Eötvös Loránd Tudományegyetem

Informatikai Kar

Információs Rendszerek Tanszék

# FELHŐALAPÚ OSZTÁLYOZÓRENDSZER

dr. Kiss Attila Elemér Gyenes Adrienn

Tanszékvezető Programtervező informatikus

Budapest, 2018

Tartalomjegyzék

1. Bevezetés

1.1 Big Data, mint fogalom

1.2 Machine Learning: Classifierek

1.3 Megoldandó feladat és megvalósítás

2. Felhasználói dokumentáció

2.1 Rendszerkövetelmények

2.2 Szerver beüzemelése

2.3 Installációs előfeltételek és indítás

2.4 Ismerkedés a felülettel

3. Fejlesztői dokumentáció

3.1 Problématerv

3.2 Szerver felépítése

3.2.1 Server.java

3.2.2 ClientThread.java

3.2.3 DBHandler belsőosztály

3.3 Az adatbázis

3.4 A kliens felépítése

3.4.1 A bejelentkezőablak logikai felépítése

3.4.2 A főablak logikai felépítése

3.5 Funkcionális követelmények

3.6 User-story

3.7 Use-case diagram és elemei

3.8 Nemfunkcionális követelmények

3.9 Tesztesetek

4. Hivatkozások

**1. Bevezetés**

**1.1 Big Data, mint fogalom**

Napjainkban szinte minden adat rögzítésre kerül, emiatt az egyes eszközökhöz, programokhoz tartozó gyűjtött adathalmazok rohamosan növekednek az idő múlásával. Ezen adathalmazok elemzésével rengeteg hasznos információ kinyerhető, így ezek elemzésére rohamosan nő az igény. Erre a célra alakult ki az informatika egy új, specializált ágazata, az Adattudomány. Ennek a tudományágnak a keretein belül alakult ki a Big Data, mint fogalom. A következőkben ezt a fogalmat és a hozzátartozó tulajdonságokat mutatom be.

A Big Data fogalmát John Mashney alkotta meg a 90-es években, mikor ez a tudományág elkezdett az érdeklődési középpontjába kerülni. Definíció szerint a Big Data olyan technológiák együttese, amik képesek biztosítani hatalmas méretű, diverz adathalmazok kezelését, elemzését és feldolgozását elfogadható időn belül. [1]

Ezt a definíciót tovább gondolva 2001-ben a META Group (ma Gartner) meghatározta az adatok növekedésének mértékét egy 3 dimenziós modellben, aminek elemei a növekedő mennyiség, sebesség és változatosság. A továbbiakban ezt a 3Vs-modellt használták az adatok leírására:

* Volume: a Big Data nem vételez adatot; csupán megfigyeli és követi a történteket
* Velocity: a Big Data általában valós idejű feldolgozást nyújt
* Variety: a Big Data állhat szövegből, képből, hanganyagból. A hiányzó adatok adatfúzióval helyettesíthetőek.

A 3Vs-modell jellemzőin kívül fontos megemlíteni a fogalomhoz szorosan tartozó további két tulajdonságot: a számítógépes tanulás (Machine Learning), illetve a digitális lábnyom (Digital Footprint) elméletét.

A Digital Footprint, vagy más néven Digital Shadow olyan egyedi, lekövethető digitális aktivitásokat, kommunikációkat illetve közreműködéseket takar, amik az interneten, digitálisan történnek[2], ezáltal a feldolgozni kívánt nagy adathalmazok legtöbbször ilyen interakciók lenyomatait tartalmazza. A Machine Learning pedig egy olyan megközelítési módja az adathalmazok feldolgozásának, amely nem kérdőjelez meg semmit, ami az adathalmazban fellelhető, csupán mintát keres, ami alapján a későbbiekben következtetéseket lehet levonni. [3]

**1.2 Machine Learning: Classifierek**

Dolgozatom fő témáját a Machine Learning algoritmusok csoportjába tartozó bizonyos Classifier-ek, magyarul osztályozók adják. Ezen eljárások segítséget nyújtanak abban, hogy az adathalmazok egyes elemeit logikailag különböző csoportokba tudjuk osztani. Ez hasznos lehet például olyan esetekben, amikor egy kereskedelmi cég bejövő, illetve kimenő árukészletét és fogyasztási adatait figyeljük: a korábbi feljegyzések alapján meghatározható, hogy bizonyos árucikkekre mennyi a kereslet-kínálati arány, van-e értelme ezt az árut továbbiakban készleten tartani.

A Machine Learning terminológiájában az osztályzók egyfajta felügyelt tanulást végeznek, azaz a tanulási folyamathoz elérhető egy úgynevezett training halmaz, ami alapján a betanulás, minta keresés történik, és egy test halmaz, amire már az algoritmus futása élesben történik. A két halmaz együttes vizsgálatával megkapható egy pontosságot kifejező (accuracy) érték, ami a tanulás sikerességét adja meg az adott paraméterekkel. Ezen algoritmus nem-felügyelt módja a klaszterezés, ami az elemek hasonlósága és távolsága alapján osztja csoportokba az adathalmaz tagjait.

A legismertebb ilyen algoritmusok közé tartozik a Random Forest Classifier, ami döntési fák összessége, az AdaBoost Classifier, amely más tanuló algoritmusokat használ a teljesítményének növeléséhez és a Gradient Boosting Classifier, ami egy előrejelző modellt készít, gyengébb modellek összességéből. Ezek a modellek jellemzően döntési fák.

A döntési fák olyan struktúrák, amelyek egy olyan fa-szerű gráfot használnak alapul, amelynek elemei a döntések, és a hozzá tartozó következmények továbbá tartalmazza a beteljesülés valószínűségét is, költségét és hasznosságát is.

**1.3 Megoldandó feladat és megvalósítás**

Dolgozatom célja egy olyan szerver-kliens kapcsolatot kialakító program, amely elsősorban laikus felhasználóknak segít nagyobb méretű adattáblákat előfeldolgozni, majd ezen algoritmusok végrehajtása után osztályozni is. Regisztrálás és az ezt követő bejelentkezés után egyes felhasználók egy olyan felületet érhetnek el, ahol külön munkamenetekbe feltöltött .csv formátumú adattáblákat tudnak kezelni.

Előfeldolgozás esetén lehetőség van a hiányzó értékek helyettesítésére, és felesleges oszlopok szelektálására is, mely utóbbi attól függ, hogy a felhasználó, vagy egy beépített algoritmus (feature selection) ítéli-e meg feleslegesnek. Ezentúl az adatok faktorizálására, vagyis számszerűsítésére, illetve normalizálásra is lehetőséget ad.

Ezt követően, illetve ha a tábla szerkezete megengedi, akár ezt kihagyva, a kliensoldalon osztályozó válaszható, aminek paramétereit egy grafikus felületen lehet kiválasztani, majd ezekkel futtatni a kívánt algoritmust egy előre kiválasztott adattáblára. Az ismert osztályozó algoritmusokon kívül (AdaBoost, Random Forest) van lehetőség célzott osztályzásra is, amely Sentiment Analysis, azaz érzelmi analízis formájában jelenik meg. Ez az algoritmus lehetővé teszi, hogy a kiválasztott tábla egy adott attribútumára vagy attribútum csoportjára olyan műveleteket hajtsunk végre, ami visszaadja azok érzelmi töltetét.

