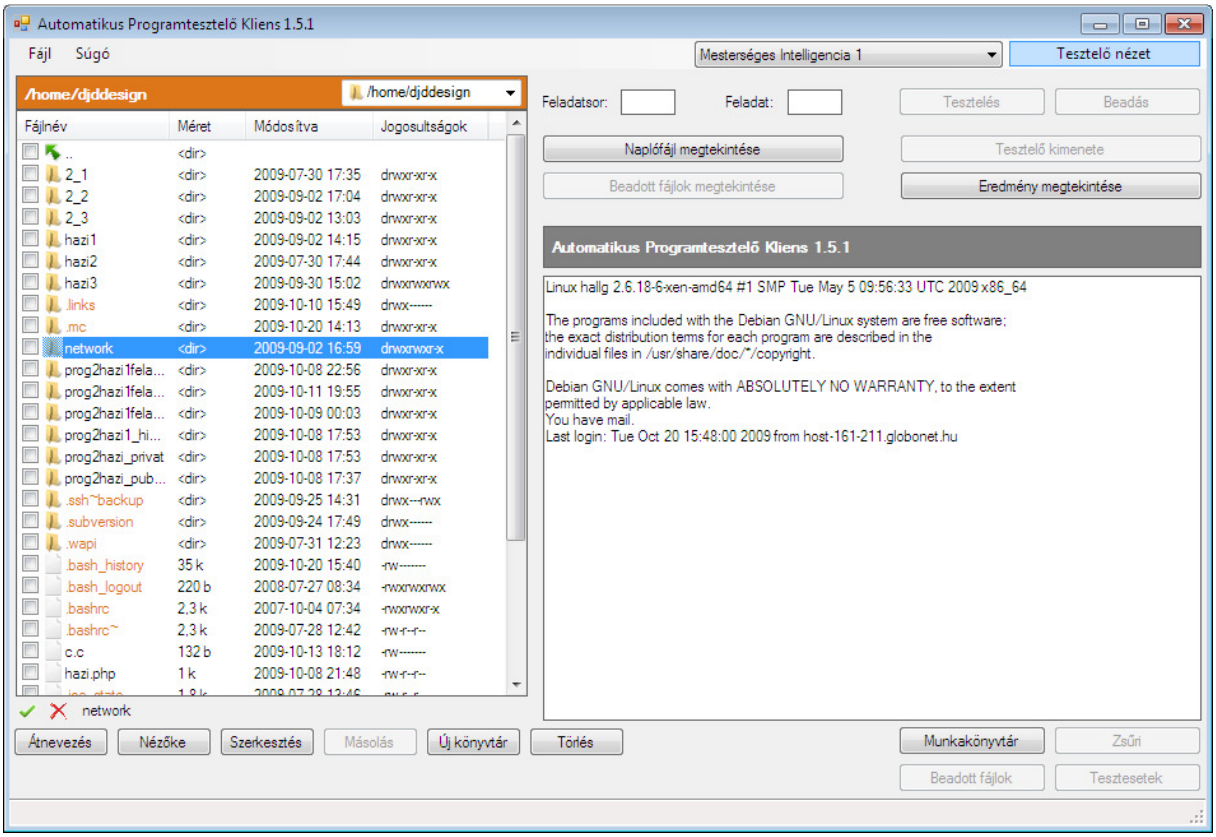


2. ábra: Fájl menü



3. ábra: Csatlakozás a szerverhez

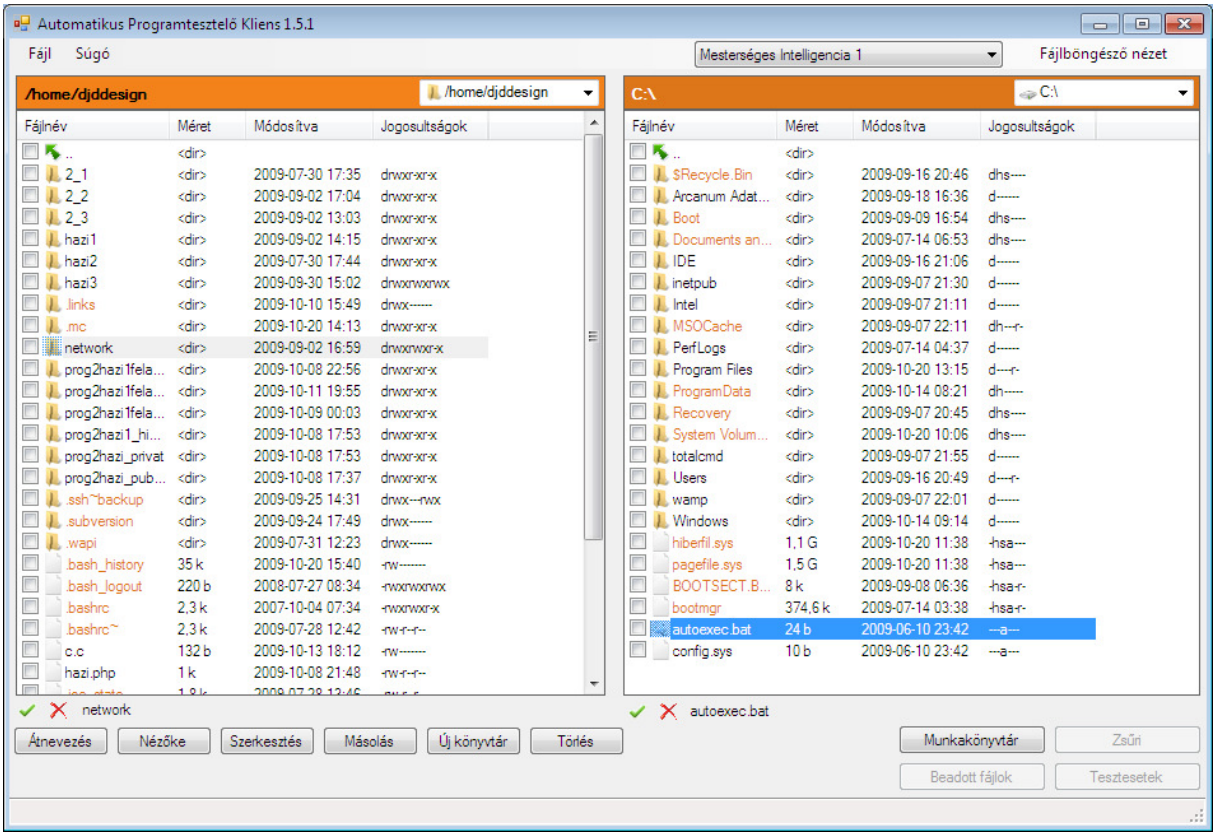
Sikeres bejelentkezést követően elérhető a program által nyújtott összes lehetőség. A programtesztelő felületen – az ún. tesztelő nézetben – (4. ábra) nyílik lehetőség a szerverre előzőleg feltöltött programok tesztelésére, a leggyakrabban használt könyvtárakba lépésre, illetve alapvetőbb fájl- és könyvtárműveletek elvégzésére.



4. ábra: Tesztelő nézet

A hallgatók a jobb felső sarokban levő legördülő menüből választhatják ki azt a tárgyat, amely tárgy házi feladatait szeretnék tesztelni. A legördülő menü mellett található a nézetváltó gomb, amellyel a *„Tesztelő nézet”* és *„Fájlböngésző nézet”* között válthat a felhasználó (5. ábra).

Fájlböngésző nézetben nyílik lehetőség az állományok mozgatására a helyi gép és a szerver között. A két panel fejlécében található legördülő menük segítségével gyorsan navigálhatunk a két fájlrendszerben, az ablak alján elhelyezkedő gombok segítségével pedig a legfontosabb fájlműveletek (átnevezés, szerkesztés, másolás, törlés) végezhetőek el.



5. ábra: Fájlböngésző nézet

A fájlok és könyvtárak kijelölése a nevük előtti jelölőnégyzet „bepipálásával” lehetséges, az adott könyvtárban levő összes állomány egyidejű kijelölését és a kijelölés megszüntetését a panelek alatt elhelyezkedő gombok teszik lehetővé. A gyorsabb munkavégzés érdekében a két fájlböngésző panelben levő állományok az oszlopok nevére kattintva tetszőlegesen rendezhetők.

2.3. A program felépítésének és szolgáltatásainak részletes ismertetése

2.3.1. Fájl menü

A *Fájl* menüben találhatóak az alkalmazás használatához szükséges legfontosabb menüpontok: *Csatlakozás…*, *Kapcsolat bontása*, *Beállítások…* és *Bezárás*.

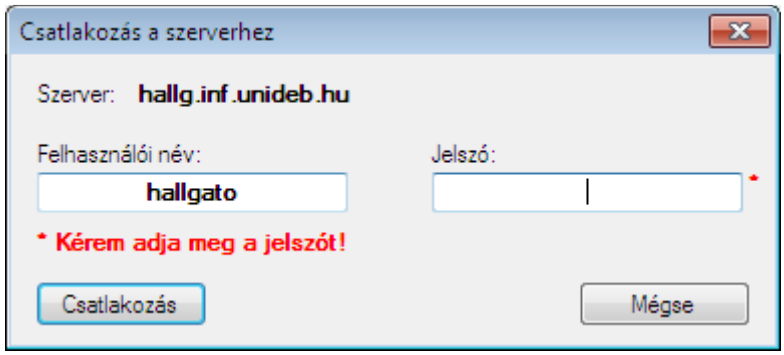
2.3.1.1. Csatlakozás…

A *Fájl* menü *Csatlakozás…* menüpontjának meghívására megjelenik az a form, amelyen megadhatjuk a felhasználói adatainkat, és csatlakozhatunk a szerverhez (3. ábra). A form (`ConnectForm.cs`) beépített validáló eszközökkel segít minden szükséges adat megadásában. A szerver címe rögzített, azt a program a `config.ini` állományban található bejegyzésből olvassa ki (6. ábra).

```
;konfigurációs fájl
;a fájl csak kommenteket, kulcs=érték párokat, és üres sorokat tartalmazhat
;különben szintaktikai hiba miatt a program nem indul el

;a szerver címe
server=hallg.inf.unideb.hu

;a minta, ameddig olvasunk
pattern=@hallg:|(igen\/\[nem\])
```

6. ábra: `config.ini`

Amennyiben a `server` kulcshoz tartozó érték, vagy maga a paraméter nem található a konfigurációs állományban, akkor a szerver címe helyett a `< Hiányzó szervercím! >` felirat jelenik meg, és a *Csatlakozás* gomb letiltásra kerül. Hasonló módon, ha rossz vagy hiányzó felhasználói nevet vagy jelszót adtunk meg, akkor azt a formon elhelyezett `errorLabel` vezérlőben megjelenített felirattal, és a problémás textbox mellett elhelyezett `*` karakterrel jelezzük, és amíg a hibát ki nem javítjuk, addig a csatlakozás művelete nem hajtható végre (7. ábra).



7. ábra: Hiányzó jelszó

A validálást a *Csatlakozás* gombhoz tartozó `connectOK_Click` eseménykezelő végzi (8. ábra).

{ConnectForm.cs}

```
...
private void connectOK_Click(object sender, EventArgs e)
{
    try
    {
        // Hibakezelés
        // ha nem adtunk meg felhasználóinevet
        if (usernameTextBox.Text.Trim() == String.Empty)
        {
            // hibajelző feliratok beállítása
            if (!errorLabel.Visible) errorLabel.Visible = true;
            if (!usernameError.Visible) usernameError.Visible = true;
            if (passwordError.Visible) passwordError.Visible = false;

            // a fókuszt a felhasználói névre állítjuk
            usernameTextBox.Focus();

            // beállítjuk a hibaüzenetet
            errorLabel.Text = "* Kérem adjon meg egy felhasználói nevet!";
        }
        // ha nem adtunk meg jelszót
        else if (passwordTextBox.Text.Trim() == String.Empty)
        ...
        else
        ...
    }
    ...
}
...
```

8. ábra: Felhasználói adatok validálása

Amennyiben minden adatot helyesen adtunk meg, akkor átadjuk a bejelentkezéshez szükséges paramétereket a `ConnectDataDelegate` callback delegate-nek (9. ábra), és bezárjuk az ablakot.

```
{ConnectForm.cs}
...
public ConnectDataDelegate ConnectDataCallback;
...
private void connectOK_Click(object sender, EventArgs e)
{
   try
   {
        // Hibakezelés
        // ha nem adtunk meg felhasználóinevet
        if (usernameTextBox.Text.Trim() == String.Empty)
        {
             ...
        }
        // ha nem adtunk meg jelszót
        else if (passwordTextBox.Text.Trim() == String.Empty)
        {
             ...
        }
        else
        {
             Cursor.Current = Cursors.WaitCursor;

             // Callback meghívása a login adatokkal
             if (ConnectDataCallback != null)
             {
                  ConnectDataCallback(serverAddressTextBox.Text,
                         usernameTextBox.Text, passwordTextBox.Text);
             }

             Cursor.Current = Cursors.Arrow;

             // Csatlakozás után az ablak bezárása
             Close();
        }
   }
   ...
}
...
```

9. ábra: Callback delegate meghívása a felhasználó adatokkal

A szerverhez csatlakozást a `MainForm` osztály `ConnectDataCallbackFn` metódusa, és az általa meghívott `BackgroundWorker` objektum végzi el (10. ábra).

A `BackgroundWorker` komponens alkalmazására azért van szükség az alkalmazásban, mert segítségével az időigényes feladatok egyszerűen egy háttérszálra helyezhetők, amíg a felhasználói felületen egyéb feladatokat végzünk el. Ezáltal a felhasználó számára is tudunk visszajelzést adni a folyamat állásáról (például a csatlakozási folyamat alatt az állapotsorban a `Csatlakozás folyamatban…` felirat jelenik meg, és ha sikeres volt a csatlakozás, akkor a szükséges vezérlők megjelenítésre kerülnek, és ez a jelzés eltűnik).

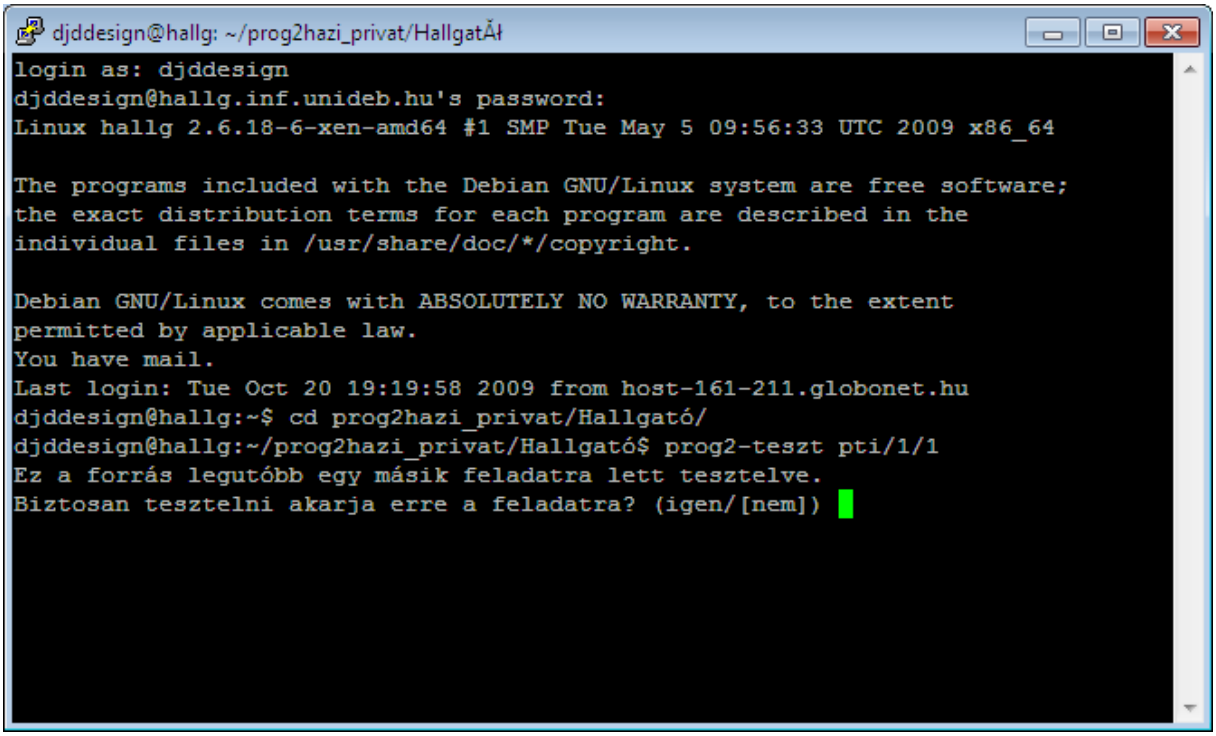
```
{MainForm.cs}
public delegate void ConnectDataDelegate(string host,
      string username,string password);
...
public partial class MainForm : Form
{
   private void ConnectDataCallbackFn(string serverAddress, string
                                      username, string password)
   {
      ...
      bw = new BackgroundWorker(){ WorkerSupportsCancellation = true };
      bw.DoWork += new DoWorkEventHandler(bw_DoWork);
      bw.RunWorkerAsync(serverAddress + "|" + username + "|" + password);
   }
   void bw_DoWork(object sender, DoWorkEventArgs e)
   {
       // a bejelentkezési adatok szerver|felhasználói név|jelszó alakban
       string item = (string)e.Argument;

       // a bejelentkezési adatok külön szegmensekben
       string[] login;

       try
       {
           // Csatlakozás során letiltjuk a kapcsolatkezelő gombokat
          textWriter(errorMessageStatusLabel,
                              "Csatlakozás folyamatban...");
          ...
          // Bejelentkezési adatok 'server|username|password' formában
          login = item.Split('|');

          // a változókat feltöltjük a bejelentkezési adatokkal
          host = login[0];
          user = login[1];
          pass = login[2];

          // példányosítjuk az SshShell objektumunkat
          ssh = new SshShell(host, user) { Password = pass };

          // csatlakozunk a szerverhez
          ssh.Connect();

          // beállítjuk a mintát
          ssh.ExpectPattern = pattern;
          ssh.RemoveTerminalEmulationCharacters = true;
```

```
            ...
        }
        catch (SocketException)
        {
            textWriter(errorMessageStatusLabel, "Hálózati hiba: Nem
                   sikerült kapcsolatot létesíteni a szerverrel.");
            ssh = null;
            ExceptionHandler();
            return;
        }
        ...
        try
        {
            // beolvassuk a szerver bejelentkező üzenetét
            string logintext = read();
            textWriter(mainTextBox, logintext.Substring(0,
                   logintext.LastIndexOf('\n') - 2));

            /*
             * Felhasználói csoportok beállítása
             */
            groupsComboBox.Invoke((MethodInvoker)delegate()
            {
                groupsComboBox.Items.Clear();
                foreach (Subject s in subjects)
                {
                    groupsComboBox.Items.Add(s.Targy_nev);
                }
                if (groupsComboBox.Items.Count > 0)
                {
                    groupsComboBox.SelectedIndex = 0;
                }
            });

            changeDir(mainTextBox, remoteFilesListView,
                                                    remoteHeaderLabel);
            enableChange(connectButton, false);
            enableChange(disconnectButton, true);

            textWriter(testLabel, form_title);

            Cursor.Current = Cursors.Arrow;
        }
        ...
        catch (Exception)
        {
            textWriter(errorMessageStatusLabel, "Hálózati hiba. Kérem
                        ellenőrizze az internetkapcsolatát!");
            ssh = null;
            ExceptionHandler();
                 return;
        }
    }
}
```

10. ábra: Csatlakozás a szerverhez

A `ConnectDataCallbackFn` metódus a paramétereiként megkapott felhasználói adatok felhasználásával, egy `BackgroundWorker` példány segítségével egy háttérszálon megkezdi a csatlakozást. – A `bw_DoWork` eseménykezelőben a második paraméterből (`DoWorkEventArgs e`) meghatározva a host címét, a felhasználói nevet és a jelszót csatlakozhatunk a szerverhez.

