ЗАТВЕРДЖЕНО
Вченою радою ЗУІ
Протокол № „3” від „27” квітня 2021 р.
Ф-КДМ-2

Закарпатський угорський інститут ім. Ференца Ракоці ІІ

Кафедра математики та інформатики

Реєстраційний №_____

Кваліфікаційна робота

Створення онлайн інформаційної системи для адміністрування навчальним процесом для ЗВО на прикладі ЗУІ

Дудаш Іван Іванович

Студент ІІ-го курсу

Освітня програма 014 «Середня освіта (Математика)»

Ступінь вищої освіти: магістр

Тема затверджена Вченою радою ЗУІ

Протокол **№ 10 від 27 жовтня 2021 року**

Науковий керівник: **Головач Йожеф Ігнацович**

д. т. н., проф.

Завідувач кафедрою математики та інформатики : **Кучінка Каталін Йожефівна**

к. ф.-м. н

Робота захищена на оцінку _____, «___» _____ 202_ року

Протокол № _____ / 202_

ЗАТВЕРДЖЕНО
Вченою радою ЗУІ
Протокол № „3” від „27” квітня 2021 р.
Ф-КДМ-2

Закарпатський угорський інститут ім. Ференца Ракоці ІІ

Кафедра математики та інформатики

Кваліфікаційна робота

Створення онлайн інформаційної системи для адміністрування навчальним процесом для ЗВО на прикладі ЗУІ

Ступінь вищої освіти: магістр

Виконав: студент ІІ-го курсу

Дудаш Іван Іванович

Освітня програма 014 «Середня освіта (Математика)»

Науковий керівник: **Головач Йожеф Ігнацович**

д. т. н., проф.

Рецензент: **Міца О.В.**

зав. кафедри ІУСТ УжНУ, доктор технічних наук, доцент

Берегове
2022

ЗМІСТ

Вступ **8**

1. Теоретична частина **10**

1.1. Інформаційно-комунікаційні технології в процесах управлінння навчальними закладами 10

1.1.1. Поняття інформаційно-комунікаційних технологій (ІКТ) 10

1.1.2. Веб-додатки, як інструменти адміністрування освітніх процесів 11

1.2. Зберігання даних, бази даних та системи управління базами даних 11

1.2.1. Бази даних і системи управління базами даних 11

1.2.2. Проектування бази даних 13

1.3. Створення компонентів серверної частини інформаційної системи в середовищі NodeJS, ExpressJS, Sequelize 14

1.3.1. NodeJS 15

1.3.2. ExpressJS 16

1.4. Angular 12 як інструмент для програмування на стороні клієнта 18

1.5. Захист даних та управління авторизацією. JSON Web Token (JWT) 19

1.6. Роль оцінювання в освіті. Веб-додатки в процесі оцінювання 20

2. Практична частина **23**

2.1. Проектування та створення бази даних 23

2.1.1. Сутності, зв’язки між ними та обмеження бази даних 23

2.1.2. Система зв'язків в базі даних 39

2.1.3. Схема бази даних в MySQL 39

2.2. Створення компонентів серверної частини інформаційної системи в середовищі NodeJS, ExpressJS, Sequelize 46

2.2.1. Установка модулів та основні налаштування 47

2.2.2. Встановлення та налаштування веб-сервера ExpressJS 48

2.2.3. Sequelize та налаштувати базу даних MySQL 48
2.2.4. Створення елементів керування 50
2.2.5. Встановлення маршрутів 54
2.3. Створення компонентів клієнтної частини інформаційної системи в середовищі Angular 12 CLI 55
2.3.1. Встановлення Angular CLI, основні налаштування 55
2.3.2. Налаштування моделей, шляхів та елементів керування 58
2.4. Створення компонентів захисту даних інформаційної системи за допомогою технології JSON Web Token (JWT) 65
2.4.1. JSON Web Token в середовищі сервера 65
2.4.2. JSON Web Token в середовищі клієнта 70
Підсумок (на угорській мові) 75
Бібліографія 77
Список ілюстрацій 81
Додатки 82
Підсумок 88

ЗАТВЕРДЖЕНО
Вченою радою ЗУІ
Протокол № „3” від „27” квітня 2021 р.
Ф-КДМ-2

II. Rákóczi Ferenc Kárpátaljai Magyar Főiskola

Matematika és Informatika Tanszék

ONLINE INFORMÁCIÓS RENDSZER KÉSZÍTÉSE A FELSŐOKTATÁSI ADMINISZTRÁCIÓS FOLYAMATOK MEGOLDÁSÁRA A KMF PÉLDÁJÁN

Szakdolgozat

Képzési szint: mesterképzés

Készítette: Dudás János

II. évfolyamos hallgató

Képzési program: 014 „Középiskolai oktatás (Matematika)”

Témavezető: Holovács József

műszaki tudományok doktora, professzor

Recenzens: Mitsa O. V.

UNE IVRT tanszék tanszékvezetője, műszaki tud. doktora, docens

Beregszász – 2022

TARTALOMJEGYZÉK

Bevezetés **8**

1. Elméleti rész **10**

1.1. Az információs-kommunikációs technológiák az oktatási intézmények adminisztrációs folyamataiban 10

1.1.1. Információs-kommunikációs technológiák (IKT) fogalma 10

1.1.2. Webes alkalmazások, mint az oktatási folyamatok adminisztrálásának az eszközei 11

1.2. Az adatok tárolása, adatbázisok és adatbáziskezelő rendszerek 11

1.2.1. Adatbázisok és adatbáziskezelő rendszerek 11

1.2.2. Az adatbázis tervezése 13

1.3. NodeJS és ExpressJS mint a szerveroldali (back-end) programozás eszköze 14

1.3.1. NodeJS 15

1.3.2. ExpressJS 16

1.4. Az Angular 12, mint a kliensoldali (front-end) programozás eszköze 18

1.5. Az adatok védelme és jogosultságok kezelése. JSON Web Token (JWT) 19

1.6. Az értékelés szerepe az oktatásban. Webes alkalmazások az értékelési folyamatban 20

2. Gyakorlati rész **23**

2.1. Az adatbázis tervezése és létrehozása 23

2.1.1. Az adatbázis egyedei, kapcsolatai és megszorításai 23

2.1.2. Az adatbázis kapcsolatrendszere 39

2.1.3. Az adatbázis sémája MySQL-ben 39

2.2. Az információs rendszer szerveroldali komponenseinek létrehozása NodeJS, ExpressJS, Sequelize környezetben 46

2.2.1. Modulok telepítése, alapvető beállítások 47

2.2.2. Az ExpressJS webszerver telepítése és beállítása 48

2.2.3. Sequelize és a MySQL adatbázis beállítása 48
2.2.4. Vezérlők létrehozása 50
2.2.5. Útvonalak beállítása 54
2.3. Az információs rendszer kliensoldali komponenseinek létrehozása Angular 12 CLI környezetben 55
2.3.1. Angular CLI telepítése, alapvető beállítások 55
2.3.2. Modellek, útvonalak és vezérlők beállítása 58
2.4. Az információs rendszer adatvédelmi komponenseinek létrehozása JSON Web Token (JWT) alapú technológia segítségével 65
2.4.1. JSON Web Token a szerveroldali környezetben 65
2.4.2. JSON Web Token a kliensoldali környezetben 70
Összefoglalás 75
Irodalomjegyzék 77
Ábrajegyzék 81
Mellékletek 82
Összefoglalás (ukránul) 88

BEVEZETÉS

Az modern információs társadalmunkban az emberek életére a legnagyobb hatást az információs-kommunikációs technológiák gyakorolják. Ugyanúgy kihatással vannak a munkánkra, a kommunikációnkra, mint a vásárlási szokásainkra vagy akár az oktatásra is. Az oktatási intézmények ezeket a technológiákat felhasználják és alkalmazzák, valamint emellett tanítják is az ehhez kapcsolódó ismereteket [8].

