Kivonat

Az internetes böngészés rohamos terjedése és az egyre tágabb körű online ügyintézés ösztönöz, hogy felhasználói fiókokat hozzunk létre. Ezek által adataink biztonságosan eltárolódnak és több funkcióhoz is jutunk egy adott oldalon. Ahhoz, hogy ezekhez az adatokhoz más ne férhessen hozzá, különféle azonosítási módszereket dolgoztak ki a felhasználók védelme érdekében. Ennek ellenére, ha egy illetéktelen személy belépést nyer saját online fiókunkba, akkor nem csak felhasználói adatainkat, de akár anyagi javainkat is veszélyeztetheti.

Diplomamunkám célja egy olyan másodlagos hitelesítő rendszer létrehozása webes alkalmazások számára, amely a felhasználónév-jelszó párossal való beléptetés biztonságát növeli. Ezt úgy valósítsa meg, hogy összehasonlítsa a felhasználó által beírt karakterek leütési mintáját a már korábban bevitt mintákkal. Az alkalmazást egy PHP keretrendszerben, Laravelben készítettük el és az adatok tárolására a Firebase valós idejű adatbázist használtuk.

Az eredmény egy gyűjtő felület, ahol a kísérletben résztvevők néhány szöveg begépelésével egy billentyűzési adathalmazt szolgáltatnak. Ez által saját adathalmazra teszünk szert és kísérleteket hajthatunk rajtuk végre, amelyek hasznos eredményeket szolgáltathatnak a jövőbeli kutatásokra nézve. Emellett egy azonosító felületet is implementáltunk, ahol le tudjuk tesztelni éles helyzetekben is az alkalmazást.

***Kulcsszavak:*** billentyűzés alapú felismerés, másodlagos hitelesítés, Laravel, webes biztoság

Tartalomjegyzék

[1. Bevezető 2](#_Toc517991002)

[2. A projekt célja 3](#_Toc517991003)

[3. Bibliográfiai tanulmány 4](#_Toc517991004)

[4. Elméleti megalapozás 5](#_Toc517991005)

[4.1 A biometrián alapuló azonosítás áttekintése 5](#_Toc517991006)

[4.2 Billentyűzési biometria webes alkalmazásban 6](#_Toc517991007)

[4.2.1 Bemeneti mezők megválasztása 6](#_Toc517991008)

[4.2.2 Jellemzők kiemelése 7](#_Toc517991009)

[4.2.3 Az egyenletes hibaarány 8](#_Toc517991010)

[4.2.4 Felhasznált algoritmusok 9](#_Toc517991011)

[4.2.4.1 Manhattan scaled 9](#_Toc517991012)

[4.2.5 Kimenetek fúziója 11](#_Toc517991013)

[4.3 Felhasznált technológiák 11](#_Toc517991014)

[4.3.1 Adattárolás Firebaseben 11](#_Toc517991015)

[5. A rendszer specifikációi 12](#_Toc517991016)

[5.1 Követelmény specifikáció 12](#_Toc517991017)

[5.1.1 Kivonat 12](#_Toc517991018)

[5.1.2 Bevezető 12](#_Toc517991019)

[5.1.2.1 A dokumentum célja 12](#_Toc517991020)

[5.1.2.2 A dokumentum olvasói 12](#_Toc517991021)

[5.1.2.3 Rövid áttekintés 13](#_Toc517991022)

[5.1.3 Áttekintés 13](#_Toc517991023)

[5.1.3.1 Funkcionalitás 13](#_Toc517991024)

[5.1.3.2 Felhasználói osztályok és karakterisztikái 14](#_Toc517991025)

[5.1.3.3 Működési környezet 14](#_Toc517991026)

[5.1.3.4 Felhasználói segédletek 14](#_Toc517991027)

[5.1.3.5 Megszorítások és függőségek 14](#_Toc517991028)

[5.1.4 Rendszer követelmények 15](#_Toc517991029)

[5.1.5 Funkcionális követelmények 15](#_Toc517991030)

[5.1.6 Nem funkcionális követelmények 19](#_Toc517991031)

[5.1.6.1 Skálázhatóság és továbbfejleszthetőség 19](#_Toc517991032)

[5.1.6.2 Karbantarthatóság 20](#_Toc517991033)

[5.1.6.3 Nyelvi követelmények 20](#_Toc517991034)

[5.1.7 Interfész követelmények 20](#_Toc517991035)

[5.1.7.1 Felhasználói interfész 20](#_Toc517991036)

[6. Részletes tervezés 20](#_Toc517991037)

[6.1 Adatok tárolása Firebaseben 20](#_Toc517991038)

[6.2 Architektúra 21](#_Toc517991039)

[6.3 A rendszer tervezése és bemutatása 22](#_Toc517991040)

[6.3.1 A kontroller rész 22](#_Toc517991041)

[6.3.2 A modell rész 22](#_Toc517991042)

[6.3.3 A nézet (view) rész 22](#_Toc517991043)

[6.3.4 Segítő osztályok 22](#_Toc517991044)

[6.4 Ajax hívások bemutatása 22](#_Toc517991045)

[6.5 Adatok mozgása 23](#_Toc517991046)

[7. Gyűjtő üzembe helyezése és kísérleti eredmények 23](#_Toc517991047)

[7.1 Gyűjtő üzembe helyezése 23](#_Toc517991048)

[7.2 Felmerült problémák és megoldásaik 23](#_Toc517991049)

[7.3 A begyűjtött adathalmaz 23](#_Toc517991050)

[7.3.1 Felhasználóktól gyűjtött adathalmaz 23](#_Toc517991051)

[7.3.2 Imposztoroktól gyűjtött adathalmaz 24](#_Toc517991052)

[7.4 Kísérleti eredmények 24](#_Toc517991053)

[8. Következtetések 25](#_Toc517991054)

[8.1 Megvalósítások 25](#_Toc517991055)

[8.2 Összehasonlítás hasonló rendszerekkel 26](#_Toc517991056)

[8.3 További fejlesztési irányok 26](#_Toc517991057)

[9. Irodalomjegyzék 26](#_Toc517991058)

[10. Függelékek 27](#_Toc517991059)

# Bevezető

Az internet rohamos fejlődése által egyre több olyan tevékenységet vagyunk képesek lebonyolítani otthon, ami személyes jelenlétet igényelt volna évekkel ezelőtt. Manapság a legújabb okos telefonok megvásárlásától a banki átutalásokig bármit képesek vagyunk pár mozdulattal lebonyolítani és ezzel arányosan hatalmasra nőtt a felhasználói fiókjainknak az értéke is. Bárki, aki az adatainknak, vagy éppen azonosító tárgyainknak (pl. bankkártya, beléptető tóken) a tulajdonában van, képes arra, hogy hozzáférjen a személyes adatainkhoz, vagy akár az anyagi értékeinkhez is.

Tegyük fel, hogy nem biztonságosan kezeltük egy online banki alkalmazásnál használt fiókunkat és a belépési adatainkat megszerezte egy harmadik fél, akinek szándéka ezt a lehetőséget kihasználni. Ilyen esetben segít egy másodlagos hitelesítő rendszer, ami képes a detektálására az esetleges betolakodóknak. A rendszer észleli azt, hogy a belépett személynek egyes tulajdonságai nem egyeznek meg a regisztrált felhasználóéval, ezért jelzést küld, és további megerősítést kér. Amennyiben a megerősítés nem megy végbe, a rendszer elkönyvelheti, hogy feltörés ment végbe és segítséget tud nyújtani a felhasználónak a további biztonsági lépések megtételéhez.

Egy ilyen rendszernél például ajánlatos viselkedési biometrián alapuló eszközökkel ellenőrizni, hogy valódi-e a felhasználó. Ezek az eszközök képesek arra, hogy a felhasználót a vele született, egyedi viselkedés mintái és mikro-mozzanatai alapján összehasonlítsák a korábbi belépéseihez és megállapítsák, hogy mekkora hasonlóság mérhető nála.

Általában egy ilyen rendszernél nem csak egyfajta viselkedést ellenőriznek, ugyanis a viselkedési biometrián alapuló hitelesítők nem képesek száz százalékos eredményt garantálni és megeshet, hogy a utánzó felhasználót beengedik, vagy éppen a valós felhasználót kizárják. Ennek elkerülésére a rendszer több viselkedést is kell, hogy figyeljen és az egyes tulajdonságok (billentyűzés, egérmozgás, szóhasználat, stb.) mind visszaadnak egy szavazatot, amit összegezve megkapjuk, hogy mi a teendő az adott felhasználóval.

A fiókunk biztonsága érdekében tehát érdemes minél jobban megerősíteni a védelmet és erre egy módszer a billentyűzési ritmus figyelése. A felhasználónak tudnia nem kell róla, mert a háttérben működik. Emellett már egyre jobb és jobb eredményeket érnek el ilyen téren, tehát valószínű, hogy hamarosan egyre több oldalon lesz lehetőség ez a fajta másodlagos azonosítás is.