Az SSH kapcsolat létesítésére az `SshShell` osztály szolgál. A szerverhez a `Connect` metódus segítségével kapcsolódhatunk. Szükségünk van egy, reguláris kifejezésként megadott mintára (`pattern`), amely a `config.ini` állományban (6. ábra) megadott paraméter. A beolvasott üzenet végét ez a minta jelenti. Mivel jelen esetben a szerver címe alapján a shell promptban a `@hallg:` sztring szerepel, ezért ez kerül a mintába. Azonban akadnak olyan esetek is, amikor a szerver egy eldöntendő kérdésre várja a választ a felhasználótól, ekkor a kimenet végét az `(igen/[nem])` sztring jelzi (11. ábra), és ezt az esetet is kezelnünk kell. A `textWriter` metódus segítségével a hibaüzeneteket az állapotsorban levő `errorMessageStatusLabel` vezérlőn keresztül a felhasználóval is közöljük.



11. ábra: Üdvözlő szöveg a szerver bejelentkezése után

Sikeres csatlakozás után beolvassuk a szerver bejelentkező üzenetét, majd feltöltjük a tantárgyak listáját tartalmazó legördülő menüt (`groupsComboBox`) a program gyökérkönyvtárában levő `Subjects` alkönyvtárban található inicializáló állományok alapján. Ezután lekérjük és megjelenítjük a felhasználó munkakönyvtárában levő fájlokat és könyvtárakat a `remoteFilesListView` vezérlőben, majd a *Fájl* menü *Csatlakozás…* menüpontját inaktivizáljuk, a *Kapcsolat bontása* menüpontot pedig elérhetővé tesszük.

Ezek után megkezdődhet a tényleges munka, a feladatok tesztelése, beadása, az eredmények lekérdezése.

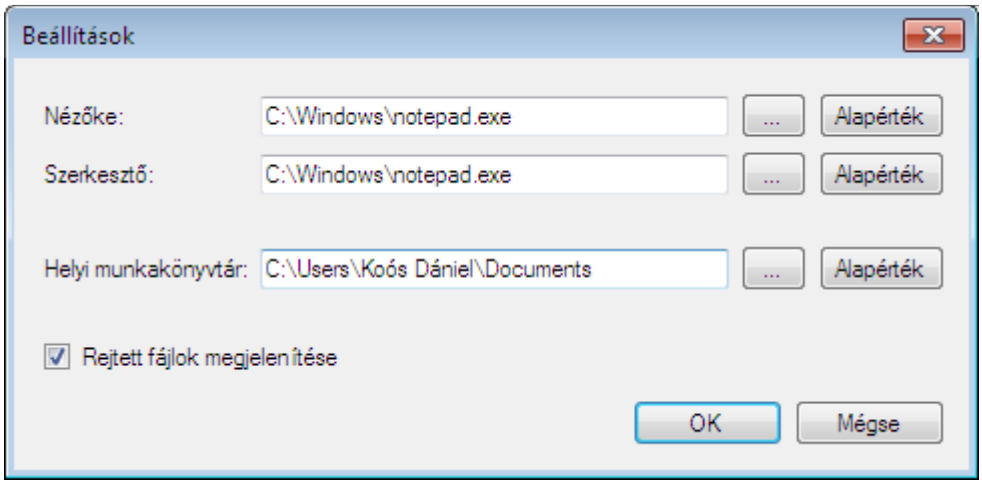
2.3.1.2. Kapcsolat bontása

Amennyiben minden feladattal végeztünk, ki kell jelentkeznünk a szerverről, amihez a *Fájl* menü *Kapcsolat bontása* menüpontját kell meghívnunk. Ekkor bontjuk a fennálló kapcsolatokat a szerverrel, majd elrejtjük azokat a vezérlőket, amelyek csak aktív kapcsolat esetén használhatóak. Így visszakapjuk a program indításakor látott képernyőt (1. ábra).

Bizonyos kivételek hatására az alkalmazás automatikusan kijelentkezik, és visszakapjuk a kezdőképernyőt. Amennyiben szeretnénk újból bejelentkezni a szerverre, ismét válasszuk a *Fájl* menü *Csatlakozás…* menüpontját. Ekkor az egyszer már használt felhasználói nevünket és jelszavunkat kínálja fel a rendszer, de ettől különböző felhasználó azonosítóval is bejelentkezhetünk. Amíg a programablakot be nem zárjuk, addig lehetőségünk van az eddig használt felhasználó név - jelszó párral bejelentkezni a szerverre.

2.3.1.3. Beállítások…

Az alkalmazáshoz tartozó néhány beállítást a *Fájl* menü *Beállítások…* menüpontja alatt változtathatjuk meg (12. ábra). Itt állíthatjuk be a használni kívánt nézőkét és szerkesztőt (alapértelmezés szerint ez a Windows beépített Jegyzettömb – Notepad – alkalmazása), megadhatjuk a helyi gépen levő munkakönyvtárunkat (alapértelmezett értéke a felhasználó *Dokumentumok* könyvtára), amelybe a *Fájlböngésző nézet*ben egy kattintással átléphetünk, és itt állíthatjuk be azt is, hogy a könyvtárak listázása során a rejtett fájlok is megjelenítésre kerüljenek-e.



12. ábra: Beállítások

2.3.2. Súgó

Az alkalmazással kapcsolatos legfontosabb információk (verzióinformáció, billentyűparancsok, a fejlesztő elérhetőségei) a *Súgó* menüponton keresztül érhetőek el.

2.3.3. Tesztelő nézet

Sikeres csatlakozás után az alkalmazás legfontosabb felülete, a *Tesztelő nézet* fogadja a felhasználót. Itt választható ki a tesztelendő feladat aktuális könyvtára, elvégezhetőek a feladatok tesztelésével és beadásával kapcsolatos műveletek, és megtekinthető a szerver válasza.

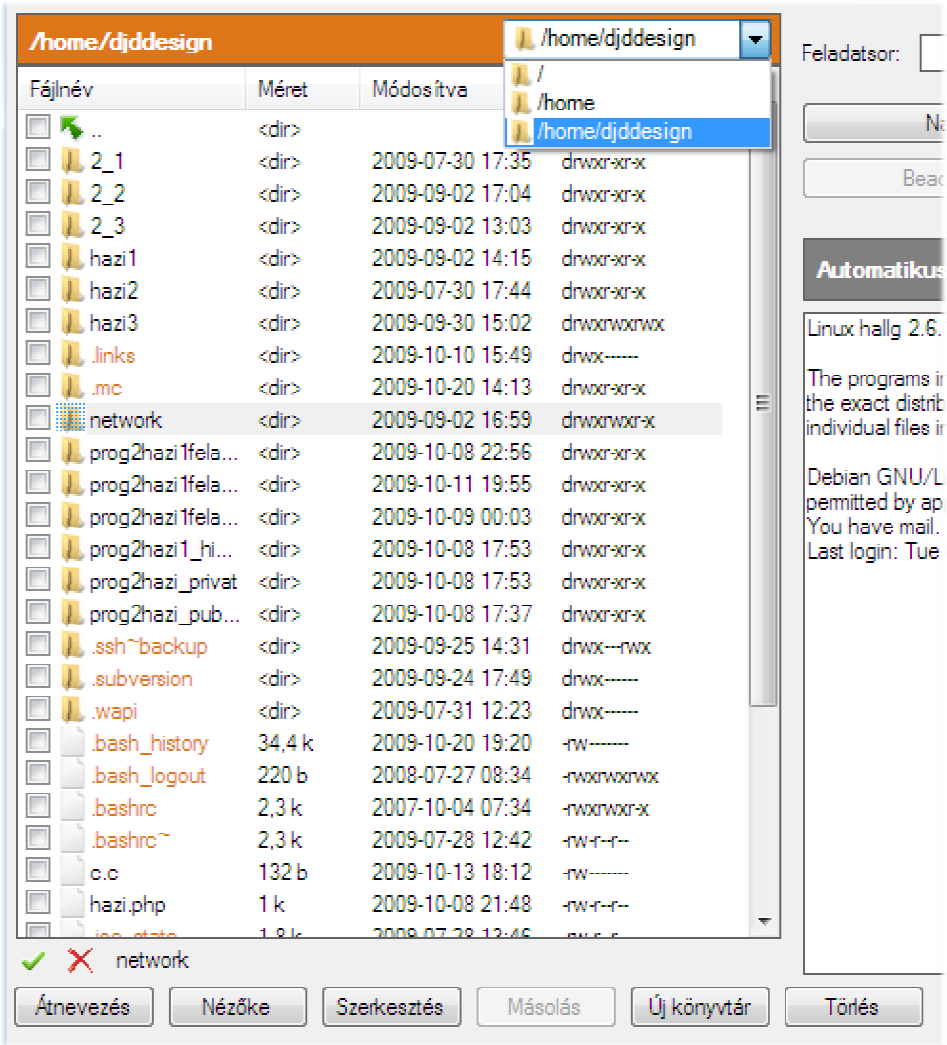
2.3.3.1. Könyvtárlista

Az ablak bal oldalán található a könyvtárlista, amely segítségével böngészhetők a szerveren tárolt állományok, illetve elvégezhetők a legfontosabb fájlműveletek (13. ábra). A fejlécben látható az aktuálisan kiválasztott könyvtár teljes elérési útja (`remoteHeaderLabel`).

A mellette található legördülő menüben (`remoteDrivesComboBox`) lehetséges a könyvtárszintek közötti gyors visszalépés.

Az aktuális könyvtár tartalma a `remoteFilesListView` vezérlőben jelenik meg. Itt jelennek meg az állományok legfontosabb tulajdonságai: a fájl/könyvtár neve, mérete (könyvtár esetén a `<dir>` felirat), az utolsó módosítás dátuma, valamint az adott fájlhoz tartozó jogosultságok.

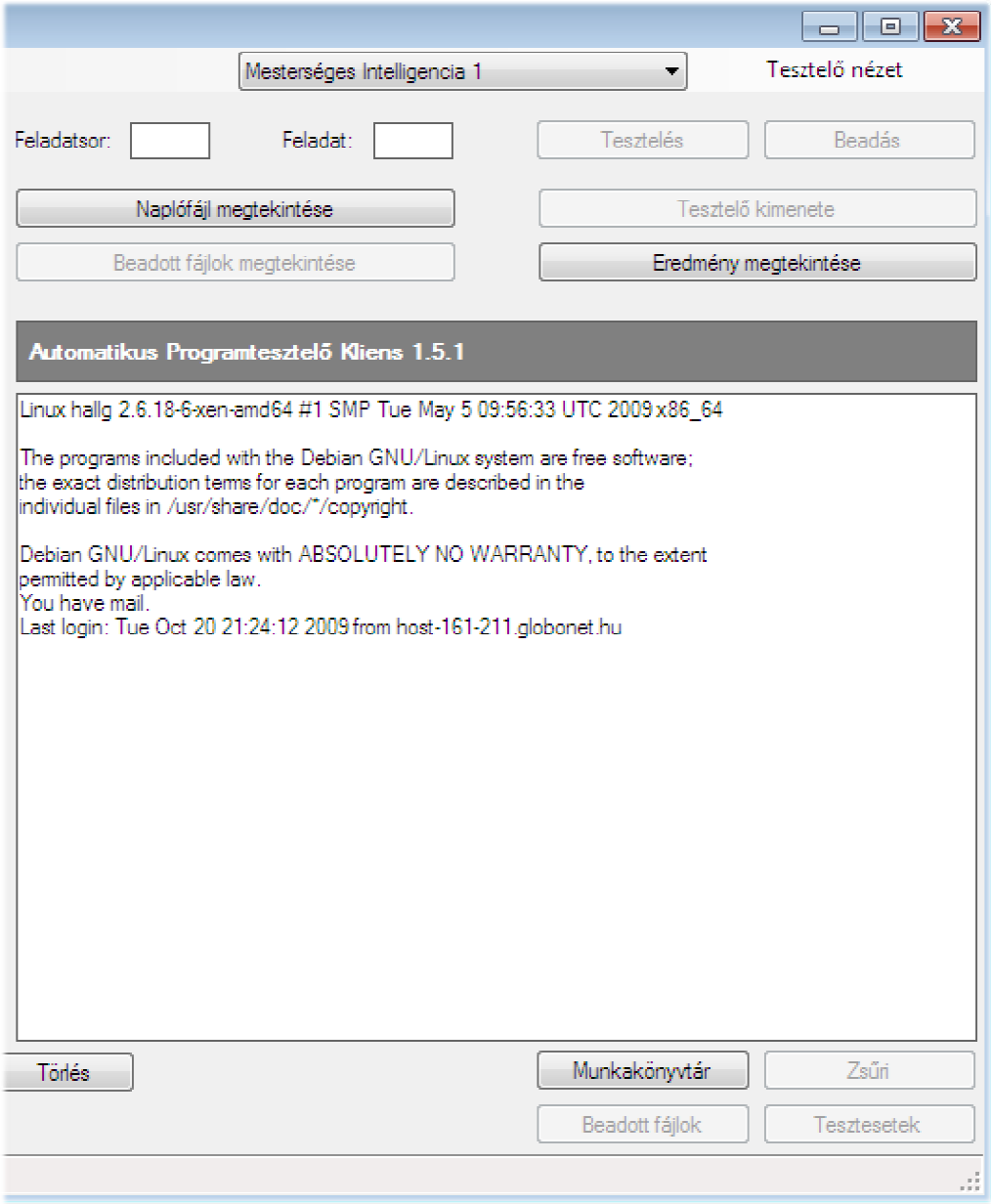
A lista az előbb említett kategóriák alapján rendezhető (a megfelelő oszlopfejlécre kattintva). A rendezést a `ListViewSorter` osztály valósítja meg. A rendezésben csak a fájlok vesznek részt, a könyvtárak ABC sorrendben listázódnak ki (hasonló megoldással találkozhatunk a népszerű fájlmenedzser programokban is).



13. ábra: Könyvtárlista

2.3.3.2. Tesztelő felület

A tesztelő felület (14. ábra) az alkalmazás egyik legfontosabb része. Az ablak jobb felső részében található legördülő menüből kiválaszthatjuk az aktuális tárgyat, majd miután a könyvtárlistában beleléptünk abba a könyvtárba, amelyben a tesztelni kívánt feladatunk található, a *Feladatsor* és a *Feladat* értékeinek megadása után elvégezhetjük a feladat tesztelését, beadását, megtekinthetjük az eredményeket és az adott feladatra beadott állományainkat.



14. ábra: Tesztelő felület

Fontos szempont volt az alkalmazás fejlesztése során a rugalmasság. Mivel a tesztkörnyezet maga is rugalmas, bármikor lehetővé tehetjük újabb tárgyak tesztelését, ehhez mindössze néhány shell szkript módosítására van szükség. Emiatt ezt az egyszerű, az alkalmazás módosítása nélkül történő konfigurációt itt is lehetővé kellett tenni. A megoldást a konfigurációs állományok használata jelentette. Az alkalmazás gyökérkönyvtárának `Subjects` alkönyvtárban található, `.ini` kiterjesztésű állományok tartalmazzák a tantárgyak adatait, amelyekkel a tesztelés elvégezhető. Egy állomány a 15. ábrán látható tartalommal rendelkezik. Látható, hogy ebben az állományban szerepelnek a tesztkörnyezetben is előforduló parancsok, az adott tárgyhoz tartozó megfelelő paraméterekkel. Ezeket a konfigurációs állományokat elegendő a félév elején elkészíteni, és a frissített állományokkal újra lefordítani majd publikálni az alkalmazáshoz tartozó telepítő állományt.

```
targy_nev=Magas szintű programozási nyelvek 2 - PTI
zsuri=/home/prog2zsuri/
feladat=/home/prog2zsuri/feladatok/pti/{0}/{1}
tesztesetek=/home/prog2zsuri/feladatok/pti/{0}/{1}/test/
beadott_fajlok=/home/prog2zsuri/feladatok/pti/{0}/{1}/submit/
tesztel=prog2-teszt pti/{0}/{1}
bead=prog2-bead pti/{0}/{1}
beadva=prog2-beadva pti/{0}/{1}
eredmeny=prog2-eredmeny
```

15. ábra: A `Subjects/prog2pti.ini` fájl tartalma

Az ebben a fájlban található kulcs-érték párok határozzák meg, hogy milyen műveletet kell elvégeznie a tesztelőnek (azaz melyik parancsot kell elküldenie a szervernek, és mit kell visszaolvasnia).

A `targy_nev` értéke fog megjelenni a legördülő menüben, ez szolgál a tantárgy azonosítására. Mivel egyes kurzusokon több szak hallgatói is részt vesznek, és olykor szakonként különböző házi feladatokat kell beadniuk a hallgatóknak, ezért ugyanahhoz a tárgyhoz több shell szkript készül a szerveren, és a parancsok is ennek megfelelően változnak. Például az előbb látott, programtervező informatikusok számára készült `prog2pti.ini` fájl mellett szerepel egy hasonló tartalmú, `prog2mi.ini` fájl is, ami a Magas szintű

programozási nyelvek 2 mérnök informatikus szakos hallgatói számára teszi lehetővé a tesztelést.