Tehát az információs-kommunikációs technológiák nemcsak a tanulás-tanítási folyamatban játszanak fontos szerepet, hanem megjelennek az oktatási intézmények adminisztrációs folyamatainak a szervezésében és végrehajtásában is. Ezeket a folyamatokat legkönnyebben web alapú rendszerek segítségével lehet kivitelezni, így a szakdolgozat keretein belül a fő célunk egy online információs rendszer készítése az oktatási folyamatok adminisztrálására a II. Rákóczi Ferenc Kárpátaljai Magyar Főiskola példáján.

Viszont az általunk létrehozni kívánt adminisztrációs platformnak kettős célja van, amiknek a megvalósítását a Covid19 járvány elterjedése miatti lezárások tettek egyre inkább sürgetővé [29]. Célunk egyrészt egy felhasználóbarát felület biztosítása az oktatási intézmény fő adminisztrációs folyamatainak a webes megvalósítására. Másrészt pedig a félév végi értékelések gördülékennyé tétele a távoktatás során, ami kiemelt szerepet játszik, akár az alap- vagy középfokú oktatásban résztvevő diákok [15, 31], vagy akár a felsőoktatásban tanuló hallgatók [20] motivációjának és teljesítményének a megőrzésében is.

Egy ilyen információs rendszer kialakításában az első és legfontosabb feladat az alapadatok tárolásának hatékony megvalósítása, és ehhez a munka során MySQL relációs adatbáziskezelő rendszert fogunk alkalmazni. Ennek a munkafolyamatnak a keretein belül a fő célunk az adatok közötti kapcsolatok feltárása és megfelelő adatszerkezetek meghatározása.

Valamint fontos még figyelembe venni, hogy a különböző adminisztrációs folyamatok, különböző jogosultsági szinttel rendelkező személyekhez kapcsolódnak. Ennek a megvalósításához szükséges a különböző adminisztrációs folyamatok szereplőinek hatáskörét, azaz jogosultságát meghatározni. A munkánk során a következő jogosultsági szintekkel fogunk dolgozni:

- szuper adminisztrátor – teljes körű hozzáférés a rendszerhez írás- és olvasás joggal;
- vezetőség (igazgatóság) – teljes körű hozzáférés a rendszerhez olvasás joggal;
- tanulmányi osztály – teljes körű hozzáférés a rendszerhez olvasás joggal, valamint írási jog a következő adatokon: hallgatók- és oktatók adatai (pl. névváltoztatás) valamint a hallgató vizsgázási jogát tudja szerkeszteni (pl. pótvizsgadíj befizetés esetén engedélyezni a vizsgázást);
- tanszéki adminisztrátor – teljes körű hozzáférés a saját tanszékének az adataihoz olvasási joggal, valamint írás jog a következő adatokon: tantárgyak adatai (pl. új tantárgy felvétele, szerkesztése, régi tantárgy törlése) tantárgyak kiosztása az oktatóknak (óraszám elosztás), mintatanterv létrehozása;
- tanárok – saját adatainak olvasása, valamint írás joggal rendelkezik azokon az e-leckekönyv bejegyzéseken, ahol az adott tantárgyat ő oktatja (jegybeírás);
- hallgatók – hozzáfér a saját adataihoz és eredményeihez.

Ezeknek a jogosultsági szinteknek a kezeléséhez pedig JSON Web Token (JWT) alapú technológiát fogunk alkalmazni.

Valamint a munkánk során kialakítunk az adminisztrációs folyamat termékének tekinthető elektronikus leckekönyvet, valamint az adminisztrációs folyamatok elvégzését megkönnyítő webes felületet is. Fő célunk ezeknek a zökkenőmentes működése, és felhasználóbarát felületének biztosítása. Hogy ezeket a célokat el tudjuk érni, ezért a platform kialakításához NodeJS, Express és Angular 12 keretrendszer rétegek fogunk használni.

1. FEJEZET

ELMÉLETI RÉSZ

1.1. Az információs-kommunikációs technológiák az oktatási intézmények adminisztrációs folyamataiban

1.1.1. Információs-kommunikációs technológiák (IKT) fogalma

Az információs-kommunikációs technológiák (továbbiakban: IKT) meghatározásához többféle megközelítést is lehet alkalmazni. Molnár 2008-as tanulmánya szerint a fogalom értelmezése a következő szempontok szerint lehetséges: IKT mint eszköz, IKT mint ellenőrzési eszköz és automata technológia, IKT mint szervezési technika, IKT mint média és összekapcsolható technika, IKT mint fejlesztési és társadalomalakító folyamat, valamint IKT mint technikai gyakorlat [14, 18].

Tehát az IKT eszközök lehetővé teszik az egész információs folyamat tervezését. Azaz az információ hozzáféréstől kezdődően, az információ átadásával bezárólag segítséget nyújtanak a különböző információs folyamatok végrehajtásában. Ebből kifolyólag IKT eszköznek tekinthetjük az okostelefonunkat, a hardver és szoftver eszközeinket, akár egy szuperszámítógépet, valamint magát az internetet is. Az IKT megjelenik a szervezésben is, ami főleg a munkafolyamatok felgyorsításában és automatizálásban mutatkozik meg. És ebből kifolyólag ezek a munkafolyamatok jobban ellenőrizhetővé válnak, viszont nem szabad elfelejteni azt a problémát, hogy azáltal, hogy a vezetők akár közvetlenül, azonnal be tudnak avatkozni a munkafolyamatba, az egyéni kreativitás és individualitás háttérbe szorul [18, 26].

Továbbá a fogalmat médiaként is fel lehet fogni, mert a feldolgozott információkat közvetíti az egyes szervezeti egységek között, ahol a folyamat- és az információmegosztás felgyorsítása is megjelenik. Így a nem azonos térben jelen lévő személyek is képesek a közös munkára, könnyen megoszthatóvá válik a tudás. És ezzel egyidejűleg megjelenik a fogalom fejlesztési és társadalomalakító aspektusa is, mivel ezeket az eszközöket, legyenek azok hardverek vagy szoftverek, folyamatosan fejleszteni kell, hogy az állandóan változó információs társadalom igényeit ki tudják elégíteni [18].