# A projekt célja

Az első fejezetben említett billentyűzés alapú azonosítási modul megvalósításában nyilvánul meg a célom. Ez magába foglalja egy gyűjtő rendszer létrehozását, mely segítségével a további kísérletekhez tudunk tanító és tesztelő adathalmazokat biztosítani, valamint egy azonosító rendszer létrehozását, ami segítségével a felhasználót egy második biztonsági réteggel is meg tudjuk védeni.

Jelenleg számos kísérlet folyik jobb és jobb eredményekért a billentyűzési ritmus alapú felismeréssel kapcsolatosan. Már a telegráf korában felfigyeltek az emberek a tényre, hogy be tudják egymást azonosítani a beütési ritmusuk alapján. Ez többnyire annak is köszönhet, hogy hogy az emberi agynak ugyanazok a neuro-fiziológiai részei kerülnek használatba gépeléskor, mint amik az aláírások esetében is. Bár egyre több kísérlet folyik ilyen irányban, teljes pontosságú megoldást még nem találtak és viszonylag kevés rendszernél alkalmazzák a módszert.

Az alkalmazás célközönsége az érzékeny adatokat kezelő alkalmazások köre, mivel ilyen esetekben kifejezetten fontos az, hogy az belépett felhasználó az legyen, aki a tulajdonosa a fióknak.

­Megvalósítás céljából a PHP egyik keretrendszerét, a Laravelt használtam. A választás azért esett a Laravel-re, mert számos ponton megkönnyíti a rendszer implementációját és amennyiben más PHP alapú rendszerbe szeretnénk a modult integrálni, kevés átalakítással képesek vagyunk rá. Az alkalmazás egyaránt képes gyűjteni adatokat a valós felhasználótól, valamint külön oldal van létre hozva arra is, hogy pár, a kísérletbe beavatott személy próbáljon belépni a tesztelt felhasználó adataival. Ezeket a belépési kísérleteket szintén elmentjük. A teljes adatbázis JSON formátumban van eltárolva a Firebase valós idejű adatbázisban, ezért a kísérletekhez, amiket Matlabban folytattunk, létrehoztunk egy átalakítót is, ami a JSON formátumban megadott adatainkat csv formátumba konvertálja.

Egy másik célja az államvizsga munkámnak, hogy minél hatékonyabb algoritmust, illetve jellemző kiemelő módszert találjak kísérletek által. Emellett az alkalmazást működővé tenni mobil eszközökön illetve táblagépeken is.

# Bibliográfiai tanulmány

A gyűjteni kívánt adatok milyenségének megfogalmazásában az [1] cikk fontos szerepet játszott. Ez bemutat egy adatgyűjtési és kísérleti fázist. Az érdekessége abban nyilvánul meg, hogy az adathalmazból a jelszavat teljesen kihagyják, és ehelyett inkább a felhasználók személyes adatait, mint név, személyi szám, nemzetiség, e-mail cím, kell, hogy begépelje. Azt feltételezték, hogy a személyes információ begépelésével jobb eredményt érnek el. Szintén itt van bemutatva egy módosított változata a Manhattan Scaled algoritmusnak, melynél a felhasználók 25% esetében az EER 0%-ra esett, és a többi mérésnél is jó eredményt értek el vele.

A Killourhy által végzett munkásságot leíró cikk [2] szintén algoritmusok összehasonlítására fókuszált amellett, hogy bemutassa saját adatgyűjtését. Az adathalmaz 51 felhasználó billentyűzési adataiból lett létrehozva, 400 mintát tartalmazva minden felhasználó esetében, valamint mivel a beírt jelszó ugyanaz, ezért külön imposztor adathalmaz is van minden felhasználó számára. A kísérleti eredmények 9.6% és 10.2%-os EER-t mutattak legjobb esetben, fúzió használata nélkül.

[3] cikkben egy összefoglaló tanulmányt olvashatunk, melyben le van írva a 2013-ig publikált eredmények nagy része. Megismerhetjük általa a di-gráf valamint az n-gráf fogalmát, és a többi tulajdonság kiemelési módszert. Emellett bemutatja, hogy az akkor elért eredmények között is már volt akár 1.401% mértékű egyenlőségi hibaarány, kevert megoldásokkal, valamint 0% közeli is többrétegű perceptronokkal és nagy mennyiségű adatokkal.

A *„Handbook of multibiometrics”* [5] könyvben rálátást nyerünk a biometriai eszközök közti fúzióra, mely segítségével akár több, különböző módon osztályozott mezőt képesek vagyunk egyként azonosítani, és így jobb eredményt elérni, mintha külön vettük volna őket.

[6] tézisben a küszöbérték meghatározásának a módszerét ismertetik, egyosztályos osztályozók esetében. Ez a mi esetünkben is hasznos, mivel valós rendszer esetében nem rendelkezünk imposztor mintákkal.

A Manhattan Scaled távolság alapú algoritmusról a [7] cikkben részletes bemutatást kaphatunk.

[8] cikk által más megközelítésből láthatjuk a Manhattan Scaled és Mahalanobi algoritmusokat, valamint a tulajdonságok közti korrelációnak a hatásaival is szembesítődünk.

# Elméleti megalapozás

## A biometrián alapuló azonosítás áttekintése

Az éppen bejelentkező felhasználóról eldönteni, hogy tényleg ő birtokolja-e az adott fiókot, több módon is lehetséges. Az azonosítási módszereket az alábbi három kategóriába soroljuk:

1. Tudás alapú azonosítás - Jelszó, minta illetve kérdés alapú ellenőrzések tartoznak ide. Legnagyobb előnyük hogy könnyű őket implementálni, míg hátrányuk, hogy el lehet őket felejteni, vagy éppen fel is törhetik pár algoritmus segítségével.
2. Token alapú azonosítás - Általában egy tárgy segítségével azonosítsák a felhasználót, ami csak az ő birtokában lehet. Ezt feltörni nehezebb és általában az elkészítése olcsó, de könnyű elveszíteni. Ide tartoznak a bankkártyák és a beléptető csipogók (például a Sapientia marosvásárhelyi székhelyének a bentlakásánál).
3. Biometrián alapuló azonosítás – A felhasználónak egy adott viselkedését vagy vele született jellemzőjét vizsgáljuk az azonosítás során. Két fajtája lehet:
   1. fiziológiai, amelyet nehéz hamisítani és magas pontossággal rendelkezik, mind például az ujjlenyomat vizsgálása vagy a retina ellenőrzése. Ezek a rendszerek viszont költségesek és mivel a felhasználó vele született fizikai tulajdonságát nézi, amennyiben egyszer sikerül megszereznie egy harmadik félnek a mintát, a felhasználó többé nem használhatja az adott módszert.
   2. viselkedési, melynél azt nézzük, hogy a felhasználó „hogyan” csinál valamit, nem pedig azt hogy mit csinál. Ilyen azonosításokat végezhetünk az egérmozgás, járás, hang, billentyű leütési ritmus és még sok egyéb egyedi mintát alkotó viselkedés elemzésével. Ezek a rendszerek általában nem biztosítanak teljes pontosságot, mivel megeshet, hogy kizárják a valódi felhasználót, vagy éppen beengedik az utánzót.

Bár a viselkedési biometrián alapuló rendszerek képesek arra, hogy egymagukban, viszonylag magas pontossággal azonosítsák a felhasználót, egy ilyen rendszer önmagában nem hatékony. Ezért is az alkalmazásnál ez a fajta hitelesítés nem veszi át a felhasználónév-jelszó páros szerepét, hanem az azonosított felhasználókat ellenőrzi le, hogy tényleg ők-e azok, akik eddig is folyamatosan bejelentkeztek.

Egy másik fontos része a billentyűzés alapú hitelesítésnek, mint bármely más viselkedésen alapuló biometriai hitelesítőnek, hogy egymagában mivel téves adatot ad, több ilyen ellenőrző modullal együtt képesek valós képet alkotni a felhasználó kilétéről. Például a billentyűzés mellett még lehet nézni az egér mozgását is. Ezért ajánlatos az alkalmazásban megírt azonosító modult egy szavazó komponensként alkalmazni az ellenőrzés során.

## Billentyűzési biometria webes alkalmazásban

Az dolgozatmunkám egy webes alkalmazásra épül, mivel sokkal könnyebben meg lehet valósítani az adatgyűjtést, ha egy szerveren rajta van az oldal és egyidejűleg korlátlan számú felhasználó írhatja be az adatokat. Ennek ellenére a felhasználónak lehetősége van arra, hogy ha nincs felügyelve, akkor egy rövid időre megszakítsa a gépelést és ez elrontsa az adott mintát. Emiatt párhuzamosan gyűjtöttem felügyelet nélküli beviteleket és általam felügyelteket is.