A műveletek, amiket a felhasználó kiválaszt, a szerverhez jutnak el és ott lokálisan, előre megírt python nyelven implementált kódokat felhasználva zajlik le az adattáblák kezelése. Ehhez a nyelvbe beépített modulokat használja fel a program, név szerint a pandas modult, ami az adattáblák könnyed kezelését teszi lehetővé, a numpy modul, ami matematikai műveleteket valósít meg és az scikit-learn, ami tartalmazza az osztályozó algoritmusokat. Ha ezek a python kódok sikeresen lefutottak, a felhasználó készen kapja vissza az adattáblákat előnézetre, majd ha ezzel elégedett, letöltheti őket a szerverről.

A szerver és a kliens megvalósítása Java nyelven történik, az adatok tárolása pedig a szerveren elhelyezett, felhasználók számára fenntartott mappákban, illetve a futás idejű adatok hsql adatbázisban helyezkednek el.

**2. Felhasználói dokumentáció**

**2.1 Rendszerkövetelmények**

A program futásához szükséges rendszerkövetelmények közé tartozik bármely olyan operációs rendszer megléte, amely .jar programok futtatását lehetővé teszi. Ajánlott operációs rendszerek közé tartoznak a Windows 7 és a nála frissebb, ezen belül is a 64 bites Windows verziók. Hardver esetben nincs megkötés, a felhasználó számítógépére feltelepített rendszer igényei szerinti beállításokat ajánlott alkalmazni.

**2.2 Szerver beüzemelése**

A szerver távolról elérhető a kliensprogramnak, de amennyiben az éppen nem üzemel, lehetőségünk van azt manuálisan bekapcsolni. Ehhez szükségünk van a PuTTY vagy a WinSCP legfrisebb verziójára, aminek segítségével kapcsolatot alakíthatunk ki a távoli gépre, ami ezesetben az erdos.inf.elte.hu. Ha beléptünk, két egymástól független parancssori ablakot nyitunk, egyet az adatbázisszerver eléréséhez, egyet pedig magának a szerverünknek. Mivel a program a ./home/gyenesadrienn mappában található, célszerű ide navigálni a parancsok futtatásához. Ha ez megtörtént, a következőket kell tennünk:

1. ablak: Adatbázisszerver elindítása a következő paranccsal:

java -classpath ./gyenesadrienn/hsqldb/lib/hsqldb.jar org.hsqldb.server.Server   
--database.0 file:hsqldb/db --dbname.0 db

1. ablak: Programszerver elindítása:

java –D –jar server.jar

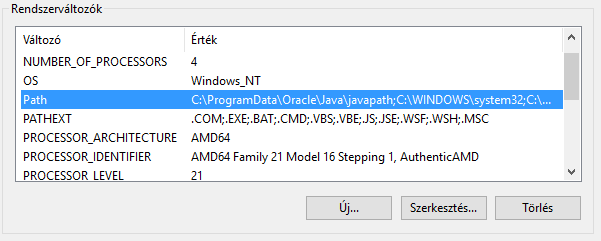
Ha mindent jól csináltunk, akkor a következő fogad minket a programszervernek nyitott parancssori ablakban:



1. ábra: Szerver helyes működése

**2.3 Installációs előfeltételek és indítás**

A program olyan 64 bites Windows operációs rendszerekre (Windows 7, 8, 10) van kiépítve, amelyek rendelkeznek aktív internetkapcsolattal, így alapkövetelmény ezen két feltétel teljesülése a felhasználó számítógépén. Bár a program logikailag két részre osztható, és kliensre és egy szerverre, a szerver távoli hozzáférést biztosít, így csak a kliensoldali program installációs előfeltételeit szükséges biztosítani a számítógépen. Ezesetben a teendő a következő: mivel kliensoldali program Java nyelven van megvalósítva, így felhasználó számítógépén szükség van a megfelelő verzióval rendelkező Java JDK telepítésére. Ezesetben a Java JDK 8-ról van szó, ami az Oracle oldalán elérhető, és ingyenes letölthető.[4] Letöltés után a JDK-t telepítjük, majd ha ez nem történt meg automatikusan, a felvesszük a környezeti változók közé. Ezek elérése Windows operációs rendszeren a vezérlőpultban található Rendszer tulajdonságain belül a Speciális fülön a Környezeti változók… gombra kattintva történik.



2. ábra: Rendszerváltozók, Path elérése

A Path-ot kiválasztva és a Szerkesztés gombra kattintva érhető el a korábbi változók listája, ezen a felületen tudjuk hozzáadni a mappát, amely tartalmazza a letöltött JDK útvonalát, ami általában így néz ki:

C:\Program Files\Java\jdk1.8.0\_111\bin

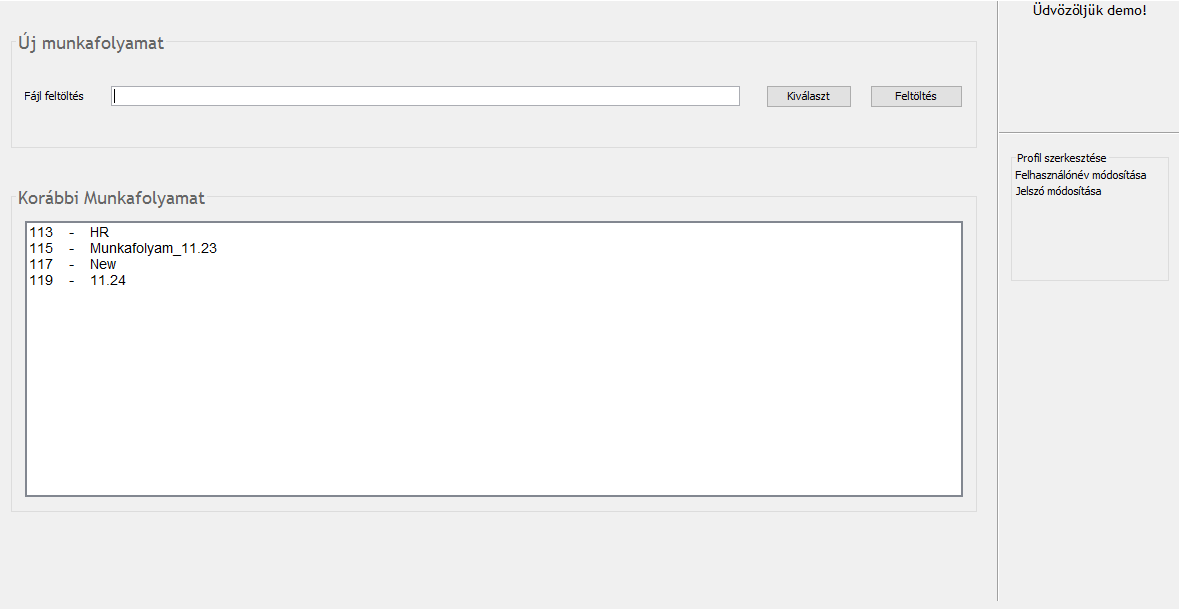
Miután a kliens telepítéséhez szükséges előfeltételeket teljesítettük, elindíthatjuk a programot. Az elindításra két lehetséges módot lehet alkalmazni: parancssori illetve grafikus felületű indítást. Ez utóbbi megoldást választva az indítás egyszerűen a cloud-based.jar ikonjára történő duplakattintással kezdhető. Ezután a program rendeltetésszerűen elindul. Parancssor választása esetén szükség van a rendszer parancssorának elindítására, amit a Futtatás>cmd paranccsal tehetünk meg. Ezt követően elnavigálunk a mappába, ami tartalmazza a .jar fájlt, és itt a következő paranccsal tudjuk azt működésre bírni:

java -D –jar cloud-based.jar

**2.4 Ismerkedés a felülettel**

Elindítás után elsőnek a bejelentkező felület jelenik meg, ahonnan át lehet navigálni a regisztrációs felületre, ahol a még nem regisztrált felhasználók megtehetik a kezdéshez szükséges lépéseket. A felhasználónévnek és az e-mail címnek egyedinek kell lennie, így ha ez nem teljesül, vagyis létezik már felhasználó ilyen adatokkal, akkor hibaüzenet érkezik. Sikeres regisztráció után ismét a bejelentkező felület látható.

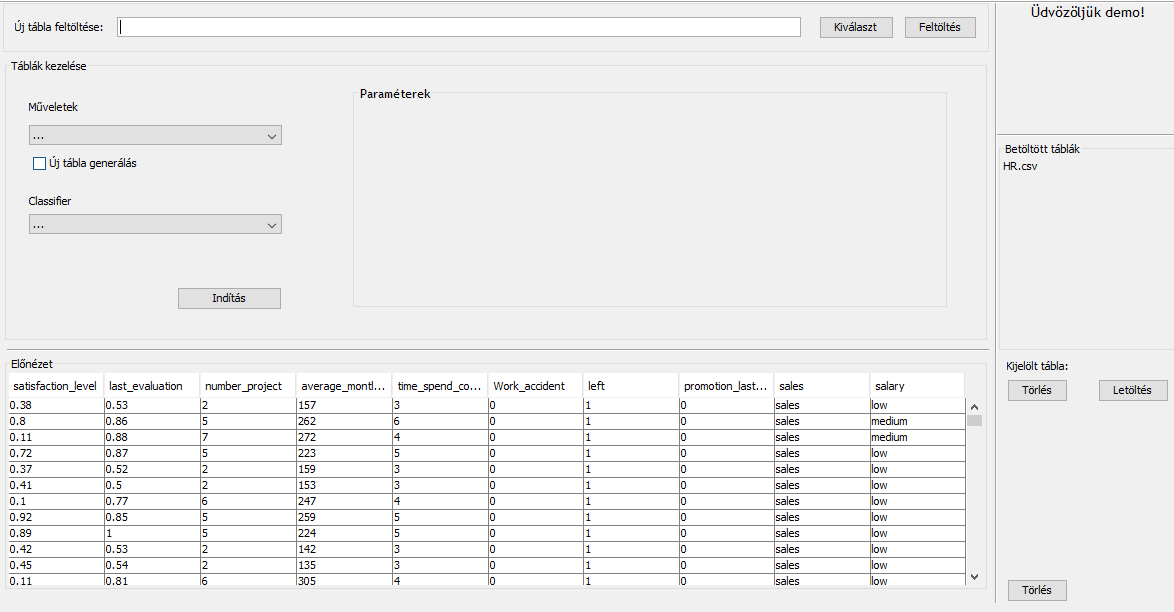
Bejelentkezés után a felhasználó egy olyan ablakot lát, ahol lehetőség van új tábla felvitelére a kiválasztás/feltöltés opciók kiválasztásával. Ezzel egyidőben létrejön egy új munkamenet, aminek a nevét a feltöltés után megadhatjuk. A táblafeltöltés ideje változó lehet, attól függően milyen méretű a kiválasztott tábla.



3. ábra: Az osztályozórendszer főablaka

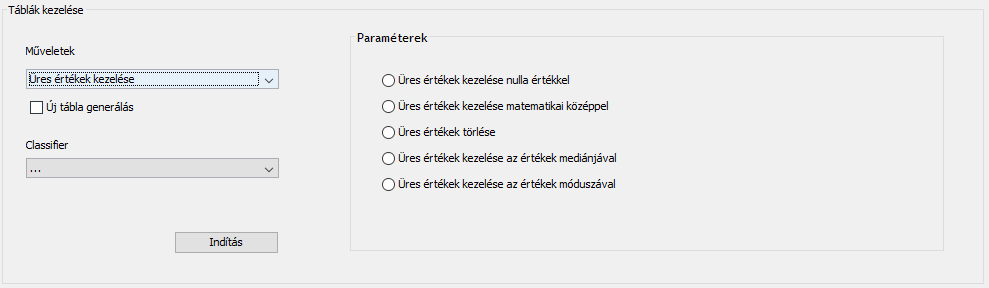
Ha a felhasználó már korábban használta a programot, és vannak korábban elkezdett munkái, akkor azok megjelennek egy listában, kiválasztva betölthetőek.

Ha a kiválasztott, vagy újonnan létrehozott munkamenet megnyitásra került, megjelenik a fő ablak, ahol a munka történik. A korábbi ablakhoz hasonlóan itt is hozzáadható új tábla, ez automatikusan a betöltött munkafolyamatba kerül, pontosabban oldalra, a betöltött táblák listájába. Ez a lista tartalmazza a kezdeti táblát is, a lista elemekre kattintva az ablak alján lévő előnézeti panelbe betöltődik a kiválasztott csv-fájl első 250 sora.



4. ábra: Munkaablak, betöltött tábla előnézetével

Ha kiválasztottuk, hogy a feltöltött táblák közül melyikkel szeretnénk tovább dolgozni, tovább mehetünk az előfeldolgozásra, illetve az osztályozó algoritmusok paraméter beállításaira is. Ezt a munkaablakon lévő *Táblák kezelése* nevű panelen tudjuk megtenni. Minden előfeldolgozó és osztályozó algoritmushoz tartozik egy felület, ahol a szükséges paramétereket be tudjuk állítani, ha az szükséges. Kivételek ezalól az egyszerűbb, paraméter nélküli műveletek, például a Faktorizálás vagy a Normalizálás, ezért azokat csak egyszerűen az *Indítás* gombra való kattintással kezdhetünk meg.

