ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

NEUMANN JÁNOS
INFORMATIKAI KAR

SZAKDOLGOZAT

OE-NIK	Hallgató neve:	Kluka Norbert
2021	Hallgató törzskönyvi száma:	T/006009/FI12904/N

Óbudai Egyetem
Neumann János Informatikai Kar
Kiberfizikai Rendszerek Intézet

SZAKDOLGOZAT FELADATLAP

Hallgató neve: **Kluka Norbert**
Törzskönyvi száma: T/006009/FI12904/N
Neptun kódja: [redacted]

A dolgozat címe:

Középiskolai belső felvételi eljárást segítő háttérrendszer
High school preliminary process supporting system

Intézményi konzulens: Sarkadi Katalin
Külső konzulens:

Beadási határidő: 2021. december 15.

A záróvizsga tárgyai: Számítógép architektúrák
Big Data és üzleti intelligencia specializáció

A feladat

A szakdolgozat kapcsán a hallgató feladata egy olyan informatikai háttérrendszer kialakítása és dokumentálása, amely segítséget nyújt a középiskoláknak a felvételi eljárás adminisztrálásban, lebonyolításában. A cél a felvételiző nyolcadikos diákok adatainak gyűjtése, tárolása és elemzése. A hallgató feladata a projekt kapcsán a rendszer felépítésének megtervezése és implementációja. A rendszer három fő részből álljon: felvételizők regisztrációja a szóbeli eljárásra; szóbeli alatt a pontok rögzítése; írásbeli pontok felvitele és elemzés, rangsor készítés. Az adatokból automatikusan frissülő nézetek és kimutatások lesznek elérhetőek.

A dolgozatnak tartalmaznia kell:

- a rendszer hasznosságát,
- Use-case meghatározása iskolai igények felmérése a rendszerrel szemben,
- a backend felépítését,
- a frontend felépítését,
- a UI bemutatását,
- adat vizualizáció megvalósítását.

ÓBUDAI EGYETEM * Neumann János Informatikai Kar * Budapest

Ph.

.....
Dr. Fleiner Rita
intézetigazgató

A szakdolgozat elévülésének határideje: **2023. december 15.**
(OE TVSz 55.§ szerint)

A dolgozatot beadásra alkalmasnak tartom:

.....
külső konzulens

.....
intézményi konzulens

HALLGATÓI NYILATKOZAT

Alulírott hallgató kijelentem, hogy a szakdolgozat / diplomamunka saját munkám eredménye, a felhasznált szakirodalmat és eszközöket azonosíthatóan közöltem. Az elkészült szakdolgozatomban / diplomamunkámban található eredményeket az egyetem és a feladatot kiíró intézmény saját céljára térítés nélkül felhasználhatja.

Budapest, 2021. december 11.

.....

hallgató aláírása

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar
9. sz. melléklet

KONZULTÁCIÓS NAPLÓ

Hallgató neve: Kluka Norbert

Neptun kód: [redacted]

Tagozat: esti

Telefon: [redacted]

Levelezési cím (pl: lakcím): [redacted]

Szakdolgozat / Diplomamunka[1] címe magyarul:

Középiskolai belső felvételi eljárást segítő háttérrendszer

Szakdolgozat / Diplomamunka[2] címe angolul:

High school preliminary process supporting system

Intézményi konzulens: Sarkadi Katalin

Külső konzulens:

Kérjük, hogy az adatokat nyomtatott nagybetűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2021. 03. 23.	Szakdolgozat téma véglegesítése	
2.	2021. 04. 13.	Szakdolgozat tematikája	
3.	2021. 05. 04.	Elkészült dolgozat értékelése	
4.	2021. 05. 11.	További feladatok egyeztetése	

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat I. / Szakdolgozat II. (BSc) vagy Diplomamunka 1 / Diplo-mamunka 2 / Diplomamunka 3 / Diplomamunka 4[3] tantárgy követelményét teljesítette, beszámolóra / védésre[4] bocsátható.

Budapest, 2021. május 11.

intézményi konzulens

[1] Megfelelő aláhúzandó!
[2] Megfelelő aláhúzandó!
[3] Megfelelő aláhúzandó!
[4] Megfelelő aláhúzandó!

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar
9. sz. melléklet

KONZULTÁCIÓS NAPLÓ

Hallgató neve: Kluka Norbert

Neptun kód: [redacted]

Tagozat: esti

Telefon: [redacted]

Levelezési cím (pl: lakcím): [redacted]

Szakdolgozat / Diplomamunka[1] címe magyarul:

Középiskolai belső felvételi eljárást segítő háttérrendszer

Szakdolgozat / Diplomamunka[2] címe angolul:

High school preliminary process supporting system

Intézményi konzulens: Sarkadi Katalin

Külső konzulens:

Kérjük, hogy az adatokat nyomtatott nagybetűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2021. 10. 12.	A fejlesztés menetének ellenőrzése	
2.	2021. 11. 08.	A fejlesztés dokumentálásának egyeztetése	
3.	2021. 11. 19.	Az elkészült munka véleményezése	
4.	2021. 12. 03.	Végső pontosítások	

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat I. / Szakdolgozat II. (BSc) vagy Diplomamunka 1 / Diplo-mamunka 2 / Diplomamunka 3 / Diplomamunka 4[3] tantárgy követelményét teljesítette, beszámolóra / védésre[4] bocsátható.

Budapest, 2021. december 3.

intézményi konzulens

[1] Megfelelő aláhúzandó!
[2] Megfelelő aláhúzandó!
[3] Megfelelő aláhúzandó!
[4] Megfelelő aláhúzandó!

TARTALOMJEGYZÉK

I. BEVEZETÉS

Szakdolgozatomban egy olyan komplex informatikai háttérrendszert készítek el, amely a középiskolai felvételi eljárást hivatott megkönnyíteni.

Magyarországon a középiskolai felvételi eljárás kötelező elemeiben iskolatípustól függetlenül mindenhol megegyezik. Szakdolgozatom tárgyát, az informatikai háttérrendszert egy 5 évfolyamos budapesti technikum működésén keresztül mutatom be, ahol jelenleg oktatóként, munkaközösség-vezetőként, és rendszergazdaként dolgozom.

A középiskolai felvételi menete

A középiskolákba készülő tanulók első körben egy központi írásbeli feladatot oldanak meg egy általuk választott intézményben. Függetlenül attól, hogy négy, hat vagy nyolc osztályos középiskolai képzésre jelentkeznek, az írásbelin magyar nyelvből és matematikából kell számot adniuk a tudásukról. Mindkét tesztfeladaton 50-50 pontot szerezhetnek. [1]

Az írásbelit követheti egy szóbeli forduló. Ez nem kötelező eleme a felvételi eljárásnak, az adott iskola dönthet róla, hogy szeretne-e szóbeli felvételit is, vagy csak a központi írásbeli alapján rangsorolják a jelentkezőket. A szóbelivel a jelentkező plusz pontokat szerezhet az adott intézményben.

Miután előálltak a jelentkezők végleges pontszámai a hozott pontokból, az írásbeli pontjaiból és a szóbeli pontjaiból: az intézményben tagozatonként sorrendbe állítják őket, ez az ún. ideiglenes felvételi jegyzék.

Összeségében három ponton szeretném informatikai megoldásokkal támogatni a felvételi eljárást.

I. 1 Jelentkezés a szóbeli felvételire

A középiskolák a szóbeli előtt csak pár nappal kapják meg az intézménybe felvételizők listáját. Ez két problémát vet fel:

- Nem tudnak előre tervezni a szükséges erőforrásokkal, termek, tanárok és segítő diákok szempontjából.

- A nagyobb probléma az, hogy így a szóbeli beosztását az intézmény készíti el névsor szerint vagy véletlenszerűen. Ezzel a megoldással könnyen előállhat az a helyzet, hogy egy felvételizőnek egy időpontban két intézményben lesz a szóbeli felvételije.

Erre a problémára egy regisztrációs felületet szeretnék készíteni, ahol a jelentkezők ki tudják választani, hogy az adott napon mely időpontban szeretnének érkezni az intézménybe. Így elkerülhető lenne az időpontütközés, illetve az intézmény is látná előre a jelentkezők hozzávetőleges számát.

I. 2 Jelentkezők értékelése

Iskolánkban jelenleg a szóbeli felvételi alatt a jelentkezőknek számot kell adniuk magyar nyelv és matematika tudásukról, illetve részt kell venniük egy kötetlenebb beszélgetésen. Mind a három részt külön tanár bonyolítja le, majd a megszerzett pontszámokat egy papírlapra vezetik fel. Az adott napon lezajlott szóbeli felvételik után a tanárok leadják a papírlapon vezetett pontszámokat, amit később a tanulmányi osztály összesít. Többszáz jelentkező esetén ez nem kis feladat, és elég magas a hibázási lehetőség is.

Mivel a felvételizők már benne vannak a rendszerben, így ezt azzal szeretném segíteni, hogy a szóbeli elbeszélgetések alatt a kollégák egyenesen a programba tudják felvezetni az elért pontszámokat.

I. 3 Rangsor készítése

A szóbeli felvételi után a tanulmányi osztály a hozott pontokat, az írásbelin és szóbelin elért pontszámokat összesíti és tagozatonként rangsort készít.

Ez szintén elég monoton feladat, amelyet a rendelkezésre álló adatokból a szoftver könnyen elkészíthetne. Ehhez csak arra van szükség, hogy a hozott pontok és az írásbeli pontszámok felvételére is legyen lehetőség.

II. LEHETSÉGES MEGOLDÁSOK

II. 1 Létező megoldások

Az informatika világában a különböző részproblémák megoldására számtalan meglévő megoldás létezik már. Manapság már majdnem minden cég, így iskolánk is rendelkezik valamilyen online irodai szoftvercsomaggal, amely megoldásokat nyújt a különböző feladatokra.

Intézményünk jelenleg a Microsoft Office 365 SaaS (Software as a Service) megoldását használja. A következő alfejezetekben bemutatom, hogy ez a szoftver milyen megoldásokat nyújt a korábban felvetett problémákra.

II. 1.1 Időpontfoglalás

A Microsoft Forms program segítségével könnyedén készíthetünk űrlapokat, melyek segítségével információkat kérhetünk be a felhasználóktól. Ezzel a megoldással maga az adatbekérés egyszerűen megvalósítható lenne, és az adatokat Excel formátumban is megkapnánk. Sajnos azonban a jelenetezők időben egyenletes elosztását nem tudjuk vele biztosítani, mivel a különböző kitöltések között nem tud kapcsolatot teremteni a rendszer. Azt sem tudjuk vele biztosítani, hogy egy jelentkező csak egyszer regisztráljon.

II. 1.2 Szóbeli pontok rögzítése

A szóbeli alatt szerzett pontszámok rögzítését meg tudjuk oldani a Microsoft Excel programmal, melyben a munkafüzeteket a kollégák egyszerre tudják szerkeszteni. A kivitelezésre alapvetően két lehetőség van.

Első lehetőség, hogy minden jelentkezőt egy munkalapra teszünk. Ez a megoldás a későbbi feldolgozás szempontjából előnyös, viszont a szóbeliztető pedagógusoknak minden jelentkezőt meg kell keresniük.

A másik lehetőség, hogy tantermekként külön munkalapra tesszük a jelentkezőket behívásuk szerinti időrendben. Ez a módszer megkönnyíti az adatbevitelt, viszont jelentős plusz munkát ad az adatfeldolgozóknak.

Ezeken felül az Excel-es megoldás nagy hátránya, hogy az adatbevitel során a felhasználó egyszerre több adatsort is tud szerkeszteni, ezzel növelve a hibás adminisztráció esélyét.

II. 1.3 Rangsor készítése

Az előző feladatnál használt Excel tökéletesen alkalmas a rangsorok elkészítésére, viszont mivel ezt tagozatonként kell elkészíteni, így az eredményeket kézzel, vagy esetleg képletek segítségével tagozatonként külön-külön kell elkészíteni.