Az alkalmazás gyűjtésénél a JavaScriptben megírt JQuery könyvtár függvényeit használtam, melyek által le lehet kérni az adott billentyűnek a lenyomási és felengedési idejében kapott időbélyeget. Ezeket egy rejtett mezőben tárolva tovább lehet küldeni az űrlap beküldése során és a szerver feldolgozza, majd tárolja őket.

Egy másik előnye a webes alkalmazásoknak ilyen téren, hogy kevés módosítással képesek vagyunk arra, hogy a számítógép/laptop billentyűzetei mellett a rendszert működésre bírjuk okos telefonokon és táblagépeken is.

### Bemeneti mezők megválasztása

Tekintsünk most el pár pillanatra a programozás résztől és nézzünk rá a problémára úgy, hogy közben próbáljuk az emberi viselkedésnek a jellemzőit felhasználni.

Már számos adathalmazt hoztak létre, ezek közül hármat próbáltuk ötvözi, hogy a lehetséges legjobb jobb eredmény elérése mellett vizsgáljuk meg azt is, mekkora különbség jelentkezik a felhasználó által megadott és a felhasználót a születésétől kísérő adatok beírásának hasznosságában billentyűzés ritmus ellenőrzésnél. A három billentyűzés dinamikai adatbázis a következő:

* ATVS-Keystroke database [1] – 63 felhasználónak 12 valós és 12 imposztor általi belépése van tárolva. Minden belépés során a felhasználók a személyes adataikat adták meg, amik a vezetéknév, keresztnév, e-mail cím, nemzetiség és személyi igazolvány szám. Két szesszióban folyt le a valós felhasználótól az adatgyűjtés, szessziónként 6 belépéssel. Az adathalmaz előnye, hogy a felhasználó egy olyan szöveget kell, hogy beírjon, amit már jól ismer és ezért összefüggőbbek lesznek a mintái.
* CMU database [2] – 51 felhasználó 8 szesszióban adta le az adatait és két szesszió között legalább egy nap telt el. Ennél az adathalmaznál a felhasználók csak egy jelszavat gépeltek be, ami mindenkinek előre meghatározott volt és tartalmaz kis- és nagybetűt, speciális karaktert valamint számot. Az adathalmaz előnye, hogy nem volt szűkség külön imposztorokra, mert mindenki ugyanazt a jelszavat ütötte be.
* GREYC database [9] – A kiadott cikk alapján 83 felhasználótól gyűjtöttek adatokat. Ezen adatoknak a mennyiségük nem egyezett teljesen, mivel személyes felügyelet nélkül történt a gyűjtés, és a résztvevők jóakaratától függött a bevitt adatok mennyisége. Összesen 5439 mintát gyűjtöttek. Ez az adathalmaz erősen hasonlít az általunk gyűjtöttre, mivel mindkettő esetében van felügyelet mentes gyűjtés, ismeretlen billentyűzettel rendelkező eszközökről.

Ismerve a fent említett két adathalmazt, olyan bemeneti mezőket választottam, amik ötvözik ezeknek a tulajdonságait és általuk olyan kísérleteket hajthatunk végre, amiket külön a két adathalmazon nem tehettünk volna. Ehhez az első mezőnek a *felhasználónevet*, a másodiknak az oldalon ügyködő személynek a *valódi nevét*, míg a harmadik mezőnek egy *training password* nevű adatot kellett tartalmaznia. Ez utóbbi kötelező módon kell, hogy tartalmazzon kis- és nagybetűt, számot és speciális karaktert.

A gyűjtés során az első tíz minta kitörlésre kerül, amennyiben a beütött billentyű leütési-felengedési minták közül nincs legalább hat olyan, amely egyenlő hosszúságú. Ha ez az érték hat és tíz között van, akkor a nem ide tartozó minták törlésre kerülnek. Tíz minta után a felhasználó figyelmeztetve lesz, és nem tárolódik el a bevitele, ha nem egyezik a hosszúsága az eddigi leütési adatokéval. Ez által az adatbázisban eltárolt minták alkalmasabbak lesznek a tanításra.

A felmerült kérdések a következők voltak: Ha a legszemélyesebb a felhasználónév akkor az is lesz a leghatékonyabb azonosításnál? Amennyiben igen, a jelszó, amit nem látunk milyen mértékben lesz kevésbé hasznos az azonosításnál, mint a másik két mező. Ezeknek a megválaszolását a 8.3-as szekcióban lehet megtalálni a többi kísérleti eredmény mellett.

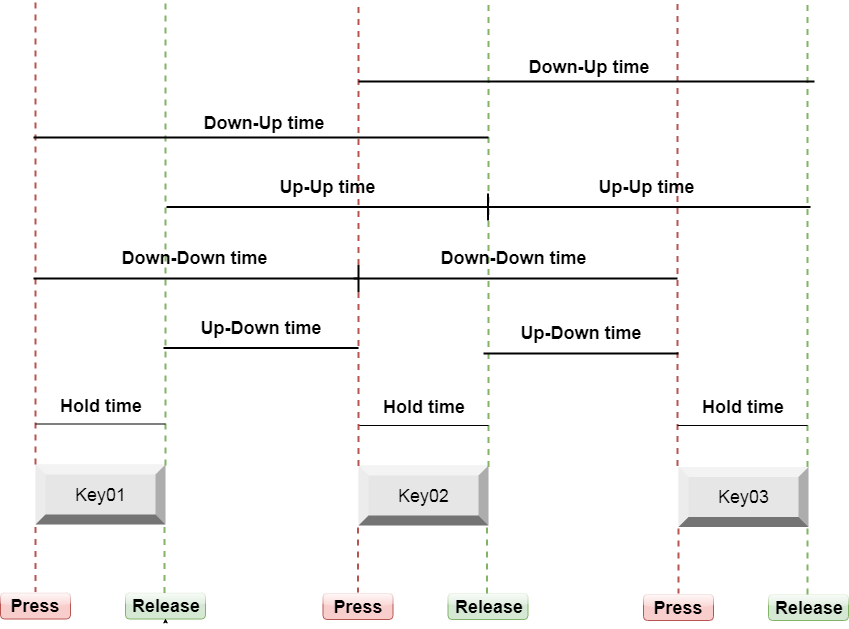
### Jellemzők kiemelése

Az adatok bevitele után következik a rendszer tanításának első része, ami a jellemzők kiemeléséből áll.

Amikor az adatok megérkeznek a szerverre, még nyers formában vannak, és úgy lehet őket kezelni, mint egy lista időbélyeg párosokból. Ezeket az adatokat nem lehetne felhasználni azonosításra, mert az időbélyegek folyamatosan változnak. Ezért a beérkező billentyűk leütési és felengedési időpontjait az őket követő billentyű leütési időiből kivonjuk. Ez által egy úgynevezett di-gráfot hozunk létre, mert kettőnként csoportosítva nyerjük ki az adatokat, amiket felhasználunk a minta elkészítésére, ami által később a felhasználót tudjuk azonosítani.

A tulajdonságok, amiket jelenleg kinyerünk a következők:

* Két egymás utáni leütés közti idő (DD = Down-Down)
* Két egymás utáni felengedés közti idő (UU = Up-Up)
* Az egyik billentyű leütése és a rákövetkező felengedése közti idő (DU = Down-Up)
* Az egyik billentyű felengedése és a rákövetkező leütése közti idő (UD = Up-Down)
* Az egyik billentyű lenyomása és felengedése között eltelt idő (H = Hold)



1. ábra - A kinyerhető tulajdonságok di-gráf esetében

Ezeket a tulajdonságokat még kibővíthetjük, hogy hármanként is nézze az időzítéseket, viszont ez az információ ugyanaz, mintha összegeznénk két di-gráf elemet, ezért nem is emeltem ki több jellemzőt.

### Az egyenletes hibaarány

A fent említett finomított adatokat átadhatjuk egy algoritmusnak, ami a szívét fogja képezni a rendszerünknek. Egy jó algoritmus nagyságrendekkel növelheti az elért eredményt, ami alatt minél több elfogadott valódi felhasználót és elutasított imposztort értünk. Az algoritmusok jóságát fontos hogy tudjuk összemérni, amire az úgynevezett Egyenletes Hibaarányt (EER = Equal Error Rate) használjuk [10] [11].

Minden kiértékelt bejelentkezési próbálkozás kap egy pontszámot, amiről a vágáspont alapján eldönti, hogy valós felhasználónak könyveli-e el az aktuálisat. Az EER-t két tényező határozza meg: a hibásan elfogadott imposztorok aránya és a hibásan elutasított valós felhasználóké. Ahol ez a kettő megegyezik ott található az EER is. Ezt legegyszerűbben úgy tudjuk vizualizálni, ha az adott belépési kísérleteket ábrázoljuk az általuk szerzett pontszám függvényében. Ezt elvégezve a valós és hamis felhasználókra ugyanazon az ábrán két görbét kapunk, melyeknek a metszéspontja adja az Egyenletes Hibaarányt.