Minden konfigurációs állományban kötelezően szerepelnie kell a `targy_nev` értékének, különben az adott tárgy nem kerül be a listába. A többi értékre ilyen követelmény nem áll fenn, amennyiben a további paraméterekhez nem tartozik érték, akkor az adott paraméterhez tartozó funkció letiltásra kerül (például Mesterséges intelligencia tárgyból csak az eredmény lekérdezésére van lehetőség, ugyanis a többi paraméterhez nem adtunk meg értéket).

A program jobb alsó sarkában található négy gomb (*Munkakönyvtár*, *Zsűri*, *Beadott fájlok*, *Tesztesetek*) gyors navigálást tesz lehetővé a különböző speciális könyvtárak között. A *Munkakönyvtár* minden esetben a felhasználó munkakönyvtárába navigál, a többi gomb funkcióját a konfigurációs állományok írják le. A `zsuri` paraméter értéke határozza meg az aktuális zsűri könyvtárát. A *Magas szintű programozási nyelvek 2 – PTI* tárgy esetében ez a könyvtár a `/home/prog2zsuri/`. A `tesztesetek` paraméter határozza meg a *Tesztesetek* gomb által végrehajtandó műveletet. A gomb megnyomásával a *Feladatsor* és a *Feladat* értékekkel megadott könyvtárba ugorhatunk, amennyiben az adott feladat könyvtára létezik. Hasonló módon, a `beadott_fajlok` a *Beatott fájlok* gomb által elvégzendő műveletet határozza meg.

A *Tesztelés* és *Beadás*, valamint a *Beadott fájlok megtekintése* és az *Eredmény megtekintése* gombok lenyomására a szervernek elküldendő parancsot rendre a `tesztel`, `bead`, `beadva` és `eredmeny` paraméterek értékei határozzák meg.

Mindegyik gomb lenyomására hasonló folyamat játszódik le: a konfigurációs állományok által meghatározott parancsot elküldjük a szervernek az `SShShell` osztály `writeLine` metódusának segítségével. A parancsok lefuttatása után a szerver válaszát az `Expect` metódussal tudjuk kinyerni, mely metódus paraméterül várja a végjelet jelentő mintát (`pattern`). A beolvasás megkönnyítésére készültek a `read` és `readText` metódusok, hiszen leggyakrabban az írás és olvasás műveletét használja az alkalmazás a szerverrel történő kommunikáció fenntartására (16. ábra).

A 17. és 18. ábrán látható a Mesterséges intelligencia tárgyhoz tartozó eredmény megtekintése egy SSH kliensben, illetve az Automatikus Programtesztelő Kliens

használatával. Az *Eredmény megtekintése* gombhoz tartozó eseménykezelő kódrészlete a 19. ábrán látható.

Az alkalmazott, nyílt forráskódú SSH könyvtár (`SharpSSH`) bármilyen, tetszőleges SSH osztálykönyvtárra lecserélhető, az alkalmazás módosítása nagyon kevés időbefektetéssel megtehető.

(A `SharpSSH` projekt `SshShell` osztályában kisebb változásokat kellett eszközölni az említett `Expect` és `writeLine` metódusokban, ugyanis az ékezetes karakterek biztos átvitele érdekében UTF-8 karakterkódolást kellett használni az alapértelmezett – `Default` – karakterkódolás helyett.)

Az alkalmazás erőssége éppen az, hogy az SSH kliensen begépelt parancsok egyetlen gomb megnyomásával helyettesíthetők, így nagyban gyorsítható, egyszerűsíthető a tesztelés folyamata.

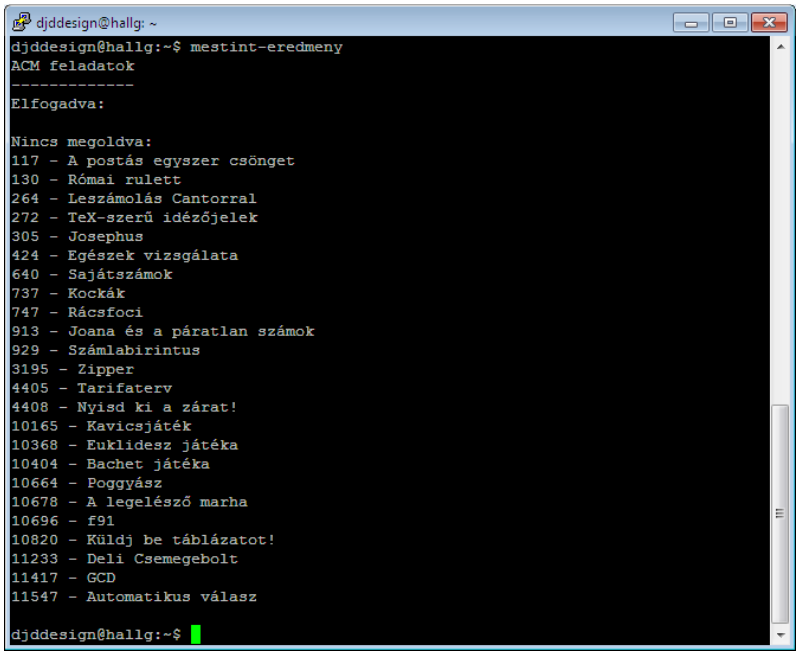
{MainForm.cs}

```
...
public partial class MainForm : Form {
   ...
   private string read(){
      try{
         return ssh.Expect(pattern).Trim();
      }
      catch (Exception){
         textWriter(testLabel, "Hiba");
         ExceptionHandler();
         return "Hiba a szerver üzenetének feldolgozása során.";
      }
   }
   ...
   private string readText(){
      try{
         return trimLines(ssh.Expect(pattern).Trim(), 1, 1);
      }
      catch (Exception){
          ExceptionHandler();
          textWriter(testLabel, "Hiba");
          return "Hiba a szerver üzenetének feldolgozása során.";
      }
   }
   ...
}
```

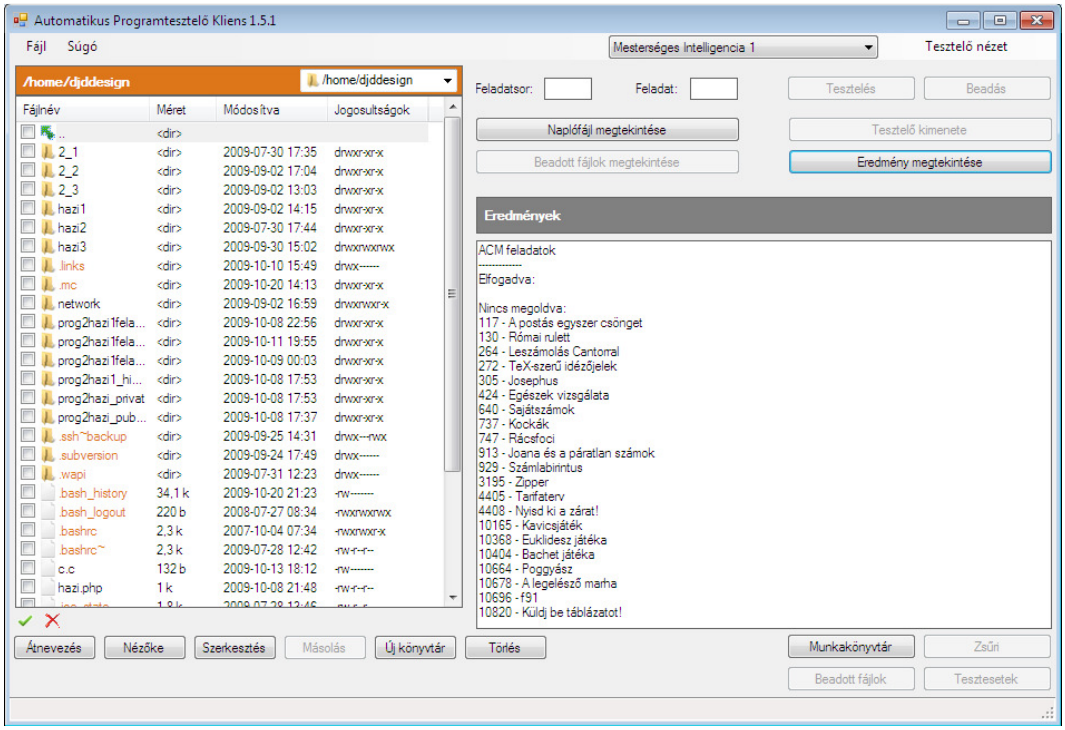
16. ábra: `read` és `readText` metódusok

A konfigurációs állományok tartalma alapján feltöltjük a `subjects` listát a `Subjects` osztály példányaival, és később a kiválasztott tárgyhoz tartozó paramétereket innen olvassuk ki.

Például az eredmények megtekintése esetén a `subjects[groupsComboBox.SelectedIndex].Eredmeny` értékét használjuk fel a lekérdezéshez. Ezt a parancsot küldjük el a szervernek, majd az eredményt megjelenítjük a `mainTextBox`-ban, illetve a `captureMethod` metódus segítségével az aktuális funkcióhoz tartozó fejlécet is generálunk.



17. ábra: Eredmény megtekintése SSH kliensben



18. ábra: Eredmény megtekintése az *Automatikus Programtesztelő Kliens*ben

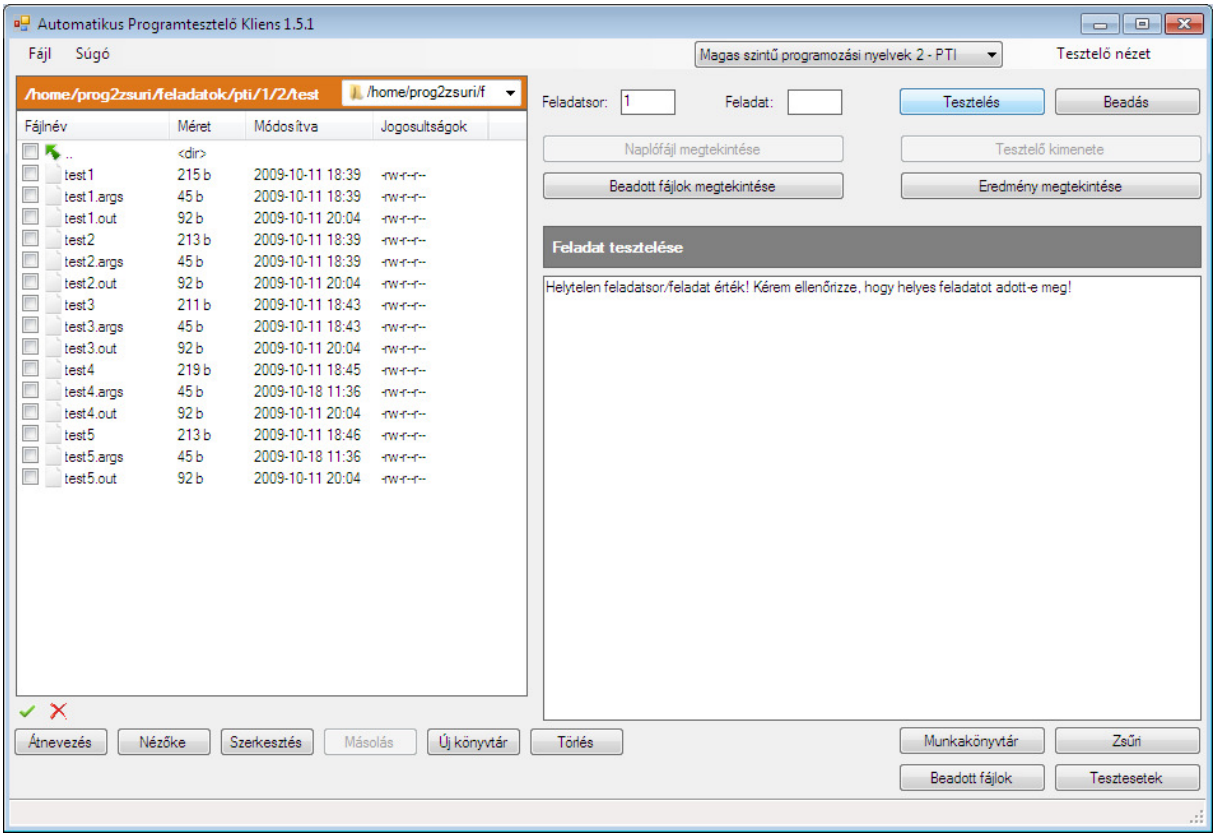
```
{MainForm.cs}
...
private static List<Subject> subjects = null;
...
public partial class MainForm : Form {
...
   private void viewResultButton_Click(object sender, EventArgs e){
      try{
         Cursor.Current = Cursors.WaitCursor;
         ssh.WriteLine(String.Format(
            subjects[groupsComboBox.SelectedIndex].Eredmeny)
         );
         captureMethod("Eredmények", readText());
         Cursor.Current = Cursors.Arrow;
      }
      catch (Exception)
      {
         Cursor.Current = Cursors.Arrow;
         captureMethod(
                "Eredmények - Hiba", "Hiba az eredmény lekérése során.");
         ExceptionHandler();
      }
   }
...
}
```

19. ábra: Az *Eredmény megtekintése* gombhoz tartozó eseménykezelő

Például ha egy feladatot tesztelünk, akkor a háttérben a következő utasítás fog lefutni:

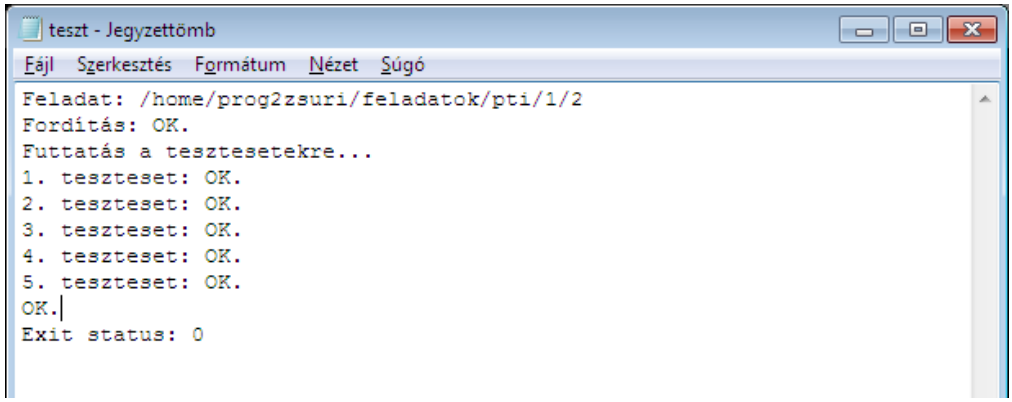
```
ssh.WriteLine(String.Format(
      subjects[groupsComboBox.SelectedIndex].Tesztel,
                                  excNum1TextBox.Text, excNum2TextBox.Text)
);
```

A legtöbb esetben ezt a módszert használjuk az adott parancs szerveren történő futtatásához, és a válasz kiíratásához. Bizonyos esetekben (például tesztelésnél, beadásnál, vagy az adott feladathoz tartozó tesztesetek könyvtárába ugrás esetén) felhasználjuk a *Feladatsor* (`excNum1TextBox`) és a *Feladat* (`excNum2TextBox`) mező értékét is. Ebben a két `TextBox`-ban csak számok szerepelhetnek, a hibás bemenetet a `correctExercise()` metódussal szűrjük ki, és erről tájékoztatjuk a felhasználót is (20. ábra). Amennyiben helyes volt mindkét paraméter értéke, a `String.Format` statikus metódus segítségével beillesztjük az értékeket a parancs vázába, majd a teljes parancsot lefuttatjuk a szerveren.



20. ábra: Hibás feladatsor/feladat érték

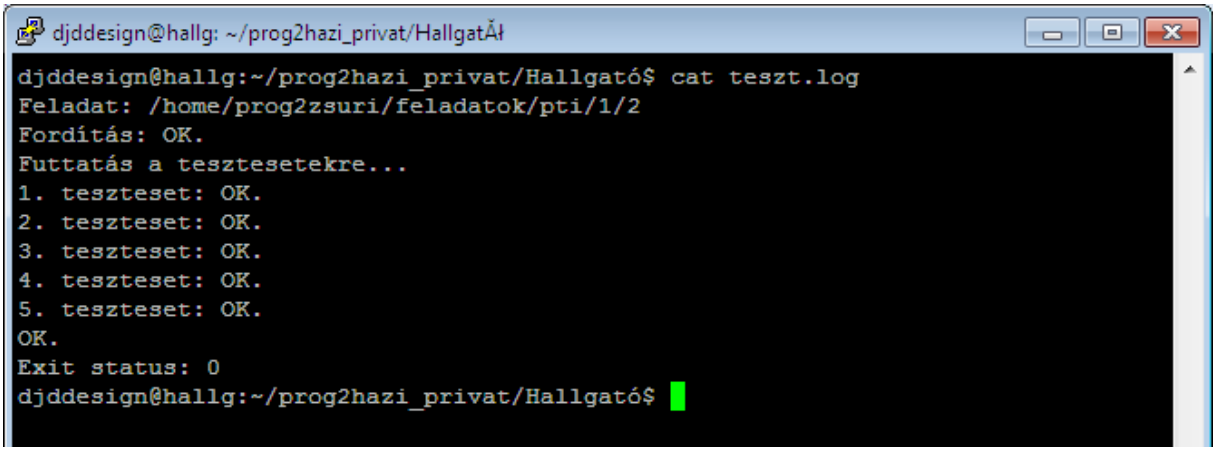
Amennyiben már teszteltünk egy feladatot, azaz létezik az adott könyvtárban `teszt.log` valamint `teszt.out` fájl, akkor a *Naplófájl megtekintése* illetve a *Tesztelő kimenete* gombok aktívvá válnak. Amennyiben megnyomjuk valamelyik gombot, akkor a szerveren levő fájlok letöltődnek a `Temp` könyvtár `apk_temp` alkönyvtárába, majd megjelenítésre kerülnek a beállított nézőkében (21. ábra).