II. 2 WordPress használata

A Wordpress [2] egy 2003-ban létrehozott szoftver, amely eredetileg bejegyzések közzétételét volt hivatott leegyszerűsíteni. Tehát tökéletes megoldás arra, ha valaki írásban szeretne tartalmat megosztani az interneten. A 2000-es évek elején még kevésbé, aztán azonban egyre gyakoribbá vált blog oldalak létrehozása. Nem csak ilyen oldalak hoztak létre, hanem már meglévő weboldalakba építettek be blog aloldalakat.

Hatalmas előnye a Wordpressnek, hogy előre kódolt sablonokkal, mintákkal és bővítményekkel lehet dolgozni. Sok minden elérhető ingyenesen hozzá, azonban vannak olyan sablonok és bővítmények, amelyek elég költségesek tudnak lenni (csak amiatt, hogy egy-két funkció bele van építve). A sablonok adják a weboldal alapját, de nélkülözhetetlenek a bővítmények, amelyek kiegészítik a funkciókat és további beállítási lehetőségeket adnak hozzá a weboldalhoz. Tehát, ha nem tudunk olyan sablont választani, amely minden igényünket teljesítene, akkor megpróbálhatjuk elérni további bővítményekkel. Ennek azonban a hátránya az lehet, hogy akár 20-30 bővítmény is telepítésre kerülhet, ami nagy hatással van az oldal betöltési sebességére.

A Wordpress tökéletes megoldás egyszerűbb, kisebb oldalak létrehozására, azonban nem ajánlott, ha valami igazán összetett és gyorsan működő felület az igény. Óriási hátránya, pont abban rejtőzik, ami a könnyebbséget adja: az egyszerűen elkészíthető oldalakhoz az előre megírt kódok. Ha megnézzük egy sablon kódolását, rengeteg felesleges sort tartalmaz egy adott projekthez. Ezeket persze lehet tömöríteni, és átnézve törölni a nem szükségeseket, de rendkívül időigényes feladat. Lehet találni olyan sablont, ami kisebb méretű és már tömörített, például a GeneratePress. [3] Ugyanez a helyzet a bővítményekkel. Minden plusz bővítmény további felesleges kódokat ad hozzá a weblaphoz. Például, ha szeretnénk mérni a weboldal látogatottságát, akkor a Google Analytics egy kézenfekvő eszköz. Ahhoz, hogy beépítsük egy bővítményre lesz szükségünk, méghozzá a Google Tag Managerre. [4] Más alternatívák is léteznek természetesen, de egyik se az igazi megoldás, hiszen feleslegesen bővítjük az oldalt.

Más probléma az oldal betöltési sebességen kívül egy Wordpress oldalnál az, amikor eljön a frissítés ideje. A bővítmények és a sablon frissítés mellett a PHP verziót és magát a Wordpress core-t is frissíteni kell időről időre. Ez havi szintű elfoglaltságot és egy hozzáértő embert is igényel. Nem szabad mindig mindent csak úgy frissíteni, hiszen, ha sok bővítménnyel dolgozunk, akkor előfordulhat, hogy az újabb sablon verzióval nem

lesznek kompatibilisek. Ez szétesett oldalhoz vezethet, és sokszor nem is lehet egyszerűen, gyorsan megoldani. Ekkor a kiesett idő több óra is lehet. A másik véglet pedig, ha nem frissítünk hónapokig, ekkor könnyedén feltörés áldozatává válhatunk. Ez a legrosszabbkor is bekövetkezhet, ami szintén több órás kiesést eredményezhet.

Nézzük specifikusan, hogy hogyan lehetne felépíteni egy weboldalt Wordpressben, hogy elkészüljön a kívánt felületünk a felvételizők számára. Szükséges lesz egy sablon, amelyet biztosan módosítanunk is kell. Ezután pedig, azok az alap bővítmények, ami mondhatni minden oldalhoz szükségesek többek közt olyan területeken, mint:

- biztonság,
- spamek elleni védelem,
- szerepkörök,
- oldal optimalizálás, feltöltött adatok optimalizálása rendszeresen,
- biztonsági mentések készítése,
- adatok importálása/exportálása,
- keresőmotor optimalizálás,
- időpontfoglaló rendszer,
- regisztrációs és adatfelvivő felületekhez,
- weboldal felépítéséhez szükségesek,
- egyéb szükséges funkciókhoz.

Ezeket tehát biztosan mind bővítményekkel kellene megoldani, ami azt jelentené, hogy rengeteg bővítményre szükség van. Tehát valószínűsíthető, hogy fenn fog állni az oldal betöltési sebesség problémája. Ahhoz, hogy találjunk megoldást minden elképzelésre, biztosan fizetős bővítményekre lenne szükség, amelyek évenként több százezer forintba is kerülhetnek.

Összesítve: nem ajánlott egy olyan rendszerben felépíteni egy ilyen funkciókkal bíró felületet, amely nem arra való. Habár alkalmazhatunk olyan bővítménykombinációkat, ami megoldást adhatnak, de ezek egy kicsit zsákbamacskák is tudnak lenni. Ha olyat kell alkalmaznunk, amelyet kevesen használnak és nem elégedettek vele a felhasználók, akkor nehéz megbízni benne, hogy hosszútávon biztosan működni fog. Továbbá semmi sem garantálja, hogy úgy fog kinézni az oldal, ami nekünk szükséges. Ebben az esetben biztosan még CSS-ben is hozzá kell nyúlni.

II. 3 Saját szoftver fejlesztése

Mint láthatjuk, a problémákra külön-külön léteznek megoldások, de ezek nem a legtökéletesebbek, nem lehet őket egyéni igények szerint testreszabni, és nincs közöttük automatikus adatmozgás. Illetve léteznek univerzális rendszerek, de ezek használata körülményes, és nem is biztos, hogy elérhető vele a kívánt működés.

Ilyen speciális problémákra sokkal célravezetőbb egy saját szoftver készítése, mert így minden apró részletét úgy tudjuk alakítani, ahogy nekünk és a felhasználónak is hasznos.

II. 3.1 A szoftverrel szemben támasztott elvárások

A következőkben megvizsgálom, hogy milyen programozási megoldás a legalkalmasabb egy ilyen rendszer elkészítésére, de előbb bemutatom, hogy milyen tulajdonságokkal kell rendelkeznie:

- könnyen kezelhető webes felület;
- adatok közös adatbázisban tárolása;
- felhasználókezelés.

III. A PROGRAM FELÉPÍTÉSE

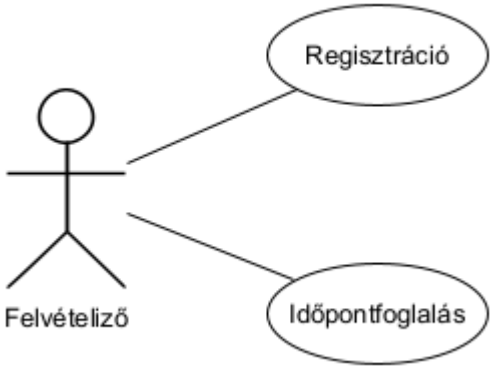
III. 1 Szerepkörök a programban

A következő alfejezetekben Use Case diagramok segítségével fogom bemutatni, hogy az öt szerepkör a rendszerben milyen funkcionalitást érhet el.

Ezek a szerepkörök:

- **Felvételiző:** a nyolcadikos diák, aki jelentkezik az iskolába és egyben szóbeli felvételire
- **Felvételiztető:** a tanár, aki a szóbeli felvételin vizsgáztatja a felvételizőt
- **Tanulmány Osztály:** az iskola Tanulmányi Osztály dolgozói
- **Vezetőség:** az iskola vezetése (igazgató és igazgatóhelyettesek)
- **Rendszergazda:** a rendszer fő adminisztrátora, aki üzemelteti a rendszert és miden paraméterét tudja állítani

III. 1.1 Felvételiző



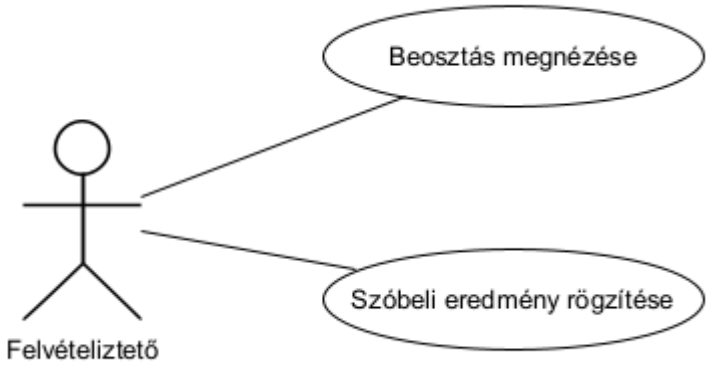
1. ábra
Felvételiző Use Case diagram

A webes felületen a nyolcadikos felvételiző adataik megadás után a rendelkezésre álló időpontok közül tudnak egyet választani.

Ez az aktor nem a klasszikus értelemben vett felhasználója a rendszernek. Regisztráció után nem jön létre új felhasználó, a továbbiakban nem tudja használni a rendszert. A program csak eltárolja az adatait.

Ha a regisztrált időponttal vagy az adatokkal bármi probléma lenne, akkor e-mailen keresztül kell felvenniük a kapcsolatot az iskolával a módosítás érdekében.

III. 1.2 Felvételiztető

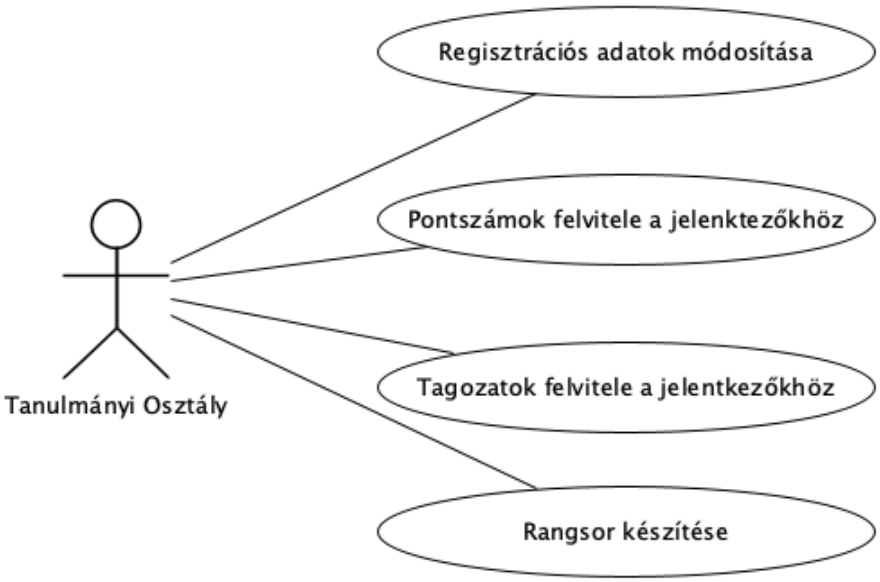


2. ábra
Felvételiztető Use Case diagram

A szóbeli felvételiztető tanárnak lehetősége van az adott terembe beosztottak listázására, és a hozzá tartozó szóbeli részen elért eredmények rögzítésére.

A felvételiztető tanárt, mint minden felhasználót a rendszerben, a rendszergazda hozza létre. A felhasználó a belépés után csak azokat a diákokat látja, akik a neki kiosztott terembe vannak beosztva. Itt rögzíteni tudja, hogy az adott diák nála (magyar, matek vagy elbeszélgetés) hány pontot ért el.