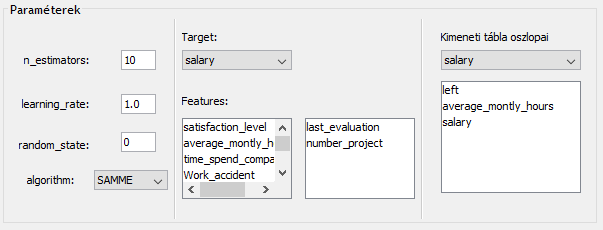


5. ábra: Üres értékek kezeléséhez szükséges paraméterablak

Mint ahogy a 4. ábrán is látszik, minden előfeldolgozó művelethez lehetőség van az *Új tábla generálás* opciót kiválasztani. Ez annyit tesz, hogy a táblán elvégzett művelet nem az eredeti táblát fogja befolyásolni, hanem létrehoz egy új, már módosított táblát, ami a művelet végrehajtása után megjelenik a *Betöltött táblák* listában. Az eredménytábla előnézete betöltődik, és alapértelmezetten ez kerül kijelölésre. Ezzel az új táblával természetesen ugyanúgy dolgozhatunk a továbbiakban, mint egy korábban általunk feltöltöttel.

Ha az előfeldolgozással végeztünk, vagy ha a tábla szerkezetéből adódóan ezt a lépést átugrottuk, a *Classifier* legördülő menüből kiválaszthatjuk, hogy melyik osztályozót szeretnénk a táblánkra alkalmazni. Ha ez megtörtént, a *Paraméterek* ablakban megjelenik az osztályozóra jellemző paraméterek megadását segítő ablak. Ezen az ablakon az osztályozó-beállításokon kívül megadhatjuk, mit szeretnénk célba venni az algoritmus folyamán. Ezt a *Target* legördülő menüből választhatjuk ki. Ez tartalmazza a kiválasztott tábla minden attribútumát.

A *Target* megadása után az alatta lévő listában megjelenik minden más, nem kiválasztott attribútum, amikből meghatározhatjuk, mi legyen a classifier *Feature* halmaza. Ezek alapján végzi el az algoritmus a tanulást, így megadásuk során szempont, hogy csak a releváns attribútumokat adjuk át. Ha ezeket mi nem tudjuk kiszűrni, az *Műveletek* fül alatti *Feature Selection* lehet a segítségünkre.

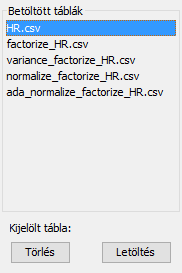


6. ábra: Az AdaBoost Classifier paraméterablakja

Ellentétben az előfeldolgozó műveletekkel, az osztályozó algoritmus automatikusan egy új táblát fog létrehozni, így amennyiben szeretnénk megadni, hogy az mit tartalmazzon a megjósolt Target értékeken kívül, akkor azt a jobboldalt lévő *Kimeneti tábla oszlopai* legördülő menüvel és listával teherjük meg. Célszerű az előfeldolgozások alkalmával mindig új táblát létrehozni, így a kimeneti tábla nem a már faktorizált, számszerű értékeket fogja megjeleníteni, hanem az eredeti tábla szöveges mezőit.

Miután minden paramétert megadtunk, az osztályozót az *Indítás* gombbal tudjuk elindítani. Az algoritmus futása a paraméterektől és a tábla méretétől függően eltérő ideig tarthat. Fontos emellett, hogy minden classifier csak számszerű mezőt tud feldolgozni mint Target és mint Feature. Kivételt képez ezalól a *Sentiment Analysis* opció, ahol csak szöveges mezőt tud elfogadni bemenetnek. Ha ezeket véletlen elfelejtettük, a program figyelmeztet, ha az algoritmus lefutásához nem megfelelő paramétereket adtunk meg. Ilyenkor újonnani, helyes beállítás után már megfelelő eredményt kapunk.

Az eredménytábla ilyen esetekben nem tartalmaz minden rekordot, amit az általunk feltöltött tábla tartalmazott. Ennek az oka a következő: mivel az osztályozó algoritmus lefuttatásához szükség van egy train és egy test halmazra is, az általunk feltöltött táblából a háttérben pár rekordot kiemelünk, mint test halmaz. Ezek az elemek fognak megjelenni az eredménytáblában. A maradék rekordokat pedig az algoritmus felhasználja a tanulási folyamathoz. Mivel maga az osztályozó eredménytáblája nem tartalmaz a felhasználó számára kellő információt az algoritmus sikerességéről, ezért ha a classifier sikeresen lefutott, egy accuracy, azaz pontossági érték is megjelenik az eredménytáblával. Ez az érték egy 0 és 1 közötti szám, ami megadja, hogy a jósolt értékek és a tanításra használt értékek mennyire hasonlítanak, vagyis mennyire pontos a becslés. Ez az érték későbbiekben hasznos lehet, ha a táblára több, más paraméterekkel beállított osztályozót is szeretnénk lefuttatni.

 A táblákon történő műveleteken kívül lehetőségünk van a számunkra megfelelő táblát letölteni, illetve pedig ha nincs rá szükségünk, a táblák listájából törölni. Ezeket a funkciókat a *Betöltött táblák* lista alatti két gombbal érhetjük el.

Ezen a panelen helyezkedik el továbbá még egy *Törlés* gomb, a bal alsó sarokban. Ha erre a gombra kattintunk, lehetőségünk van törölni az éppen betöltött munkafolyamatot. Visszavonás művelet nem elérhető, így a törléshez szükséges megerősítenünk a szándékunkat.

7. ábra Törlés és letöltés

A felhasználónak lehetősége van a felhasználónevének és jelszavának módosítására is. Ezeket a főablakban tudja megtenni, a *Profil szerkesztése* listában található opciók használatával. Továbbá megemlíteném még a sugó menüpontot, ami a felső menüsorban található, és elérhető a fő és a munkaoldalról is egyaránt. Ez a menüpont tartalmazza csoportosítva a műveletek leírását, és az osztályozó algoritmusok működéséhez szükséges paraméterek magyarázatát is, ezzel segítve a felhasználó munkáját.

**3. Fejlesztői dokumentáció**

**3.1 Feladatmegoldás eszközei**

A probléma megoldásához Java JDK 8-at és Python 2.7-et használtam. A későbbi fejlesztéshez szintén ezek az eszközök, és a hozzájuk tartozó fejlesztői környezetek (pl. NetBeans) ajánlottak. A program adatbázisa egy hsqldb nevű adatbázis, ami a szerver .jar fájljába már be van építve. Az adatbázis az interneten elérhető, letölthető.[5]

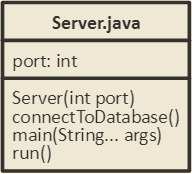
**3.2 A szerver felépítése**

A program szerver része egy console felületre íródott, grafikus felület nélküli kezelő, aminek a fő feladata, hogy a klienstől érkezett kéréseket fogadja, és a kérés minőségének megfelelően feldolgozza. A programszerver elindulásakor kapcsolódik a hsql adatbázisszerverhez, ami segít eltárolni az egyes felhasználók adatait, elősegítve ezzel az azonosításukat és a gyorsabb munkát. Két részből áll: Server.java és ClientThread.java.