21. ábra: Tesztelő kimenetének megtekintése a kliensben

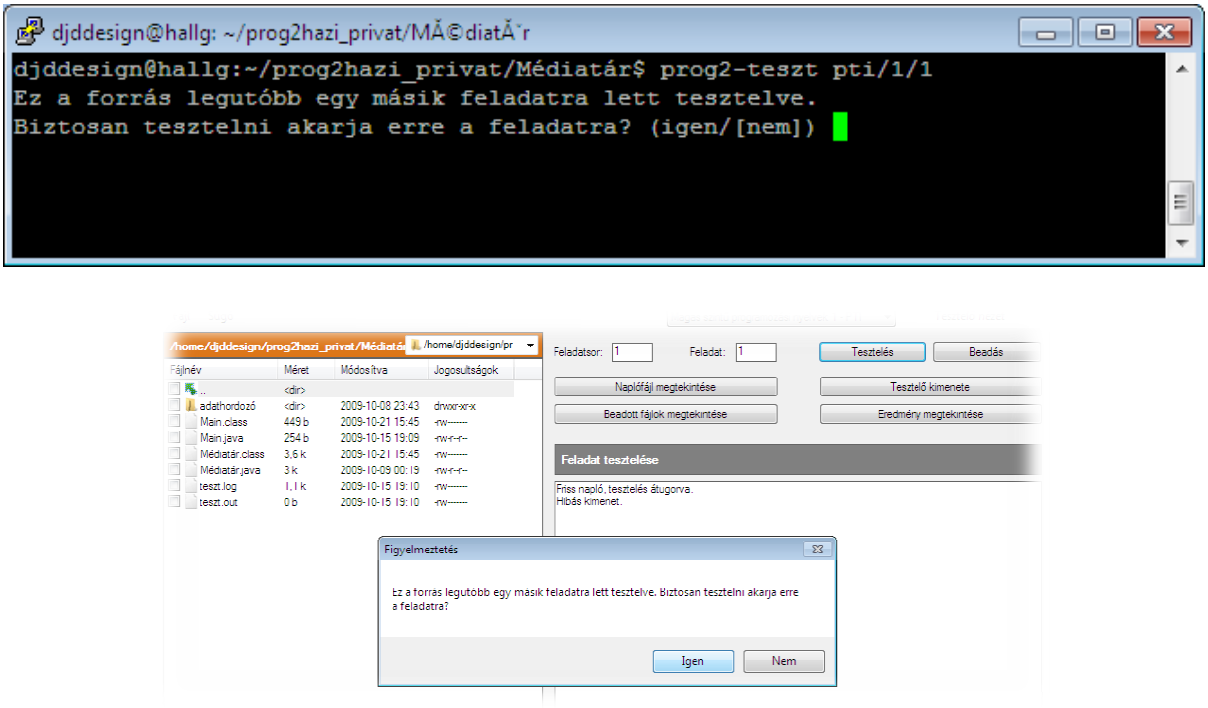
(Az `apk_temp` könyvtárban tárolt ideiglenes fájlok az alkalmazás bezárását követően törlődnek.)

A szerveren ehhez a `cat` parancs kiadására (22. ábra) vagy valamelyik beépített szövegszerkesztő (*Joe*, *VI*, *CoolEdit*) használatára van szükség, azonban ebben az esetben nincs lehetőség például a forrásállomány és a tesztelő kimenetének egymás melletti elhelyezésére a képernyőn, amelyre bizonyos esetekben szükségünk lehet.



22. ábra: Tesztelő kimenetének megtekintése PuTTY használatával

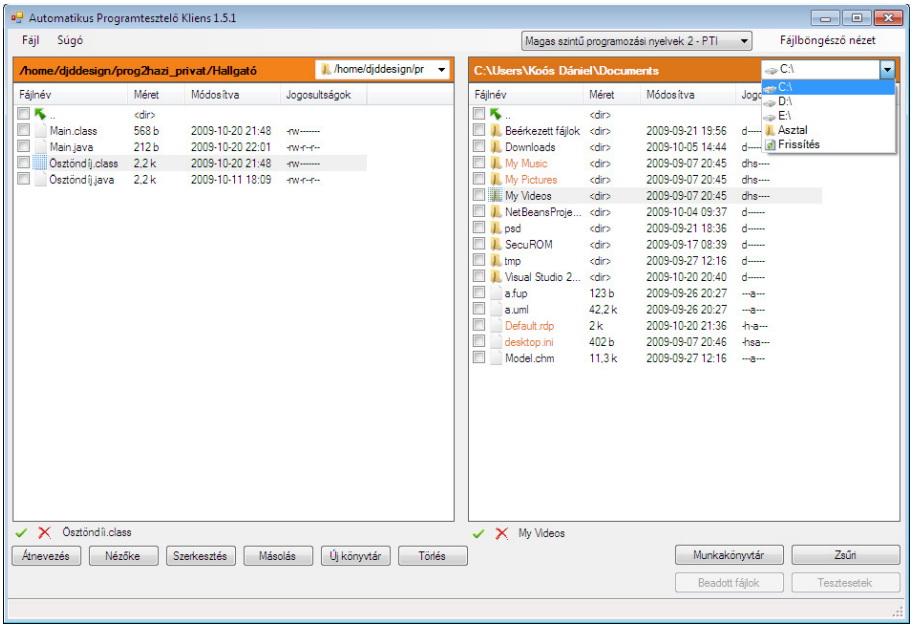
Tesztelés és beadás esetén a szerveren futó `test.sh` és `submit.sh` szkript bizonyos kritériumokat ellenőriz, és egyes esetekben a felhasználó megerősítését várja. Ez a karakteres felülettel ellentétben már egy `MessageBox` megjelenítésével lehetséges (23. ábra).



23. ábra: A feladatot előzőleg egy másik feladatra tesztelték

2.3.4. Fájlböngésző nézet

Fájlböngésző nézetben (24. ábra) egy egyszerű fájlmenedzserként és SCP kliensként állományok mozgatására használhatjuk az alkalmazást a szerver és a helyi gép között. Ez egy egyszerű funkcionalitású megoldás, amely elegendő a kisebb méretű állományok mozgatására és az egyszerűbb fájlműveletek elvégzésére, így a tesztelés során nem kell másik programot elindítanunk, ha fel szeretnénk tölteni egy állományt a szerverre. Amennyiben nagy adatmennyiség mozgatására vagy több funkcióra van szükség, akkor továbbra is használhatóak a teljes értékű SCP/SFTP segédprogramok (például *WinSCP*).



24. ábra: Fájlböngésző nézet

2.3.4.1. Távoli könyvtárlista

A szerver állományait megjelenítő ún. távoli könyvtárlista a tesztelő nézetben megismert könyvtárlistával teljesen egyenértékű, minden funkció, ami a tesztelő nézetben rendelkezésre állt, az itt is megtalálható.

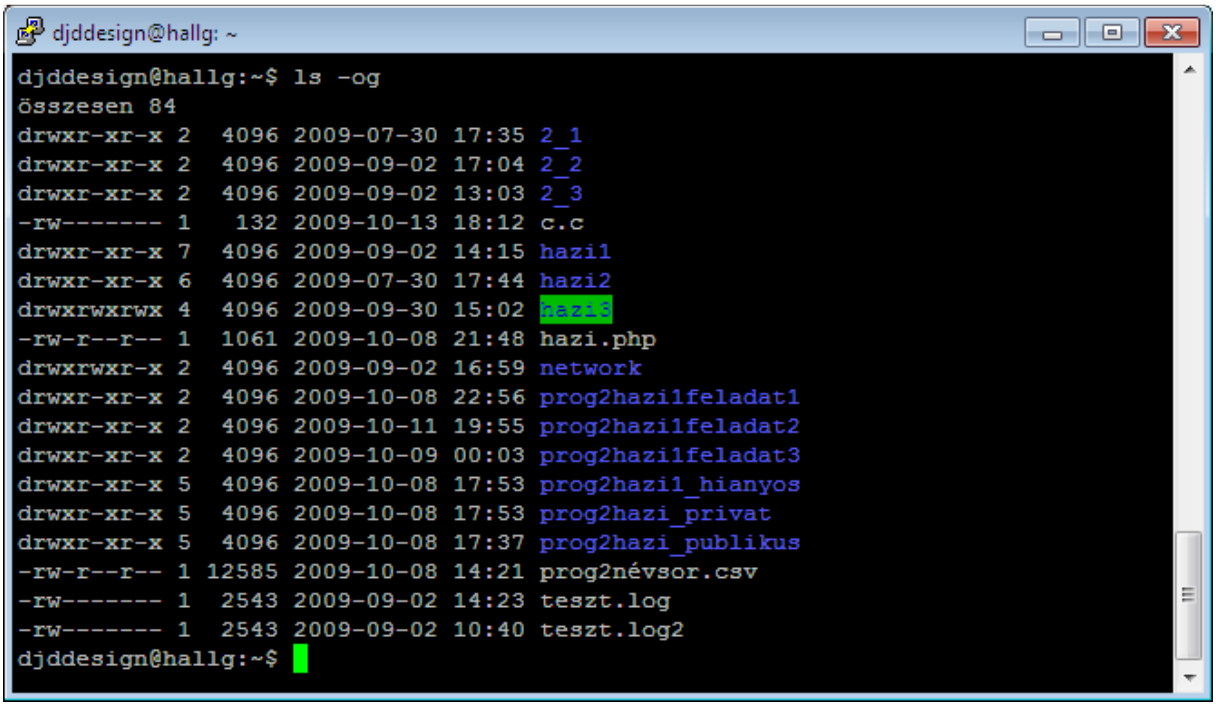
Az aktuális könyvtár tartalmát a `MainForm` osztály `changeDir` metódusa listázza ki (25. ábra)

```
{MainForm.cs}
...
private void changeDir(TextBox mainTextBox, ListView remoteFilesListView,
            Label remoteHeaderLabel){
   ...
   string command = string.Empty;
   string result = string.Empty;

   try{
      new Thread((ThreadStart)delegate()
      {
         this.BeginInvoke((ThreadStart)delegate()
         {
            ...
            // a beállításoktól függően jelenítjük meg a rejtett fájlokat
            if (hiddenFiles) { command += "ls -aog"; }
            else { command += "ls -og"; }

            command += ";echo \"---LIST---\"; pwd; echo \"---PWD---\"";
            ssh.WriteLine(command);
            result = read();
            ...
            // a kimenet feldolgozása, a fájllista előállítása,
            // a fejléc tartalmának feltöltése
            ...
            viewLogButton.Enabled =(remoteFilesListView.FindItemWithText(
                  PROG_LOG) != null) ? true : false;
            viewTestOutButton.Enabled =
                  (remoteFilesListView.FindItemWithText(PROG_OUT) != null)
                        ? true : false;
            // legördülő menü elemeinek feltöltése
            ...
         });
      }).Start();
   }
   catch { ... }
}
...
```

25. ábra: Távoli könyvtárlista feltöltése

Amennyiben a *Beállítások...* menüben beállítottuk, hogy a rejtett fájlok is megjelenítésre kerüljenek, akkor az `ls -aog`, különben az `ls -og` parancsot futtatjuk a szerveren, és a visszakapott listából (26. ábra) állítjuk elő a grafikus könyvtárlistát. Az aktuális könyvtárt a `pwd` parancs segítségével tudhatjuk meg, ez szolgál a későbbiekben a fejléc (`remoteHeaderLabel`) szövegéül, valamint a legördülő menü (`remoteDrivesComboBox`) elemeinek felépítéséhez is.

A könyvtár tartalmát vizsgálva állítjuk be bizonyos gombok (`viewTestButton`, `viewTestOutButton`) láthatóságát, illetve az adott állományhoz tartozó ikont is.



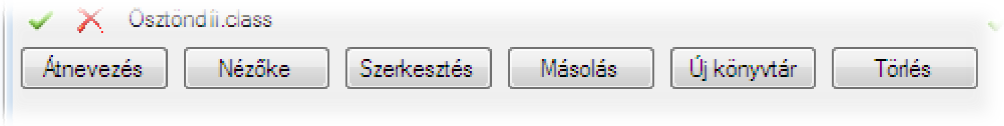
26. ábra: `ls -og` parancs kimenete

2.3.4.2. Helyi könyvtárlista

Fájlböngésző nézetben lehetőségünk van a számítógépen található fájlok és könyvtárak között tallózni. A távoli könyvtárlistához hasonlóan ebben a panelben is megjelenítésre kerülnek az állományokhoz tartozó legfontosabb adatok (*Fájnév*, *Méret*, *Módosítva*, *Jogosultságok*), amelyek alapján a listát rendezni is lehet. A fejléc mellett található legördülő menüből választhatóak ki a számítógépen található aktív meghajtók, és egyszerűen átléphetünk az *Asztal* könyvtárába is. Amennyiben egy új adathordozót (például pendrive) csatlakoztattunk a számítógéphez vagy helyeztünk a meghajtóba (például DVD), akkor a *Frissítés* opciót választva a rendelkezésre álló meghajtók listáját frissíthetjük, így a programból az újonnan behelyezett médiumon található adatokhoz is hozzáférhetünk.

2.3.4.3. Fájl- és könyvtárműveletek

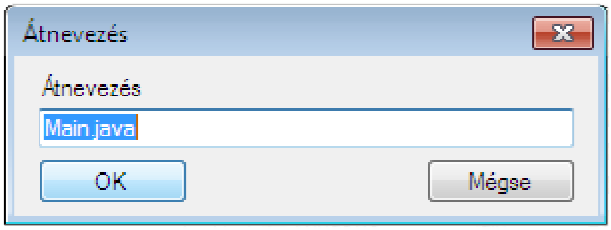
Tesztelő és fájlböngésző nézetben is rendelkezésre állnak az alapvető fájl- és könyvtárműveletek, amelyeket az ablak alján elhelyezkedő gombokkal (27. ábra) érhetünk el. A másolás kivételével minden funkció rendelkezésre áll mindkét nézetben, és a működésük mindkét környezetben (a szerveren vagy a helyi gépen) alkalmazva hasonló.



27. ábra: Fájl- és könyvtárműveleteket vezérlő gombok

2.3.4.3.1. Átnevezés

Az aktuálisan kijelölt fájl vagy könyvtár átnevezésére szolgál. A megjelenő formon (`RenameForm.cs`) adhatjuk meg az új állomány nevét (28. ábra). A karakterkódolási problémák elkerülése végett amennyiben a szerveren levő állományt nevezzük át, a megadott névben csak a Unix fájlnév-konvenciónak megfelelő karakterek szerepelhetnek. A helyi gépen történő átnevezésnél erre nincs kikötés.



28. ábra: Átnevezés

A beírt fájlnév ellenőrzésére a legegyszerűbb és leghatékonyabb módszernek a reguláris kifejezések használata bizonyult (29. ábra). Amíg helyes fájlnevet nem írunk be, addig az *OK* gomb letiltásra kerül. Amennyiben a beírt fájlnév helyes, az *OK* gomb megnyomására átadjuk az adatokat a `RenameDelegate` delegate-nek, ami meghívja a `MainForm` osztály `RenameCallbackFunction` metódusát.

Ez a callback metódus összehasonlítja az eredeti és az új fájlnevet, és amennyiben minden adat helyes, elvégzi az átnevezést. A helyi gépen az átnevezés a `Directory.Move` és `File.Move` metódusokkal történik, a szerveren történő átnevezés ennél egy kicsit komplikáltabb. Ugyanis a szerveren történő átnevezésre az `mv` parancsot használjuk. Ez szolgál az áthelyezésre is, ami kicsit bonyolulttá teszi a műveletet, hiszen Unix környezetben nem tudjuk egyértelműen megkülönböztetni a fájl- és könyvtárneveket (például fájlt nevezhetnénk át könyvtárra), ezért ezeket az eseteket a programkódban kell kezelnünk. Amennyiben ilyen ütközést tapasztalunk, azt a felhasználóval egy `MessageBox`-ban közöljük, és várjuk a megerősítést a művelet folytatására vagy megszakítására.

{RenameForm.cs}

```
...
public partial class RenameForm : Form
{
   ...
   private void renameTextBox_TextChanged(object sender, EventArgs e)
   {
      if ( !isLocal )
      {
         Regex regex = new Regex( "[^0-9a-zA-Z._-]" );
         if ( renameTextBox.Text.Trim() == string.Empty )
         {
            okButton.Enabled = false;
         }
         else if ( regex.IsMatch( renameTextBox.Text ) )
         {
            okButton.Enabled = false;
         }
         else
         {
         okButton.Enabled = true;
         }
      }
   else
   {
      okButton.Enabled =
            ( renameTextBox.Text == string.Empty.Trim() ? false : true );
   }
...
}
```

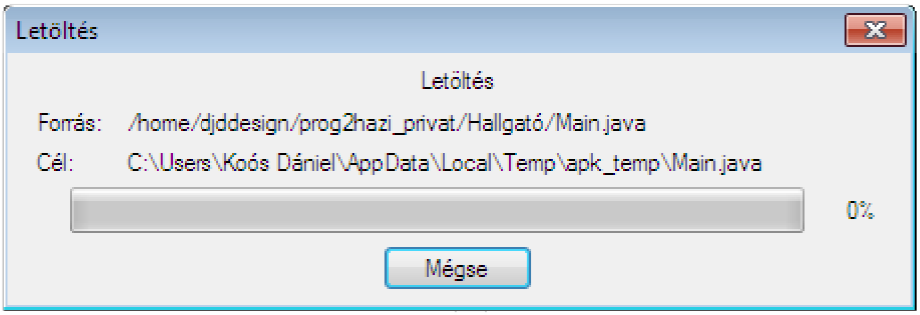
29. ábra: Átnevezés során a fájlnév helyességének ellenőrzése

Az átnevezés megtörténte után frissítjük a könyvtárlistát a `changeDir` illetve a `displayFolderList` metódusokkal.

2.3.4.3.2. Nézőke

Az aktuálisan kiválasztott fájl tartalmának megtekintésére szolgál a beállított nézőke segítségével. (A funkció természetesen könyvtárak esetén nem elérhető.)

A szerveren tárolt fájlok megtekintése esetén a fájlok előzőleg letöltésre kerülnek a felhasználó `Temp` könyvtárának `apk_temp` alkönyvtárába. A letöltés folyamatát nyomon követhetjük a `DownloadForm` segítségével (30. ábra). A formról leolvasható a forrás és a cél címe, illetve a letöltés aktuális állapota százalékban. A folyamat a *Mégse* gombra kattintva szakítható meg.