1.1.2. Webes alkalmazások, mint az oktatási folyamatok adminisztrálásának az eszközei

Az IKT-k egyre nagyobb szerepet játszanak az oktatási intézmények adminisztrációs folyamataiban is. Ezen technológiák- és eszközök segítségével végrehajtott munkafolyamatok nagyban egyszerűsödnek. Segítségükkel automatikussá válnak a különböző monoton adminisztrációs feladatok végrehajtásai is, azaz hatékonyabbá teszik a munkát. Viszont nem szabad figyelmen kívül hagyni, hogy ezáltal egyre bővül a munkavégzéshez szükséges ismeretek köre is [8, 30].

Az iskolai adminisztrációs rendszerek legfontosabb feladatai Buda 2017-ben megjelent könyve szerint a következők: az információs rendszer működéséhez szükséges alapadatok nyilvántartása; a hallgatók és oktatók adatainak tárolása; a tantervek és egyéb, az oktatási folyamatban használt dokumentumok tárolása; valamint kimutatások és statisztikák készítése. Az elvégzendő feladatokból is látszik, hogy az ilyen és ehhez hasonló rendszerek létrehozására a web alapú technológiák a legalkalmasabbak [8]. Legfontosabb előnyük, hogy nincsenek helyhez kötve, és a rendszer résztvevői távolról is hozzáférhetnek az adataikhoz, végezhetik tevékenységeiket.

A 2020-as évi események [29] – a Covid19 járvány elterjedése, a lezárások bevezetése – következtében a II. Rákóczi Ferenc Kárpátaljai Magyar Főiskola esetében is egyre inkább sürgetőbbé vált az adminisztrációs folyamatok informatizálása, online formába történő szervezett átállása. Viszont minden felsőoktatási intézmény strukturális, valamint szervezeti felépítése más és más. Ennek következtében standard, minden intézményben egységesen használható rendszer kiépítése és használata nem lehetséges. Ezáltal egy saját tervezésű, az említett felsőoktatási intézményeben használható Web API fejlesztését és tesztelését kezdtük el a 2020/2021-es tanévben NodeJS, ExpressJS és Angular 12 CLI keretrendszer rétegek segítségével, az adatok tárolását és kezelését MySQL relációs adatbáziskezelő rendszerben végezzük.

1.2. Az adatok tárolása, adatbázisok és adatbáziskezelő rendszerek

1.2.1. Adatbázisok és adatbáziskezelő rendszerek

Hatékony információs-kommunikációs rendszer kialakításához elsődleges lépés az adatok megfelelő tárolásának és hozzáférhetőségének a biztosítása. Ennek megvalósítása érdekében a munkánk során az adatokat adatbázisban fogjuk tárolni, valamint

feldolgozásukat és hozzáférhetőségüket MySQL adatbáziskezelő rendszer fogja biztosítani. De mi is az az adatbázis és az adatbázis-kezelő rendszer?

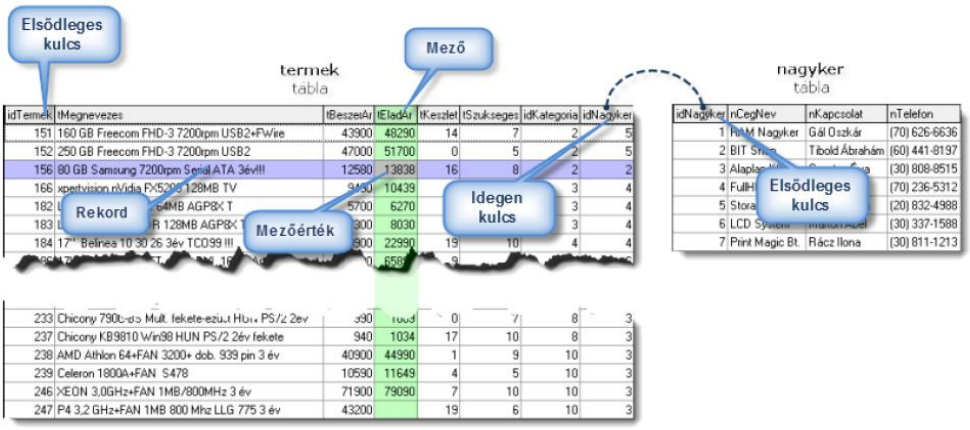
Az adatbázis az adatoknak egy olyan rendezett gyűjteménye, amely hozzáférést biztosít a feladatokhoz kapcsolódó adatokhoz és biztosítja az azok védelmét és integritásának megőrzését. Az adatbázisokat és a számítógépes erőforrásokat összefoglaló rendszert adatbázis-kezelő rendszereknek vagy DBMS (Database Management System) nevezzük. Ezeknek a rendszereknek a legfontosabb feladatai a következők: az adatbázisok tartalmának létrehozása és tárolása, az adatok lekérdezése, védelme, titkosítása, valamint a hozzáférési jogok kezelése [16, 22, 23, 28].

Az adatbázis-kezelő rendszerek általában szerverként működnek és nem rendelkeznek kezelőfelülettel. Vezérlésük adatkezelő nyelven megírt utasításokkal történik, ami úgynevezett kliensalkalmazásokon keresztül valósul meg [23, 28].

Minden adatbázisnak van egy belső struktúrája, amely tartalmazza az adatok és ezeknek az adatoknak az egymáshoz viszonyított leírását, amit az adatbázis sémájának nevezünk. Ez a séma az alkalmazott modelltől függően más és más. Az adatbázis-kezelés fejlődése során a következő adatmodellek alakultak ki: hierarchikus, hálós és relációs [22]. A rendszerünk alapjául szolgáló MySQL adatbázis-kezelő rendszer relációs alapú adatmodellt használ.

A relációs adatmodell alapját a matematikai reláció képezi. Ez napjainkban a legelterjedtebb adatmodellnek számít [22, 28]. A modell alapja, hogy az adatok egymással kapcsolatban vannak, és ezeket a kapcsolatokat, és az adatok egymáshoz való viszonyát kétdimenziós táblázatokban elrendezve ábrázolhatjuk. Az egyes táblákban az úgynevezett egyedtípusok és hozzájuk tartozó tulajdonságokat tároljuk. Az itt tárolt egyedtípusok egyedeinek az azonosíthatóságát az elsődleges kulcs biztosítja, a különböző egyedek közötti kapcsolatokat pedig az idegen kulcs (lásd: 1. ábra) [28].

Tehát a relációs adatbázisban a logikailag összetartozó adatokat a reláció soraiban fognak elhelyezkedni, ezeknek a sorrendje nem fontos, viszont semmiképpen sem tartalmazhat két azonos adatot. A mennyiségek a sorok és az oszlopok metszetében tárolódnak, ezek az adatok lehetnek szöveges-, numerikus-, valamint logikai adatok is. Tehát tábla vagy táblázat elnevezés magára a relációra utal [16, 22]. A relációs alapú adatmodell használó adatbázis-kezelő rendszerek adatkezelő nyelve az SQL [28].