III. 1.3 Tanulmányi Osztály



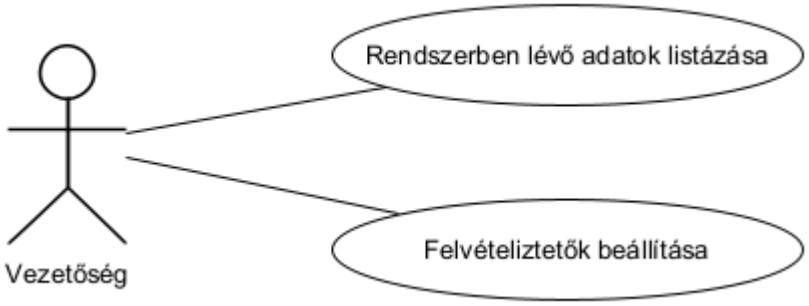
3. ábra
Tanulmány Osztály Use Case diagram

A Tanulmányi osztály:

- módosíthatja a regisztráltak adatait és időpontfoglalásait,
- felviheti a rendszerbe a jelentkezők hozott és az írásbelin elért pontjait,
- felviheti a rendszerbe, hogy a jelentkezők az iskolán belül melyik tagozatra vagy tagozatokra jelentkeztek,
- majd a felvételi folyamata végén, mikor már minden adat bekerült a rendszerbe, elkészítheti az ideiglenes rangsorokat tagozatonként.

A tanulmányi osztály felhasználóit szintén a rendszergazda rögzíti a rendszerbe. Bejelentkezés után lehetőségük van a fent felsorolt tevékenységek elvégzésére.

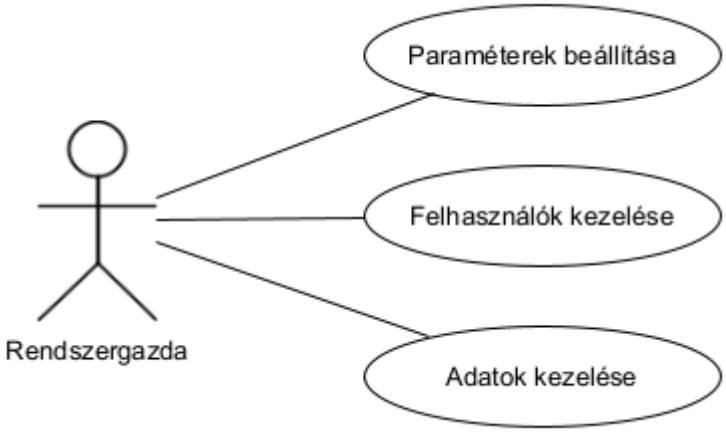
III. 1.4 Vezetőség



4. ábra
Vezetőség Use Case diagram

Az iskola vezetőségének lehetősége van a rendszerben éppen aktuálisan megtalálható adatokhoz való hozzáférésére. Mivel egy iskolában a elvégzendő feladatok kiosztása a vezetőség feladata, így a jelentkezők regisztrálásának lezártával hozzárendelhetik, hogy melyik felvételiztető tanár melyik teremben és milyen feladatkörben (magyar, matek és elbeszélgetés) vesz majd részt a szóbeli felvételi folyamán.

III. 1.5 Rendszergazda



5. ábra
Rendszergaza Use Case diagram

A rendszergazda felelős a program megfelelő működésért.

- Beállíthatja a program paramétereit:
    - Jelentkezési időszak

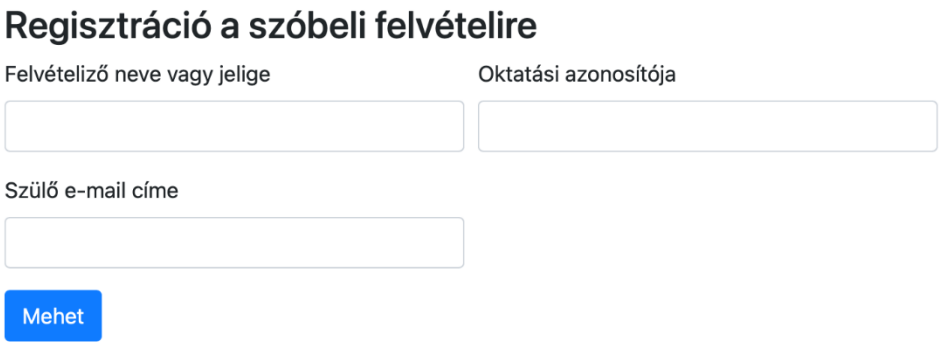
    - Jelentkezési felület üdvözlő üzenete
    - Tagozatok kezelése
    - Termek kezelése
    - Időpontok kezelése
- Kezelheti a felhasználókat
    - Létrehozás
    - Módosítás
    - Szerepkörök beállítása
- Az egyszerűbb segítségnyújtás érdekében a rendszergazda hozzáfér és módosíthat minden adatot a rendszerben.

A rendszer telepítésnél egy rendszergazda felhasználó jön létre. Ezzel a felhasználóval belépve van lehetőség további felhasználók felvételére. Mint láthatjuk ez a felhasználó a legfontosabb a rendszerben: minden adatot tud kezelni, amit a többi felhasználó, illetve a rendszer paraméterezését csak ő tudja elvégezni.

III. 2 A felhasználói felület

A következő alfejezetekben bemutatom látványtervek segítségével a program főbb felületinek elrendezését és az azokon megvalósítható funkciókat.

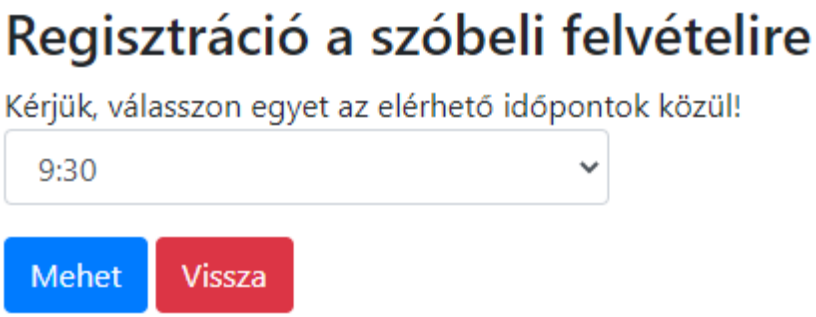
III. 2.1 Regisztrációs felület



6. ábra
Adatok megadása

A szóbeli felvételire regisztráló diákoknak első körben adataikat kell megadniuk. A regisztrációhoz szükséges adatok láthatóak a 6. ábra. A felvételiző azonosítására az oktatási azonosítóját használja a rendszer, de az egyszerűbb kezelés érdekében a felvételiző megadhatja a nevét, vagy egy jeligét (pont úgy, mint a hivatalos eljárás folyamanán). A szülő e-mail címét a regisztráció megerősítése és későbbiekben a szóbeli felvételivel kapcsolatban az esetleges plusz információk közlésére használja a rendszer. A felvételi lezártával az e-mail címek törlésre kerülnek.

Az adatok megadása után a rendszer felajánlja az éppen rendelkezésre álló időpontokat, és a jelentkező ezek közül tud választani.

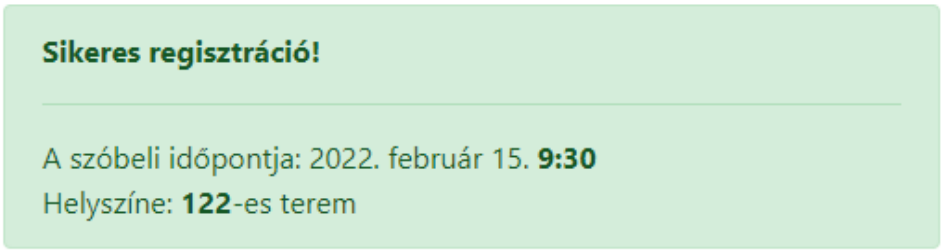


7. ábra
Időpont kiválasztása

Az adatok megadását és az időpont kiválasztását követően a regisztráció véglegesítése előtt a felvételizőknek a beírt adatokat a rendszer összesítve kiírja. Ez a felület látható a következő 8. ábra ábrán. Ha az adatok helyesek, akkor a *Regisztráció* gomb megnyomásával élesedik a regisztráció.

8. ábra
Adatok összesítése

Sikeres regisztrációt követően a rendszer azonnal és emlékeztető jelleggel e-mailben is tájékoztatja a felhasználót a szóbeli pontos időpontjáról, illetve a regisztrációhoz generált terem számáról, ahol majd meg kell jeleni.



9. ábra
Információ a sikeres regisztrációról

III. 2.2 Adminisztrációs felületek

III. 2.2.1 Jelentkezők listázása

Név	Okt. azon.	Terem	Időpont	
Teszt Elek	70123456789	122	9:30	✎
Nagy Péter	71234567890	122	10:00	✎

10. ábra
Jelentkezők listázása

A felvételire jelentkezettek listáját a fontosabb információkkal a program egy táblázat formájában mutatja az erre jogosult felhasználóknak. A tanulmányi osztálynak és a rendszergazdának minden sor mellett egy szerkesztés ikon is megjelenik. Erre kattintva egy űrlapon a jelentkező minden adata megjelenik. Itt lehetőség van az adatok szükségszerű módosítására.

III. 2.2.2 Szóbeli pontok rögzítése

Szóbeli felvételi
matematika — 122-es terem

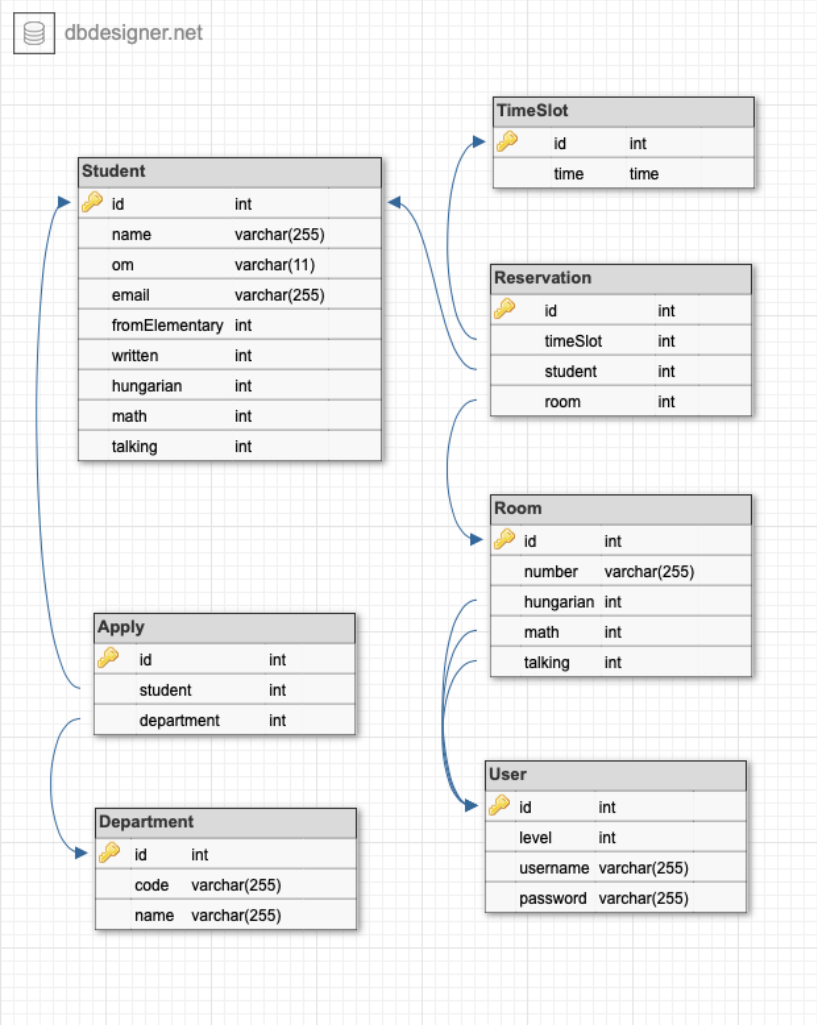
Felvételiző	Időpont	
Teszt Elek 70123456789	9:30	✓
Nagy Péter 71234567890	10:00	✎

11. ábra
Szóbeli pontok rögzítése

A szóbeli felvételi közben a felvételiztető tanár a 11. ábrán látható felület fogadja. Itt csak azok a felvételizők jelennek meg, akik abba terembe érkeznek, ahol ő van. A felvételizők melletti szerkesztés gombbal tud pontszámot rögzíteni az őhozzá rendelt felvételi részhez. A pontszám rögzítése után a szerkesztés ikon átalakul pipává, ezzel jelezve, hogy az adott felvételiző már kapott pontot.