* + 1. *Server.java*

Ennek az osztálynak a feladata a szerver alapjainak kiépítése. Ezekhez a feladatokhoz szükségesek a hozzájuk tartozó függvények és adattagok. Ezen adattagok és függvények leírása a következő:

* A *port* (integer) az osztály saját adattagja, amit a konstruktorban inicializál. Ezen a nyitott porton keresztül jön létre a kommunikáció.
* Az adatbázishoz való csatlakozás. Ezt a *connectToDatabase()* függvény egy *DriverManager* segítségével hozza létre. A függvény egy *Connection* objektumot ad vissza, amin keresztül az adatbázis elérhető. Az adatbázis beállításai a kódba vannak égetve.
* A szerver egy *Runnable interface*-t valósít meg, ezáltal elengedhetetlen a *run()* metódus megléte. Ebben a metódusban kerül létrehozásra a *ServerSocket*, illetve itt fogadjuk a csatlakozást kérő klienseket is egy végtelen ciklusban. Fontos megemlíteni, hogy minden kliens egy saját szálon indul el, amit egy *ExecutorService* objektum *submit* függvényével hozunk létre. Ennek köszönhetően egyszerre több felhasználó is használni tudja a szerverünket, összeakadás nélkül.

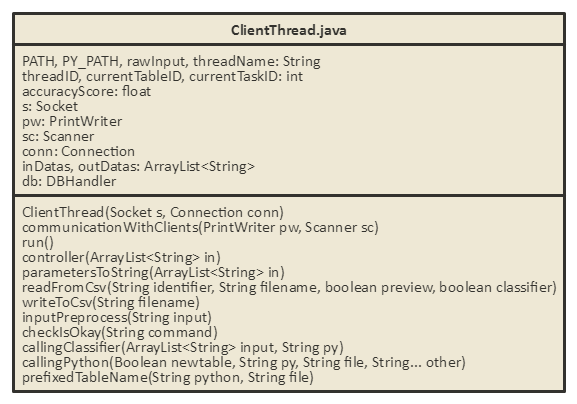


8.ábra: Server.java osztálydiagramja

*3.2.2 ClientThread.java*

Ha a *Server.java* osztály leírásánál említett végtelen ciklusban klienst fogadunk, a kliens ennek az osztálynak egy példánya lesz. Ha a kapcsolat a socket-en keresztül létrejött, a klienseket ezentúl ez az osztály fogja kezelni. A *Server.java*-hoz hasonlóan ez az osztály is egy *Runnable interface*-t terjeszt ki. Ebben az osztályban található a feladatok elosztása, a külső python kódok meghívásáért felelős metódusok, továbbá az adatbázis kezelésére használt *DBHandler* belsőosztály. Adattagjai és függvényei a következők:

* A *PATH* és *PY\_PATH* adattagok a szerveren lévő mappák elérési útvonalát tartalmazzák, ahova a felhasználók által feltöltött .csv fájlok kerülnek, és ahol a python kódok helyezkednek el.
* A *threadID*, *currentTableID*, *currentTaskID* és a *threadName* változók a szálra és a munkamenetre specifikus tulajdonságokat tárolják el. A *threadID* és a *threadName* nem változik, (hacsak a felhasználó nem változtat nevet), viszont a *currentTaskID* és a *currentTableID* az éppen kezelt tábla és munkafolyamat azonosítóját tárolja el. Ez a futás folyamán többször is megváltozhat.
* Az *inDatas* és *outDatas* dinamikus listák a kommunikáció lebonyolítását segítik. Nevükből adódóan a kimenő és bejövő adatokat tárolják el.
* A további adattagok (pl. pw, sc…stb) a socket kapcsolat lebonyolításához szükséges változók, használatuk egyezik a JAVA APIban leírtakhoz, plusz vonzatuk ebben az esetben nincs.
* Az osztály konstruktora, ami paraméterként megkapja a kapcsolat kialakításához szükséges *Socket*-et és a korábban a *Server.java*-ban létrehozott *Connection*-t. A *Socket* alapján létrehozza az alapértelmezett ki- és bejövő stream-et.
* Az interface implementálása miatt muszáj jelen lennie egy *@Override* annotációval ellátott *run()* függvénynek, amivel az osztály működésbe lép. Itt hívjuk meg a *communicationWithClients* nevű függvényt.
* A *communicationWithClients* egy void típusú függvény. Feladata az *inDatas* és *outDatas* listákban lévő adatok küldése illetve fogadása. Ha a kimenő adatoknak fenntartott lista üres, nem történik semmi. Fogadás, kiküldés és feldolgozás után a listákat ürítjük.
* Az osztály legösszetettebb függvénye, az *ArrayList<String>* típust visszaadó *controller* függvény. Itt történik a bejövő adatok feldolgozása, az alapján, hogy milyen azonosítóval érkezik a kérés. A függvény törzsében egy switch-case szerkezetben minden azonosítóhoz a szükséges utasítások kerülnek meghívásra. A visszatérési érték pedig a szerver válasza az adott kérésre. Ezt a választ az *outDatas* változóba tölti, ami ezután már nem lesz üres, így a szerver a socket-en keresztül elküldi annak tartalmát a kliensoldalnak.
* Ahhoz, hogy a bejövő adatokat könnyedén tudjuk kezelni a *controller* függvényben, szükség van egy segédfüggvényre, ami a bejövő nyers adatokat feldolgozza a megfelelő formátumba. Ezt a feladatot az *inputPreprocess* nevű függvény látja el.
* A *readFromCsv* és a *writeToCsv* segédfüggvények kezelik a .csv fájlokat.
* Ezután következnek a python scriptek hívásáért felelős függvények. Ez a három függvény a *callingClassifier*, *callingPython* és a *checkIsOkay*. Az első kettő rendre az osztályozó- és az előfeldolgozó algoritmusok meghívásáért felelősek. Mindkét függvény törzsében a megfelelő python kód meghívásra kerül egy *Process* formájában. A függvény megvárja a script lefutásának végét, és ez alapján ad visszajelzést, hogy sikeres volt, vagy sem. Az osztályozó algoritmusok továbbá visszaadnak egy pontossági értéket is, amit a szerver eltárol, majd az eredménytáblával együtt vissza is küld a kliensoldalnak. Hasonló elven működik az *checkIsOkay* függvény, ami az osztályozóalgoritmusok lefutása előtt kerül meghívásra. Ez egy logikai értéket ad vissza. Ha az érték igaz, lefuthat az osztályozó. Ha nem, hiba lépett fel, ezt jelzi a felhasználónak, aki újra próbálkozhat, más paraméterek megadása után.