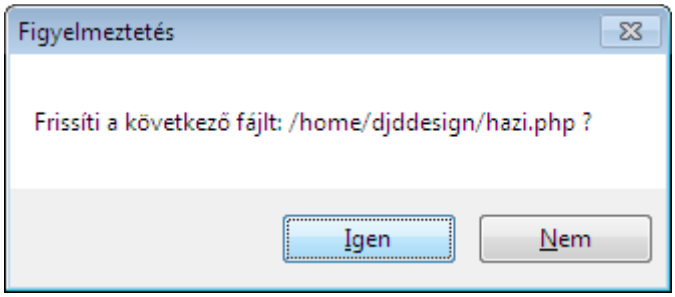
30. ábra: Letöltés

2.3.4.3.3. Szerkesztés

A szerkesztés nagyon hasonló a nézőke funkcióhoz, azonban amíg az utóbbi esetben a fájlt csak olvasásra nyitjuk meg, addig szerkesztés esetén a fájlt módosíthatjuk is. Amikor a szerveren szerkesztünk egy fájlt, az a nézőkénél leírt módon letöltődik az ideiglenes könyvtárba, majd létrejön egy `EditForm`, ami végigkíséri a szerkesztés folyamatát. Ez a form felelős a módosított fájlok szerverre való feltöltéséért. A form példányosításakor feliratkozunk a szerkesztő `Process`-ének `Exited` eseményére, ami a szerkesztő bezárása után az `UploadDelegate`-en keresztül meghívja a feltöltést végző formot, amennyiben a felhasználó jóváhagyja a módosítást (31. ábra) (ez a `DownloadForm` meghatározott paraméterű konstruktorának hívását jelenti).

A nézőke és szerkesztés folyamata során a fájlra vonatkozó adatokat egy `ViewerSettings` objektumban tartjuk számon. A letöltés/feltöltés pedig a `BackgroundWorker` osztályból származó `Viewer` komponens segítségével valósul meg. Ez

a komponens valósítja meg a háttérszálon történő letöltést/feltöltést, így a folyamat során a programablak tartalma folyamatosan frissül, amellyel nyomon követhető a folyamat előrehaladása.



31. ábra: Megerősítés várása módosított fájl feltöltése előtt

A szerverről és a szerverre történő állománymozgatást az `Scp` osztály valósítja meg (32. ábra). Az `Scp` osztály konstruktorában paraméterként a szerver címét, a felhasználói nevet és a jelszót várja. Ezután a `connect` metódus segítségével csatlakozunk a szerverhez, majd attól függően, hogy a szerverre töltünk fel vagy a szerverről töltünk le, az `Scp` osztály `To` vagy `From` metódusát hívjuk meg a forrás és a cél teljes elérési útvonalának megadásával. A folyamat befejezése után a `Close` metódussal zárjuk a kapcsolatot.

Az `Scp` osztály a folyamat kontrollálására három eseménykezelőt biztosít: `OnTransferStart`, `OnTransferProgress`, és `OnTransferEnd`. Az eseménykezelők paraméterei között szerepel az átvitt és az összes bájtok száma, amelyek alapján a `ProgressBar`-on visszajelzést adhatunk a folyamat állapotáról.

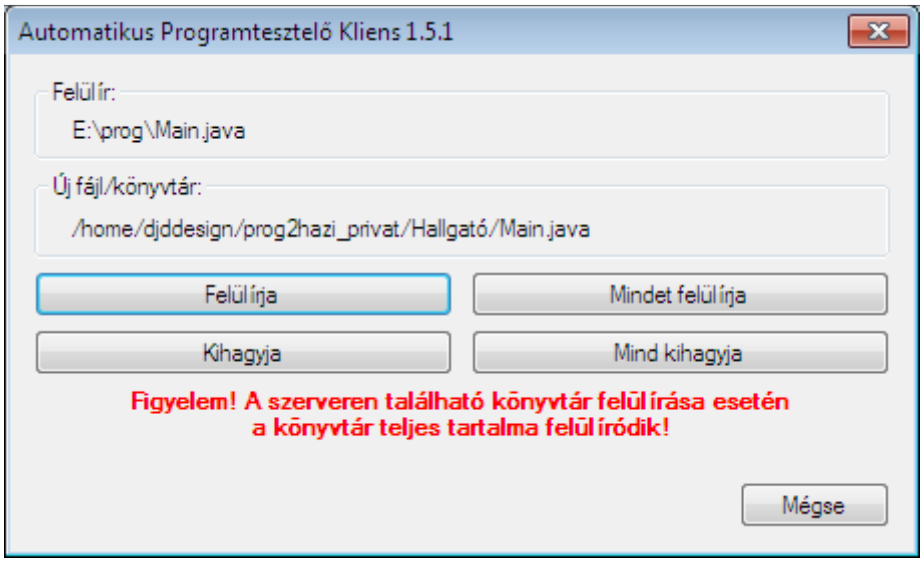
2.3.4.3.4. Másolás

Az alkalmazás segítségével állományokat másolhatunk a szerver és a kliens számítógép között. A másolandó állományok kijelölése után a *Másolás* gombra kattintva, a felhasználói megerősítés után megkezdődik a másolás folyamata, amelynek előrehaladásáról a `CopyForm` ad visszajelzést. Ebben az osztályban a `DownloadForm` mintájára létrehozott `Copier` komponens gondoskodik a többszálúságról, és az átviendő állományok adatait egy `CopierSettings` objektum tartja nyilván. A másoláshoz az előző pontban ismertetett `Scp` osztály metódusait használjuk.

```
{Viewer.cs}
...
public partial class Viewer : BackgroundWorker
{
   private static BackgroundWorker bw;
   private static ViewerSettings viewerSettings;
   private static Scp scp;
   ...
   private void Viewer_DoWork(object sender, DoWorkEventArgs e){
      try{
         bw = sender as BackgroundWorker;
         viewerSettings = e.Argument as ViewerSettings;

         scp = new Scp(viewerSettings.Host, viewerSettings.User,
               viewerSettings.Pass);

         scp.OnTransferStart +=
                        new FileTransferEvent(scp_OnTransferStart);
         scp.OnTransferProgress +=
                        new FileTransferEvent(scp_OnTransferProgress);
         scp.OnTransferEnd += new FileTransferEvent(scp_OnTransferEnd);
         ...
         scp.Connect();

         // Az Upload tag határozza meg a fájlátvitel irányát, ha értéke
         // igaz, akkor a szerverre töltünk fel, ha hamis, akkor
         // a szerverről töltünk le.
         if (viewerSettings.Upload == true){
            scp.To(viewerSettings.Tempfilename,
                                          viewerSettings.Selectedfile);
         }
         else{
            scp.From(viewerSettings.Selectedfile,
                                          viewerSettings.Tempfilename);
         }
         if (scp.Connected) scp.Close();
      }
      catch (Exception exc){
         MessageBox.Show(
            "Hiba a folyamat végrehajtása során." + exc.Message
         );
      }
   }
...
}
```

32. ábra: Az `Scp` osztály használata

Amennyiben egy fájl vagy könyvtár már létezik az ellentétes oldalon, akkor a felhasználó választhat a 33. ábrán látható lehetőségek közül (`OverwriteForm.cs`). Az `Scp` osztály eszközkészlete miatt amennyiben egy könyvtár tartalmát írjuk felül a szerveren, az az új könyvtár tartalmával íródik felül.



33. ábra: Létező állomány esetén a felülírás lehetőségei

A folyamat befejeződésével ismét frissítjük a könyvtárlistákat, hogy azok az aktív könyvtár aktuális tartalmát mutassák.

2.3.4.3.5. Új könyvtár

Mind a szerveren, mind a helyi gépen levő fájlrendszerben létrehozhatunk új könyvtárt (illetve a helyi menüt használva új fájlt is), ehhez csak ki kell választanunk az aktuális panelt (szerver vagy helyi gép), majd az *Új könyvtár* gombra kattintva, a megjelenő ablakban megadva a könyvtár nevét (amennyiben lehetséges) létrejön a kívánt könyvtár. Helyi gépen erre a `Directory` osztály `CreateDirectory` metódusa szolgál, a szerveren pedig az `mkdir` paranccsal hozhatjuk létre a kívánt könyvtárat.

2.3.4.3.6. Törlés

Lehetőségünk van kijelölt állományok törlésére is. A folyamat állását a törlés állapotáról visszajelzést adó `DeleteForm`-on követhetjük nyomon, ahol lehetőségünk van a folyamat megszakítására is.

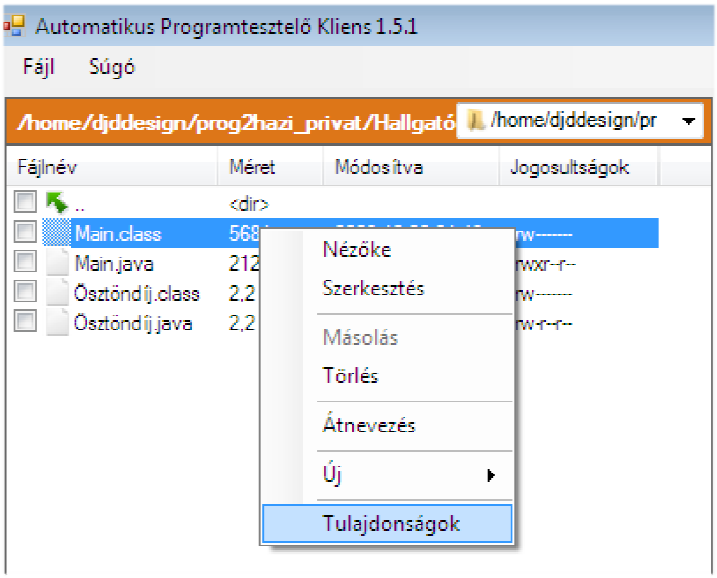
A `DeleteForm` osztály a `Viewer` komponens mintájára létrehozott `Deleter` komponens, és a szükséges adatokat nyilvántartó `DeleterSettings` osztály

felhasználásával többszálúan hajtja végre a törlési folyamatot, amiről a felhasználó így folyamatos visszajelzést kap. Szerverről való törléshez az `rm` (könyvtárak esetén az `rm -r`) parancsot, a helyi gépen pedig a `FileSystem` `DeleteDirectory` és `DeleteFile` metódusait használjuk. Utóbbi esetben a törölt állományok a lomtárba kerülnek.

2.3.4.3.7. Helyi menü

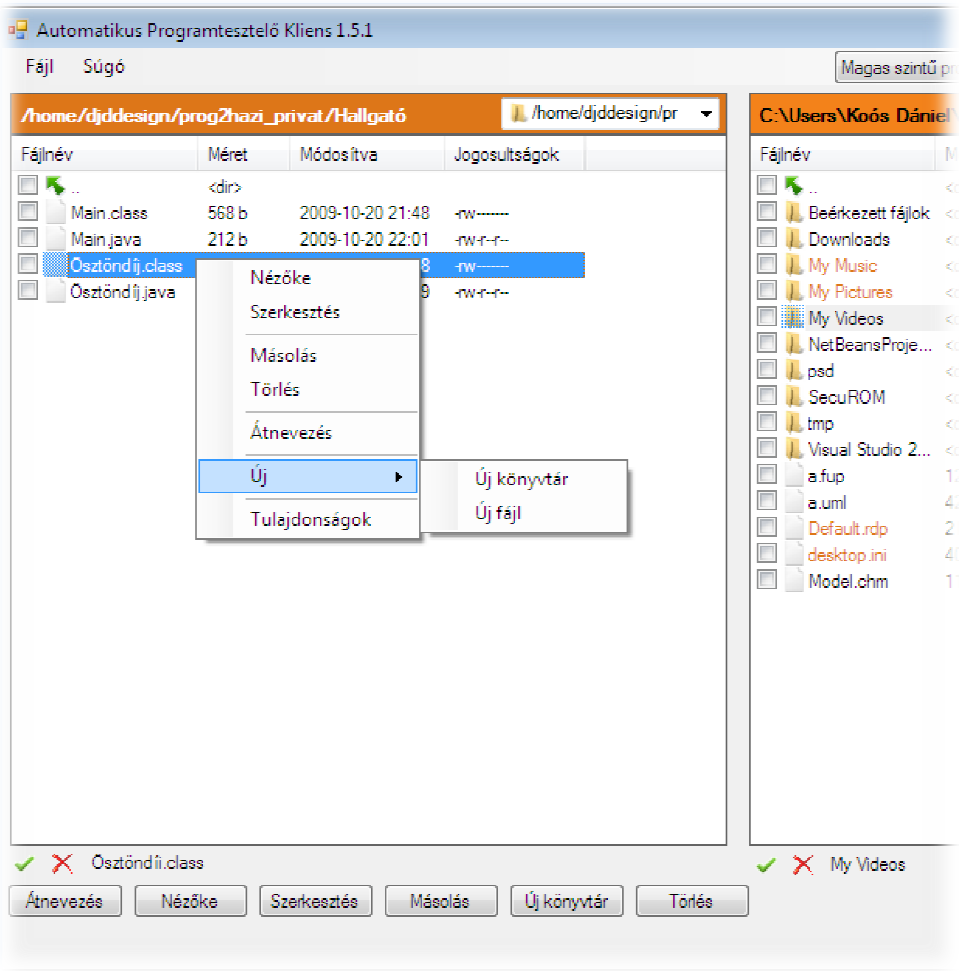
A legfontosabb fájlműveleteket a helyi menüből is elérhetjük, ehhez vagy a távoli vagy a helyi könyvtárlistára kell jobb klikkel kattintanunk, majd a megjelenő menüben (34. ábra) a kívánt parancsot választani.

A *Nézőke* és *Szerkesztés* menüpontok csak fájlok kijelölése esetén működnek, a *Másolás* pedig csak *Fájlböngésző nézet*ben aktív. A többi menüpont minden esetben használható.



34. ábra: Létező állomány esetén a felülírás lehetőségei

Az *Új* menüponton belül választhatunk az *Új fájl* és *Új könyvtár* opciók közül (35. ábra), a *Tulajdonságok* menüpontot választva pedig a kijelölt állomány tulajdonságait kérdezhetjük le és állíthatjuk be.

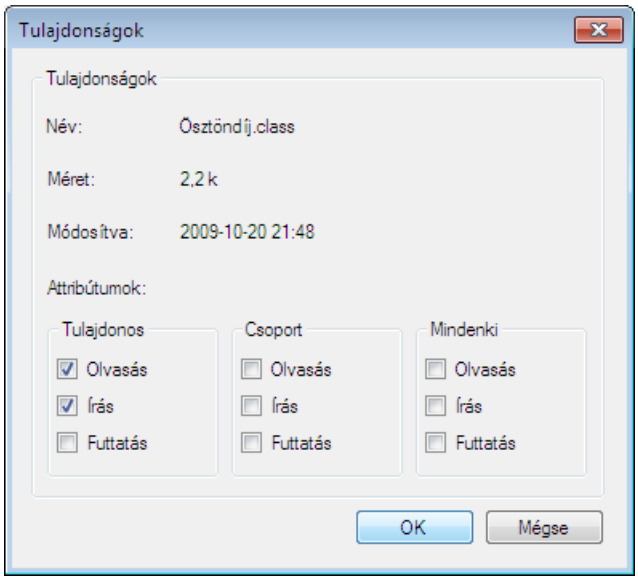


35. ábra: Új könyvtár és új fájl létrehozása a helyi menü használatával

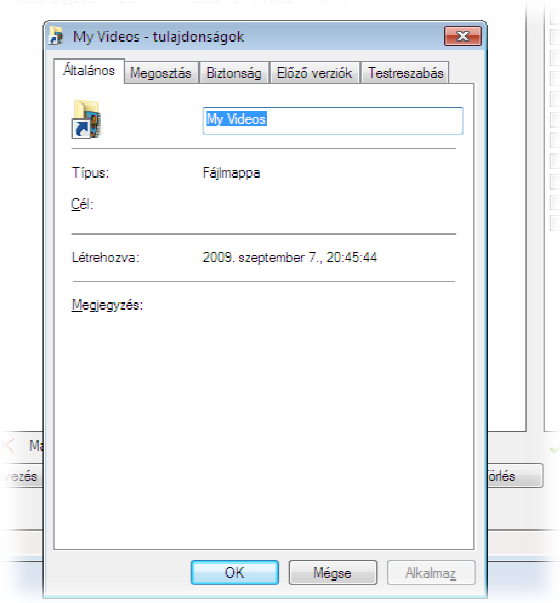
A szerveren levő állomány tulajdonságainak megjelenítésére a `PropertiesForm` osztály szolgál. A *Tulajdonságok* menüpont meghívására megjelenő formról (36. ábra) leolvasható az állomány neve, mérete, az utolsó módosítás dátuma, és a jelölőnégyzetek segítségével beállítható a kívánt attribútum-érték. Az *OK* gombra kattintva a `PropertiesDelegate` meghívja a `MainForm modifyPropertiesCallback` metódusát, ami a jelölőnégyzetek értékeiből kiszámítja a `chmod` parancsnak paraméterül átadott oktális értéket, majd lefuttatja a szerveren a `chmod` parancsot a kiszámított paraméterekkel.

A helyi állományok tulajdonságainak megjelenítése a Windows beépített *Tulajdonságok* panelének meghívásával történik. Ennek eléréséhez a `ShellExecuteEx` metódus használatára van szükség, ami a `SHELLEXECUTEINFO` struktúra értékeinek helyes

beállításaival meghívva az adott állomány *Tulajdonságok* ablakát jeleníti meg, ahol a Windows felhasználók által már ismert paramétereket lehet lekérdezni és beállítani (37. ábra).



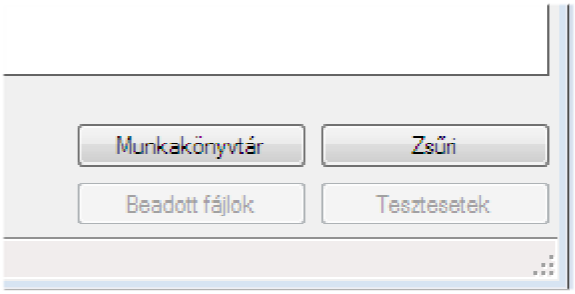
36. ábra: A szerveren tárolt állomány tulajdonságai



37. ábra: Egy helyi gépen tárolt könyvtár tulajdonságai

2.3.4.3.8. Speciális könyvtárakba ugrás

Fájlböngésző nézetben is egy gombnyomással elérhetjük a fontosabb könyvtárakat (38. ábra), azonban a tesztelő nézettől eltérően itt a *Beadott fájlok* és a *Tesztesetek* gombok nem aktívak, mivel azok csak egy bizonyos feladatsor és feladat érték megadása esetén használhatóak.