III. 2.3 Az adatbázis felépítése

A program az adatokat MySQL relációs adatbázisban fogja tárolni. Azért esett a választásom erre a megoldásra, mert ez egy elterjedt megoldás, így sok platformra elérhető, illetve, hogyha a rendszert egy webhosting megoldással szeretnék üzemeltetni, akkor biztosan támogatott lesz.



12. ábra
Az adatbázis felépítése

User tábla

Ebben a táblában tárolódnak a rendszer belépésre jogosult felhasználó.

- **id:** a felhasználó azonosítója (elsődleges kulcs)
- **level:** a felhasználó jogusultsági szintje, azaz, hogy milyen szerepkörű felhasználóról van szó
- **username:** a felhasználó belejelentkezési neve
- **password:** a felhasználó jelszava hash-elve

Room tábla

Ebben a táblában tárolódnak a termek ahová a jelentkező diákokat a rendszer be tudja osztani.

- **id:** a terem egyedi azonosítója (elsődleges kulcs)
- **number:** a terem valóságos azonosítója (egyedi érték)
- **hungarian:** felvételiztető tanár magyarból az adott teremben (idegen kulcs)
- **math:** felvételiztető tanár matematikából az adott teremben (idegen kulcs)
- **talking:** az elbeszélgető felvételiztető tanár az adott teremben (idegen kulcs)

A három idegen kulcs alapértelmezett értéke NULL, ezzel jelezve, hogy még nincsenek beállítva.

TimeSlot tábla

Ebben a táblában tárolja a rendszer, hogy a felvételiző diákok milyen időpontok közül tudnak választani.

- **id:** az időpont egyedi azonosítója (elsődleges kulcs)
- **time:** az időpont

Reservation tábla

Ebben a kapcsolótáblában tárolódnak a konkrét időpont foglalások, hogy egy adott diák melyik terembe és milyen időpontba jön felvételizni.

- **id:** a foglalás azonosítója (elsődleges kulcs)
- **timeSlot:** a fogaláshoz tartozó időpont (idegen kulcs)
- **student:** a foglaláshoz tartozó diák (idegen kulcs)
- **room:** a foglaláshoz tartozó terem (idegen kulcs)

Student tábla

Ebbe a táblába kerülnek a jelentkező diákok, illetve majd később a felvételi folyamán itt tárolódnak a pontszámaik is.

- **id:** a diák egyedi azonosítója a rendszerben (elsődleges kulcs)
- **name:** a diák neve
- **om:** a diák 11 jegyű oktatási azonosítója (egyedi)
- **email:** e-mail cím, amin keresztül az iskola kapcsolatba tud lépni a szülővel
- **fromElementary:** hozott pontok
- **written:** az írásbelin elért pontszám
- **hungarian:** a magyar szóbelin elért pontszám
- **math:** a matek szóbelin elért pontszám
- **talikng:** a szóbeli elbeszélgetésen elért pontszám

A pontszámok alapértelmezett értéke NULL, ezzel jelezve, hogy az adott adat még nincs kitöltve.

Department tábla

Ebben a táblában tárolónak az iskolában meghirdetésre került tagozatok.

- **id:** a tagozat rendszerbeli azonosítója (elsődleges kulcs)
- **code:** a tagozat kódja (egyedi)
- **name:** a tagozat megnevezése

Appy tábla

Ebben a kapcsolótáblában tárolja rendszer, hogy melyik diák milyen tagozatokra jelentkezett az iskolába.

- **id:** a jelentkezés egyedi azonosítója (elsődleges kulcs)
- **student:** a jelentkezéshez tartozó tanuló (idegen kulcs)
- **department:** a jelentkezéshez tartozó tagozat (idegen kulcs)

Sok táblában van olyan egyedi oszlop, amely technikailag megfelelne elsődleges kulcsnak, de a könnyebb módosítás érdekében nem ezeket használom, hanem minden táblánál van egy rendszerbeli automatikus azonosító oszlop.

Az előbb felsorolt táblákon kívül a rendszer a működéshez szükséges paramétereket szintén az adatbázisban fogja tárolni.

III. 3 A rendszer megvalósításához választott technológia

Mint az adatbázis kiválasztásánál, itt is fontos szempont, hogy egy olyan technológiát használjak, ami széleskörben támogatott. Így esett a választás a PHP nyelvre. A PHP-t és a MySQL-t használó rendszereket szinte minden webhosting megoldás támogat, illetve, ha saját magunk (on-premises) szeretnénk futtatni, akkor minden operációsrendszeren kis erőforrással megoldhatjuk.

IV. A RENDSZER MEGVALÓSÍTÁSA

A következő alfejezetekben található programkódok nem minden esetben tartalmazzák a tényleges fájlok teljes tartalmát, – például hivatkozás más fájlokra, fejléc sorok – illetve a mappastruktúráról sem esik részletesen szó.

A PHP kódok a 7.3-as verzióhoz készültek, amely jelen szakdolgozat írásakor a legelterjedtebb mind hosting megoldások, mind saját üzemeltetésű linux operációs rendszerek esetén.

A rendszert egy Raspberry Pi 2 B modellen fejlesztettem, teszteltem. Egyrészt a Raspberry Pi OS operációs rendszerre szintén a 7.3-as PHP a legfrissebb elérhető verzió, másrészt jól látható, hogy milyen kevés erőforrás elég a rendszer futtatásához. A tesztelésről még részletesebben szó fog esni később.

IV. 1 Az adatbázis létrehozása

Első lépésként az adatbázist kell kialakítani, ahol a rendszer az adatokat fogja tárolni. A következő MySQL szkriptek létrehozzák az *Adatbázis felépítése* című fejezetben részletezett táblákat.

IV. 1.1 User tábla

Első lépésként a User táblát hozom létre. Itt is, mint majd minden következő táblánál beállítom, hogy az elsődleges kulcs automatikusan növekedjen minden egyes rekord beszúrásánál. Így egy új rekord beszúrásánál nem kell foglalkoznunk az egyedi azonosító (*id*) oszlop érekével, ezt az adatbázis magától kezelni fogja.

Szintén minden tábla esetén beállításra fog kerülni, hogy az oszlopok értéke nem lehet üres (not null). Ez alól kivétel lesz, mikor a NULL értéknek külön jelentése van.

Mivel a legtöbb esetben tanár felhasználót fogunk létrehozni a rendszerben, így a *level* (felhasználói szint) oszlop értékét alapértelmezetten 1-re állítom. Ezzel a megoldással lehetőségünk lesz úgy létrehozni egy tanár szintű felhasználót az adatbázisban, hogy csak a felhasználónevét és a jelszavának a hash értékét adjuk meg.

A felhasználók a rendszerbe felhasználónevük és jelszavuk megadásával fognak bejelentkezni, így elengedhetetlen, hogy egy felhasználónév csak egyszer szerepelhessen az adatbázisban. Erről gondoskodik a unique kényszer.

```
CREATE TABLE User (
    `id` INT AUTO_INCREMENT PRIMARY KEY,
    `level` INT NOT NULL DEFAULT 1,
```

```
    `username` VARCHAR(255) NOT NULL
UNIQUE,
    `password` VARCHAR(255) NOT NULL
);
```

1. kódrészlet

IV. 1.2 Department tábla

```
CREATE TABLE `Department` (
    `id` INT AUTO_INCREMENT PRIMARY KEY,
    `code` VARCHAR(255) NOT NULL,
    `name` VARCHAR(255) NOT NULL
);
```

2. kódrészlet

IV. 1.3 Student tábla

Ennél a táblánál a **fromElementary**, **written**, **hungarian**, **math**, és **talking** oszlopok esetén a NULL érték azt jelenti, hogy az az adat még nincs megadva. Ennek értelmében ezen oszlopok nem rendelkeznek a NOT NULL kényszerrel. Mivel ezek az adatok később kerülnek megadásra – vagyis nem akkor, mikor létre jön a diák az adatbázisban – így ezen oszlopok alapértelmezett értéke NULL, hogy ne kelljen feleslegesen megadni őket az új rekordok beszúrásakor.

```
CREATE TABLE `Student` (
    `id` INT AUTO_INCREMENT PRIMARY KEY,
    `name` VARCHAR(255) NOT NULL,
    `om` VARCHAR(11) NOT NULL,
    `email` VARCHAR(255) NOT NULL,
    `fromElementary` INT DEFAULT NULL,
    `written` INT DEFAULT NULL,
    `hungarian` INT DEFAULT NULL,
    `math` INT DEFAULT NULL,
    `talking` INT DEFAULT NULL
);
```

3. kódrészlet

IV. 1.4 Apply tábla

Ebben a kapcsolótáblában lesznek tárolva, hogy egy diák milyen szakokra jelentkezett. Azért van szükség a kapcsolótáblára, mert egy diák több szakra is jelentkezhet. A táblának a rangsorok létrehozásakor lesz jelentősége, hiszen ezek szakoként fog történni.

Az adatbázis konzisztenciája és a véletlen adattörlések elkerülése végett beállításra kerülnek az idegen kulcsok.

```
CREATE TABLE `Apply` (
    `id` INT AUTO_INCREMENT PRIMARY KEY,
    `student` INT NOT NULL,
    `department` INT NOT NULL,
    FOREIGN KEY (`student`) REFERENCES
`Student`(`id`),
    FOREIGN KEY (`department`) REFERENCES
`Department`(`id`)
);
```

4. kódrészlet

IV. 1.5 TimeSlot tábla

```
CREATE TABLE `TimeSlot` (
    `id` INT AUTO_INCREMENT PRIMARY KEY,
    `time` TIME NOT NULL
);
```

5. kódrészlet

IV. 1.6 Room tábla

Ennél a táblánál is beállításra kerülnek az idegen kulcsok, melyek alapértelmezett értékei NULL, mivel a szoba létrehozásakor még nem állítjuk be, hogy mely felhasználók fognak abban a teremben felvételiztetni.

A *number* (a terem száma/neve) oszlop kap egy *unique* kényszert, hiszem a valóságban az emberek ez alapján fogják megtalálni a termet, így nem szeretnénk, hogy ha két azonos elnevezésű lenne.

```
CREATE TABLE `Room` (
    `id` INT AUTO_INCREMENT PRIMARY KEY,
    `number` VARCHAR(255) NOT NULL UNIQUE,
```

```
    `hungarian` INT DEFAULT NULL,
    `math` INT DEFAULT NULL,
    `talking` INT DEFAULT NULL,
    FOREIGN KEY (`hungarian`) REFERENCES
`User`(`id`),
    FOREIGN KEY (`math`) REFERENCES
`User`(`id`),
    FOREIGN KEY (`talking`) REFERENCES
`User`(`id`)
);
```

6. kódrészlet

IV. 1.7 Reservation tábla

Ebben a táblában fog eltárolódni, hogy melyik diák mikor és melyik teremben fog felvételizni. Ennek értelmében itt is beállításra kerülnek az idegen kulcsok.

A tervezési fázistól elérően ebben a táblában helyet kap még egy *timeStamp* (időbélyeg) nevű oszlop, amelynek az alapértelmezett értéke az aktuális időpont, így elmentve az időpont regisztráció pontos idejét. Ez az oszlop hasznos lehet későbbi hibakeresések esetén.

```
CREATE TABLE `Reservation` (
    `id` INT AUTO_INCREMENT PRIMARY KEY,
    `timeSlot` INT NOT NULL,
    `student` INT NOT NULL,
    `room` INT NOT NULL,
    `timeStamp` DATETIME DEFAULT NOW(),
    FOREIGN KEY (`timeSlot`) REFERENCES
`TimeSlot`(`id`),
    FOREIGN KEY (`student`) REFERENCES
`Student`(`id`),
    FOREIGN KEY (`room`) REFERENCES
`Room`(`id`)
);
```

7. kódrészlet

IV. 2 Az üzleti logika

Az adatbázis kezelését a PHP-s alkalmazásban osztályok segítségével fogom megvalósítani. A program bonyolultsági szintje nem követeli meg, hogy egy ORM

(objektum-relációs leképező) keretrendszert építsek be. Így a programban nem lesz felesleges többletmunka.