1. ábra. Táblák a relációs adatmodellben (Forrás: Szabó, 2013)

Radványi 2014-ben megjelent könyve szerint a relációval kapcsolatos legfontosabb követelmények a következők: a különböző táblázatokat egyértelműen meg lehet különböztetni egymástól; a sorok és az oszlopok metszéspontjában található adatok az elemi adatok; egy oszlopon belül azonos típusú adatok szerepelnek; az oszlopokat nevük segítségével egyértelműen meg tudjuk különböztetni egymástól; minden sor ugyanannyi adatot tartalmaz; nem létezhet két azonos sor; a sorok és oszlopok sorrendje tetszőleges.

A relációs séma követelményeinek a betartása az általunk létrehozni kívánt információs rendszer esetében is nagyon fontosak, hiszen ez az alapja egy platform gördülékeny működésnek, valamint adatmodell hibák elkerülésének. Emiatt az adattárolás megvalósítása mellett, az adatbázis tervezésére is nagyon nagy hangsúly kell fektetnünk a munkánk során.

1.2.2. Az adatbázis tervezése

A tervezés hosszú és összetett folyamat, mivel többféle kritériumnak is meg kell felelnie az adatbázisunknak.

Első és legfontosabb feladat a cél egyértelmű kijelölése, aminek a felhasználók igényeivel összehangban kell lennie. Fontos meghatározni, hogy milyen információkat szeretnénk megkapni az adatbázisból, tehát meg kell ismerni a felhasználói igényeket, és tudni kell tájékozódni az adott rendszer felhasználóinak szakterületén. Tisztáznunk kell azt is, hogy milyen alapvető adatokat kell tárolnunk az egyedtípusokról. Érdemes minél

több, tervezési forrásként felhasználható dokumentumot gyűjteni, ez nagyban megkönnyíti a későbbi munkát [16, 22].

Ha az első szakasszal megvagyunk, akkor kezdődhet a logikai tervezés. Itt történik az egyedtípusok és tulajdonságaiknak a leírása, valamint a közöttük lévő kapcsolatok feltérképezése. Fontos kiemelni az elsődleges kulcs, valamint az egyedtípusok közötti kapcsolatok meghatározásának a fontosságát, hogy adatainkat egyértelműen tudjuk majd a későbbiekben azonosítani [22]. Radványi 2014-ben megjelent könyve szerint három típusú elsődleges kulcs alkalmazható: számláló, egy mezőből álló, valamint több mezőből álló elsődleges kulcs.

Az egyedtípusok közötti kapcsolatok meghatározásához a táblák sorait kapcsoljuk össze elsődleges kulcsok segítségével. Számosságuk szerint a kapcsolatokat három csoportba oszthatjuk: egy az egyhez kapcsolat; egy a többhöz kapcsolat; több a többhöz kapcsolat [16, 22].

Kiemelet szerepet kap még a normalizálás, azaz az adatok elhelyesének optimális módjának a kidolgozása. Segítségével a különböző ellentmondások, azaz anomáliák kiküszöbölése, valamint a redudancia (adatismétlés) csökkentése valósítható meg.

És végül az elkészült rendszer fizikai tesztelése következik, aminek a keretein belül a számítógépen létrehozzunk az adatbázisrendszer első verzióját. A hangsúly ennél a fázisnál azon van, hogy megvizsgáljuk azt, hogy a logikai szerkezet mennyire felel meg a hatékony végrehajtásnak. Ha szükséges módosításokat is végrehajthatunk [22].

Ha sikeresen elkészítettük és teszteltük az adatbázisunkat, akkor a továbbiakban szükségünk lesz egy olyan felület kialakítására is, ahol a felhasználók kényelmesen, akár távolról is tudnak majd dolgozni az eltárolt adatokkal. És erre a legalkalmasabb egy dinamikus webes platformnak az elkészítése, aminek a kivitelezéséhez a mi választásunk a NodeJS, ExpressJS és Angular 12 CLI keretrendszer rétegekre esett.

1.3. NodeJS és ExpressJS mint a szerveroldali (back-end) programozás eszköze

Jól működő web alkalmazásokat tervezni és létrehozni nem egyszerű feladat, hiszen ezek az alkalmazások magas szintű funkcionalitást, felhasználói élményt, és költséghatékony karbantartást, valamint alacsony futtatási költségeket igényelnek. Viszont a JavaScript,

NodeJS és az ExpressJS kombinációja ideális választás lehet azoknak, akik nagy teljesítményű, gyorsan telepíthető és karbantartható technológiákat akarnak létrehozni [13]. Ezáltal a mi választásunk is erre párosításra esett, hogy az általunk tervezett webes platform keretrendszerének a (rejtett) feldolgozást végző rétegként (back-end) funkcionáljon. Valamint ez a réteg kommunikál a háttérben az adatok feldolgozásával- és tárolásával foglalkozó MySQL adatbáziskezelő-rendszerrel is, amit Sequelize modul segítségével fogunk majd megvalósítani.

A Sequelize egy ORM eszköz, amely felhasználóbarát felülettel rendelkezik. Az objektum-relációs leképzés (angolul Object-Relational Mapping) egy programozási technika adatok konvertálására nem kompatibilis típusos rendszerek és objektumorientált programozási nyelvek között. Lényegében létrehoz egy „virtuális objektum-adatbázist”, amit a programozási nyelvben használhatunk [9, 25].

A Sequelize modul keretein belül lehetőségünk van az adatok manipulálására SQL parancsokkal, valamint az eszköz támogatja a fő SQL adatbázisokat is, mint például a Postgres, MariaDB, MySQL, SQL Server és SQLite3 [25].

1.3.1. NodeJS

A NodeJS egy C/C++-ban íródott, esemény alapú I/O rendszer a Google V8 JavaScript motorja felett. Használatával JavaScript nyelven fejleszthetünk skálázható szerveroldali alkalmazásokat egy egyszerű felület segítségével [2, 9].

A NodeJS úgy lett kifejlesztve, hogy aszinkron eseményekkel dolgozzon, ezért a programban nem kell várni, hogy egy művelet befejeződjön, hanem vele párhuzamosan más műveleteket is futtathatunk. Ez a rendszer optimálisabb működését és több alkalmazás egyidejű futását teszi lehetővé. A NodeJS sok hasonlóságot mutat a népszerű webszerverekkel (IS, Apache stb), viszont érdemes először is a különbségekkel foglalkoznunk [9, 13].

A NodeJS a többi webszerverrel ellentétben (IS, Apache stb) nagyon könnyen beállítható és konfigurálható. Ez nem azt jelenti, hogy a csomópontkiszolgálók beállítása a maximális teljesítmény érdekében éles környezetben triviális kérdés: csak a konfigurációs lehetőségek egyszerűbbek és könnyebben értelmezhetőek [13].