Egy másik módszer erre a tévesen elfogadott felhasználók ábrázolása a tévesen elutasítottak függvényében. Ahol a két hibaarány megegyezik, ott található az EER.

|  |  |
| --- | --- |
| D:\Államvizsga\Meglévő Elemek\Keystroke-plots\keystrokePHP\FIGS\user_2_manscaledFAR_FRR.png  . ábra -   A két görbe metszéspontjánál van az EER, ha ott helyezzük el  a metszéspontot ugyanannyit fog hibázni a valódi és utánzó felhasználóknál is. | D:\Államvizsga\Meglévő Elemek\Keystroke-plots\keystrokePHP\FIGS\user_2_manscaled.png  3. ábra -  Az egyenlőségi hibaarány az első szögfelezőn,  a False Positive (FAR) és False Negative (FRR) arányok alapján felírva. |

Az EER értéke minél kisebb, az algoritmus annál jobban működik. Napjainkban már vannak algoritmusok, amik közel 0% EER-t érnek el, viszont nagyon sok tanító mintát igényelnek és ezért valós rendszereknél nem működnének.

Mikor küszöbértéket, vagy más néven vágáspontot választunk, egy olyan függőleges egyenest helyezünk el a grafikonon, ami vagy az EER ponton helyezkedik el, vagy abban az irányban mozog, amerre több hamis felhasználót ki tud zárni, ahogy az az ábrán (ábra) is látszik. Minden következő belépési kísérlet pontszáma akkor lesz elfogadva, ha meghaladja ezt az értéket. Ez lehet minden felhasználóra egy globális érték, vagy felhasználónkét váltakozó.

Valós rendszereknél nem áll rendelkezésre az imposztor adathalmaz, ezért a vágáspontot más módon kell, hogy beállítjuk [ref]. Erre az egyik megoldás az, hogy kizárjuk a felhasználó mintáinak leggyengébb 10 százalékát. Ezzel általában elég magasra tudjuk helyezni a vágáspontot ahhoz, hogy kizárjuk a lehetséges imposztorok nagy részét. (ábra)

### Felhasznált algoritmusok

A következőben tárgyalt algoritmusokat először Matlab alatt teszteltük le a CMU [2] adathalmazon. Tesztelés után integráltuk az algoritmusokat a webes alkalmazásunkba is.

#### Manhattan scaled

Az első algoritmus, amit kipróbáltunk, mivel egyszerűsége ellenére az egy nagyon jó eredményt ad az eddigi felmérések szerint [3][ref]. Az említett adathalmazon tesztelve 0.09% EER-t kaptunk, amire a vágáspont 0.047. Az algoritmus a tulajdonságok közti távolságon alapszik [4], és egy átlagolásból, valamint egy standard eltérés számolásból áll.[ábra]

Első lépésként nyers adatokból kinyert tulajdonságokat (DD, UU, DU, UD, H) sorra átlagoljuk a következő képlet szerint:

(1) Ahol az tulajdonság átlagát, a minták számát, és a az tulajdonság mintáját jelenti.

Eredményképpen egy vektort kapunk, ami sorra tartalmazza az összes leütött karakter tulajdonságainak az átlagát. Következőnek standard eltérést számolunk a következőképpen:

(2) Ahol az tulajdonság standard eltérését, a minták számát, és a az tulajdonság mintáját jelenti, valamint az tulajdonság átlagát.

Ez szintén egy vektort fog adni, ami a tulajdonságok eltérését mutatja az átlagtól. Ezt az értéket a pontosabb eredmény érdekében minden tulajdonságnál leskáláztuk –el. Az átlag és standard eltérés segítségével már ki tudjuk számolni egy megadott minta pontosságát, ami egy pontszám lesz. Hogy a pontszámot megkapjuk, a következő képletet alkalmaztuk:

(3) Ahol a végső pontszámot jelenti.

A fenti képlet alapján minél közelebb van egy minta, annál kisebb lesz a végső pontszám [ábra]. Hogy 0 és 1 értékek közé skálázzuk, és megfordítsuk a pontszám növekedését, hogy a közeli minták nagyobb pontokat adjanak, a (4) képletet használtuk. A felhasználónkénti vágáspont meghatározásáért kizártuk a legkisebb pontszámot elért 10%-ot.

### Kimenetek fúziója

A biometriai eszközök esetében, amennyiben több tulajdonságot vizsgálunk egyszerre, ajánlott a kimeneteket valamilyen módon fuzionálni, mivel így jobban tudja a rendszer azonosítani a felhasználót, mint ahogy egyenként tudná a kimenetek alapján. Fúziót több módon is el végbe lehet vinni, mi kettőt próbáltunk ki ezek közül:

* Középérték alapú fúzió: Az eredményeket minden bemenetre összeadjuk, majd ezt az értéket vizsgáljuk meg, hogy egy adott vágáspont felett van-e
* Szorzás alapú fúzió: Az eredményeket összeszorozzuk, és a kapott értéket vizsgáljuk, hogy a vágáspont felett van-e

Egy másik módszer a tulajdonságok egy vektorban való elhelyezése, majd erre a vektorra való tanítás.

A különböző kimeneteken alkalmazott algoritmus nem szükséges, hogy ugyanaz legyen, de minél jobb eredményt ér el egy algoritmus annál jobb lesz a fúzió is.

A következő ábrán látható a példa a fúziós módszerekre: [ábra]

## Felhasznált technológiák

### Adattárolás Firebaseben

Választásunk azért esett a Firebase felhő alapú, valós idejű adatbázisra, mert gyorsan le lehet kérni belőle az adatokat, könnyű szemmel tartani a változásokat, valamint a JSON típusú adatok tárolása helyzetünkben sokkal előnyösebb, mint egy relációs adatbázisban táblákat használni.

Direkt Laravel támogatást a Firebase nem nyújt, ezért egy külön modul segítségével oldottuk ezt meg, amit GitHub-on publikáltak. Miután hozzáadtuk a Laravel projektünkhöz az adott modult, egy Firebase által generált kulcs JSON file segítségével bárhol le tudtuk kérni az adatbázis referenciát. Miután a referenciát megkaptuk, a következő műveleteket lehet végre hajtani:

* getReference(útvonal)
  + Visszatérít az adott útvonalra egy hivatkozást.
* orderByChild(csomópont neve).
  + Az adott csomópont szerint fogja rendezni a lekért adatokat, amennyiben a csomópont létezik.
* equalTo(érték)
  + Amennyiben egy adott érték szerint már rendezve van az orderByChild tagfüggvény által a lekérendő érték, ez a függvény csak azokat a csomópontokat kéri le, amelyek egyenlők a megadott értékkel.
* getValue()
  + Visszatéríti a kért csomópontokat.
* set(érték)
  + Beállítja az adott referenciára a megadott értéket. Ha egy asszociatív tömböt adunk meg akkor az elemek egymás alatt, hierarchikusan fognak elhelyezkedni.

A következőkben néhány példát láthatunk az itt megemlített függvények működésére: [példák: kép–nodark + szöveg]

# A rendszer specifikációi

## Követelmény specifikáció

### Kivonat

A dokumentum tartalmazza azon követelményeknek leírását, melyek betartásával az alkalmazást sikeresen el lehet készíteni. A specifikáció tartalmazza az alkalmazásra vonatkozó követelményeket, mind a gyűjtő, mind a tesztelő részére vonatkozólag.

A dokumentum első szekciója a bevezető, amely általánosan bemutatja a dokumentum célját, valamint rövid áttekintést nyújt arról. Ezt követi az alkalmazás részletes áttekintése, ami során említést teszünk a funkcionalitásokról, felhasználói osztályokról, függőségekről és megszorításokról. Ezek után a követelmények lesznek tárgyalva, és minden funkcionalitás részletes leírása.

### Bevezető

#### A dokumentum célja

A dokumentum tartalmazza a billentyűzési adatokat gyűjtő és ezen adatok alapján azonosító alkalmazás specifikációját. A cél a követelmények részletet és egyértelmű megfogalmazása. Ezek a követelmények a projekt megtervezéséhez szükségesek, és érvényesülniük kell a végleges termékre.

#### A dokumentum olvasói

A dokumentum célközönsége magába foglalja:

* Az alkalmazás fejlesztőit
* Az alkalmazás továbbfejlesztőit
* Az alkalmazás tesztelőit

#### Rövid áttekintés

A projekt célja egy olyan webes alkalmazás létrehozása, ami képes egy felhasználót hitelesíteni a bejelentkezési mezőkbe írt karakterek leütési ritmusa alapján. A hitelesítés mellett egy olyan lehetőséggel is kell, hogy rendelkezzen az alkalmazás, mely biztosít egy könnyen kezelhető adatgyűjtő felületet. A weboldal külsejénél fontos az esztétikus megszerkesztés, valamint a dinamikus navigálási lehetőség.