Alapvetően minden táblához fog tartozni egy PHP osztály, amivel lehet kezelni az adatbázist. Bizonyos esetekben lesznek eltérések, mivel az egyszerűség jegyében igyekeztem minden üzleti folyamatot egy metódussal lefedni. Ennek értelmében a program által nem használt adatbázis műveletek nincsenek megvalósítva.

A HTTP protokollból adódóan ez az alkalmazás kapcsolatmenetes, – azaz miután a webszerver legenerálta az adott oldalt, megszűnik a kapcsolat a felhasználóval, majd újra létrejön egy másik – így az osztályokban található metódusok nagy része statikus.

IV. 2.1 Adatbázis kapcsolat létrehozása

Az adatbázis kapcsolat létrehozásáért két osztály felelős.

Az első osztály a **Settings** osztály, amelyben az adatbázis kapcsolat paraméteri (szerver címe, adatbázis neve, felhasználónév, jelszó és karakterkódolás) vannak eltárolva. Ezeket az értékeket mindig az adott környezethez (adatbázis szerverhez) kell igazítani.

```
class Settings
{
    public static $server = 'localhost';
    public static $database = 'felveteli';
    public static $username = 'felveteli';
    public static $password = 'felveteli';
    public static $charset = 'utf8mb4';
}
```

8. kódrészlet

A **Database** osztályban található **GetDatabase** statikus metódus felelős az adatbáziskapcsolat tényleges létrehozásáért. A **Settings** osztályban található paramétereket felhasználva létrehozza az adatbáziskapcsolatot és visszatérési értékként visszaadja a kapcsolat objektumát. Kapcsolódási hiba esetén megállítja a kód további futását és kiírja a hibát.

```
class Database {
    public static function GetDatabase()
    {
        $conn = new
mysqli(Settings::$server,
```

```
Settings::$username, Settings::$password,
Settings::$database);
        $conn-
>set_charset(Settings::$charset);

        if($conn->connect_errno) {
            die("Couldn't connect to
database!\n" . $conn->error);
        }
        return $conn;
    }
}
```

9. kódrészlet

IV. 2.2 Táblákat kezelő osztályok

A következő alfejezetekben bemutatom a táblákat kezelő osztályokat. Minden osztály rendelkezik a táblákban található oszlopnévvel megegyező nevű publikus adattagokkal. Idegen kulcsok esetén általában az adott azonosítóhoz tartozó osztálypéldányt tartalmazzák.

Az adatbázis lekérdezések esetén a prepared statementet használom az SQL injection megelőzése végett.

IV. 2.2.1 User osztály

Ez az osztály felelős a felhasználók kezelésére. Lehetőség van hitelesíteni, lekérni, menteni és törölni felhasználót.

Authenticate metódus

Ennek a metódusnak a feladata, hogy a paraméterül adott felhasználónév és jelszó hash-t felhasználva visszaadja az adott *User* objektumát. Ha a paraméterek hibásak – azaz nincs ilyen felhasználó, vagy hibás a jelszó – akkor a lekérdezés nem egy darab sorral fog visszatérni, így a metódus null értéket ad vissza. Ellenkező esetben létrehoz egy új *User* objektumot, beállítja az adattagjait és azt adja vissza.

```
public static function
Authenticate($userName, $password)
{
    $conn = Database::GetDatabase();
```

```
    $stmt = $conn->prepare("SELECT id,
level FROM User WHERE username = ? AND
password = ?");
    $stmt->bind_param("ss", $userName,
$password);
    $stmt->execute();
    $result = $stmt->get_result();
    if ($result->num_rows != 1) {
        return null;
    }
    $user = new User();
    $user->user = $userName;
    $userRecord = $result->fetch_array();
    $user->id = $userRecord[0];
    $user->level = $userRecord[1];
    return $user;
}
```

10. kódrészlet

GetUserById metódus

Működése szinte megegyezik az előző metóduséval, viszont a feladata egy adott azonosítójú felhasználó megkeresése. Találat hiánya esetén szintén null értékkel tér vissza.

```
public static function GetUserById($id) {
    $conn = Database::GetDatabase();
    $stmt = $conn->prepare("SELECT
username, level, password FROM User WHERE
id = ?");
    $stmt->bind_param("i", $id);
    $stmt->execute();
    $result = $stmt->get_result();
    if ($result->num_rows != 1) {
        return null;
    }
    $user = new User();
    $userRecord = $result->fetch_array();
    $user->id = $id;
    $user->user = $userRecord[0];
    $user->level = $userRecord[1];
    $user->password = $userRecord[2];
```

```
        return $user;
    }
```

11. kódrészlet

GetAllUsers metódus

Ez a metódus egy tömbben visszaadja a táblában található összes felhasználót.

```
    public static function GetAllUsers() {
        $conn = Database::GetDatabase();
        $stmt = $conn->prepare("SELECT * FROM
    User");
        $stmt->execute();
        $result = $stmt->get_result();
        $users = array();
        while ($userRecord = $result-
    >fetch_assoc()) {
            $user = new User();
            $user->id = $userRecord['id'];
            $user->user =
    $userRecord['username'];
            $user->level =
    $userRecord['level'];
            $user->password =
    $userRecord['password'];
            $users[] = $user;
        }
        return $users;
    }
```

12. kódrészlet

SaveUser metódus

Ennek a metódusnak a feladata kettős. Új felhasználó esetén (amikor az *id* adattag értéke null) a felhasználót beszúrja az adatbázisba, az objektum azonosítóját beállítja és annak értékét visszaadja. Meglévő felhasználó esetén az adott azonosítóhoz tartozó rekordot frissíti az adatbázisban.

A többi osztályban is lesz hasonló metódus, ott azokat nem mutatom be, működésük hasonló, csak az SQL utasításban és a paraméterek hozzárendelésében térnek el.

```
    public function SaveUser() {
```

```
    if ($this->id == null) {
        $conn = Database::GetDatabase();
        $stmt = $conn->prepare("INSERT INTO
User (level, username, password) VALUES
(?,?,?)");
        $stmt->bind_param("iss", $this-
>level, $this->user, $this->password);
        $stmt->execute();
        $id = $stmt->insert_id;
        return $stmt->insert_id;
    }
    else {
        $conn = Database::GetDatabase();
        $stmt = $conn->prepare("UPDATE User
SET password = ?, level = ? WHERE id = ?");
        $stmt->bind_param("sii", $this-
>password, $this->level, $this->id);
        $stmt->execute();
        return $id;
    }
}
```

13. kódrészlet

DeleteUser metódus

A paraméterül adott azonosítóhoz tartozó felhasználó rekordját törli az adatbázisból. Visszatérési értékként visszaadja, hogy az érintett sorok száma egy e, azaz sikeres volt-e a törlés.

A továbbiakban más osztályoknál a hasonló működésű metóduskat nem mutatom be – a metódus nevében és tábla nevében különböznek.

```
public function DeleteUser() {
    $conn = Database::GetDatabase();
    $stmt = $conn->prepare("DELETE FROM
User WHERE id = ?");
    $stmt->bind_param("i", $this->id);
    $stmt->execute();
    return $stmt->num_rows == 1;
}
```

14. kódrészlet

IV. 2.2.2 TimeSlot osztály

Ez az osztály az időpontokat kezeli az adatbázisban. Lehetőség van létrehozásra, törlésre, lekérdezésre és szabad időpont keresésére.

GetTimeSlotFromRow metódus

Mivel a lekérdezések eredményének feldolgozása során sokszor kell egy asszociatív tömbből (olyan tömb, ahol nevesítve vannak az indexek) létrehozni egy objektumpéldányt, így ezt a feladatot egy külön privát metódusba kiszerveztem. A metódus paraméterül megkapja az adatbázis rekordot, létrehoz egy objektumpéldányt, beállítja az adattagjait, majd az objektumot visszaadja.

Ez a fajta metódus más osztályokban is, hasonló néven előfordul. Működésük megegyezik, csupán az tömb méretében térnek el, így azokat külön nem fogom többször bemutatni.

```
static function GetTimeSlotFromRow($row) {
    $timeSlot = new TimeSlot();
    $timeSlot->id = $row['id'];
    $timeSlot->time = $row['time'];
    return $timeSlot;
}
```

15. kódrészlet

GetTimeSlots metódus

Lekérdezi az összes időpontot és visszaadja azokat egy tömbben – hasonlóan, mint az **User** osztály megfelelő metódusa. Azért mutatom meg itt is, mert az előző **GetTimeSlotFromRow** metódust használja. A továbbiakban a hasonló működésű metódusokat más osztályban nem fogom bemutatni.

```
public static function GetTimeSlots() {
    $conn = Database::GetDatabase();
    $stmt = $conn->prepare("SELECT * FROM
TimeSlot");
    $stmt->execute();
    $result = $stmt->get_result();

    $timeSlots = array();
    while ($row = $result->fetch_assoc()) {
```

```
            $timeSlots[] =
self::GetTimeSlotFromRow($row);
        }
        return $timeSlots;
}
```

16. kódrészlet

TimeSlotById metódus

Visszaad egy adott azonosítóhoz tartozó időpont objektumot.

Az előző metódushoz hasonlóan csak azért kerül megint bemutatásra, mert a **GetTimeSlotFromRow** metódust használja. A továbbiakban szintén nem lesz bemutatva.

```
public static function TimeSlotById($id) {
    $conn = Database::GetDatabase();
    $stmt = $conn->prepare("SELECT * FROM
TimeSlot WHERE id = ?");
    $stmt->bind_param("i", $id);
    $stmt->execute();
    $result = $stmt->get_result();
    if($result->num_rows != 1) {
        return null;
    }
    $record = $result->fetch_assoc();
    return
self::GetTimeSlotFromRow($record);
}
```

17. kódrészlet

GetFreeTimeSlots metódus

A metódus visszaadja azokat az időpontokat, ahol még van szabad időpont. A szabad időpontok keresése adatbázis szinten történik meg.

A lekérdezés úgy működik, hogy egy allekérdezésben megszámolja a termek számát (ennyi foglalás lehet maximum egy időpontra), majd szintén egy allekérdezésben kiválasztjuk azokat az időpont azonosítókat a foglalások közül, ahol (időpontokként csoportosítva) a foglalások száma legalább a lehetséges maximum foglalások száma. Így megkapjuk azokat az időpontokat, amelyek beteltek. Végül kiválasztjuk az összes időpontot, amely nincs benne a beteltek között.

```
public static function GetFreeTimeSlots() {
    $conn = Database::GetDatabase();
    $stmt = $conn->prepare("SELECT * FROM
TimeSlot
            WHERE id NOT IN (
                SELECT timeSlot FROM
Reservation GROUP BY timeSlot
                    HAVING COUNT(id) >=
(SELECT COUNT(id) FROM Room))");
    $stmt->execute();
    $result = $stmt->get_result();

    $timeSlots = array();
    while ($row = $result->fetch_assoc()) {
        $timeSlots[] =
self::GetTimeSlotFromRow($row);
    }
    return $timeSlots;
}
```

18. kódrészlet

Az osztály ezeken kívül rendelkezik **SaveToDatabase** és **DeleteById** publikus metódusokkal.

IV. 2.2.3 Student osztály

Ez az osztály felelős a diákok kezelésért, illetve az objektumot, de más táblákat is érintő egyéb folyamatokért.

GetDepartmentsByStudentId metódus

Ez a metódus azokat a szakokat fogja egy tömbben visszaadni, ahova az adott azonosítójú diák jelentkezett.