Egy másik nagy különbség a NodeJS és a hagyományos webszerverek között az, hogy a NodeJS egyszálú (egyetlen esemény hurokban (event loop) fut a program). Első

pillantásra ez visszalépésnek tűnhet. Viszont ez egy zseniális ötlet. Az egyszálas futás nagyban leegyszerűsíti a webalkalmazások írását, és ha szükségünk van egy többszálas alkalmazás teljesítményére, akkor egyszerűen beállíthatjuk a NodeJS több példányát, és hatékonyan élvezhetjük a többszálúság előnyeit [13].

Az programok megírásának módját tekintve a NodeJS alkalmazások jobban hasonlítanak a PHP- vagy Ruby-alkalmazásokhoz, mint a .NET- vagy a Java-nyelven megírtakhoz. Míg a NodeJS által használt JavaScript-motor (a Google V8-as verziója) natív gépi kódra fordítja a JavaScriptet (hasonlóan a C-hez vagy a C++-hoz), ezt átláthatóan teszi, így a felhasználó szemszögéből tisztán értelmezett nyelvként viselkedik. A külön fordítási lépés hiánya csökkenti a karbantartási és telepítési nehézségeket: mindössze egy JavaScript-fájlt kell frissítenünk, és a módosítások automatikusan elérhetők lesznek [9, 13].

A rendszer egy másik előnye, hogy a NodeJS platformfüggetlen. A NodeJS egy pillanat alatt beállítható az összes fő operációs rendszeren (Windows, OS X és Linux), és egyszerű együttműködést tesz lehetővé [9, 13].

1.3.2. ExpressJS

Az ExpressJS egy ingyenes és nyílt forráskódú szoftver MIT licenc alatt, ami a NodeJS háttér-keretrendszereként funkcionál (framework). Felhasználása egyszerűbbé teszi a webalkalmazások fejlesztését NodeJS alapokon. Az ExpressJS a NodeJS http modulon és az úgynevezett connect komponenseken alapul, és egy olyan alkalmazást definiál, ami elfedi számunkra a nyelv nehézségeit. Ezeket a komponenseket köztes szoftvereknek nevezik, és ezek a keret filozófiájának sarokkövei [2, 13].

Brown 2014-ben megjelent könyve szerint az ExpressJS-t a következő funkciókkal lehet jellemezni:

- Minimális: az ExpressJS filozófiája az, hogy minimális réteget biztosítson az úgynevezett „agy” és a szerver között. Viszont ez nem jelenti azt, hogy nem rendelkezik elegendő hasznos funkcióval, hanem kevésbé akadályozza az egyéni utat, azaz lehetővé teszi a saját ötletek kifejezését;
- Rugalmas: másik kulcsfontosságú perspektívája pedig az, hogy bővíthető. Ez azt jelenti, hogy az ExpressJS egy nagyon minimális keretrendszert biztosít, és ha szükség van rá hozzáadhatjuk az ExpressJS funkciók különböző részeit, valamint

lehetővé teszi, hogy lecseréljük azokat a részeket, amelyek nem felelnek meg az igényeinknek. Más szóval, a fejlesztők szabadon választhatnak bármilyen könyvtárat, amire szükségük van egy adott projekthez, ami rugalmasságot és nagy testreszabhatóságot biztosít számukra;

- Alkalmas webes alkalmazás keretrendszer-rétegnek: az ExpressJS segítségével webalkalmazásokat készíthetünk. De mi is az a webalkalmazás? A webhely webalkalmazás, a weboldal is webalkalmazás. De egy webalkalmazás több is lehet: funkcionalitás biztosít más webalkalmazások számára. Általánosságban elmondható, hogy az „alkalmazás" kifejezést olyan dolgok jelölésére használjuk, amelyeknek van funkcionalitása: ez nem csak egy statikus tartalomgyűjtemény (bár ez egy nagyon egyszerű példa egy webalkalmazásra);
- Alkalmas egyoldalas webes alkalmazások készítésére: az egyoldalas webalkalmazásoknál ahelyett, hogy egy webhely hálózati kérést igényelne minden alkalommal, amikor a felhasználó egy másik oldalra lép, ahelyett az egyoldalas webalkalmazás letölti a teljes webhelyét (vagy annak egy jó részét) a felhasználó böngészőjébe. A kezdeti letöltés után a navigáció gyorsabb, mert aztán kevesebb lesz, vagy egyáltalán nincs is kommunikáció a szerverrel. Az egyoldalas alkalmazásfejlesztést megkönnyíti az olyan népszerű keretrendszerek használatát, mint az Angular.
- Alkalmas többboldalas, vagy hibrid webalkalmazások készítésére: a többboldalas webes alkalmazások a webhelyek hagyományosabb megközelítése. A weboldal minden egyes oldala külön kéréssel érkezik a szerverhez. Csak azért, mert ez a megközelítés hagyományosabb, nem jelenti azt, hogy az egyoldalas alkalmazások jobbak. Egyszerűen több lehetőség áll rendelkezésre a fejlesztés során, és eldönthetjük, hogy a tartalom mely részeit szeretnénk egyoldalas alkalmazásként, és mely részeit egyedi kéréseken keresztül kézbesíteni. A „hibrid" olyan webhelyeket ír le, amelyek mindkét megközelítést alkalmazzák [2, 13].

Azonban a NodeJS, ExpressJS csak az adatok (rejtett) feldolgozásával foglalkozik, ezért a webalkalmazásunk létrehozásához szükségünk van egy olyan platformra is, ahol be tudjuk vinni a szükséges adatokat, s a már feldolgozott adatokat meg tudjuk jeleníteni. Tehát a weboldalnak annak a részére, ami a felhasználóval közvetlen kapcsolatban áll. Ezt a funkciót a mi esetünkben, a keretrendszer felső rétegeként (front-end), az Angular 12 framework látja majd el.

1.4. Az Angular 12, mint a kliensoldali (front-end) programozás eszköze

Az Angular CLI egy platform és keretrendszer egyoldalas alkalmazások (single-page application) készítéséhez HTML és TypeScript használatával. Az egyoldalas alkalmazás (single-page application) olyan weblap vagy webalkalmazás, amely interakcióba lép a felhasználóval azáltal, hogy dinamikusan átírja az aktuális weboldalt a webszerverről származó új adatokkal, ahelyett, hogy új oldalakat töltene be emiatt. A cél a gyorsabb átmentek biztosítása, valamint az egységes felhasználói élmény elérése, hasonlóan egy asztali alkalmazáshoz. Tehát gyakorlatilag letöltődik egy weboldal és soha nem navigálunk át egy másikra [11, 19, 27].

Az Angular CLI az alapvető- és opcionális funkciókat az alkalmazásokba importált TypeScript könyvtárak készleteként valósítja meg. A keretrendszer alapvető építőelemei az Angular CLI komponensek, amelyek úgynevezett NgModulokba (NgModules) vannak rendezve. Az NgModulok a kapcsolódó kódot funkcionális halmazokba gyűjtik. Tehát egy alkalmazást az NgModulok halmaza határoz meg [19, 27].