38. ábra: Speciális könyvtárak elérése a fájlböngésző nézetben

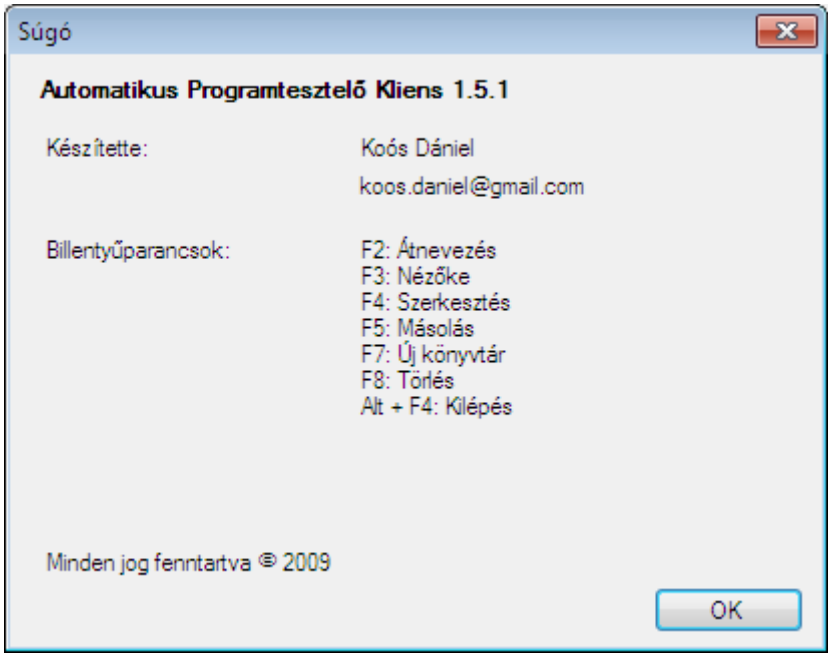
Ebben a nézetben a *Munkakönyvtár* gomb megnyomására a könyvtárlista a *Beállítások…* menüpontban megadott könyvtár tartalmára ugrik, a *Zsűri* gomb megnyomására pedig a távoli könyvtárlista az aktuálisan kiválasztott tárgyhoz tartozó zsűri könyvtárába navigál (amely értéket a konfigurációs állományokban határoznak meg).

2.3.5. Billentyűparancsok

A legfontosabb funkciókat nem csak a felületen található gombokkal, hanem a legtöbb alkalmazásban megismert billentyűkombinációkon keresztül is elérhetjük. Ezek a billentyűkombinációk a *Súgó* ablakában (39. ábra) is szerepelnek.

A billentyűparancsok a `MainForm` osztály `FilesListView_KeyDown` eseménykezelőjében lettek definiálva, mivel a legtöbb művelet csak akkor hajtható végre, hogyha valamelyik `ListView` aktív, és van kijelölt elem. *Fájlböngésző nézet*ben a `Tab` billentyűvel válthatunk a két panel között, a könyvtárváltásra és a könyvtár tartalmának böngészésére használhatjuk az `Enter` és a kurzormozgató billentyűket is.

A *Súgó* az `F1` gomb leütésére jeleníthető meg, azonban ennek a billentyűnek a leütését a `MainForm_KeyDown` eseménykezelőjében figyeljük, hiszen a súgótartalom megjelenítésére az egész alkalmazásban, globálisan van szükségünk.



39. ábra: Billentyűparancsok listája a *Súgó* ablakában

3. Automatikus Programtesztelő Portál

Az *Automatikus Programtesztelő Portál* az *Automatikus Programtesztelő Kliens* tesztelő nézetét helyezi teljesen webes alapokra, így a kliens által kínált alapvető funkciók bárhol, bármikor elérhetőek. A portál használatához csak egy működő internetkapcsolatra és egy böngészőre van szükség.

A portál a kliens alkalmazáshoz képest korlátozott funkciókkal rendelkezik, azonban jól jön a használata, ha például a határidő előtt eszünkbe jut, hogy elfelejtettük beadni a feladatunkat. A portál segítségével bárhonnan lekérdezhetjük az eredményünket is, így gyorsan értesülhetünk a zárthelyi dolgozatok és házi feladatok eredményeiről bármilyen segédprogram használata nélkül.

3.1. Platform, fejlesztői környezet

A portál fejlesztése PHP nyelven történt, ehhez MySQL alapú adatbázis társul. A felhasználói felület HTML nyelven készült, az esztétikus elrendezéshez CSS stíluslapokkal, a felhasználói élmény fokozásához JavaScript technológiával kiegészítve. Az SSH kapcsolat a PHP_SSH2 kiterjesztés használatával valósul meg.

3.2. Fájlstruktúra

A fejlesztés az MVC személet figyelembe vételével történt, így a könyvtárstruktúrában elkülönítésre kerültek a konfigurációs állományok, az adatbevitelt kezelő, az adatokat feldolgozó, és az eredményt kiíratók kódrészek.

Így a portál gyökérkönyvtárában elhelyezkedő `Subjects` alkönyvtárban találhatóak az *Automatikus Programtesztelő Kliens*nél már megismert konfigurációs állományok, a `core/code` könyvtárban helyezkednek el az alkalmazás működéséhez szükséges osztályok, és a `core/design` könyvtárban találhatóak a megjelenítésért felelős állományok (ezen belül a CSS stíluslapok, a képek, és a JavaScript kódok külön alkönyvtárakba csoportosítva). A teljes fájlstruktúra a 40. ábrán tekinthető meg.

```
.
.error.php
.index.php
.login.php
.logout.php
.updateSubjects.php
/-core
      |---index.html
      /---code
            |---configError.php
            |---config.php
            |---helper.class.php
            |---index.html
            |---progTesztelo.class.php
            |---subject.class.php
            |---subjectList.class.php
      /---design
            |---directoryContent.php
            |---footer.php
            |---foot.php
            |---header.php
            |---headError.php
            |---head.php
            |---index.html
            |---index.php
            |---labels.php
            |---login.php
            |---nav.php
            |---testPanel.php
            /-----css
                  |---common.css
                  |---index.html
                  |---reset.css
            /-----images
                  |---delete.gif
                  |---document.gif
                  |---folder.gif
                  |---header_bg.gif
                  |---index.html
                  |---pre_code_bg_blk.gif
                  |---rename.gif
                  |---view.gif
            /-----js
                  |---coreFunctions.js
                  |---index.html
/-Subjects
      |---index.html
      |---mestint.ini
      |---prog1mi.ini
      |---prog1pti.ini
      |---prog2mi.ini
      |---prog2pti.ini
      |---proglab.ini
```

40. ábra: Fájlstruktúra

3.3. Adatbázis

Amíg az *Automatikus Programtesztelő Kliens*nél a `Subjects` könyvtárban tárolt konfigurációs állományok tartalmát elég volt a program betöltődése során beolvasni, addig a webes felületen történő felhasználáshoz célszerű ezeket az információkat egy adatbázisban tárolni.

A `prog` nevű adatbázisban egyetlen, `prog` nevű adattáblában tároljuk a konfigurációs állományok tartalmát (41. ábra). A tábla szerkezete a 42. ábrán, tartalma a konfigurációs állományokkal történő feltöltés után a 43. ábrán látható. A konfigurációs állományok beolvasásáért és adatbázisba töltéséért az `updateSubjects.php` fájl felelős. Ezt a fájlt csak akkor kell futtatni, hogyha a konfigurációs állományokban változás történt (valójában elég minden félév elején, az aktuális tárgyaknak megfelelő konfigurációs állományok feltöltése után frissíteni az adatbázis tartalmát).

A kliens és a portál közötti átjárhatóság biztosítása érdekében a portálnál alkalmazott konfigurációs állományok felépítése teljesen megegyezik a kliensnél látottakkal.

```
mysql> use prog
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| prog               |
+--------------------+
2 rows in set (0.00 sec)

mysql> show tables;
+----------------+
| Tables_in_prog |
+----------------+
| prog           |
+----------------+
1 row in set (0.00 sec)
```

41. ábra: Az alkalmazott adatbázis és adattábla

Mező	Típus	Egybevetés	Tulajdonságok	Null	Alapértelmezett	Extra
targy_nev	varchar(128)	utf8_hungarian_ci		Nem		
zsuri	varchar(256)	utf8_hungarian_ci		Igen	NULL	
feladat	varchar(256)	utf8_hungarian_ci		Igen	NULL	
tesztesetek	varchar(256)	utf8_hungarian_ci		Igen	NULL	
beadott_fajlok	varchar(256)	utf8_hungarian_ci		Igen	NULL	
bead	varchar(256)	utf8_hungarian_ci		Igen	NULL	
beadva	varchar(256)	utf8_hungarian_ci		Igen	NULL	
eredmeny	varchar(256)	utf8_hungarian_ci		Igen	NULL	
tesztel	varchar(256)	utf8_hungarian_ci		Igen	NULL	

42. ábra: A `prog` adattábla szerkezete

targy_nev	zsuri	feladat	tesztesetek	beadott_fajlok	bead	beadva	eredmeny	tesztel
Magas szintű programozási nyelvek 1 - MI	/home/prog1zsuri/	/home/prog1zsuri /feladatok/mi/{0}/{1}	/home/prog1zsuri /feladatok/mi/{0} /{1}/test/	/home/prog1zsuri /feladatok/mi/{0} /{1}/submit/	prog1-bead mi/{0}/{1}	prog1-beadva mi/{0}/{1}	prog1-eredmeny	prog1-teszt mi/{0}/{1}
Magas szintű programozási nyelvek 1 - PTI	/home/prog1zsuri/	/home/prog1zsuri /feladatok/pti/{0}/{1}	/home/prog1zsuri /feladatok/pti/{0} /{1}/test/	/home/prog1zsuri /feladatok/pti/{0} /{1}/submit/	prog1-bead pti/{0}/{1}	prog1-beadva pti/{0}/{1}	prog1-eredmeny	prog1-teszt pti/{0}/{1}
Magas szintű programozási nyelvek 2 - MI	/home/prog2zsuri/	/home/prog2zsuri /feladatok/mi/{0}/{1}	/home/prog2zsuri /feladatok/mi/{0} /{1}/test/	/home/prog2zsuri /feladatok/mi/{0} /{1}/submit/	prog2-bead mi/{0}/{1}	prog2-beadva mi/{0}/{1}	prog2-eredmeny	prog2-teszt mi/{0}/{1}
Magas szintű programozási nyelvek 2 - PTI	/home/prog2zsuri/	/home/prog2zsuri /feladatok/pti/{0}/{1}	/home/prog2zsuri /feladatok/pti/{0} /{1}/test/	/home/prog2zsuri /feladatok/pti/{0} /{1}/submit/	prog2-bead pti/{0}/{1}	prog2-beadva pti/{0}/{1}	prog2-eredmeny	prog2-teszt pti/{0}/{1}
Mesterséges Intelligencia 1							mestint-eredmeny	
Programozás Labor 1	/home/proglabzsuri/	/home/proglabzsuri /feladatok/{0}/{1}	/home/proglabzsuri /feladatok/{0}/{1} /test/	/home/proglabzsuri /feladatok/{0}/{1} /submit/	proglab-bead {0}/{1}	proglab-beadva {0}/{1}	proglab-eredmeny	proglab-teszt {0}/{1}

43. ábra: A `prog` adattábla tartalma

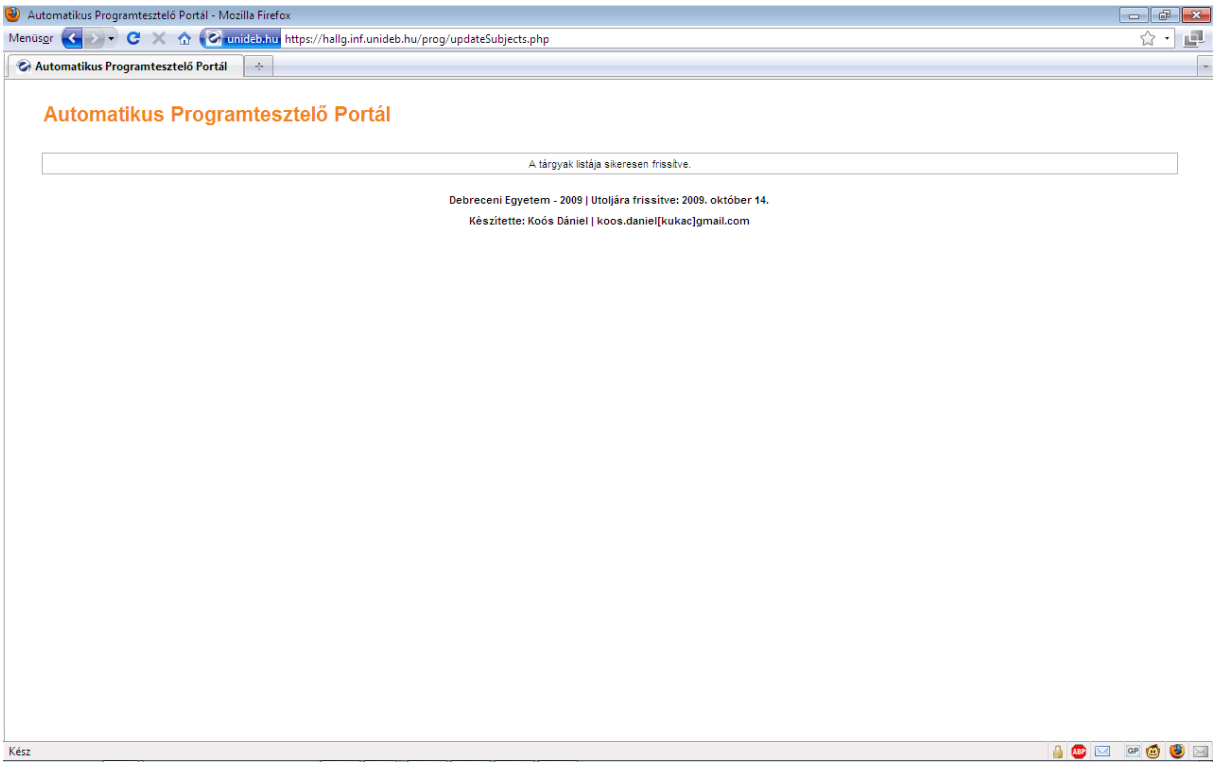
3.3.1. Adatbázis frissítése

Az adatbázis tartalmának frissítése az `updateSubjects.php` fájl futtatásával történik. Ez a fájl a `core/code/subjectList.class.php` fájlban található `SubjectList` osztály példányosítása és az adatbázishoz való csatlakozás (`connect`) után az `updateEntries` metódussal frissíti az adatbázis tartalmát, majd a kapcsolat lezárása (`close`) után a művelet sikerességéről üzenetet küld a felhasználónak (44. ábra).

Az `updateEntries` metódus az *Automatikus Programtesztelő Kliens*nél megismert feltételek figyelembe vételével beolvassa az állományok adatait, a beolvasott adatokat

felhasználva készít egy-egy `Subject` objektumot, majd a létrejött objektumok attribútumainak felhasználásával felülírja az adatbázis tartalmát az új konfigurációs adatokkal.

Az adatbázis tartalmának aktualizálása után készen áll az alkalmazás a használatra.



44. ábra: A tárgyak listája sikeresen frissítve.

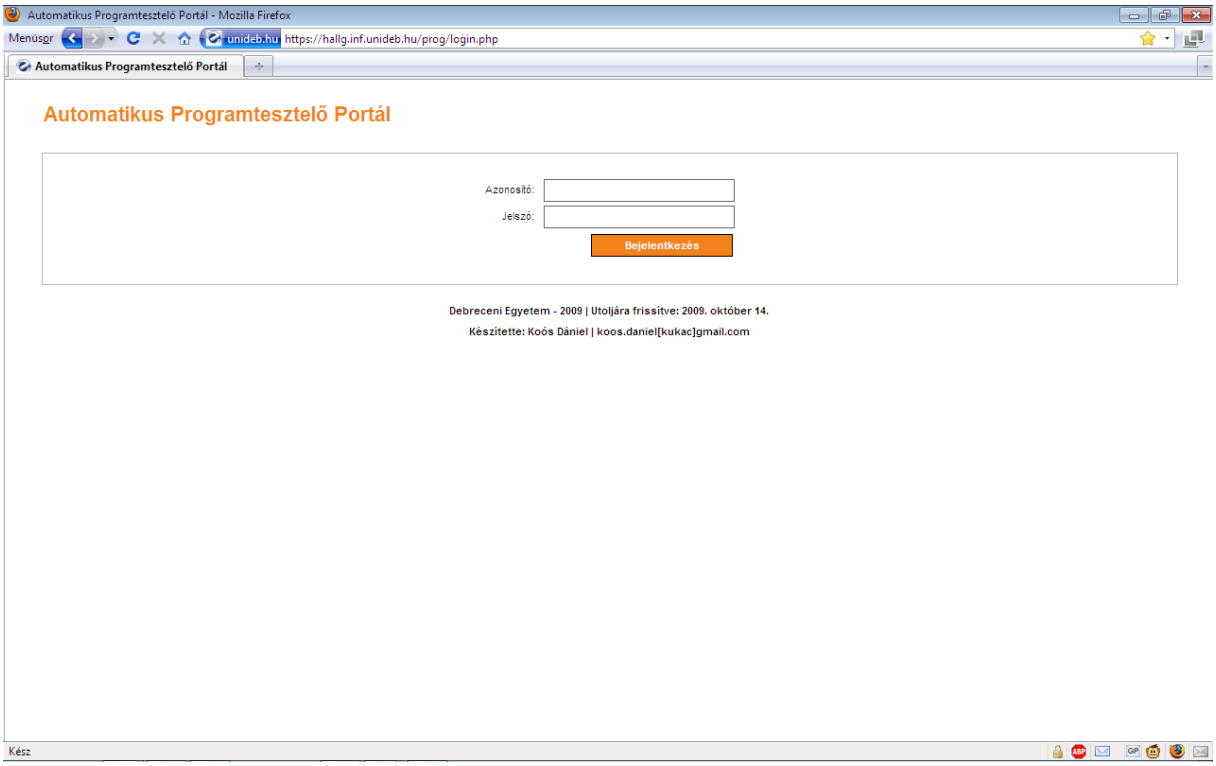
3.4. Felhasználói felület

3.4.1. Bejelentkező oldal

A portálra látogató felhasználókat a 45. ábrán látható képernyő fogadja, ahol a hálózati azonosítójukkal és jelszavukkal tudnak belépni az oldalra. Amennyiben minden adatot helyesen adtak meg, akkor használhatják a portál által nyújtott összes lehetőséget.