A projekt két részre bontható: először el kell készíteni a bejelentkezési felületet, majd a JavaScript gyűjtőt, ami által kinyerjük a felhasználó billentyű leütési időbélyegjeit.

### Áttekintés

#### Funkcionalitás

A funkcionalitások a bejelentkezés illetve a tanítás köré csoportosulnak leginkább. Emellett fontos, hogy a felhasználó tudjon regisztrálni az oldalra, az elfelejtett jelszavát tudja újra beállítani e-mail cím által, valamint tudja megtekinteni tanítás esetében a saját folyamat állapotát.

A projektre vonatkozó funkcionalitások listája a következő:

* Regisztrálási funkció
  + A szükséges adatok megadásával felhasználói fiók létrehozása
* Bejelentkezési funkció
  + Valós felhasználónév-jelszó kombinációval belépés a tanító oldalra
  + Tesztelés esetében, ha a bejelentkező személy átmegy a másodlagos ellenőrzésen is, megjeleníteni, hogy sikeresen bejelentkezett
* Jelszó újraállítási funkció
  + A felhasználó az e-mail címét megadva egy link által újra tudja állítani a bejelentkező jelszavát
* Tanítási funkció
  + A felhasználó beírt adatainak és rögzített billentyűzési adatainak a lementése, amennyiben azok valósnak bizonyulnak
  + A ritmust lehetségesen elrontó karakterek letiltása vagy figyelmen kívül hagyása, helyzettől függően
  + Letiltott karakterek esetén hibaüzenet megjelenítése
* Folyamat megtekintési funkció
  + A felhasználó kell, hogy lássa, hány helyes mintát küldött be, mennyi kell egy szesszió befejezéséhez, és hány szessziót vitt végig
* Nyelvi funkció
  + Angol és magyar nyelvű felület
* Segítő funkció
  + A választott nyelven visszaküldött üzenet érvénytelen adatok esetén
  + Leírása a kísérletnek, a letiltott és ignorált karaktereknek, valamint az egyéb lehetséges félreértések tisztázása, mind például a tanító jelszó és a bejelentkezési jelszó közti különbség
  + Megjelenítése tanító módban a beírandó adatoknak

#### Felhasználói osztályok és karakterisztikái

Figyelembe véve, hogy a felhasználók lehetnek tesztelők, vagy a gyűjtésben résztvevők, akik a tanító halmazt bővítik, a következő két csoportba sorolhatjuk őket:

* Tesztelő felhasználók
  + Leginkább használt funkcionalitások, sorrendben:
    - Regisztrálási funkció
    - Bejelentkezési funkció
    - Segítő funkció
    - Jelszó újraállítási funkció
    - Nyelvi funkció
* Tanító felhasználók
  + Leginkább használt funkcionalitások, sorrendben:
    - Tanítási funkció
    - Bejelentkezési funkció
    - Regisztrálási funkció
    - Segítő funkció
    - Folyamat megtekintési funkció
    - Jelszó újraállítási funkció
    - Nyelvi funkció

#### Működési környezet

Az alkalmazás egy internetes böngésző segítségével érhető el a kliens számára.

#### Felhasználói segédletek

A segítő funkció által a felhasználó rálátást nyer arra, milyen lehetőségei és megkötései vannak az oldal használata során. A regisztrálás nézeten belül specifikálva van, hogy milyen karaktereket használhat a felhasználó egyes mezőknél, valamint tisztázva vannak a mezők jelentései.

A tanító oldalon minden bejelentkezés során megjelenik egy jelzés, ami figyelmezteti a felhasználót arra, milyen karaktereket üthet le, valamint milyen megszorításokhoz kell, hogy tartsa magát az adatbevitel alatt illetve milyen segítséget nyújt az oldal. Az említett oldalon szintén lehetősége van arra a felhasználónak, hogy megtekintse a beírandó adatokat.

#### Megszorítások és függőségek

Az alkalmazás használatához első sorban az internetes kapcsolat elengedhetetlen.

Ezen kívül ajánlott minél újabb böngészőt használni. A böngészőnél be kell, hogy legyen kapcsolva a JavaScript, különben sem a billentyű leütési időbélyegek lekérése és továbbküldése, sem a többi JavaScript alapú funkció nem lesz elérhető.

### Rendszer követelmények

Az alkalmazást egy szerverre kell feltölteni, hogy elérhető legyen az interneten. Szükséges egy web szerver, ajánlott az Apache. A Laravel 5.5 verziója igényeli a PHP 7.2 verzióját, valamint a Composert. Adatbázis konfigurációra nincs szűkség, mivel az összeköttetés a kulcs segítségével megoldódik.

### Funkcionális követelmények

.. Táblázat - Regisztrálási funkció

|  |  |
| --- | --- |
| **Regisztrálási Funkció** | |
| **Leírás** | A szükséges adatok megadásával felhasználói fiók létrehozása |
| **Prioritás** | Nagyon fontos |
| **Kiváltás** | 1. A felhasználó a bejelentkezési oldalon rákattint a *Fiók regisztrálása* linkre. |
| **Funkcionális követelmény** | A regisztrációs oldalra való navigálás után a felhasználó ki kell, hogy töltse a következő mezőket:   * Teljes név – szöveg, legalább 8 karakter * Életkor – szám 10 és 120 között * Felhasználónév – szöveg, legalább 8 karakter * E-mail cím – valós e-mail cím formátum * Tanítandó jelszó – ezt fogja a felhasználó a többször bevinni, legalább 8 karakteres szöveg, mely tartalmaz kis- és nagybetűt, számot és speciális karaktert * Nemzetiség – legördülő menü, lehet: Román, Magyar, Angol * Jelszó – a belépéshez van rá szűkség, rejtett szöveg, legalább 10 karakteres * Jelszó megerősítés – rejtett szöveg * Nem: legördülő menü, lehet: Férfi, Nő * Domináns kéz: legördülő menü, lehet: Bal, Jobb * Eszköz: legördülő menü, lehet: Laptop, PC, Táblagép, Okos telefon (az utolsó kettő le van tiltva, még nem működőképes) * Szesszió kód - ellenőrzött adatbevitel esetére, ha nem ellenőrzött maradhat üresen   Emellett fontos, hogy a felhasználó elfogadja a Felhasználói Feltételeket.  Amennyiben minden mező helyesen ki van töltve és a felhasználó elküldte az űrlapot, átirányítódik a bejelentkezés oldalra. Hiba esetén az alkalmazás jelez, hogy melyik mezőben, és milyen típusú hiba jelent meg. |
| **Megszorítás** | - |

.. Táblázat - Bejelentkezési funkció

|  |  |
| --- | --- |
| **Bejelentkezési Funkció** | |
| **Leírás** | Valós felhasználónév-jelszó kombinációval belépés a tanító oldalra.  Tesztelés esetében, ha a bejelentkező személy átmegy a másodlagos ellenőrzésen is, megjeleníteni, hogy sikeresen bejelentkezett. |
| **Prioritás** | Nagyon fontos |
| **Kiváltás** | 1. A felhasználó a főoldalra navigál. 2. A felhasználó nincs bejelentkezve, de olyan útvonalat próbál elérni, amit csak bejelentkezett felhasználók használhatnak. 3. A regisztrációs oldalon a felhasználó a *Vissza a bejelentkezéshez* linkre kattint. 4. A felhasználó kijelentkezik. |
| **Funkcionális követelmény** | A felhasználónak rendelkeznie kell egy valós fiókkal. Felhasználónév-jelszó párossal tud regisztrálni.  A felhasználó amennyiben adatokat akar bevinni az adathalmaz bővítéséhez, a billentyűzési stílusát ellenőrző algoritmus is érvénybe lép, és amennyiben a mintája nem elég pontos, vissza kerül a főoldalra hibával.  Sikeres bejelentkezés esetén:   * Tanító módban: A felhasználó a tanítást lebonyolító oldalra lesz átirányítva. * Tesztelő módban: A felhasználó vissza lesz küldve a bejelentkezési oldalra egy sikeres belépést közlő üzenettel. |
| **Megszorítás** | Amennyiben a felhasználó teszteli a rendszert:  A böngészőben be van kapcsolva a JavaScript. |