Egy Angular CLI alkalmazásnak mindig van legalább egy gyökérmodulja (app.module.ts), amely lehetővé teszi a rendszerindítást. Viszont a program jellemzően sokkal több funkciómodullal is rendelkezik, amik különböző feladatokat látnak el:

- Az egyes összetevők (components) nézeteket határoznak meg, amelyek közül az Angular CLI, a programunk logikájának és adatainak megfelelően, választhat és módosíthat. Tehát ezeket az összetevőket dinamikus képernyőelemek halmazainak lehet tekinteni [27].
- Az összetevők (components) olyan speciális szolgáltatásokat is felhasználnak, amelyek a nézetekhez nem közvetlenül kapcsolódó funkciókat biztosítanak. Ezek a funkciók függőségként is beilleszthetők az összetevőkbe így a kód modulárissá, újra felhasználhatóvá és hatékonyabbá válik [27].

Viszont az információs rendszerünk működéséhez nem elegendő a hatékony webes felület és adatfeldolgozás biztosítása, hanem arról is gondoskodni kell, hogy a rendszerben szereplő adatokhoz csak az arra jogosult személyek férhessenek hozzá, valamint a megfelelő jogkörrel rendelkező személyek tudják ezeket az adatokat módosítani. Ezt pedig JSON Web Token (JWT) alapú hitelesítés segítségével fogjuk biztosítani.

1.5. Az adatok védelme és jogosultságok kezelése. JSON Web Token (JWT)

A JSON Web Token vagy röviden JWT egy szabvány az adatok biztonságos továbbításához az felhasználó és a szerver között. Az egyszerűség, a kompaktság és a könnyű használhatóság az architektúra kulcsfontosságú jellemzői [24]. A JWT-k mögött rejlő varázslat az, hogy szabványosítanak bizonyos állításokat, amelyek hasznosak néhány gyakori művelettel összefüggésben. Például az egyik ilyen gyakori művelet egy bizonyos fél személyazonosságának megállapítása. A JSON webtoken a következő-képpen néz ki (sorokra tördelve az olvashatóság kedvéért):

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJpZCI6MTgsImlhdCI6MTY1MDczNjc0OCwiZXhwIjoxNjUwODIzMTQ4fQ.
8R2Tor48F-SDWOwd50nAGdczpBpn_54twLtqoj-YHJg
```

Bár ez halandzsának tűnik, valójában egy sor állítás tartalmaz nagyon kompakt, nyomtatható ábrázolással, valamint a hitelességét igazoló aláírással együtt. A JSON Webtokenek soha nem titkosítottak, hanem Base64 (64 karakterből álló ábécén alapuló tartalomkódolási forma) alapon kódoltak. A JWT három részből áll: fejlécből, úgynevezett hasznos terhelésből és aláírásból. Minden rész ponttal (.) van elválasztva, és egymástól függetlenek, azaz nem szükséges a teljes tokent figyelembe venni a dekódoláshoz [24]. A JSON Webtoken egyes összetevői:

- Fejléc: A JSON Webtoken első összetevője a fejléc, ami információt tartalmaz például az algoritmusról, a token típustáról stb. [21, 24].

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

- Hasznos terhelés: olyan információkat tartalmazhat, mint ügyfélazonosító, cégazonosító, a token hozzáférési joga és a token lejárati dátuma [21, 24].

```
{
  "id": 18,
  "iat": 1650736456,
  "exp": 1650822856
}
```

- JSON aláírás: mind a fejléc, mind a hasznos terhelés az aláírás titkos kulcsával van összekapcsolva. Ez az aláírás őrzi meg a token integritását [21, 24].

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
) secret base64 encoded
```

A JSON Webtokeneket főleg hitelesítésre használják, mivel a bennük lévő információk digitálisan alá vannak írva egy titkos vagy nyilvános/titkos kulcspárral, tehát megbízhatóak. Működésük alapja, hogy miután a felhasználó bejelentkezetett, azaz hitelesítette magát, visszakap egy tokent, ami alapján a szerver el tudja majd dönteni, hogy az adott felhasználó milyen útvonalakhoz, tartalmakhoz, illetve szolgáltatásokhoz tud általa hozzáférni, tehát jogosultságokat is kezel [21, 24].

Összefoglalva tehát az általunk tervezett online információs rendszer kialakításához felhasznált legfontosabb eszközök a következők:

– backend: NodeJS, ExpressJS, Sequelize;
– front-end: Angular 12 CLI;
– adattárolás és adatfeldolgozás: MySQL adatbáziskezelő rendszer;
– adatok védelme és jogosultságok kezelése: JSON Web Token (JWT).

Viszont a rendszer az online adminisztráció megvalósításán túl didaktikai szereppel is bír, hiszen a hallgatók értékelése is része az adminisztrációs folyamatnak. És egy online e-leckekönyv szinte azonnali visszajelzést nyújt a hallgatóknak az eddigi teljesítményükről, az érdemjegyektől kezdődően egészen a féléves átlagokkal bezárólag. És ez a fajta visszajelzés a táv-, vagy digitális oktatás esetében rendkívül fontos és motiváló lehet a hallgatók számára [20]. De mi is a pedagógia értékelés szerepe az oktatásban?

1.6. Az értékelés szerepe az oktatásban. Webes alkalmazások az értékelési folyamatban

Az értékelés hagyományos értelemben a tanulót megcélzó tanári tevékenységként jelent meg az oktatás folyamatában, és következményei a tanulókat érintették (dicséret, bukás esetleg büntetés). Viszont napjainkban az értékelés célja nemcsak a tanulók tárgyi tudásának megismerése, esetleges mérése, hanem megjelenik az oktatás minden szintjén.

Célja lehet az oktatási rendszer, az iskola személyzete, egy tanterv vagy akár egy tanóra értékelése is, és ezáltal lehetővé teszi, hogy eredményesen szabályozni tudjuk az oktatási folyamatokat [12, 17]. Az általunk létrehozni kívánt webes adminisztrációs platform ebben nagy segítséget nyújthat a távoktatás, azaz a digitális oktatás során, mivel megkönnyítik a tantervek és elért eredmények egységes áttekinthetőségét, ami így akár távolról is elérhetővé válik.

Mégis az értékelés legfontosabb célja annak megállapítása, hogy a tanulók milyen mértékben érték el a tanítási-tanulási folyamat során a tervezett célokat és követelményeket [17]. A visszacsatoló funkció alapján az értékelésnek három típusát különböztetjük meg: diagnosztikus-, formatív- és szummatív értékelés [12].

A diagnosztikus értékelés célja, hogy a döntéshozók (pl. tantervfejlesztők, tanárok) információt szerezzenek a diákok képességeiről a tanítási folyamat megkezdése előtt. Azaz képet kapjanak azokról a területekről, ahol esetleg a tanulók lemaradtak, vagy éppen kiemelkedőek. Segítségével besorolási döntéseket hozhatunk meg, és ezáltal csoportra, vagy egyénre szabott oktatási-nevelési stratégiákat alakíthatunk ki, amelyek segítik a hatékony oktatás-szervezést [12, 17].