A bejelentkezés folyamata a `Helper` osztály `login` metódusának meghívásával kezdődik el, ami a paraméterként kapott felhasználó név és titkosított jelszó alapján a PHP `setcookie()` függvényével létrehozza a felhasználó számítógépén azokat a sütiket, amelyek a szerverrel történő kommunikációhoz elengedhetetlen fontosságúak. Ugyanis amíg a kliens

alkalmazásnál a bejelentkezés után felépült egy kommunikációs csatorna a szerver és a kliens gép között, ami a kapcsolat lezárásáig fenn is állt, addig a webes portál esetében minden parancs lefuttatása előtt csatlakozni kell a szerverhez, majd a parancs végrehajtása után bontani a kapcsolatot.



45. ábra: Bejelentkező oldal

A sütik létrehozása után az oldal átirányításra kerül az `index.php` oldalra, és használatba vehető az alkalmazás. A felhasználó mindaddig bejelentkezve marad, amíg a `config.php`-ban megadott `$cookieTimeout` változó által meghatározott időtartam el nem telik két parancs végrehajtása között.

Ahogy az a 46. ábrán is látható, hogyha létezik érvényes süti a felhasználó számítógépén (ezt a `Helper` osztály `cookieExists` metódusa vizsgálja meg), akkor automatikusan az `index.php` oldalra irányítjuk át, és csak akkor engedjük belépni, ha nincs érvényes süti a számítógépen tárolva (azaz vagy nem jelentkezett még be, vagy letelt az időlimit).

```
{./login.php}
<?php
   require_once("./core/code/helper.class.php");

   $loginhelper = new Helper();

   if ($loginhelper->cookieExists())
   {
      header("Location: index.php");
   }
   else if (isset($_POST['username']) && !empty($_POST['username']) &&
            isset($_POST['password']) && !empty($_POST['password']))
   {
      $loginhelper->login($_POST['username'], mcrypt_encrypt(MCRYPT_3DES,
            $salt, $_POST['password'], MCRYPT_MODE_ECB, $iv));
   }
   require_once("./core/design/head.php");
   require_once("./core/design/login.php");
   require_once("./core/design/foot.php");
?>

{./core/code/helper.class.php}
<?php
require_once("config.php");

class Helper
{
   ...
   public static function login($username, $password)
   {
      global $cookieTimeout;
      setcookie("prog_tesztelo_username", $username, time() +
                  $cookieTimeout, '/', '.hallg.inf.unideb.hu', true);
      setcookie("prog_tesztelo_password", $password, time() +
                  $cookieTimeout, '/', '.hallg.inf.unideb.hu', true);
      header("Location: index.php");
   }
   ...
}
?>
```

46. ábra: Bejelentkezés

3.4.1.1. Kapcsolat létesítése a hallgatói szerverrel

A webes alkalmazás esetén is létre kell hozni a kapcsolatot a hallgatói szerverrel, hiszen itt is a kliensnél látott módon, a különböző Unix parancsok kimenetének feldolgozásával áll elő a modern, grafikus felület.

Az SSH kapcsolat kiépítését és a hallgatói szerverrel történő kommunikációt a PHP_SSH2 kiterjesztés függvényei biztosítják. A `ProgTesztelo` osztály `connect` metódusa segítségével kapcsolódhatunk a szerverhez, a `disconnect` metódus segítségével pedig lezárhatjuk a kapcsolatot, miután minden szükséges parancsot lefuttattunk és beolvastuk a szerver válaszát. Az `execute` és `readResponse` metódusok a szerver válaszát dolgozzák fel, és hozzák olyan formára, amelyből már könnyen előállíthatóak az oldal egyes moduljait felépítő kódrészletek.

Amennyiben megadtuk a bejelentkezéshez szükséges adatokat (amelyeket a sütikben tároltunk), az `index.php` állományban példányosítjuk a `ProgTesztelo` osztályt – a konstruktorban megadjuk az oldal címét (amit az adott funkciónak megfelelően változatunk), valamint a kapcsolatfelvételhez szükséges adatokat: a szerver címét, a felhasználói nevet és a jelszót, majd a `connect` metódus meghívásával csatlakozunk a szerverhez (47. ábra).

{./core/code/progTesztelo.class.php}

```
<?php
public function connect(){
   try
   {
      if (!($this->connection = @ssh2_connect($this->serverAddress, 22)))
      {
        throw new Exception("Nem hozható létre kapcsolat a szerverrel!");
      }
      if (!(@ssh2_auth_password(
                  $this->connection, $this->username, $this->password)
      ))
      {
        throw new Exception("Hibás felhasználói név vagy jelszó!");
      }
      $this->shell = @ssh2_shell($this->connection, false);
   }
   catch (Exception $e)
   {
      global $errorMessage;
      $errorMessage[] = $e->getMessage();
      return;
   }
}
?>
```

47. ábra: SSH authentikáció

Az `ssh2_auth_password` függvény plaintext authentikációt tesz lehetővé, a jelenlegi környezetben ez volt a legegyszerűbb mód a szerverre történő bejelentkezéshez. A függvény első paramétere az `ssh2_connect` függvény által visszaadott erőforrás, második és harmadik paramétere pedig a felhasználói név és jelszó, amellyel be szeretnénk jelentkezni a szerverre. Ha a folyamat sikeresen lezajlik, akkor az `ssh2_shell` függvény visszaad egy adatfolyam erőforrást, melyet az `fread` és `fgets` függvényekkel olvashatunk, és az `fwrite` függvénnyel írhatunk.

A szervernek küldött parancsokat az `execute` metódus dolgozza fel (48. ábra) – a szerver kimenetének sorait egy tömbben tárolja, amelyből a `readResponse` metódus készít folyamatos szöveget. A parancsainkat összefűzve, egy összetett parancsként küldjük el a szervernek, így a válasz elejét és végét, illetve a köztes parancsokat meghatározott kiíratásokkal határoljuk el egymástól, amellyel megkönnyítjük az adatok feldolgozását.

3.4.2. Tesztelő nézet

Sikeres bejelentkezést követően a 49. ábrán látható képernyő fogadja a felhasználót. Ezen a felületen az *Automatikus Programtesztelő Kliens* tesztelő nézetének azon főbb funkciói érhetőek el, amelyek a teszteléshez és beadáshoz elengedhetetlenek.

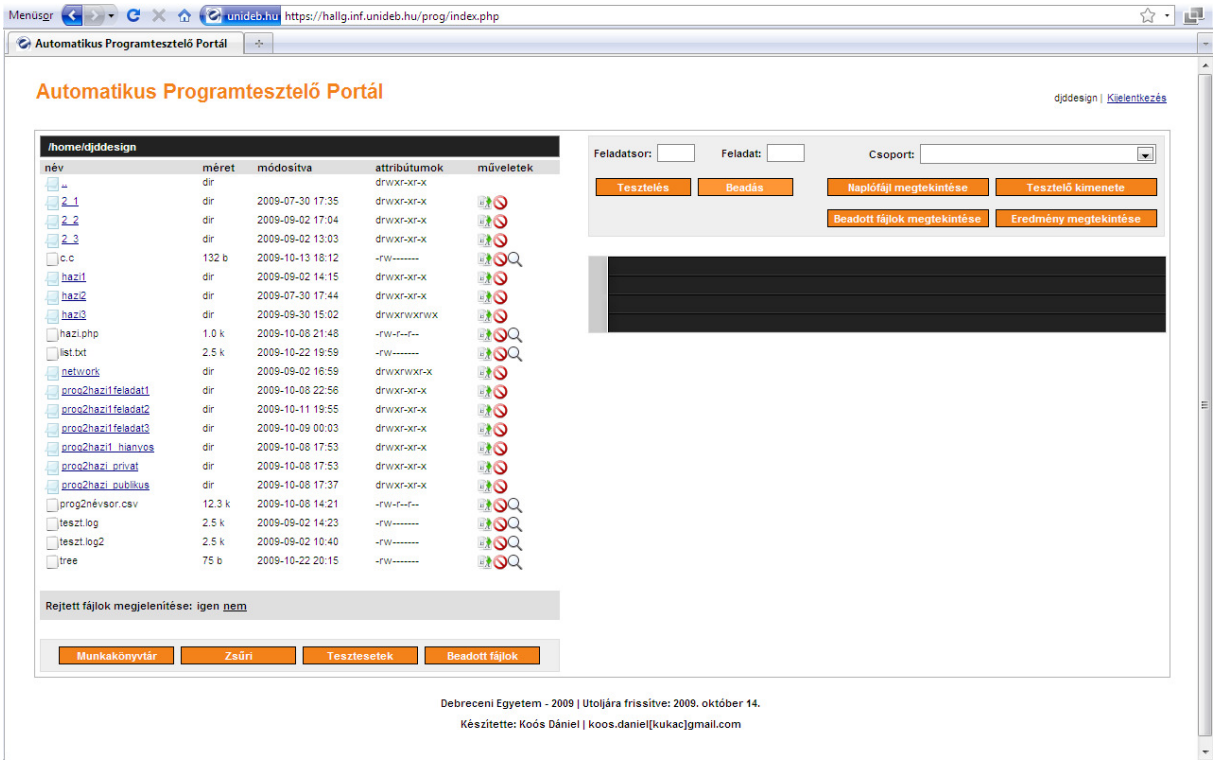
Az egyes műveletek végrehajtásához szükséges változók értékét az URI-ben megadott query string paraméterekből határozzuk meg, és az adott oldalon található linkek (`<a>` tagok) `href` paramétereit, és az oldalon található vezérlők értékeit dinamikusan, az aktuális query string paramétereknek megfelelően állítjuk elő.

```
{./core/code/progTesztelo.class.php}
<?php
public function execute($shell, $cmd)
{
   try
   {
      if (!$shell)
      {
         throw new Exception("Hiba a kommunikáció során.");
      }
      stream_set_blocking($shell, true);
      $cmd = "echo '[start]'; " .
             ($cmd[strlen($cmd)-1] == ";" ? $cmd : $cmd . ";" ) .
             " echo '[end]'";
      fwrite($shell, $cmd . "\n");
      $output = null;
      $start = false;
      $start_time = time();
      $max_time = $this->connectionTimeout; // időtúllépés

      while(((time()-$start_time) < $max_time))
      {
         $line = (fgets($shell));
         if(!strstr($line, $cmd))
         {
            if(preg_match('/\[start\]/',$line))
            {
               $start = true;
            }
            else if(preg_match('/igen\/\[nem\]/',$line))
            {
               $output[] = $line;
               return $output;
            }
            else if(preg_match('/\[end\]/',$line))
            {
               return $output;
            }
            else if($start)
            {
               $output[] = $line;
            }
         }
      }
      throw new Exception("Időtúllépés.");
   }
   catch (Exception $e)
   {
      global $errorMessage;
      $errorMessage[] = $e->getMessage();
      return;
   }
}
?>
```

48. ábra: Parancsfeldolgozó

3.4.2.1. Könyvtárlista

A portál esetében csak a szerveren található fájlok között lehetséges a böngészés. A könyvtárlistában a kliensnél látottakhoz hasonló módon a *név* mellett a *méret*, *módosítva* és *attribútumok* mezők szerepelnek, viszont itt nincs lehetőség sorba rendezésre. Könyvtárak esetében az állomány nevének helyén egy link található, amire kattintva beléphetünk az adott könyvtárba.



49. ábra: *Automatikus Programtesztelő Portál – Tesztelő nézet*

Az aktuális könyvtár a `pwd` parancs kimenetéből származik, a fájllistát pedig az `ls` parancs kimenetének feldolgozásával nyerjük. – A könyvtárlista alatt állíthatjuk be a rejtett fájlok megjelenítését/elrejtését, választásunknak megfelelően a `hidden` query string paraméter értéke `true` vagy `false` lesz, ami az `ls` parancs paramétereit határozza meg. Könyvtárváltás pedig a `cd` paranccsal történik. Ezeket a parancsokat, és az aktuális művelet által meghatározott parancsot összefűzve átadjuk az `execute` metódusnak, és az eredményt a kimenet feldolgozásával állítjuk elő.

A könyvtárlista felépítéséért és az aktuális műveletek elvégzéséért felelős kódrészletet a 50. ábra mutatja.

```
{./index.php}
<?php
...
$_cd;                               // Könyvtárváltás teljes útvonala
$_pwd;                              // Az aktuális könyvtár
$_directory_content_string = null;  // A könyvtár tartalma
$_subject = ($_GET['subject']);     // Az aktuális tárgy

// A rejtett fájlok megjelenítése
$_hidden = isset($_GET['hidden']) && !empty($_GET['hidden']) &&
                        $_GET['hidden'] == "true" ? "true" : "false";
...
$_directory_content = array();      // Könyvtár tartalma
$_command_output;                   // Az aktuális parancs kimenete

$prog_tesztelo = new ProgTesztelo(
        $title, $server, $helper->getUsername(), $helper->getPassword()
);

$prog_tesztelo->connect();

// könyvtárváltás
if (isset($_GET['cd']) && !empty($_GET['cd'])){
   $command = "echo 'CHANGE START'; cd " . $_GET['cd'] .
              "; echo 'CHANGE END';";
}
else{
   $command = "echo 'CHANGE START'; cd ~; echo 'CHANGE END';";
}

// aktuális könyvtár
$command .= "echo 'PWD START'; pwd; echo 'PWD END';";

// műveletek
if (isset($_GET['action']) && !empty($_GET['action'])){
   $action = trim($_GET['action']);
   $value = (isset($_GET['value']) && !empty($_GET['value'])
            ? $_GET['value'] : "");

   switch($action){
      ...
   }
}

// rejtett fájlok megjelenítése
if ($_hidden == "true")
{
   $command .= "echo 'LIST START'; ls -Aog; echo 'LIST END';";
}
else
{
   $command .= "echo 'LIST START'; ls -og; echo 'LIST END';";
}
```

```
$command .= "echo 'COMMAND START'; $s echo 'COMMAND END';";

$output = $prog_tesztelo->execute($prog_tesztelo->getShell(), $command);

$response = $prog_tesztelo->readResponse($output);

/*
 * Válasz feldolgozása
 */
if (isset($response) && !empty($response))
{
   $cs = "CHANGE START";
   $ce = "CHANGE END";
   $ps = "PWD START";
   $pe = "PWD END";
   $ls = "LIST START";
   $le = "LIST END";
   $ds = "COMMAND START";
   $de = "COMMAND END";

   $_cd = trim(substr($response,
   strpos($response, $cs) + strlen($cs),
   strpos($response, $ce) - (strpos($response, $cs) + strlen($cs))));

   $_pwd = trim(substr($response,
   strpos($response, $ps) + strlen($ps),
   strpos($response, $pe) - (strpos($response, $ps) + strlen($ps))));

   $_directory_content_string = trim(substr($response,
   strpos($response, $ls) + strlen($ls),
   strpos($response, $le) - (strpos($response, $ls) + strlen($ls))));

   $directory_content_array = split("\n", $_directory_content_string);

   $_command_output = substr($response, strpos($response, $ds) +
                      strlen($ds)) . "\n\n";

   $_command_output = str_replace($de, "", $_command_output);
   $_command_output = str_replace("<", "&lt;", $_command_output);
   $_command_output = str_replace(">", "&gt;", $_command_output);
   $_command_output = str_replace("\r\n", "\n", $_command_output);
}
...
?>
```

50. ábra: A könyvtárlista felépítéséért és az aktuális műveletek elvégzéséért felelős kódrészlet

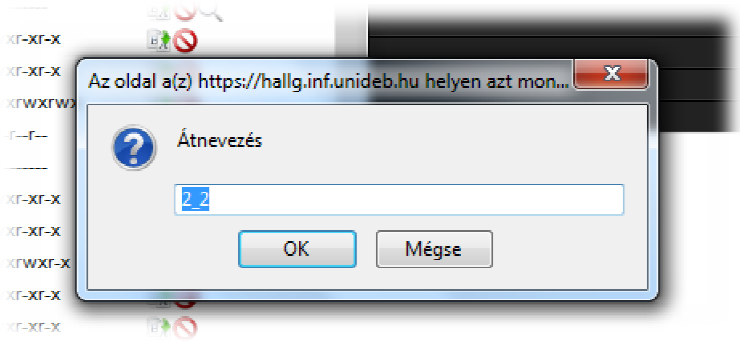
3.4.2.2. Fájlműveletek

Helytakarékossági okokból a legfontosabb fájlműveletek (*Átnevezés*, *Törlés*, és fájlok esetén *Nézőke*) a *műveletek* oszlopban kaptak helyet. Azt, hogy éppen milyen műveletet kell

elvégezni, az `action`, a művelet elvégzéséhez szükséges változókat – például a fájlnevet – a `value` query string paraméter értéke alapján határozzuk meg, a felhasználó megerősítését váró dialógus ablakokat és az adatok validálását pedig JavaScript függvények segítségével végezzük el.

3.4.2.2.1. Átnevezés

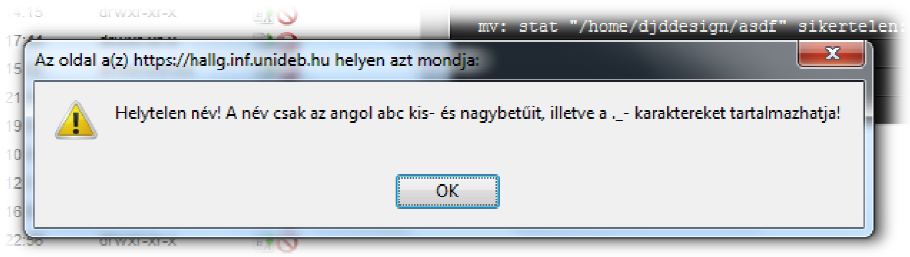
Amennyiben egy adott fájl/könyvtár adatai mellett levő *Átnevezés* ikonra kattintunk, akkor a 51. ábrán látható felugró ablakkal találkozunk. Itt adhatjuk meg az állomány új nevét.