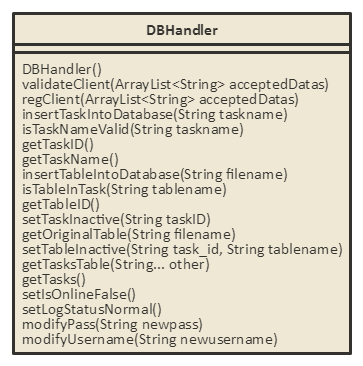


9. ábra: ClientThread.java osztálydiagramja

*3.2.3 DBHandler belsőosztály*

Ez az osztály nem egy különálló .java fájl, hanem a ClientThread.java-ban található belsőosztály. Ez az osztály tartalmazza az adatbáziskezelő függvényeket. Adattagjai nincsenek, a tartalmazó osztály adattagjait használja fel. Fontosabb metódusainak leírása:

* A felhasználók regisztrációjának jóváhagyása, illetve a belépés validálása a *regClient* és a *validateClient* függvény segítségével működik. A *regClient* törzsében az adatbázisba való adatfelvitelen kívül, ha az adatok helyesek, létrejön a felhasználó számára a szerver tárhelyén egy mappa. A mappa neve a felhasználó azonosítója lesz. Ide kerülnek majd később a feltöltött munkái. A *validateClient*ben a validálás után pedig a *ClientThread* leírásánál említett *PATH* változó értéke ez a tárhelymappa lesz. Visszatérési értékük boolean, ezt a kliens felé a szerver visszaküldi. Ezalapján tudhatja meg a felhasználó, hogy a regisztráció vagy a bejelentkezés sikeres volt-e.
* Az *insertTaskIntoDatabase* és az *inserTableIntoDatabase* függvény feladata a létrehozott munkamenet és a feltöltött tábla adatainak felvitele az adatbázis megfelelő tábláiba.
* Fontos, hogy a felhasználó bejelentkezés után visszakapja a már korábban használt munkameneteket, és a hozzájuk tartozó feltöltött táblákat. Erre ad lehetőséget a *getTasks* függvény, ami visszaadja a korábbi munkákat, a *getTasksTable* pedig a kiválasztott munkafolyamatban használt adattáblákat. A lekért adatokat visszaküldi a kliensnek, az pedig megjeleníti őket a felhasználó számára.
* Ha egy tábla vagy munkamenet törlése kerül, akkor magáról az adattárolóról nem törlődik le, csupán a rá való referencia szűnik meg. Ezt a funkciót valósítja meg a *setTaskInactive* és a *setTableInactive*. Minden táblához és taskhoz tartozik egy *isActive* nevű attribútum, ami hamis értékre vált, ha törlésre kerül sor. A törölt objektumok nevei egységesen megváltoznak. Ezeket a táblákat és taskokat a későbbiekben a *getTask* és a *getTasksTable* már nem adja vissza.
* A *getOriginalTable* adja vissza azt a táblát az adatbázisból, ami a már módosított tábla őse. Ez segíti az osztályozóalgoritmusok lefutása után az eredeti attribútumok visszaadását.
* Ha a kliensoldal felöl a felhasználó adatait módosító kérés érkezik, akkor a *modifyPass* és a *modifyUsername* metódusok kerülnek meghívásra attól függően, hogy mit kíván változtatni a felhasználó. Ha a jelszó módosul, csak az adatbázisban kell a rekordokat frissíteni, viszont ha a felhasználónév is megváltozik, akkor a *ClientThread* *threadName* tulajdonságát is megváltoztatjuk.
* A továbbiak segédfüggvények, amik a korábban leírt függvények egy-egy részfeladatáért felelősek. Logikailag nem összetettek, nevük elárulja funkciójukat, így ezeket nem részletezem.



10. ábra: DBHandler osztálydiagram

**3.3 Az adatbázis**

A szerver felé küldött adatokat egy háttérben futó hsql adatbázis tárolja. A Server.java osztály ehhez az adatbázishoz kapcsolódva kéri le és kezeli a szükséges adatokat, mint például a regisztrált felhasználók adatait, a feltöltött táblákat és a táblák kezeléséhez szükséges munkafolyamatokat. Az adatbázis sémája három táblából áll. Ezek a táblák a users, a tables és a tasks nevet viselik.

IDEKERÜL AZ EGYED-KAPCS DIAGRAM, amit meglesz

**3.4 A kliens felépítése**

A program kliensoldala egy háromrétegű architektúra alapján készült, ahol a egymástól elkülönül a logikai réteg (*Model*), a program felhasználó felületét leíró réteg (*View*) és a vezérlő (*Controller*) réteg. Ezt az architektúrát MVC néven ismerhetjük. Mivel a kliensoldal többablakos, ezért minden ablak esetén fontos volt ezeket a rétegeket megfelelően logikai egységekbe elkülöníteni, ezáltal a különböző komponensek, ahol volt rá lehetőség, különböző osztályokba, vagy éppen belsőosztályokba kerültek.

A felület kialakítását a fejlesztői környezet segítségével alakítottam ki, a felület egységei, mint a panelek és egyéb elemek a .java classban kerültek deklarálásra, a .form fájlban pedig a formázásra került sor. Ezek a .form fájlok xml felhasználására épülnek fel.

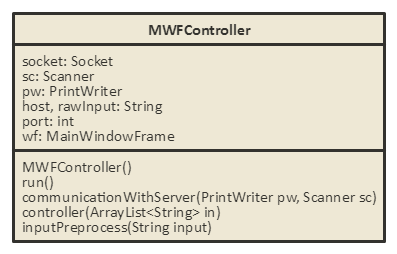
A program főosztálya a Main.java, ami nem tartalmaz mást, csak egy main függvényt, aminek segítségével a program első ablaka és a hozzá tartozó vezérlő létrejön.

*3.4.1 A bejelentkező ablak logikai felépítése*

Ha a felhasználó bejelentkezik, elsőnek a bejelentkező ablakot látja meg. Mivel ez logikailag elkülönül a főprogram ablakától, ezt külön osztályokban kezeljük. Ahogy korábban említettem, a felhasználói felületekre érvényes háromrétegű architektúra ebben az esetben is jelen van. Minden rétegnek egy-egy osztályt feleltethetünk meg.

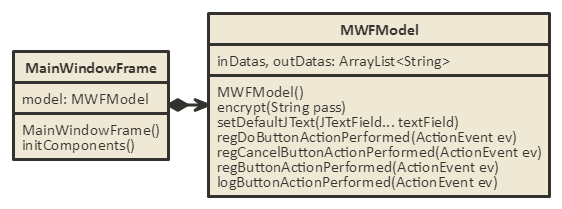
A **vezérlő réteg**, vagyis a *MWFController* osztály tartalmazza a futáshoz szükséges *run()* függvényt, ami a *Runnable* interface megvalósítása miatt szükséges. Hasonlóan a szerveroldali megvalósításhoz, itt is ez a függvény hívja meg a kommunikációért felelős *communicationWithServer* metódust. A vezérlő a socket létrehozatalához szükséges változókat nem tartalmaz, csupán a felület réteget megvalósító *MainWindowFrame* egy példányát, amin keresztül a felület és model adattagjait eléri.

Az osztály *controller* függvénye felelős azért, hogy a model által feltöltött listák tartalmát továbbítsa a szerver irányába, ami azokat a korábban említett módon feldolgozza, majd a választ visszaküldi. A fogadás ugyanitt a függvénytörzsben történik.