A formatív értékelés a tanítási folyamat közbeni sikeres irányítást teszi lehetővé. Feltárja a hibákat és nehézségeket, ami lehetővé teszi a különböző oktatási folyamatoknak a felülvizsgálatát, és ezáltal a nevelési-oktatási célok és tartalmak finomhangolását, valamint a tanulási sikerek megerősítését. Használatával akkor tudjuk a legjobb hatást elérni, ha minden elvárható tudáselemre kiterjed (mit tudnak és mit nem tudnak a tanulók) [12, 17].

A szummatív értkelés a nevelési-oktatási folyamat záró szakasza, aminek a legfontosabb célja az minősítés. A folyamat során a tanulókat csoportokba soroljuk, aminek hatására megjelenik a tanulók szelektálása, és ez kihatással van a diákok későbbi életére, akár a továbbtanulás, akár a munkába állás feltételeinek a tekintetében. A szummatív értékelés akkor hatékony, ha azonos feltételekkel méri a tanulók tudását, azaz objektív, megbízható, korrekt és nyilvános. Gyakori formája a vizsga [12, 17].

A tanári munka során fontos odafigyelni arra, hogy a diagnosztikus-, formatív- és a szummatív értékelések akkor tudják jól ellátni a feladatukat, ha egymástól elkülönítve alkalmazzuk őket, a megfelelő időben, valamint a funkcióiknak megfelelően [12].

A pedagógiai értékelés és visszajelzés, az alap-, középfokú- és a felsőoktatásban is az egyik legfontosabb tényező a hallgatók motiválásában [15, 20, 31]. A digitális személyre szabott eredményjelentések elengedhetetlenek a felsőoktatásban, mivel ezek a visszajelzések rendkívül fontos szerepet játszanak a távoktatás során a hallgatók teljesítményének és motivációjának javításában [20]. Ezért az általunk létrehozni kívánt online adminisztrációs rendszerben is kiemelkedő szerepet kap ezeknek a funkcióknak a megvalósítása.

2. FEJEZET

GYAKORLATI RÉSZ

2.1. Az adatbázis tervezése és létrehozása

Az általunk létrehozni kívánt webes információs rendszer kialakításához elengedhetetlen lépés a platform mögött működő adatbázis alapos megtervezése. A folyamat során meghatározzuk az egyedtípusokat és azok tulajdonságait, valamint megállapítjuk a tárolt egyedek között fennálló kapcsolatokat. Ezeknek az információknak a pontos definiálása alapvetően fontos az információs rendszer későbbi gördülékeny működéséhez.

2.1.1. Az adatbázis egyedei, kapcsolatai és megszorításai

A létrehozni kívánt információs rendszer feladatainak, és a meghatározott céloknak megfelelően az adatbázis a következő egyedekből tevődik össze: Tanszékek, Szakok, Tantárgyak, Munkatársak, Diákok, Diákok száma, Mintatantervek, Eredmények Tarifikációs változók, Tantárgy tarifikációk, Gyakorlat tarifikációk, valamint a Szakdolgozat és évfolyammunka tarifikációk.

Hogy elkerüljük a párhuzamos adatokat jelenlétét az adatbázisban (redudancia), valamint elejét vegyük a különböző anomáliáknak, ezért az egyedeket harmadik normálformában alakítjuk ki.

Tanszékek

A Tanszékek elnevezésű egyed, mint ahogy a neve is mutatja a II. Rákóczi Ferenc Kárpátaljai Magyar Főiskola (továbbiakban: II. RFKMF) tanszékeit fogja tárolni. Ide tartoznak a különböző tanszékek és tanszéki csoportok. A következő attribútumok halmazából áll:

id	egész szám, amely a sorok azonosítására szolgál, azaz elsődleges kulcs;
tanszek_nev	karakterlánc, ami a tanszékek neveit tárolja, nem vehet fel üres értéket és egyedinek kell lennie;
tanszek_tipus	felsorolás, ami a lehetséges tanszékek típusait tárolja, üres értéket nem vehet fel.

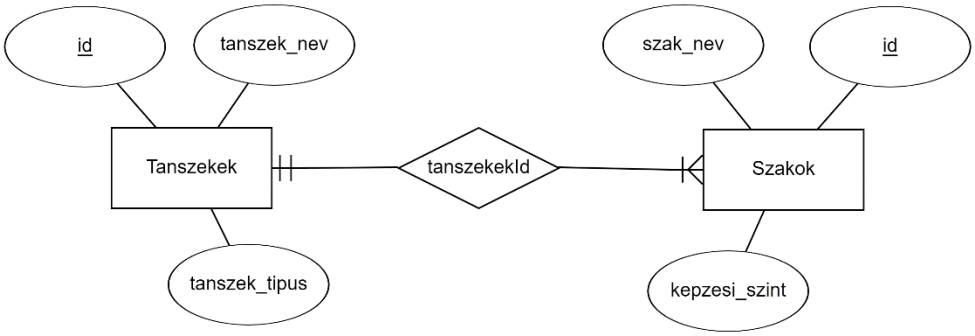
Szakok

A Szakok egyed a II. RFKMF szakjait fogja tárolni. Ezek egyértelmű azonosításához szükséges a képzési szint tárolása is, mivel léteznek ugyanazzal a szaknévvel, de különböző képzési szinttel rendelkező szakok (pl. Középiskolai oktatás (Matematika) – BSc és MSc képzési szint). Attribútumai a következők:

id	egész szám, amely a sorok azonosítására szolgál, azaz elsődleges kulcs;
szak_nev	karakterlánc, ami a szakok neveit tárolja és nem vehet fel üres értéket;
kepzesi_szint	felsorolás, ami a lehetséges képzési szinteket tárolja, üres értéket nem vehet fel.

A megszorítások meghatározásánál fontos még arra is figyelni ebben az esetben, hogy szak_nev és kepzesi_szint párosnak minden esetben egyedinek kell lennie. Ez megakadályozza a későbbiekben az ismétlődő adatok felvételének a lehetőségét.

A tanszékek és a szakok között kapcsolat áll fent. A szak mindig valamilyen tanszékhez kapcsolódik, és csakis egyhez. Viszont egy tanszékhez legalább egy, vagy akár több szak is kapcsolódhat. Tehát a két egyed között a kapcsolat egy a többhöz (2. ábra).