A lekérdezésben INNER JOIN segítségével az Apply és a Depatment táblát összekapcsoljuk és leszűrjük az adott diák azonosítóra, kód alapján rendezzük, majd ebből létrehozzuk a tömböt és azt visszaadjuk.

```
public static function
GetDepartmentsByStudentId($id) {
    $conn = Database::GetDatabase();
```

```
    $stmt = $conn->prepare("SELECT * FROM
Apply INNER JOIN Department D on
Apply.department = D.id WHERE student = ?
ORDER BY D.code");
    $stmt->bind_param("i", $id);
    $stmt->execute();
    $result = $stmt->get_result();
    $departments = array();
    while ($row = $result->fetch_assoc()) {
        $department = new Department();
        $department->id =
$row['department'];
        $department->code = $row['code'];
        $department->name = $row['name'];
        $departments[] = $department;
    }
    return $departments;
}
```

19. kódrészlet

UpdateDepartmentsByStudentId metódus

Ez a metódus a diákhoz tartozó szakokat frissíti az adatbázisban.

Első lépésként a paraméterül kapott azonosítóhoz tartozó szakokat törli a kapcsolótáblából, majd a második paraméterként kapott tömbben található szak azonosítókhoz rögzít jelentkezéseket.

```
public static function
UpdateDepartmentsByStudentId($id,
$departments) {
    $conn = Database::GetDatabase();
    $stmt = $conn->prepare("DELETE FROM
Apply WHERE student = ?");
    $stmt->bind_param("i", $id);
    $stmt->execute();
    foreach ($departments as $department) {
        $stmt = $conn->prepare("INSERT INTO
Apply (student, department) VALUES (? ,
?)");
        $stmt->bind_param("ii", $id,
$department);
```

```
            $stmt->execute();
        }
    }
```

20. kódrészlet

IsPointFilled metódus

A metódus visszaadja, hogy a paraméterül adott témakörben a diáknak van-e már pontszáma.

```
    public function IsPointFilled($topic) {
        switch ($topic) {
            case "hungarian": return $this-
    >hungarian != null;
            case "math": return $this->math !=
    null;
            case "talking": return $this-
    >talking != null;
            default: return false;
        }
    }
```

21. kódrészlet

GetPointAtTopic metódus

A metódus visszaadja, hogy a paraméterül adott témakörben a diáknak mennyi pontja van.

```
    public function GetPointAtTopic($topic) {
        switch ($topic) {
            case "hungarian": return $this-
    >hungarian;
            case "math": return $this->math;
            case "talking": return $this-
    >talking;
            default: return null;
        }
    }
```

22. kódrészlet

GetAllPoints metódus

A metódus visszaadja, hogy a diáknak összesen mennyi pontja van.

Az összpontszám a hozott, az írásbeli és a három szóbeli témakör pontszámának összege. Ha valamelyik pontszám még nincs megadva, akkor a metódus 0 értékkel tér vissza, ezzel jelezve, hogy még nincs meg minden szükséges adat.

```
public function GetAllPoints() {
    if ($this->fromElementary == null ||
        $this->written == null ||
        $this->hungarian == null ||
        $this->math == null ||
        $this->talking == null) {
        return 0;
    }
    return $this->fromElementary + $this-
>written + $this->hungarian + $this->math +
$this->talking;
}
```

23. kódrészlet

Compare metódus

Ez a segéd metódus a rangsor készítésekor lesz hasznos. A paraméterül adott két diák összpontszáma alapján visszaad:

- nullát, ha egyenlők a pontok,
- egyet, ha második diáknak több a pontja
- és mínusz egyet, ha az első diáknak több a pontja.

```
public static function Compare($a, $b) {
    if ($a->GetAllPoints() == $b-
>GetAllPoints()) {
        return 0;
    }
    return ($a->GetAllPoints() > $b-
>GetAllPoints()) ? -1 : 1;
}
```

24. kódrészlet

Az osztály ezeken kívül rendelkezik **SaveToDatabase**, **GetStudentById**, **GetStudentById** és **RemoveFromDatabase** publikus metódusokkal.

IV. 2.2.4 Room osztály

GetRandomFreeRoomAtTimeSlot metódus

A metódus feladata, hogy a paraméterül adott időpont azonosítóhoz tartozó időpontban megadjon egy véletlenszerű termet. Ez azért fontos, mert a szóbelire jelentkező diákok csak az elérhető időpontok közül tudnak választani. Ez után a program magától osztja be őket a termekbe. Ezzel a metódussal biztosítható, hogy a terem kiválasztás esetén az összes terem lehetőség szerint körülbelül azonos terheléssel legyen lefedve.

A lekérdezésben először egy allekérdezés segítségével lekérjük az adott időpontban a foglalásokhoz tartozó terem azonosítókat, majd a termek közül lekérjük azoknak a termeknek az azonsítóját melyek nem szerepelnek az allekérdezésben – azaz a kérdéses időpontban szabadok.

Ha a lekérdezés nulla sort ad vissza akkor a metódus null értékkel visszatér, jelezve, hogy nincs több szabad terem az időpontban.

Ezután az azonosítókat a lekérdezésbe kigyűjtjük egy tömbbe, majd innen egy véletlenszerű elemet kiválasztva létrehozzuk az azonosítóhoz tartozó terem objektumát és azt visszaadjuk.

```
public static function
GetRandomFreeRoomAtTimeSlot($timeSlot) {
    $conn = Database::GetDatabase();
    $stmt = $conn->prepare("SELECT id FROM
Room WHERE id NOT IN (SELECT room FROM
Reservation WHERE timeSlot = ?)");
    $stmt->bind_param("i", $timeSlot);
    $stmt->execute();
    $result = $stmt->get_result();
    if ($result->num_rows == 0) {
        return null;
    }
    $freeRoomIds = array();
    while ($row = $result->fetch_assoc()) {
        $freeRoomIds[] = $row['id'];
    }
    $randomId = $freeRoomIds[rand(0,
count($freeRoomIds) - 1)];
```

```
        return self::GetRoomById($randomId);
    }
```

25. kódrészlet

GetRoomsAndTopicsByUserId metódus

A metódus egy tömbben visszaadja, hogy a paraméterül megadott azonosítójú felhasználó mely termekbe és témakörökbe van beosztva.

A lekérdezés lekéri azokat a terem azonosítókat, ahol a felhasználó a három szóbeli témakör egyikénél is szerepel.

Ez után megvizsgáljuk a lekérdezett sorokat. Ha valamelyik oszlopban szerepel a felhasználó, akkor hozzáadjuk a tömbhöz egy tömb formájában, hogy melyik terem, milyen témakörénél szerepel.

```
public static function
GetRoomsAndTopicsByUserId($id) {
    $roomsAndTopics = array();
    $conn = Database::GetDatabase();
    $stmt = $conn->prepare("SELECT * FROM
Room WHERE hungarian = ? OR math = ? OR
talking = ?");
    $stmt->bind_param("iii", $id, $id,
$id);
    $stmt->execute();
    $result = $stmt->get_result();
    while ($row = $result->fetch_assoc()) {
        if ($row['hungarian'] == $id) {
            $roomsAndTopics[] =
array("room" => $row['id'], "topic" =>
"hungarian");
        }
        if ($row['math'] == $id) {
            $roomsAndTopics[] =
array("room" => $row['id'], "topic" =>
"math");
        }
        if ($row['talking'] == $id) {
            $roomsAndTopics[] =
array("room" => $row['id'], "topic" =>
"talking");
```

```
            }
        }
        return $roomsAndTopics;
    }
```

26. kódrészlet

Az osztály ezen kívül rendelkezik **GetRoomById, GetAllRooms, SaveToDatabase** és **DeleteById** metódusokkal.

IV. 2.2.5 Reservation osztály

Place metódus

A metódus feladata az időpont foglalás rögzítése az adatbázisba és a terem hozzárendelése.

A paraméterül kapott diáknak rögzít egy időpontot a paraméterül kapott időpontban egy az adott időben szabad terembe. Visszatérési értékként visszaadja a kisorsolt terem azonosítóját.

```
public static function Place($timeSlot,
$student) {
    $conn = Database::GetDatabase();
    $stmt = $conn->prepare("INSERT INTO
Reservation (timeSlot, student, room)
VALUES (?, ?, ?)");
    $room =
Room::GetRandomFreeRoomAtTimeSlot($timeSlot
);
    $stmt->bind_param("iii", $timeSlot,
$student, $room->id);
    $stmt->execute();
    return $room;
}
```

27. kódrészlet

GetReservationsAtRoom metódus

A metódus visszaadja tömbként a paraméterül adott szobához tartozó foglalásokat időrendben.

```
public static function
GetReservationsAtRoom($roomId) {
    $conn = Database::GetDatabase();
    $stmt = $conn->prepare("SELECT * FROM
Reservation WHERE room = ? ORDER BY
timeSlot");
    $stmt->bind_param("i", $roomId);
    $stmt->execute();
    $result = $stmt->get_result();
    $reservations = array();

    while ($row = $result->fetch_assoc()) {
        $reservations[] =
self::GenerateReservationFromRow($row);
    }

    return $reservations;
}
```

28. kódrészlet

GetReservationAtRoomAtTimeSlot metódus

A metódus visszaadja a paraméterül kapott teremben és időpontban található foglalást. Ha nincs ilyen, akkor null értékkel tér vissza.

```
public static function
GetReservationAtRoomAtTimeSlot($room,
$timeSlot) {
    $conn = Database::GetDatabase();
    $stmt = $conn->prepare("SELECT * FROM
Reservation WHERE room = ? AND timeSlot =
?");
    $stmt->bind_param("ii", $room,
$timeSlot);
    $stmt->execute();
    $result = $stmt->get_result();

    if ($result->num_rows == 0) {
        return null;
    }
```

```
    return
self::GenerateReservationFromRow($result-
>fetch_assoc());
}
```

29. kódrészlet

Az osztály ezen kívül rendelkezik **GetAllReservations** metódussal.

IV. 2.2.6 Department osztály

FullName metódus

A metódus visszaadja az osztálypéldányhoz tartozó tagozatot *kód – megnevezés* formátumban

```
public function FullName() {
    return $this->code . " - " . $this-
>name;
}
```

30. kódrészlet

Az osztály ezen kívül rendelkezik **GetAllDepartments**, **GetDepartmentById**, **SaveToDatabase** és **DeleteById** metódusokkal.

IV. 2.2.7 Apply osztály

GetAppliesAtDepartmentId metódus

A metódus egy tömbben visszaadja a paraméterül kapott szak azonosítóhoz tartozó jelentkezéseket egy tömbben.

```
static function
GetAppliesAtDepartmentId($id) {
    $conn = Database::GetDatabase();
    $stmt = $conn->prepare("SELECT * FROM
Apply WHERE department = ?");
    $stmt->bind_param("i", $id);
    $stmt->execute();
    $result = $stmt->get_result();
    $departments = array();
    while ($row = $result->fetch_assoc()) {
```

```
            $departments[] =
self::GetAppyFromRow($row);
        }
        return $departments;
}
```

31. kódrészlet

GetRankingAtDepartmentId metódus

A metódus elkészíti a paraméterül kapott szak azonosítóhoz tartozó rangsort.

Az előző metódust felhasználva egy mappelés segítségével kigyűjtjük a jelentkezésekhez tartozó diákokat, majd azokat a **Student** osztályban található **Compare** metódus segítségével csökkenő sorrendbe rendezzük. Ezután a rangsorba rendezett diákokat tartalmazó tömböt visszaadjuk.

```
public static function
GetRankingAtDepartmentId($id) {
    $students = array_map(function ($a)
{return $a->student;},
self::GetAppliesAtDepartmentId($id));
    uasort($students, function ($a, $b) {
return Student::Compare($a, $b);});
    return $students;
}
```

32. kódrészlet

IV. 3 A felhasználói felület

A felhasználói felület szintén PHP segítségével van megvalósítva. Ennek a megoldásnak az előnye, hogy a felhasználó böngészőjében csak a megjelenítés történik, így nincs sok erőforrásigénye. Hátránya, hogy a szerverre hárul a HTML oldalak generálása, de a tervezett felhasználószám és az oldal bonyolultságából adódóan ez nem jelentős. Másik hátránya, hogy az alkalmazás ebben a formájában nem egészíthető ki más fajta kliensekkel, de ez ennél az alkalmazásnál nem volt szempont.