11. ábra: MWFController osztálydiagram

A bejelentkezési ablak **felületi** és model **rétegét** a *MainWindowFrame.java* tartalmazza. Ez az osztály tartalmaz egy belső osztályt is, ami a már említett model réteg implementálásáért felelős. A külső osztály lényegében egy darab *initComponents()* függvényt tartalmaz ami a felület komponenseit deklarálja, és a hozzátartozó tulajdonságokat megadja. Ez a függvény egy drag-and-drop funkció által automatikusan generált, levédett kódrészlet. A felület nem tartalmaz semmilyen logikai funkciót, csupán a megjelenítéshez szükséges. Ez az osztály konstruktorában kerül meghívásra.

A bejelentkező ablak felülete két panelre osztható. Első panel, vagyis a *loginPanel* tartalmazza a bejelentkezési opciót, itt található a Bejelentkezés és Regisztráció gomb. A második panel felel a regisztráció lebonyolításáért, ennek neve *regPanel*. Ezekhez hozzáadva további adattagok elérhetőek, ezeket nem részletezem.

12. ábra: MainWindowFrame és MWFModel kapcsolata.

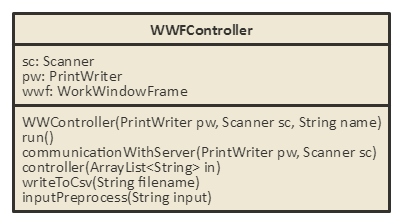
A harmadik réteg, vagyis a **logikai réteg** tartalmazza a felületen elérhető elemekhez tartozó *ActionListener* függvényeket. Az osztály globális változója két *ArrayList*, ami ugyanazt a szerepet látja be, mint a szerver esetében: a bejövő és kimenő adatok eltárolását. A kimenő adatokat az *ActionListener* függvények töltik fel, kiolvasva a megfelelő elemek értékét. A metódusok törzse a feltöltésen kívül ellenőrzi is, hogy az elvártnak megfelelő adat kerüljön csak bele. A függvények feladata részletesen:

* *regButtonActionPerformed* a bejelentkező panelről navigál át a *regPanel*-re, ahol az adatok helyes megadása után a felhasználó regisztrálni tud. A *Mégse* gomb semmissé teszi, visszairányít a korábbi panelre. A *regPanel* *Regisztráció* gombjához társított *regDoButton* *ActionListener* pedig felküldi a szerver irányába az adatokat, majd ha azokat jóváhagyta, bejegyzi a felhasználót.
* Regisztráció után a *loginPanel* lesz újra látható, adatok megadása után a Bejelentkezés gombra kattintva meghívódik a *logButtonActionPerformed* metódus. A megadott adatokat betölti az *outDatas* listába, amit a vezérlő réteg felküld a szervernek. Ugyanitt kapjuk meg a választ a *controller* függvénybe, ami ha helyes, akkor a jelenlegi ablak megszűnik, és a főablak egy példánya jön létre.
* A segédfüggvények közé tartozik az *encrypt* és a *setDefaultJText*. Előbbi a jelszó kódolásáért, míg utóbbi rossz adatok esetén a mezők törléséért felelős.

*3.4.2 A munkaablak logikai felépítése*

Ha a bejelentkezés sikeres, a munkaablak vezérlőjének egy példányán keresztül megjelenik az új felület, amit a WorkWindowFrame osztály ír le. Ez a munkaablak felületi rétege, ami csakúgy, mint az előző esetben, szintén tartalmaz egy beágyazott model osztályt. Ez a felület már jóval összetettebb, mint a bejelentkező ablak, több panel egymásba ágyazva jelenik meg.

A főablak vezérlő rétege nem sokban különbözik a bejelentkező ablakétól, itt is ugyanazok a függvények találhatóak meg, csak a törzsükben különböznek.



13. ábra: Főablak vezérlőosztálya

Egyedüli különbség csak a *writeToCsv* függvény, ami funkcióját tekintve ugyanaz, mint a szerverben megtalálható ugyanilyen nevű függvény. Akkor kerül meghívásra, mikor a felhasználó a tábla letöltése funkcióját szeretné igénybe venni.

IDE KELL MAJD A WWF ÉS A MODEL – KÉSŐBB

**3.5 Funkcionális követelmények**

A program felhasználását illetően két esetet különböztetünk meg: vendég és regisztrált felhasználóként való hozzáférést. Ebben a két esetben elérhető funkciók a következők:

*Vendégként* szeretnénk:

* megtekinteni a regisztrációs felületet
* a regisztrációs felületen keresztül szabadon felhasználói fiókot létrehozni
* regisztráció után bejelentkezni

*Regisztrált* *felhasználóként* szeretnénk:

* sikeres regisztráció után bejelentkezni a programba
* kezdő táblát feltölteni új munkamenetbe
* korábbi munkafolyamatot betölteni, és ezzel dolgozni
* kiválasztott munkafolyamatba további táblákat feltölteni
* kiválasztott munkafolyamatot törölni
* betöltött munkafolyamatban kiválasztott táblákat törölni
* kiválasztott tábla előnézetét megtekinteni, illetve táblát letölteni
* táblákra előfeldolgozó műveleteket futtatni (ezen belül eredményként új táblát létrehozni)
* táblákra osztályozó algoritmusokat futtatni, ezek paramétereit előzetesen megadni
* kijelentkezni

**3.6 User-story**

Az előző fejezetben leírt funkcionális követelményeknek megfelelő user-story, azaz felhasználói történet a következő:

Mint *vendég*, szeretnék regisztrálni az oldalra.

* *Amennyiben* a Regisztráció gombot választottuk, *ha* megadjuk az egyedi felhasználónevet, egyedi e-mail címet és a tetszőleges jelszót, *akkor* az adataink felvitelre kerülnek a szerveren keresztül a *users* táblába.
* *Amennyiben* a Regisztráció gombot választottuk, *ha* a mezők bármelyike üres, vagy nem helyes, *akkor* a program újra kéri a szükséges adatokat.
* *Amennyiben* a Bejelentkezés gombot választottuk, *ha* megadjuk a korábban megadott felhasználónevet és jelszavunkat, *akkor* az adatok helyességének ellenőrzése után a program hozzáférést enged a további felülethez.
* *Amennyiben* a Bejelentkezés gombot választottuk, *ha* a mezők bármelyike üres, vagy nem helyes, *akkor* a program újra kéri a szükséges adatokat.

Mint *felhasználó,* szeretnék táblát feltölteni.