.. Táblázat - Tanítási funkció

|  |  |
| --- | --- |
| **Tanítási Funkció** | |
| **Leírás** | A felhasználó beírt adatainak és rögzített billentyűzési adatainak a lementése, amennyiben azok valósnak bizonyulnak.  A ritmust lehetségesen elrontó karakterek letiltása vagy figyelmen kívül hagyása, helyzettől függően.  Letiltott karakterek esetén hibaüzenet megjelenítése. |
| **Prioritás** | Nagyon fontos |
| **Kiváltás** | 1. A felhasználó sikeresen bejelentkezett |
| **Funkcionális követelmény** | A felhasználó három mezőbe kell, hogy tudjon írni adatokat:   1. Felhasználónév – szöveg, legalább 8 karakter 2. Teljes név – szöveg, legalább 8 karakter 3. Tanítandó jelszó - legalább 8 karakteres szöveg, mely tartalmaz kis- és nagybetűt, számot és speciális karaktert   Amennyiben a felhasználó a letiltott billentyűk közül valamelyiket lenyomja, a mező üresre állítódik, nullázódik a billentyűzési adat az adott mezőre, és megjelenik egy hibaüzenet 5 másodpercre, majd eltűnik.  A letiltott billentyűk a következőek: *Backspace, Delete, Shift/Alt* lenyomása majd felengedése, közben leütött karakter nélkül, illetve *CapsLock* lenyomása kétszer, anélkül hogy közben karakterek is leütődtek volna.  A *Tab, Enter, Ctrl* és navigáló billentyűk figyelmen kívül vannak hagyva billentyűleütési szempontból.  Ha a felhasználó három másodpercen keresztül nem ír egy adott mezőbe, akkor az ezt követő karakter beírás lenullázza a mezőt, így megoldva azt, hogy a felhasználónak, ha elterelődik a figyelme, ne romoljon az adat.  Sikeres adatbevitel esetén az oldalon megjelenítődik, hogy egyel több minta van leadva, a mezők újra állítódnak, és a felhasználó beviheti a következő adatot, |
| **Megszorítás** | 1. A böngészőben be van kapcsolva a JavaScript. |

.. Táblázat - Jelszó újraállítási funkció

|  |  |
| --- | --- |
| **Jelszó Újraállítási Funkció** | |
| **Leírás** | A felhasználó az e-mail címét megadva egy link által újra tudja állítani a bejelentkező jelszavát. |
| **Prioritás** | Közepesen fontos |
| **Kiváltás** | 1. A bejelentkezés oldalon az *elfelejtett jelszó* linkre kattintva. |
| **Funkcionális követelmény** | A felhasználónak kell rendelkeznie egy már meglévő fiókkal, és a fiókhoz tartozó e-mail címnek valódinak kell lennie. Ez után az adott e-mail címre elküldődik egy link, amit követve a felhasználó újra be tudja állítani a jelszavát. |
| **Megszorítás** | 1. A böngészőben be van kapcsolva a JavaScript. |

.. Táblázat - Folyamat megtekintési funkció

|  |  |
| --- | --- |
| **Folyamat Megtekintési Funkció** | |
| **Leírás** | A felhasználó kell, hogy lássa, hány helyes mintát küldött be, mennyi kell egy szesszió befejezéséhez, és hány szessziót vitt végig. |
| **Prioritás** | Kevésbé fontos |
| **Kiváltás** | 1. Sikeres bejelentkezés során |
| **Funkcionális követelmény** | A felhasználó láthatja számszerűen és egy fejlődésmutatón, hogy eddig hány adatot írt be. Minden sikeres leküldéssel növekszik ennek az értéke. Mikor a szessziót a felhasználó befejezte, egy üzenet jelenik meg, ami közli a sikeres kitöltést. |
| **Megszorítás** | 1. A böngészőben be van kapcsolva a JavaScript. |

.. Táblázat - Nyelvi funkció

|  |  |
| --- | --- |
| **Nyelvi Funkció** | |
| **Leírás** | Angol és magyar nyelvű felület biztosítása. |
| **Prioritás** | Kevésbé fontos |
| **Kiváltás** | 1. Az oldal bal felső sarkában lévő legördülő listából kiválaszt a felhasználó egy nyelvet. |
| **Funkcionális követelmény** | A nyelv váltás hatására az adott oldalon, és minden lapon, ahova az aktuális oldalról navigálunk, a nyelv megváltozik aszerint, hogy melyiket választotta a felhasználó. |
| **Megszorítás** | - |

.. Táblázat - Segítői funkció

|  |  |
| --- | --- |
| **Segítői Funkció** | |
| **Leírás** | A választott nyelven visszaküldött üzenet érvénytelen adatok esetén.  Leírása a kísérletnek, a letiltott és ignorált karaktereknek, valamint az egyéb lehetséges félreértések tisztázása, mind például a tanító jelszó és a bejelentkezési jelszó közti különbség.  Megjelenítése tanító módban a beírandó adatoknak. |
| **Prioritás** | Közepesen fontos |
| **Kiváltás** | 1. A felhasználó a regisztrációs vagy tanító oldalra navigál. |
| **Funkcionális követelmény** | Útmutatót adni arról, hogy miket kell tegyen a felhasználó, mik a lehetőségei és megszorításai. |
| **Megszorítás** | 1. A böngészőben be van kapcsolva a JavaScript. |

A felhasználói eset diagram itt látható: [diagram]

### Nem funkcionális követelmények

#### Skálázhatóság és továbbfejleszthetőség

Az alkalmazás legyen skálázható, és könnyen továbbfejleszthető. Ajánlott az MVC tervezési mintát alkalmazni.

#### Karbantarthatóság

A bekövetkezett hibák legyenek naplózva a könnyű karbantartás végett.

#### Nyelvi követelmények

Az oldal több nyelven is megírható, de minimálisan angol nyelven legyen megírva. Ajánlott a Laravel lokalizációs módszerét használva nyelvi fájlokban tárolni a megírt szöveget, a későbbi többnyelvűsítés érdekében.

### Interfész követelmények

#### Felhasználói interfész

A felhasználói interfész HTML-ben készüljön, böngészőből lehessen elérni. Ajánlott a HTML5 dokumentum típus, valamint Bootstrap 4.0 vagy annál újabb verzió, hogy könnyedén ki lehessen alakítani egy esztétikus felhasználói felületet. Emellett a Bootstrap segítségével könnyedén tudunk egy olyan nézetet létre hozni, melynek formája és elrendezése a használt eszköz kijelzőjétől függően változik

A felület vázlata a következőképpen néz ki: [ábra]

# Részletes tervezés

## Adatok tárolása Firebaseben

A felhasználói és kísérleti adatok tárolására a Firebase valós idejű NoSQL adatbázisára esett a választásunk. Ennek fő okai közé sorolható a nagyobb mennyiségű adatok gyors kezelése valamint a tulajdonsága, hogy a PHP asszociatív tömbjeit képes egyben feltölteni és átalakítani őket JSON formátumba, ez által nincs szűkség külön konverzióra.

A Firebase adatbázisban eltárolt csomópontoknak a következő a struktúrája: [ábra]

Magyarázat: Az egész adatbázis egy csomópont alatt helyezkedik el, ami a [*state-exam-project-keystroke*](https://console.firebase.google.com/u/0/project/state-exam-project-keystroke/database/state-exam-project-keystroke/data/). Ez alatt a következő csomópontok helyezkednek el:

1. *sessions* – a felügyelt tanításokhoz tartozó szesszió kódokat itt tároljuk
2. *user\_keystroke\_raw* – itt találhatóak a felhasználók nyers billentyűzési adatai, a következőképpen:
   1. *username(s)* – a felhasználónév alapján lehet megtalálni valakinek az adatait
      1. *timestamp(s)* – egy két részből álló azonosítója egy adott beküldésnek, az első része a beírás kezdeti dátumát, míg a második az aktuális időbélyeget tartalmazza
         1. *input\_field(s)* – az adott beküldés során kapott mezőket számokkal jelöltük, ahol 0 a felhasználónév, 1 a teljes név és 2 a tanítandó jelszó
            1. *keystroke(s)* – egy bemeneti mezőben leütött billentyű, aminek két értéke van: egy leütési és egy felengedési időbélyeg
      2. *forgeries* – a felhasználón végzett utánzások száma
         1. *username(s)* – az utánzó vagy utánzók neve, ez alatt ugyanolyan időbélyegek vannak, mint a 2.1. alatt
3. *user\_templates* – a felhasználók mintái, különböző algoritmusok által
   1. *username(s)* - a felhasználónév alapján lehet megtalálni valakinek a mintáit
      1. *algorithm(s)* – az algoritmus neve, amelyik által készült a minta
         1. *pattern* – a minta, ami általában pár vektorból áll
         2. *threshold* – a küszöbérték az adott felhasználóra
4. *users* – a felhasználók adatai, amiket a regisztrációnál adtak meg
   1. *username(s)* - a felhasználónév alapján lehet megtalálni valakit
      1. *device* – a felhasználó által választott eszköz
      2. *email* – a felhasználó e-mail címe
      3. *forgeries* – a utánzások száma
         1. *username(s)* – a utánzók nevei
            1. *number\_of\_samples* – az aktuális utánzó által leadott minták száma
      4. *full\_name* – a felhasználó teljes neve
      5. *input\_lengths* – a beírt karakterek hossza, mezőnként
         1. *input* – a mező számmal jelölve, mint a 2.1.1.1. csomópontnál, az adott mezőbe beírt karakterek hosszát tartalmazza
      6. *number\_of\_samples* – a felhasználó által beadott minták száma
      7. *password\_hash* – a bejelentkezéshez használt jelszó titkosítva
      8. *training\_password* – a tanuló jelszó, amit a felhasználó ír be több alkalommal
      9. *user\_age* – a felhasználó életkora
      10. *user\_gender* – a felhasználó neme
      11. *user\_nationality* – a felhasználó nemzetisége
      12. *username* – a felhasználónév