Szintén további hátrány, hogy a böngészőben dinamikusan frissülő tartalom nehezebben megoldható, de ez szintén nem elvárás az alkalmazással szemben.

A felhasználói felület megjelenítéséért felelős állományai az adott oldal URL-jét reprezentáló mappában *index.php* néven található. Ezekben a mappákban található a

mappa nevével megegyező nevű php kiterjesztésű állomány, amely az adott oldalhoz tartozó felhasználói interakciót kezeli. Ez alól kivételek az olyan interakciót kezelő állományok, melyekhez nem tartozik külön grafikus interfész, illetve a különböző kiegészítő állományok.

Az oldalak stílusáért a Bootstrap keretrendszer felelős. A Bootstrap egy ingyenes, nyílt forráskódú, széleskörben alkalmazott CSS és JavaScript keretrendszer. Segítségével egyszerűen, CSS osztályok használatával lehet igényesen kinéző, reszponzív – azaz, a böngésző szélességéhez igazodó tartalmú – weboldalakat készíteni.

IV. 3.1 A felület kialakítása

Minden a felhasználói felületet megjelenítő oldal legelejére be van emelve (include) a *header.php* állomány. Ennek a fájlnak a tartalma tartalmazza a HTML DOM head részét (ebben többek között a Bootstap keretrendszerre való hivatkozás) és a body részét a konkrét tartalom kezdetéig. Így itt található meg a navigációs sáv, ami dinamikusan kerül generálásra: minden felhasználó csak a számára elérhető menüpontokat látja, illetve az aktív menüpont ki van emelve.

Példa egy menüpont elemre:

```
<?php if($_SESSION['level'] > 2): ?>
<li class="nav-item">
<a class="nav-link <?php
if($_SERVER['REQUEST_URI'] ==
"/admin/listRooms/") echo "active";?>"
href="/admin/listRooms">Termek</a>
</li>
```

33. kódrészlet

A kód első sora azért felelős, hogy az alatta lévő HTML sorok csak akkor jelenjenek meg, ha a felhasználó szintje kettőnél nagyobb. (A szintek kezeléséről később bővebben szó lesz.) A következő sorokban egy Bootstrap navigációs elemet láthatunk. Az *a* elem osztály listájához egy feltételvizsgálat segítségével hozzákerül az *active* osztály, ha az aktuális URL megegyezik a megadottal.

Szintén minden oldal legvégén be van emelve a *footer.php* állomány. Ebben az állományban található a HTML DOM lezárása, illetve a Bootstrap JavaSrcipt beemelése.

IV. 3.2 Publikus felület

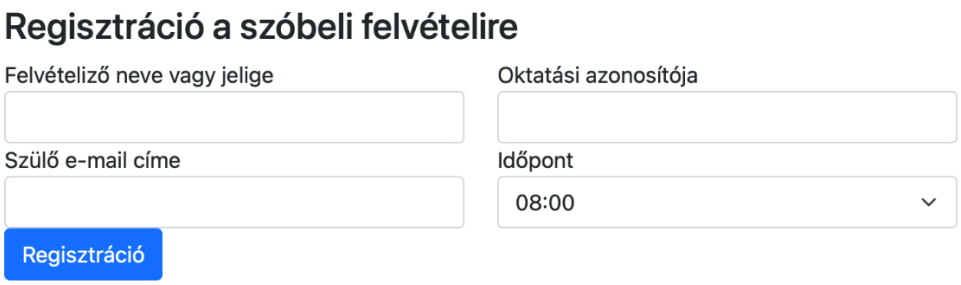
A program ezen részét a felhasználók bejelentkezés nélkül tudják használni. Itt van lehetősége a felvételiző diákoknak regisztrálni egy időpontra.

IV. 3.2.1 Regisztrációs űrlap

Ez az oldal az alkalmazás nyitólapja. A regisztráló diákok meg tudják adni:

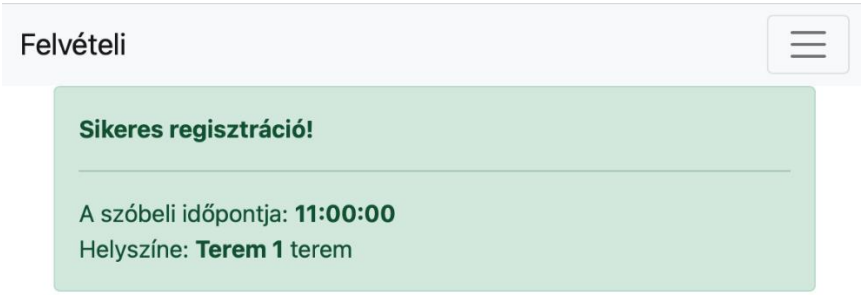
- nevüket vagy jeligét,
- oktatási azonosítójukat,
- szülő e-mail címét
- és azt az időpont, amikor szeretnének érkezni.

Az időpontokat megjelenítő legördülő listában csak azok az időpontok jelennek meg, amelyekre még van szabad terem.



13. ábra: regisztrációs űrlap

Az űrlap elküldése után a rendszer ellenőrzés céljából egy új lapon összesíti megadott adatokat. Itt a felhasználónak lehetősége van visszalépni, vagy folytatni a regisztrációt. Ez után az adatokat feldolgozó oldal ellenőrzi, hogy minden adat helyesen lett-e kitöltve, illetve, hogy a kiválasztott időpont továbbra is szabad e. Ha valamely adat rosszul lett megadva, vagy az időpont már betelt visszairányítja a felhasználót az űrlapra. Sikeres regisztráció esetén a felhasználó tájékoztatást kap ennek tényéről, illetve, hogy melyik terem lett neki kiválasztva.



14. ábra: sikeres regisztráció

IV. 3.3 Védett oldalak

A rendszer nagy részét azok az oldalak teszik ki, amik csak bejelentkezés ellenében érhetőek el. Ezek az oldalak a **/admin** könyvtárban találhatóak.

A bejelentkezést a **/login** oldal végzi. A felhasználó az űrlapon megadja a felhasználónevét és a jelszavát, majd az adatok elküldésre kerülnek a feldolgozó oldalnak.

Itt a jelszó hash-elésre kerül, majd lekéri a megadott felhasználónév-jelszó pároshoz tartozó felhasználót az adatbázisból. Ha nem sikerült a hitelesítés, akkor a felhasználó visszakerül a bejelentkezési felületre. Siker esetén indít egy SESSION-t (ez egy olyan tároló, amelyet létre lehet hozni minden kliens számára, és a webszerver az itt tárolt értékeket minden oldal betöltéskor újra rendelkezésünkre bocsájtja), ha még nem indult el és eltárolja a felhasználó azonosítóját és szintjét. Ez után átirányítjuk a védett felületre.

```
$userName = $_POST['user'];
$password = md5($_POST['password']);
$user = User::Authenticate($userName,
$password);
if($user === null) {
    header("Location: /login");
}
else {
    if (session_status() ===
PHP_SESSION_NONE) {
        session_start();
    }
    $_SESSION['user'] = $user->id;
    $_SESSION['level'] = $user->level;
    header("Location: /admin");
}
```

34. kódrészlet

A bejelentkezett felhasználók között az alábbi szinteket és elérhető menüpontokat különbözteti meg a rendszer. A magasabb jogosultsági szinttel rendelkező felhasználók az alacsonyabb szintű menüpontokat is elérik.

- 1. szint: tanár
    - Szóbeli pontszámok rögzítése
- 2. szint: tanulmányi
    - Adatrögzítés: hozott és központi pontok, jelentkezett szakok rögzítése
    - Foglalások: időpont foglalások listázása
    - Tanulók: tanulók listázása
    - Rangsor: felvételi rangsorok szakokként
- 3. szint: vezetőség
    - Termek: tanárok termekhez és témakörökhöz rendelése

      Itt van lehetőség termek kezelésére is, de ez csak 4-es szinten érhető el
    - Termek foglaltsága: időpont regisztrációk táblázatos nézete
- 4. szint: rendszergazda
    - Felhasználók: felhasználók kezelése
    - Szakok: szakok kezelése
    - Időpontok: időpontok kezelése

A megfelelő jogosultság ellenőrzését minden védett oldalon a checkLevel metódus meghívása ellenőrzi.

A metódus paraméterül kapja, hogy az adott oldal milyen szintű jogosultsággal érhető el.

```
function checkLevel($level)
{
    if (session_status() ===
PHP_SESSION_NONE) {
        session_start();
    }
    if (!isset($_SESSION['user']) ||
$_SESSION['level'] < $level) {
        header("Location: /");
    }
}
```

35. kódrészlet

Ezen oldalak alapvetően az osztályokban található üzleti logikai metóduskat illesztik a webes felületre.

Mivel a legtöbb oldal ugyanarra a logikára (listázó oldal akció gombokkal, adatbeviteli űrlap és űrlapkezelő) épül fel, így csak egy példát mutatok be.

IV. 3.3.1 Példa: felhasználók kezelése

A felhasználók kezelése három felhasználói felületből áll. Ezek közül az első a felhasználók listázása.

15. ábra: felhasználók listázása

Ezen az oldalon található egy link, amely az új felhasználó rögzítése oldalra mutat, majd alatta egy táblázat, ahol a rendszerben található felhasználók vannak megjelenítve.

A **User** osztály **GetAllUsers** metódusát felhasználva lekéri a rendszerben található felhasználókat. Ezeken egy foreach ciklussal végig lépkedve készít egy HTML táblázat sort, melybe belehelyezi a felhasználó nevét, szintjét és készít két linket a felhasználó módosítása és törlése oldalhoz, ahol paraméterként beágyazza az adott felhasználó azonosítóját.

```
<h3>Felhasználók</h3>
<div>
    <a class="btn btn-success"
href="../addUser">Új felhasználó</a>
</div>
<table class="table table-striped">
    <thead>
    <tr>
        <th>Név</th>
        <th>Szint</th>
        <th> </th>
    </tr>
    </thead>
    <tbody>
```

```
    <?php
    foreach (User::GetAllUsers() as $user)
{
        echo "<tr>" . PHP_EOL;
        echo "<td>" . $user->user . "</td>"
. PHP_EOL;
        echo "<td>" . $user->level .
"</td>" . PHP_EOL;
        echo "<td align='right'>" .
PHP_EOL;
        echo "<a class='btn btn-primary'
href='../modifyUser/index.php?id=$user-
>id'>Módosítás</a>" . PHP_EOL;
        echo "<a class='btn btn-danger'
href='../modifyUser/deleteUser.php?id=$user
->id'>Törlés</a>" . PHP_EOL;
        echo "</td>" . PHP_EOL;
        echo "</tr>" . PHP_EOL;
    }
    ?>
    </tbody>
</table>
```

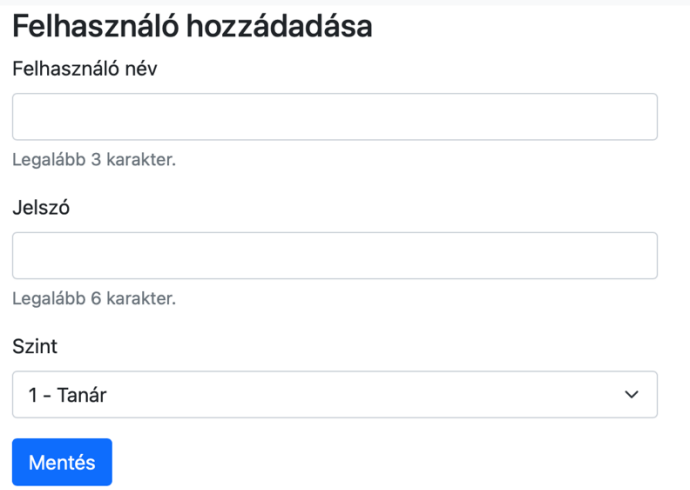
36. kódrészlet

Felhasználó törlése esetén nincs más felület, a művelet azonnal megtörténik, majd a felhasználó megint visszakerül a listához.

```
$user = User::GetUserById($_GET['id']);
$user->DeleteUser();
header("Location: /admin/listUsers");
```

37. kódrészlet

Új felhasználó felvétele esetén egy űrlapot mutat a rendszer a felhasználónak, ahol meg tudja adni az új felhasználó nevét, jelszavát és szintjét.