* *Amennyiben* a Kiválasztás gombot választottuk, *ha* megadjuk a helyes élérési útvonalat, *akkor* a tábla útvonala a megfelelő mezőbe kerül.
* *Amennyiben* a Feltöltés gombot választottuk, *ha* helytelen kiterjesztésű fájlt, vagy üres útvonalat adunk meg, *akkor* a program hibaüzenetet ad, újra kéri az útvonalat.
* *Amennyiben* a Feltöltés gombot választottuk, *ha* megadjuk a helyes elérési útvonalat, *akkor* a tábla feltöltésre kerül az újonnan létrehozott munkafolyamatba.
* *Amennyiben* a Feltöltés gombot választottuk, *ha* az elérési útvonal és kiterjesztés helyes, de a munkafolyamat neve helytelen, vagy üres, *akkor* a program hibaüzenetet ad, a tábla nem kerül feltöltésre, a munkafolyamat nem jön létre.
* *Amennyiben* a munkafolyamaton belüli Feltöltés gombot választottuk, *ha* az elérési útvonal és a kiterjesztés helyes, de a tábla neve már szerepel betöltve, *akkor* a program hibaüzenetet ad, a tábla nem kerül feltöltésre.

Mint *felhasználó* szeretnénk munkafolyamatot folytatni.

* *Amennyiben* a korábbi munkafolyamatokra kattintunk, *ha* létezik korábbi munkafolyamat, *akkor* a folyamat és a hozzátartozó táblák betöltődnek.

Mint *felhasználó* szeretnénk táblát előfeldolgozni.

* *Amennyiben* az Indítás gombra kattintunk, *ha* van kiválasztott és betöltött tábla, és kiválasztott előfeldolgozó művelet, *akkor* a kliens felküldi az adatokat, amikkel a szerver végrehajtja a kiválasztott műveletet, az eredménytáblát visszatölti.
* *Amennyiben* az Indítás gombra kattintunk, *ha* az Új tábla létrehozása be van jelölve, és minden adat helyes, *akkor* a szerver új táblában küldi vissza a művelet eredményét.
* *Amennyiben* az Indítás gombra kattintunk, *ha* nincs kiválasztott művelet, *akkor* nem történik semmi.

Mint *felhasználó* szeretnénk osztályozóalgoritmus végrehajtani.

* *Amennyiben* az Indítás gombra kattintunk, *ha* van kiválasztott és betöltött tábla, és kiválasztott osztályozó művelet, *akkor* a kliens felküldi az adatokat, a szerver végrehajtás után az eredménytáblát visszaküldi.
* *Amennyiben* az Indítás gombra kattintunk, *ha* nincs kiválasztott algoritmus, *akkor* nem történik semmi.
* *Amennyiben* az Indítás gombra kattintunk, *ha* van kiválasztott tábla és algoritmus, de paraméterek hibásak, vagy üresek, *akkor* hibaüzenet érkezik, újra próbálkozás lehetséges.

Mint *felhasználó* szeretnénk táblát letölteni.

* *Amennyiben* a Letöltés gombra kattintunk, *ha* van kiválasztott tábla, amit sikerült betölteni, *akkor* a tábla letöltése megkezdődik, amint végzett a célmappában megjelenik.
* *Amennyiben* a Letöltés gombra kattintunk, *ha* van kiválasztott tábla, amit nem sikerült betölteni, *akkor* a tábla letöltése nem lesz sikeres.

Mint *felhasználó* szeretnénk táblát törölni.

* *Amennyiben* a Törlés gombra kattintunk, *ha* van kiválasztott tábla, *akkor* a kérést elküldi a szerver felé, ott inactive-ra állítódik a tábla állapota, többet nem kerül betöltésre.
* *Amennyiben* a Törlés gombra kattintunk, *ha* nincs kiválasztott tábla, *akkor* nem történik semmi.
* *Amennyiben* a Törlés gombra kattintunk, *ha* megerősítjük a törlési szándékot, *akkor* a kérés a szerver felé elindul, a tábla inactive lesz, többet nem kerül betöltésre.
* *Amennyiben* a Törlés gombra kattintunk, *ha* nem történik megerősítés, a tábla továbbra is betölthető marad.

Mint *felhasználó* szeretnénk munkafolyamatot törölni.

* *Amennyiben* a Törlés gombra kattintunk, *ha* megerősítjük a törlési szándékot, *akkor* a kérés a szerver felé elindul, a task inactive lesz, többet nem kerül betöltésre.
* *Amennyiben* a Törlés gombra kattintunk, *ha* nem történik megerősítés, *akkor* a task továbbra is betölthető marad.

Mint *felhasználó* szeretnénk jelszót változtatni.

* *Amennyiben* a Jelszó módosítása pontra kattintunk, *ha* helyes jelszót adunk meg, ami több, mint 3 karakter, *akkor* a kérés megindul a szerver felé, ha megváltozott a jelszó, a felület jelzi.
* *Amennyiben* a Jelszó módosítása pontra kattintunk, *ha* a jelszó beviteli mezője nem tartalmaz semmit, illetve az új jelszó rövidebb, mint 3 karakter, *akkor* a kérés nem indul meg, nem történik változás, hibaüzenet jelenik meg.
* *Amennyiben* a Jelszó módosítása pontra kattintunk, *ha* a kétszer egymásután bekért jelszó nem egyezik, *akkor* a kérés nem indul meg, nem történik változás, hibaüzenet jelenik meg.

Mint *felhasználó* szeretnénk felhasználónevet változtatni.

* *Amennyiben* a Felhasználónév módosítása pontra kattintunk, *ha* a felhasználónév nem üres, illetve érvényes, *akkor* a kérés megindul a szerver felé, megváltozik a név.
* *Amennyiben* a Felhasználónév módosítása pontra kattintunk, *ha* a felhasználónév korábban már foglalt, *akkor* a hibaüzenet jelenik meg.

|  |  |  |
| --- | --- | --- |
| AS A | | Vendég |
| I WANT TO | |  |
| 1. | GIVEN |  |
|  | WHEN |  |
|  | THEN |  |
| 2. | GIVEN |  |
|  | WHEN |  |
|  | THEN |  |

**3.y Nemfunkcionális követelmények**

A program megfelelő működéséhez szükséges nemfunkcionális követelmények az alábbiak:

* Legyen felhasználóbarát, vagyis a felhasználó legyen biztos egyes funkciók jelentésében.
* Biztonság, vagyis a felhasználó által megadott adatok (például jelszó), ne lehessenek visszafejthetők.
* Letisztult és tiszta felület
* Hibatűrés, vagyis egy kisebb hiba esetén ne omoljon össze a program alapja.

**Hivatkozások**

[1]C. Snijders, U. Matzat, U-D. Reips; *’Big Data’: Big gaps of knowledge in the field of Internet*, International Journal of Internet Science, 2012

ISSN 1662-5544

[2] [*https://techterms.com/definition/digital\_footprint*](https://techterms.com/definition/digital_footprint) *-* 2017.11.05.

[3] V. Mayer-Schönberger, K. Cukier; *Big Data: a revolution that will transform how we live, work and think*, Houghton Mifflin Harcourt, 2013

ISBN 0522002695

[4] [*http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html*](http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html) *-* 2017.11.24

[5] [*https://sourceforge.net/projects/hsqldb/files/*](https://sourceforge.net/projects/hsqldb/files/) *-* 2017.11.24

[5] sentiment training set