51. ábra: Átnevezés

Az átnevezés funkció esetén a kliens alkalmazás esetében megismert módon a karakterkódolási problémák elkerülése végett csak a szigorú Unix fájlnév-konvencióknak megfelelő neveket fogadjuk el. A beírt nevet a `nameValidator` JavaScript függvény segítségével ellenőrizzük (52. ábra), és helytelen fájlnév esetén az 53. ábrán látható hibaüzenetet jelenítjük meg.

52. ábra: Fájlnév-validáló függvény



53. ábra: Helytelen fájlnév

Helyes név megadása esetén frissítjük az oldalt, és az URI-ben az `action=rename` paraméterrel jelöljük a vezérlő számára, hogy átnevezés történik. A régi és az új nevet egy `|` karakterrel elválasztva a `value` paraméternek adjuk értékül – ezeket a fájlneveket kapja meg az `mv` parancs, ami elvégzi az átnevezést (54. ábra).

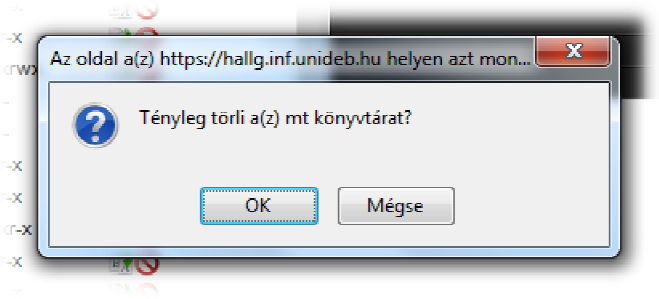
```
                                                    e'])
                                                    $filename')
                                                    bject=$_subject
                                                    |$_pwd/'+_rename;
```

```
                                                    kis- és
                                                    azhatja!');
```

54. ábra: Átnevezés

3.4.2.2.2. Törlés

Egy fájl vagy könyvtár törléséhez kattintsunk a *Törlés* ikonra. Ekkor a rendszer először a felhasználó megerősítését kéri (55. ábra) – ha az *OK* gombra kattintunk, akkor töröljük az adott állományt a szerverről.



55. ábra: Törlés megerősítése

Az állományok törlésére az `rm -r` parancs szolgál. A törlés végrehajtása az előző pontban látottakhoz hasonlóan történik, csak ebben az esetben az `action` paraméter értéke `delete` lesz, és a `value` paraméter a törlendő állomány teljes elérési útvonalát tartalmazza.

```
{./index.php}
<?php
...
// műveletek
if (isset($_GET['action']) && !empty($_GET['action']))
{
   $action = trim($_GET['action']);
   $value = (isset($_GET['value']) && !empty($_GET['value'])
            ? $_GET['value'] : "");
   switch($action)
   {
      ...
      // Törlés
      case "delete":
      {
         $s = "rm -r $value;";
         break;
      }
      ...
   }
}
...
$_delete =
   "var _delete=confirm('Tényleg törli a(z) $filename " .
                                ($isDir ? "könyvtárat" : "fájlt") . "?');
   if (_delete) {
      window.location = '$path/index.php?cd=$_pwd&subject=$_subject&
                          hidden=$_hidden&action=delete&value=$filepath';
   }";
...
?>
```

56. ábra: Törlés

3.4.2.2.3. Nézőke

A webes felület esetében is lehetőségünk van megtekinteni egy fájl tartalmát. Ehhez mindössze a kiválasztott fájl sorában található *Nézőke*  ikonra kell kattintanunk. Ekkor az oldal jobb oldalán található `<div>`-ben jelenik meg az adott fájl tartalma (57. ábra), amely kód akár tetszőleges külső alkalmazásba is átmásolható.

57. ábra: Nézőke – `Alakzat.java`

A portál jelenleg csak a fájlok tartalmának megtekintését teszi lehetővé. A művelet végrehajtása az előző két pontban leírtakhoz nagyon hasonló módon történik, csak ebben az esetben a `cat` parancsot kell használnunk a fájl tartalmának megjelenítéséhez (58. ábra).

```
{./index.php}
<?php
...
// műveletek
if (isset($_GET['action']) && !empty($_GET['action']))
{
   $action = trim($_GET['action']);
   $value = (isset($_GET['value']) && !empty($_GET['value'])
             ? $_GET['value'] : "");
   switch($action)
   {
      ...
      // Nézőke
      case "view":
      {
         $s = "cat $value;";
         break;
      }
      ...
   }
}
...
$_view = "window.location =
         '$path/index.php?cd=$_pwd&subject=$_subject&hidden=$_hidden&
          action=view&value=$filepath';";
...
?>
```

58. ábra: Nézőke

3.4.2.3. Speciális könyvtárakba ugrás

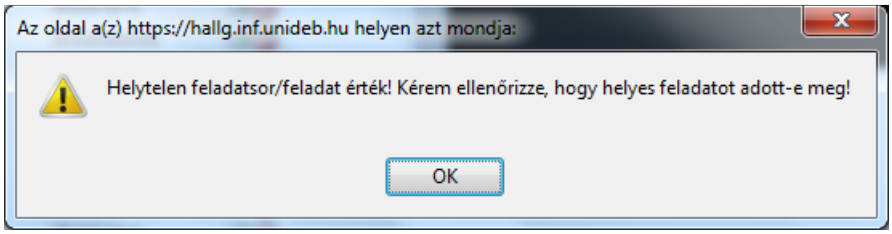
A főbb könyvtárak (*Munkakönyvtár*, *Zsűri*, *Tesztesetek* és *Beadott fájlok*) a portálon keresztül is egy kattintással elérhetőek (59. ábra). A linkekre kattintva elérhető könyvtárakat az aktuális tárgyhoz tartozó adatbázis rekord mezői határozzák meg (60. ábra).

59. ábra: Speciális könyvtárak elérése a portál használatával

```
{./index.php}
<?php
...
// Munkakönyvtár
$_goto_home_dir =
   "$path/index.php?cd=~&subject=$_subject&hidden=$_hidden";

// Zsűri
$_goto_jury_dir = $_current_subject_params['zsuri'] == "" ? "#" :
   "$path/index.php?cd=" . $_current_subject_params['zsuri'] .
   "&subject=$_subject&hidden=$_hidden";

// Tesztesetek
$_goto_test_dir = $_current_subject_params['tesztesetek'] == "" ? "" :
   "var c = correctExercise('".$_current_subject_params['tesztesetek'].
   "');
   if (c != '') {
      window.location = '$path/index.php?cd=' + c +
                        '&subject=$_subject&hidden=$_hidden'; }";

// Beadott fájlok
$_goto_submitted_dir = $_current_subject_params['beadott_fajlok'] == ""
   ? "" : "var c = correctExercise('" .
           $_current_subject_params['beadott_fajlok'] . "');
   if (c != '') { window.location = '$path/index.php?cd=' + c +
          '&subject=$_subject&hidden=$_hidden'; }";
...
?>
```

60. ábra: Speciális könyvtárak hivatkozásai

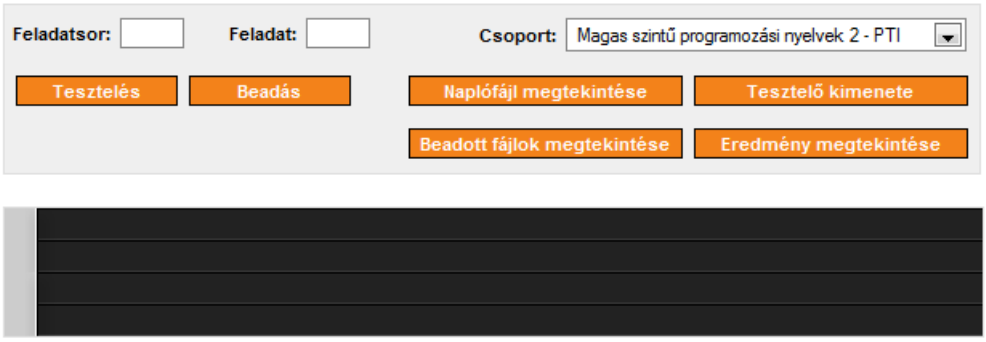
A *Tesztesetek* és *Beadott fájlok* esetében kötelező megadni a feladatsor és feladat értékét, ezek meglétét és helyességét a `correctExercise` JavaScript-függvénnyel ellenőrizzük (**`./core/code/js/coreFunctions.js`**), amennyiben helytelenek az értékek, az 61. ábrán látható figyelmeztető üzenetet küldjük a felhasználónak.



61. ábra: Helytelen feladatsor/feladat érték figyelmeztetés

3.4.2.4. Tesztelő felület

A tesztelő felület az *Automatikus Programtesztelő Kliens* tesztelő felületének pontos másá, ugyanazok a funkciók érhetőek el a portálon is, mint a kliens esetében (62. ábra).



62. ábra: Az Automatikus Programtesztelő Portál tesztelő felülete

A legördülő menüből választhatóak ki az elérhető csoportok. A lista az adatbázisban levő rekordok alapján épül fel, és a különböző funkciók csak akkor válnak csak elérhetővé, ha a kiválasztott csoporthoz tartozó adatbázis-rekord adott mezője tartalmaz értéket.

Az elérhető funkciók (*Tesztelés*, *Beadás*, *Naplófájl megtekintése*, stb.) az *Automatikus Programtesztelő Kliens* esetében részletezett algoritmusok alapján lettek ebben az esetben is megvalósítani, csak a platform, és ennek megfelelően a megvalósítás módja változott meg.

Azoknál a funkcióknál, amelyekhez kötelező megadni a feladatsor és feladat értékeit, a speciális könyvtárak hivatkozásainál használt `correctExercise` JavaScript-függvénnyel ellenőrizzük a beírt adatokat, és amennyiben helyes értéket adtunk meg, a `window.location` függvény segítségével az adott funkciónak megfelelő query string paraméterekkel újratöltjük az oldalt. A műveleteket feldolgozó kódrészlet a 63. ábrán látható.

Az egyes műveletek eredménye a gombok alatt található -ben jelenik meg, így itt követhető nyomon a művelet sikeressége vagy sikertelensége is.

```
{./index.php}
<?php
...
// Tesztelés
$_test = $_current_subject_params['tesztel'] == ""
   ? ""
   : "var c = correctExercise('" . $_current_subject_params['tesztel'] .
      "');
       if (c != '') {
           window.location =
           '$path/index.php?cd=$_pwd&subject=$_subject&hidden=$_hidden&
            action=test&value=' + c; }";

// Beadás
$_submit = $_current_subject_params['bead'] == ""
   ? ""
   : "var c = correctExercise('" . $_current_subject_params['bead'] .
     "');
      if (c != '') {
         window.location =
         '$path/index.php?cd=$_pwd&subject=$_subject&hidden=$_hidden&
          action=submit&value=' + c; }";

// Naplófájl megtekintése
$_view_log_file = "window.location =
   '$path/index.php?cd=$_pwd&subject=$_subject&hidden=$_hidden&
    action=view_log_file';";

// Tesztelő kimenete
$_view_test_out = "window.location =
   '$path/index.php?cd=$_pwd&subject=$_subject&hidden=$_hidden&
    action=view_test_out';";

// Beadott fájlok megtekintése
$_view_submitted_files = $_current_subject_params['beadva'] == ""
   ? ""
   : "var c = correctExercise('" . $_current_subject_params['beadva'] .
      "'); if (c != '') { window.location =
       '$path/index.php?cd=$_pwd&subject=$_subject&
        hidden=$_hidden&action=view_submitted_files&value=' + c; }";

// Eredmény megtekintése
$_view_results = $_current_subject_params['eredmeny'] == ""
   ? ""
   : "window.location = '$path/index.php?cd=$_pwd&subject=$_subject&
      hidden=$_hidden&action=view_results&value=" .
      $_current_subject_params['eredmeny'] . "';";
```

```
...
// műveletek
if (isset($_GET['action']) && !empty($_GET['action'])){
   $action = trim($_GET['action']);
   $value = (isset($_GET['value']) && !empty($_GET['value']) ?
             $_GET['value'] : "");
   switch($action){
      ...
      // Tesztelés
      case "test":
      {
         $s = "rm teszt.log; $value > test.tmp;
               cat test.tmp; rm -r test.tmp;";
         break;
      }
      // Beadás
      case "submit":
      {
         $s = "yes igen | $value > submit.tmp;
               cat submit.tmp; rm -r submit.tmp;";
         break;
      }
      // Naplófájl megtekintése
      case "view_log_file":
      {
         $s = "cat teszt.log;";
         break;
      }
      // Tesztelő kimenete
      case "view_test_out":
      {
         $s = "cat teszt.out;";
         break;
      }
      // Beadott fájlok megtekintése
      case "view_submitted_files":
      // Eredmény megtekintése
      case "view_results":
      {
         $s = "$value;";
         break;
      }
      default:
      $s = "";
   }
}
...
?>
```

63. ábra: Tesztelő felület műveletei

3.4.3. Kijelentkezés

Feladatunk végeztével a szerverről a jobb felső sarokban található *Kijelentkezés* linkre kattintva jelentkezhetünk ki. Ekkor a felhasználó számítógépén tárolt sütik törlődnek, és a bejelentkező oldalra irányítódik át a honlap.

3.5. Az adatok biztonsága

A webes felület esetében fontos megoldandó problémának bizonyult az adatok biztonságos továbbítása a felhasználó számítógépe és a szerver között, hiszen ezen a platformon tárolni kell a felhasználói adatokat, amihez titkosítás nélkül illetéktelenek is könnyen hozzáférhetnek.

Ezt kiküszöbölendő a portál csak biztonságos kapcsolaton (HTTPS) keresztül érhető el, illetve a felhasználó számítógépén létrehozott sütikben az adatok titkosítva lettek (3DES titkosítás), ezzel biztosítva, hogy az adott felhasználó adataihoz illetéktelenek ne férjenek hozzá.

4. Tapasztalatok, visszajelzések

Az *Automatikus Programtesztelő Kliens* letölthető az Információ Technológiai Tanszék *it.inf.unideb.hu/honlap/prog2* oldaláról, az *Automatikus Programtesztelő Portál* pedig a *hallg.inf.unideb.hu/prog* címen érhető el.

A két alkalmazást a 2009/10-es tanév első félévétől használhatják a hallgatók, a visszajelzéseket pedig a *hallg.inf.unideb.hu/phpbb* fórumon, illetve személyesen vagy emailben közölhetik.

Az eddigi tapasztalatok nagyon pozitívak, a hallgatók előszeretettel használják, tesztelik az alkalmazásokat. Az eddigi hibajelentések alapján számos kisebb-nagyobb hiba került kijavításra, és az alkalmazások tökéletesítése a folyamatosan beérkező visszajelzések alapján minél előbb megtörténik.

5. Jövőbeni fejlesztések

A két alkalmazás korántsem tökéletes, éppen ezért folyamatos fejlesztésre, csiszolásra szorul, amihez a személyes tesztelés mellett nagyban hozzájárulnak a hallgatók visszajelzései is.

A közeljövőben szeretném a fejlesztésben a hangsúlyt az *Automatikus Programtesztelő Portál*ra fektetni, ezáltal egy olyan komplex webes alkalmazást elkészíteni, ami a desktop alkalmazás összes funkcióját ötvözi az online platform nyújtotta szabadsággal. A jelenlegi portál köré tervben van egy komplex rendszer kiépítése, ami nemcsak a tesztelést biztosítaná, hanem egy globális hallgatói tudásbázist is jelentene – lehetőséget biztosítva például jegyzetek és tapasztalatok megosztására, hallgatók és oktatók közötti villámgyors kommunikációra.

A jelenlegi alkalmazás és a tesztrendszer rugalmassága miatt könnyen illeszthető további tantárgyak vagy versenyek (például ACM) tesztelési feladatainak megvalósításához, ezáltal a jelenlegi alkalmazás egy komplex, több területet érintő programtesztelő- és nyilvántartó rendszerré válhat.

6. Összefoglalás

Szakdolgozatomban az automatikus programtesztelő rendszerekben rejlő lehetőségekre szerettem volna felhívni a figyelmet. Munkám eredményeként létrejött két olyan alkalmazás, amely modern, a huszonegyedik század trendjeinek megfelelő felületen keresztül biztosít hozzáférést az Informatikai Karon működő programtesztelő rendszerhez, ezáltal egyszerűbbé és hatékonyabbá téve a hallgatók számára a programok tesztelését és beadását, valamint az eredmények megtekintését.

A rendszerben további lehetőségek rejlenek, ezért szeretném a későbbiek során továbbfejleszteni a dolgozatban tárgyalt alkalmazásokat.

7. Irodalomjegyzék

[1] Espák Miklós (2008): Automatikus tesztelő rendszerek szerepe a tömegképzésben. Informatika a felsőoktatásban 2008, Debrecen

[2] Dr. Kovács Emőd, Hernyák Zoltán, Radványi Tibor, Király Roland (2005): A C# programozási nyelv a felsőoktatásban Programozási tankönyv. Eger: Eszterházy Károly Főiskola.
http://aries.ektf.hu/csharptk/ (2009.11.01.)

[3] George Schlossnagle (2004): PHP fejlesztés felsőfokon. Budapest: Kiskapu Kft.

[4] MSDN Library
http://msdn.microsoft.com/en-us/library/default.aspx (2009.11.01.)

[5] PHP Manual
http://php.net (2009.11.01.)

[6] PHP: SSH2 – Manual
http://php.net/manual/en/book.ssh2.php (2009.11.01.)

[7] Tamir Gal (2007): SharpSSH - A Secure Shell (SSH) library for .NET.
http://www.tamirgal.com/blog/page/SharpSSH.aspx (2009.11.01.)

[8] Sindhumn (2005): Image ComboBox Control
http://www.codeproject.com/KB/combobox/ImageComboBoxControl.aspx
(2009.11.01.)

[9] Abhinav Misra (2008): Create a File Browser using C#
http://www.dotnetspider.com/resources/23565-Create-File-Browser-using-C.aspx
(2009.11.01.)

[10] The ShellExecuteEx API
http://www.pinvoke.net/default.aspx/shell32/ShellExecuteEx.html (2009.11.01.)

[11] Rockin J: Get all files of directory and sub directories using ASP.NET, C#, and VB.NET
http://www.progtalk.com/ViewArticle.aspx?ArticleID=77 (2009.11.01.)

[12] Tim Mackey (2006): FtpWebRequest with ProgressBar [C# Ftp]
http://tim.mackey.ie/FtpWebRequestWithProgressBarCFtp.aspx (2009.11.01.)