16. ábra: felhasználó felvétele

```
<h3>Felhasználó hozzádadása</h3>
<?php
if (isset($_GET['error'])) {
    echo "<div class='alert alert-danger'>"
. $_GET['error'] . "</div>";
}
?>
<form action="addUser.php" method="post">
    <div class="mb-3">
        <label for="userName" class="form-
label">Felhasználó név</label>
        <input type="text" class="form-
control" id="userName" name="user" aria-
describedby="userHelp" required>
        <div id="userHelp" class="form-
text">Legalább 3 karakter.</div>
    </div>
    <div class="mb-3">
        <label for="password" class="form-
label">Jelszó</label>
        <input type="password" class="form-
control" id="password" name="password"
aria-describedby="passHelp" required>
        <div id="passHelp" class="form-
text">Legalább 6 karakter.</div>
    </div>
    <div class="mb-3">
```

```
        <label for="level" class="form-
label">Szint</label>
        <select class="form-select"
id="level" name="level">
            <option value="1">1 -
Tanár</option>
            <option value="2">2 -
Tanulmányi</option>
            <option value="3">3 -
Vezető</option>
            <option value="4">4 -
Rendszergazda</option>
        </select>
    </div>
    <button type="submit" class="btn btn-
primary">Mentés</button>
</form>
```

38. kódrészlet

Az űrlap kitöltése után a feldolgozó oldal ellenőrzi az adatokat, majd felveszi a felhasználót az adatbázisban. Siker esetén a felhasználó visszakerül a listához, hiba esetén az űrlapra, ahol a hiba oka megjelenik számára.

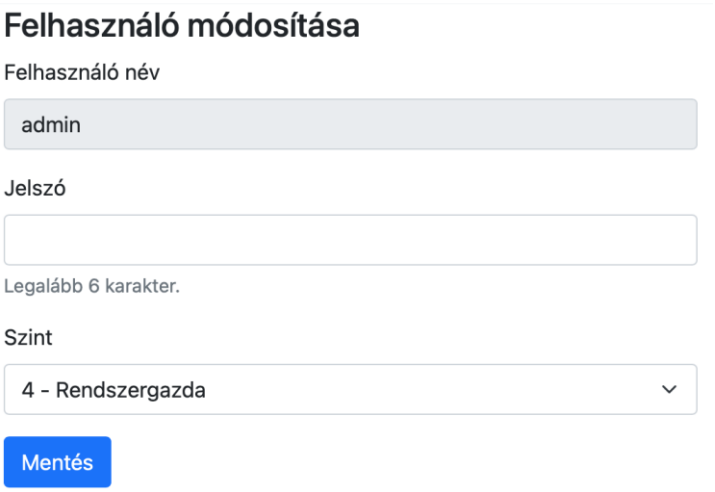
```
$user = new User();
$user->user = trim($_POST['user']);
$user->password = md5($_POST['password']);
$user->level = $_POST['level'];
if(strlen($user->user) < 3 ||
strlen($_POST['password']) < 6) {
    header("Location:
/admin/addUser?error=A%20név%20vagy%20a%20j
elszó%20nem%20megfelelő!");
}
else {
    $id = $user->SaveUser();

    if($id == 0) {
        header("Location:
/admin/addUser?error=A%20felhasználó%20már%
20létezik!");
    }
```

```
    else {
        header("Location:
/admin/listUsers");
    }
}
```

39. kódrészlet

Felhasználó módosítása esetén hasonló űrlap jelenik meg a felhasználó számára, mint hozzáadáskor, de a felhasználónév (amit nem lehet változtatni) és a szint már ki vannak töltve.



17. ábra: felhasználó módosítása

Az űrlap elküldése után a feldolgozó oldal ellenőrzi, hogy meg lett-e változtatva a jelszó, ha igen, frissíti azt és a felhasználó szintjét is. Ez után menti a módosításokat az adatbázisba és visszairányítja a felhasználót a listára.

```
$user = User::GetUserById($_POST['id']);
if (strlen($_POST['password']) > 0) {
    if(strlen($_POST['password']) < 6) {
        header("Location:
/admin/addUser?error=A%20jelszó%20nem%20meg
felelő!");
    }
    else {
        $user->password =
md5($_POST['password']);
    }
}
```

```
$user->level = $_POST['level'];
$id = $user->SaveUser();
header("Location: /admin/listUsers");
```

40. kódrészlet

V. A RENDSZER TESZTJE

A rendszer tesztje két lépcsőben zajlott. Egy funkcionális teszt, majd egy felhasználói teszt.

V. 1 Funkcionális teszt

A funkcionális tesztet pár kollégám végezte. Feladatuk az volt, hogy minden szerepkör tekintetében próbálják ki a rendszert felhasználói szemmel. Ellenőrizzék, hogy mindig az történik, aminek történnie kellene.

A teszt alatt szerencsére semmilyen hibás működést nem tapasztaltak, minden felhasználói interakció az elvárt módon működött, így a rendszer üzleti logikai részén nem kellett javítani.

De a teszt során felmerült pár javítási és fejlesztési ötlet a felhasználó felülettel kapcsolatban, így azokat a lehetőségekhez mérten be is építettem a rendszerbe.

V. 2 Felhasználói teszt átlagos forgalommal

Az elkészült és már (az előző fejezet tekintetében) hibamentes rendszer második megmérettetése az éles felhasználói teszt, amikor egyszerre annyi felhasználó teszteli a rendszer, mint amennyi felhasználó várható majd az rendszer éles bevetésekor.

Ebben a tesztben a tanulóim voltak segítségemre. Első lépésként meg kellett határozni, hogy ehhez hány tesztfelhasználóra van szükség. A rendszert egyidőben két esetben fogja használni egyszerre sok felhasználó:

- időpont foglaláskor (ekkor pontosan nem becsülhető meg az egyidejű felhasználók száma) és
- a szóbeli felvételi alatt.

Előző évek tapasztalatai alapján iskolánkba 300-500 tanuló jelentkezik. A szóbelire való jelentkezés legalább egy hétig tart, így ez átlagosan 3 felhasználót jelent óránként.

A szóbeli felvételi alatt a szóbeliztető tanárok használják intenzívebben a rendszert. 500 jelentkezővel és 20 perces idősávokkal 7 óra hosszan számolva 24 teremre van szükség. Ez megszorozva a témák számával 72 felhasználót jelent.

Egy átlagos tanulói csoport 16 fős. Az előző számításokat figyelembe vége elfogadható létszám a teszteléshez. Az első esetben mivel a regisztráció maga 1-2 percet vesz igénybe az áltagos felhasználó terhelést jóval túl lehet szimulálni. A második esetben is

elfogadhatónak tűnik a 16 fő, mivel valós esetben az elkerülhetetlen csúszások miatt biztosan nem fog az összes felhasználó egyszerre kattintani a rendszerben.

Ennél a két esetnél azt kellett tesztelni, hogy a rendszer mennyire reszponzív, illetve, hogy minden bevitt adat pontosan jelenik meg a rendszerben.

Szerencsére itt sem volt semmi fennakadás, annak ellenére sem, hogy a teszt szerver funkcióját egy Raspberry Pi látta el.

Terheléses tesztet – amikor az egyidejű maximális felhasználói számmal teszteljük a rendszert – nem végeztem, mivel a rendszer gyorsasága az azt kiszolgáló eszközön múlik. A weboldal kiszolgálásért az éles használat során egy webhosting fog gondoskodni, így, ha kevés lenne az erőforrás, akkor azt menetközeben lehet növelni.

VI. FEJLESZTÉSI LEHETŐSÉGEK

A rendszer több szempontból tartogat fejlesztési lehetőségeket, amiket a jövőben tanácsos lenne kiaknázni.

VI. 1 A felhasználói élmény javítása

E tekintetben főként a rendszer adminisztratív részén lehetne fejleszteni. Több esetben is a felhasználó nem kap hibaüzenet, hogy egy adott művelet (elsősorban törlés) pontosan miért nem történt meg. Mivel a rendszer elsősorban saját felhasználásra készült, így a rendszergazda szerepét én fogom betölteni, így az első verziónál ez megfelelő. Viszont ezen a jövőben informatív hibaüzenetekkel lehetne javítani.

VI. 2 Dinamikus működés

Bizonyos beállításokat jelenleg a rendszeren csak a rendszergazda, mint a rendszer üzemeltetője tud megváltoztatni a megfelelő állományokban a paraméterek átírásával. Ezeket a rendszerszintű beállításokat lehetne szintén az adatbázisban tárolni és a webes felületen lehetőséget biztosítani ezek módosítására.

Szintén idekapcsolódik, hogy az előző évekkel ellentétben, a következő szóbeli felvételi alkalmával a diákoknak a jelentkezett szakmához kapcsolódóan is számot kell adniuk a tudásukról. E tekintetben máig még nem körvonalazódott, hogy ez a szóbeli pontok összetételét miként érinti. Ennek vonatkozásában implementálni lehetne azt a lehetőséget a rendszerbe, hogy a témaköröket dinamikusan, azaz globális beállításként lehessen kezelni.

VI. 3 Naplózás

A rendszer jelenleg csak a regisztrációkhoz rendel időbélyeget. A könnyebb visszakövethetőség és hibakeresés érdekében a rendszert ki lehetne bővíteni egy naplózó modullal, amely minden adatmódosítást rögzítene az adatbázisban, vagy egy fájlban. Ehhez csak az adatbázist kezelő osztályok mentési metódusát kellene módosítani.

VI. 4 Rendszerek egységesítése

Iskolánkban több apróbb, bizonyos feladatra megalkotott rendszer létezik. Ilyen például a szakképzésre való jelentkeztetési, a fogadóórára jelentkezési, a nyílt napokra jelentkezési és a jelen rendszer. Ezen rendszerekben közös, hogy céljuk az adatbekérés, és azok prezentálása az iskola felé. A rendszerek között még adatátfedés is található, például jelen rendszerem és a fogadóóra rendszerében a tanárok azonosak.

Ezeket a rendszereket lehetne egységesíteni: az adatokat egy közös adatbázisban tárolni, és a belső felület nézeteit egy közös felületen prezentálni. Ez könnyebbség lenne az iskolának, mert az adatokat egy helyen láthatják, illetve ezzel a megoldással a későbbiekben egy új funkció könnyebben implementálható lenne. Ez tulajdonképpen hasonló lenne, mint egy ERP rendszer.

VII. KÖSZÖNETNYILVÁNÍTÁS

Szeretnék köszönetet mondani első sorban konzulensemnek, Sarkadi Katalinnak, aki végig kisért és segített ebben a két félévben.

Továbbá szeretnék köszönetet mondani kollégáimnak, akik sokat segítettek a rendszer tesztelésében. Illetve szeretnék még köszönetet mondani minden olyan barátomnak, ismerősömnek és diákomnak, akik valamilyen formában segítettek a szakdolgozatom elkészülésében.

VIII. IRODALOMJEGYZÉK

[1] Oktatási Hivatal: Tájékoztató a középfokú beiskolázás egységes írásbeli felvételi vizsgáinak - magyar nyelvi és matematika – feladatlapjairól (https://www.oktatas.hu/kozneveles/kozepfoku_felveteli_eljaras/tajekoztato_felvetelizoknek/2020_2021beiskolazas/tajekoztato_matek_magyar), utoljára megtekintve: 2021. 04. 05.

[2] Wikipedia: WordPress (https://en.wikipedia.org/wiki/WordPress), utoljára megtekintve: 2021. 05. 10.

[3] GeneratePress (https://generatepress.com), utoljára megtekintve: 2021. 05. 10.

[4] Google Tag Manager (https://hu.wordpress.org/plugins/duracelltomi-google-tag-manager), utoljára megtekintve: 2021. 05. 10.