## Architektúra

Az alkalmazás architektúrája két fő komponensből tevődik össze:

* Kliens rész
  + A kliens által látható, publikus rész
  + Itt található a JavaScriptben megírt gyűjtő is
* Szerver rész
  + Itt megy végbe a szerver oldali hitelesítés, valamint az adatok tárolása

A következő diagramon látható a rendszer architektúrája: [ábra]

## A rendszer tervezése és bemutatása

A rendszer Laravel 5.5 keretrendszerben készült, és követi az MVC tervezési mintát. Az osztályok, amiket vagy saját kezűleg hoztunk létre, vagy a Laravel parancsainak segítségével a következő osztálydiagramon láthatóak:[ábra]

### A kontroller rész

A Laravel keretrendszer által minden oldalhoz saját kontrollert készíthetünk, ami felelős a háttérben elvégzett logikáért. A kontrollerek függvényeket tartalmaznak, melyek a hozzájuk rendelt webes útvonal felkeresése esetén hívódnak meg. Ennek ellenére a kontroller nem minden esetben hívódik meg az URL beütése után, ha az útvonalra egy Middleware van csatolva. A Middleware-ek a kontroller előtt lépnek érvénybe, és képesek átirányítani a felhasználót egy másik URL-re még azelőtt, hogy elérnék az eredetileg kitűzött kontrollert. A következő állapotdiagram szemlélteti a felhasználó lehetséges eseteit az oldalon való navigálásnál. [ábra]

### A modell rész

A modell részét a mi esetünkben a bárhol lekérhető adatbázis kapcsolat a Firebase-el helyettesíti.

### A nézet (view) rész

A nézetek azok az erőforrások, melyek megjelenítődnek a felhasználó számára. Laravel esetében támogatva vannak az úgynevezett *blade* sablonok, amikkel könnyedén tudunk bonyolult logikát bevinni nézetünkbe. A nézethez néhány adat még a megjelenítés előtt hozzá kell, hogy rendelődjön, amiket a *View Composerek* segítségével oldottunk meg. Az nézetek elkészítéséhez Bootstrap 4.0 volt használva design szempontjából. Emellett a nézeteknél a többnyelvűsítés érdekében nyelvi változók vannak jelen statikus szövegek helyett

A következőkben látható néhány kép az elkészített nézetekről:

### Segítő osztályok

A billentyűzési algoritmusok strukturálása érdekében létre hoztunk egy absztrakt osztály, amiből származik minden billentyűzési algoritmus osztálya. Az osztályok a következő módon működnek: [ábra]

## Ajax hívások bemutatása

A kliens oldalon a tanításnál aszinkron hívással küldjük el a szervernek az űrlap tartalmát és a billentyűzési időket. Ehhez a JQuery könyvtár *post* metódusát használtuk. Az adatok megérkeznek egy kontroller függvényéhez, és amennyiben valóban Ajax hívás történt a függvény elvégzi a feldolgozást és vissza küldi az eddig összegyűjtött minták számát. Ezt a fogadó szkript feldolgozza, és megjeleníti a változásokat az oldalon.

## Adatok mozgása

A teljes adatmozgás az oldalon a következő diagramon látható:[ábra]

# Gyűjtő üzembe helyezése és kísérleti eredmények

## Gyűjtő üzembe helyezése

Az alkalmazás a kliens részéről nem igényel semmiféle üzembe helyezést. A kliensnek egy böngészővel, amiben engedélyezve van a JavaScript és internetkapcsolattal kell rendelkeznie.

Szerver oldalon egy PHP 7.2 vagy újabb verzióra van szűkség, valamint a *Composer* függőség kezelőre. Az üzemeléshez ajánlott az Apache web szerver használata

## Felmerült problémák és megoldásaik

Az egyik felmerült probléma abból származott, hogy a Firebase harmadik fél által készített összekötő modulja *localhost-on* rendesen működött, de az Ubuntu alapú szerveren nullás érték feltöltése esetén hibát adott. Ezt úgy orvosoltuk, hogy az adatbázisban minden értéket karakterláncként tároltunk.

## A begyűjtött adathalmaz

### Felhasználóktól gyűjtött adathalmaz

A gyűjtés során az oldalra 73 felhasználó regisztrált, viszont ezek közül csak 56 adott le adatokat. Minden regisztrált felhasználó a tanító felületen kellett beírja a felhasználónevét, valós nevét és az általa választott tanítandó jelszavat. Amennyiben valamelyik adatot elfelejteti, az *Információ* linkre kattintva képes megtekinteni minden bevitelt igénylő információt. A tanítandó jelszó bevezetése biztonsági és kényelmi megfontolásból történt, mivel a valós jelszó elfelejtése esetén azt újra be kéne állítani, ami elrontaná a kísérletet. Amennyiben a valós jelszavat nem titkosítva tárolnánk, szerver oldalon megoldható lenne a megjelenítése, de könnyen feltörhető lenne a felhasználók fiókja.

A gyűjtés alatt szessziónként 25 adatot kellett bevigyen minden felhasználó, és 43 személynek sikerült kettő, vagy több szessziót bevinnie. Eddig 3628 begépelt felhasználónév, jelszó, valamint teljes név időbélyegekből álló mintája van az adathalmazunkban (összesen 10884), amibe nem számoltuk bele azon felhasználók mintáit, akik 50-nél kevesebbet adtak le. A kinyert adatmennyiségek eloszlása a következő hisztogramon látható: [ábra]

Bár szerver oldalon ellenőrizve vannak a bevitt adatok, de a leütött karaktereket nem lehet egy általános mintához kötni, mert a lenyomott karakterek száma személyenként változhat. Erre példa a *Shift* és *CapsLock* közötti különbség: Mikor nagybetűt akarunk írni, a *Shift* egy karakternek számít, aminek leütési ideje a betű előtt, és felengedési ideje a betű után van időrendi sorrendben. Ezzel ellentétben a *CapsLock* leütése és felengedése nagybetűs módra váltáshoz egy karakternek számít, míg a vissza váltás kisbetűre még egynek. Így az első megoldással két karakterünk lesz, míg a másodikkal három.

A beírt adatok első tíz mintája elvetődik, ha a felhasználó több mint négy nem ugyanolyan hosszúságú karaktersort írt le. A tízedik minta bevitelénél szerver oldalon egy ellenőrzés megy végbe: megkeressük a leggyakrabban megjelenő hosszúságokat mind a három mezőre, majd megnézzük, hány meglévő bevitelre érvényes az, hogy mindhárom bemenet ezekkel a hosszúságokkal rendelkezzen. Amennyiben hat és tíz közötti az elfogadott bevitelek száma, a visszautasítottak törlésre kerülnek, és a felhasználó értesül erről az által, hogy csökken a mintáinak a száma a felhasználói felületen is. Tíz helyes minta a felhasználó profiljához hozzárendelődik a három bemenetnek a hossza, és minden ezt követő bevitelt ezek alapján ellenőrizünk, ez által biztosítva az elkövetkező adatok helyességét.

A felhasználókról való felmérések a következő ábrán láthatóak: [ábra]

### Imposztoroktól gyűjtött adathalmaz

Miután elég aktív felhasználó regisztrált, nekifogtunk az imposztorok gyűjtésének. Az alkalmazáson belül létre hoztunk egy új útvonalat (Route-ot), amit csak azok érhetnek el, akikkel megosztjuk ezt. Itt a résztvevő beléphet a felhasználójával, illetve, amennyiben nem rendelkezik fiókkal az oldalon, vagy elfelejtette adatait, beléphet vendégfelhasználóként is.

A valós felhasználóktól való adat gyűjtéséhez hasonló felületen történik az utánzó felhasználók belépéseinek gyűjtése is. Egyetlen különbség, hogy egy más felhasználónak az adatait kell, hogy beírják, és ezek megjelennek, a könnyítés érdekében, a bemeneti mezőkben is helykitöltőként.

Az előbb említett valós adatok gyűjtésénél szó esett a *CapsLock* és *Shift* között lévő problémáról. Az imposztor adatok gyűjtésénél emiatt az eltérés miatt a begyűjtött 43 utánzásból, amiből minden felhasználót legalább egy szesszióban utánoztak (*1 szesszió = 25 minta*), csupán 27 felhasználónak volt elfogadható az utánzása a különböző karakter leütési hosszok miatt, a többi minta kísérleti szempontból selejtnek minősül. Emiatt 675 mintát tudtunk gyűjteni az imposztoroktól, minden mezőre (összesen 2025 minta).

A rendszer ennél a gyűjtésnél mindig olyan felhasználót adott az éppen belépőnek, amelynek még nincs, vagy a lehető legkevesebb számú utánzása van. A közeljövőben tervezzük a fentebb említett hibát kijavítani a *CapsLock* és *Shift* karakterek figyelmen kívül hagyásával.

## Kísérleti eredmények

A kísérleti adathalmaz, az előző szekcióban tárgyalt okok miatt, 27 felhasználóból áll. Minden felhasználó 50 adattal rendelkezik, amiből xx tanításra, és xx tesztelésre van felhasználva. Emellett 25 utánzás is hozzá van rendelve minden felhasználóhoz, melyek segítségével tudunk egyenlő hibaarányt számolni.

A kísérletek során sikerült egy olyan algoritmushoz jussunk, mely xx EER-el, valamint xx globális EER-el rendelkezik. A próbálkozások során teszteket végeztünk a Manhattan Scaled, Mahalanobis, valamint egy módosított Manhattan Scaled osztályozóval. A kísérletekből kiderült, hogy a mezők különbözősége ellenére nagyon hasonló egyenlő hibaarány értékeket adnak az eddig tesztelt osztályozók használatánál. A különböző mezők pontszámait átlagoló valamint szorzó fúziónak vetettük alá, melynek hatására a következő eredményeket értük el: [ábrák, táblázatok és mérést magyarázó szövegek]

# Következtetések

## Megvalósítások

Eddigi megvalósításaink közé tartozik egy Laravelben megírt webes alkalmazás, ami tartalmaz egy adatgyűjtő valamint egy azonosító (tesztelő) részt. Az adatgyűjtő komponens által képesek vagyunk egyidejűleg több felhasználótól billentyű leütési mintákat gyűjteni, valamint az utánzásokat is ugyanilyen módon az alkalmazással ki lehet nyerni. Emellett az éles tesztelés is lehetséges a teszt bejelentkezésért felelős URL-en keresztül.

Emellett sikerült gyűjteni 43 felhasználótól 50-nél több adatot, és 27 felhasználónak 25 vagy több utánzást is. Hogy az adatokat egyszerűen lehessen használni más programozási nyelvek által is, megírtunk egy átalakítót, ami a teljes JSON-ban tárolt adathalmazból a következő *csv* fájlokat generálja:

* *user\_info.csv* – a felhasználók adatai
* *username\_keystroke\_data.csv* – a felhasználónevek nyers adataiból kinyert tulajdonságok
* *passwordTrain\_keystroke\_data.csv* – a tanítandó jelszavak nyers adataiból kinyert tulajdonságok
* *full\_name\_keystroke\_data.csv* – a teljes nevek nyers adataiból kinyert tulajdonságok
* *username\_keystroke\_forgeries.csv* – a felhasználónevek utánzásainak nyers adataiból kinyert tulajdonságok
* *passwordTrain\_keystroke\_forgeries.csv* – a tanítandó jelszavak utánzásainak nyers adataiból kinyert tulajdonságok
* *full\_name\_keystroke\_forgeries.csv* – a teljes nevek utánzásainak nyers adataiból kinyert tulajdonságok

Egy másik eredmény, amit a dolgozat alatt tárgyaltunk, a kísérletek elvégzése, melyeknek adatai hasznosak lehetnek későbbi kutatások számára. Emellett a kísérletek során egy olyan rendszert alkottunk, mely xx EER-el rendelkezik felhasználók azonosítása terén.

## Összehasonlítás hasonló rendszerekkel

Billentyűzés alapú másodlagos azonosító rendszerek már léteznek, és nagyobb kereskedelmi illetve bankozási alkalmazásoknál be vannak építve, viszont még nem található publikusan elérhető, magas pontosságú rendszer. Mi ennek a megalkotására törekedünk, miközben a lehető legjobb algoritmust keressük a rendszernek.

Más eddigi kísérletek során, például a Killourhy által kiadott cikkben [2], egy mezőre jobb eredményt értek el, viszont fúzió használatával három mezőre egy pontosabb azonosítót készítettünk.

Az egyik leghasonlóbb gyűjtő rendszer a miénkhez a Caeni egyetemen készült [9]. Itt felhasználónév-jelszó párost gyűjtöttek felügyelet nélkül az egyetem hallgatóitól. A leírt felhasználói felület is hasonló funkcionalitásokkal rendelkezik a miénkhez, a cikk alapján. Fő különbségek a kísérletek végzésénél, és azok eredményeinél jelentkeznek.

## További fejlesztési irányok

A rendszer még fejlesztést igényel könnyebb adat kinyerés, pontos utánzások, pontosabb osztályozó algoritmus és tanítási mintaszám szempontjából.

* Az adat kinyerésénél a következőkben nem fogjuk tárolni a *Shift* illetve *CapsLock* karakterek időit, a fentebb említett okokból, valamint tervezzük a rendszert okos telefon és táblagép eszközökre is használhatóvá tenni. Jelenleg az oldal nézetei már úgy vannak tervezve, hogy jól kezelhető legyen az alkalmazás a kisebb kijelzővel rendelkező eszközöknél is.
* Az utánzások még pontosabbá tételéhez egy új ellenőrzőt adunk a tárolás előtt, mely a nem megfelelő hosszúságú szövegeket nem menti el, és jelez az utánzónak. Emellett az oldalon megjelenítjük a felhasználó által leütött karaktereket is, arra az esetre, ha valami olyan kombinációt használt, ami nem egyértelmű.
* Az osztályozási algoritmus szempontjából a következőkben a kovariancia mátrix felhasználásával próbálunk javítani az algoritmuson, valamint a túl távoli minták kiszűrésével. Emellett a tulajdonságok szerinti fúziót is szeretnénk letesztelni.
* A valós alkalmazás fejlesztése érdekében tervezzük megkeresni azt a minimális mintaszámot, amire a rendszer már elfogadható azonosítást nyújt.

# Irodalomjegyzék

[1] Morales, A., Falanga, M., Fierrez, J., Sansone, C., & Ortega-Garcia, J. (2015, September). Keystroke dynamics recognition based on personal data: A comparative experimental evaluation implementing reproducible research. In *Biometrics Theory, Applications and Systems (BTAS), 2015 IEEE 7th International Conference on* (pp. 1-6). IEEE.

[2] Killourhy, K. S., & Maxion, R. A. (2009, June). Comparing anomaly-detection algorithms for keystroke dynamics. In *Dependable Systems & Networks, 2009. DSN'09. IEEE/IFIP international conference on* (pp. 125-134). IEEE.

[3] Teh, P. S., Teoh, A. B. J., & Yue, S. (2013). A survey of keystroke dynamics biometrics. The Scientific World Journal, 2013.

[4] Araújo, L. C., Sucupira, L. H., Lizarraga, M. G., Ling, L. L., & Yabu-Uti, J. B. T. (2005). User authentication through typing biometrics features. *IEEE transactions on signal processing*, *53*(2), 851-855.

[5] Ross, A. A., Nandakumar, K., & Jain, A. K. (2006). Handbook of multibiometrics (Vol. 6). Springer Science & Business Media.

[6] Tax, D. M. J. (2001). One-class classification: concept-learning in the absence of counter-examples [Ph. D. thesis]. Delft University of Technology, Stevinweg, The Netherlands.

[7] Araújo, L. C., Sucupira, L. H., Lizarraga, M. G., Ling, L. L., & Yabu-Uti, J. B. T. (2005). User authentication through typing biometrics features. IEEE transactions on signal processing, 53(2), 851-855.

[8] Zhong, Y., Deng, Y., & Jain, A. K. (2012, June). Keystroke dynamics for user authentication. In Computer Vision and Pattern Recognition Workshops (CVPRW), 2012 IEEE Computer Society Conference on (pp. 117-123). IEEE.

[9] Giot, R., El-Abed, M., & Rosenberger, C. (2012, July). Web-based benchmark for keystroke dynamics biometric systems: A statistical analysis. In Intelligent Information Hiding and Multimedia Signal Processing (IIH-MSP), 2012 Eighth International Conference on (pp. 11-15). IEEE.

[10] Fawcett, T. (2006). An introduction to ROC analysis. Pattern recognition letters, 27(8), 861-874.

[11] Li, S. Z., & Jain, A. (2015). Encyclopedia of biometrics. Springer Publishing Company, Incorporated.

# Függelékek