2. ábra. A tanszékek és szakok kapcsolata (Saját kép, 2022)

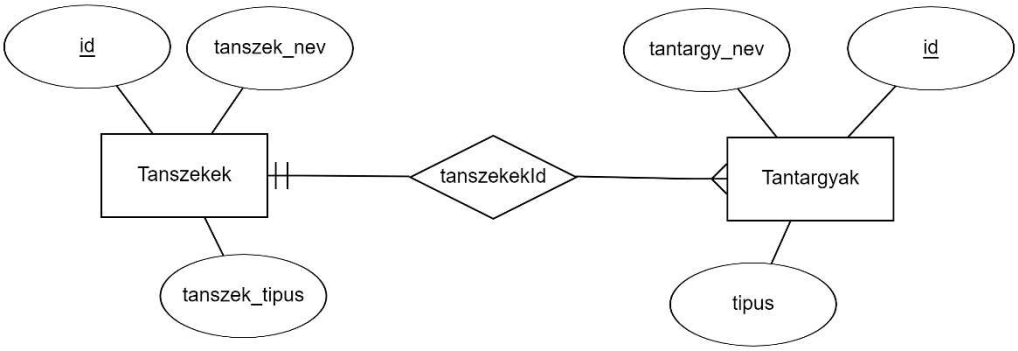
Tantárgyak

A Tantárgyak egyedtípus értelemszerűen a II. RFKMF-án oktatott tantárgyakat fogja tárolni. A tantárgyakhoz azt az információt is társítani fogjuk, hogy kötelező-, szabadon választható-, gyakorlat vagy fakultatív tantárgyról van-e szó. Továbbá mivel néhány tantárgy oktatása esetén fennállhat olyan eset, hogy egy adott tantárgyat ugyanazon a szakon belül esetleg több tanár is oktathatja egymástól függetlenül (pl. az Idegen nyelv tanítása során lehet a fele csoport angolt fog tanulni a másik fele németet), ezért a későbbi

óraelosztás (lásd: Tantárgy tarifikációk) megkönnyítése érdekében a tantárgyak típusainak a kategóriáit tovább bővítjük, ami a gyakorlatban annyit jelent, hogy ezek a tantárgyak valamilyen „osztott” jelzőt fognak kapni (pl. szabadon választható osztott). Tehát az egyed attribútumai:

id	egész szám, amely a sorok azonosítására szolgál, azaz elsődleges kulcs;
tantargy_nev	karakterlánc, ami a tantárgyak neveit tárolja, nem vehet fel üres értéket;
tipus	felsorolás, ami a lehetséges tantárgy típusokat tárolja, üres értéket is felvehet (évfolyammunka, szakdolgozat stb. esetén).

A tanszékek és a tantárgyak között kapcsolat áll fent. A tantárgy mindig valamilyen tanszékhez kapcsolódik, és csakis egyhez. Viszont egy tanszékhez több tantárgy is tartozik. Tehát a két egyed között a kapcsolat egy a többhöz (3. ábra).



3. ábra. A tanszékek és tantárgyak kapcsolata (Saját kép, 2022)

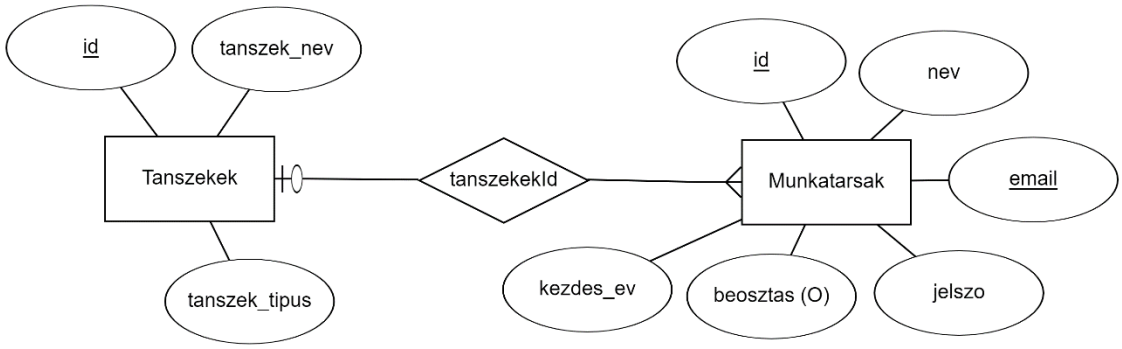
Munkatársak

A Munktársak egyed a II. RFKMF adminisztrációval és oktatással foglalkozó munkatársait fogja tárolni, beleértve a tanulmányi osztály munkatársait is. Az adminisztrációs folyamatban betöltött szerepkört JSON Web Token segítségével fogjuk meghatározni (lásd: 65. oldal). A tábla az alábbi attribútumokkal fog rendelkezni:

id	egész szám, amely a sorok azonosítására szolgál, azaz elsődleges kulcs;
nev	karakterlánc, ami a munkatársak neveit tárolja, nem vehet fel üres értéket;
beosztas	felsorolás, ami a lehetséges beosztásokat tárolja, felvehet üres értéket;
kezdes_ev	dátum, ami a munkába állás időpontját tárolja, üres értéket nem vehet fel;

email	karakterlánc, ami az adott munkatárs intézményi email címét tárolja, és a későbbi azonosítást szolgálja, csak egyedi értékeket vehet fel és nem lehet üres;
jelszo	karakterlánc, ami a munkatárs jelszavát tárolja, segítségével hozzá tud férni a saját felhasználói fiókjához, nem lehet üres.

A tanszékek és a munkatársak között kapcsolat áll fent. Egy tanszékhez mindig több munkatárs tartozik, azonban egy munkatárs csak egy tanszékhez tartozhat, vagy esetleg egyhez sem (pl. tanulmányi osztály kollégái), ezért a kapcsolat egy a többhöz (4. ábra).



4. ábra. A tanszékek és munkatársak kapcsolata (Saját kép, 2022)

Diákok

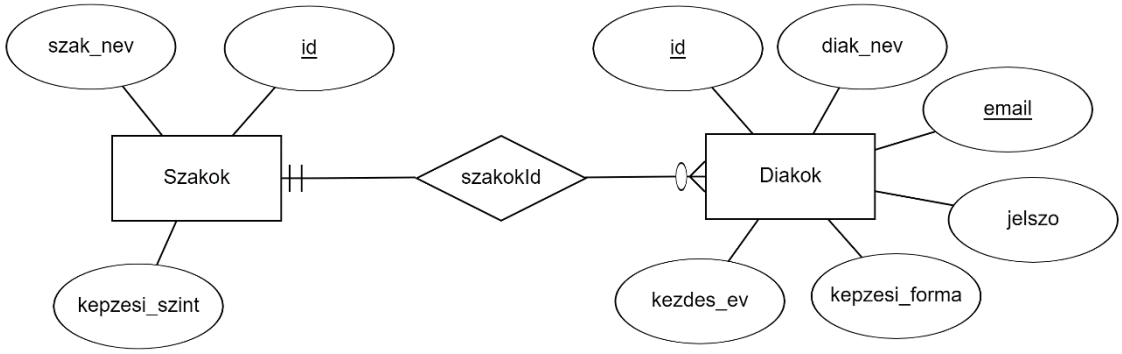
A Diákok egyed a II. RFKMF-án jogviszonnyal rendelkező hallatgatókat fogja tárolni. Ebben a relációban fogjuk elhelyezni azt is, hogy a diák milyen szakon folytat tanulmányokat. Ha egyszerre több szakon is tanul az adott hallgató, akkor a táblába is többször kerül bele a megfelelő diák-szak párosítással, valamint az intézményben belsőleg létrehozott, megfelelő egyedi email címmel együtt (ez képzési szintenként, évfolyamonként, és szakonként is eltérő azonosító kóddal rendelkezik). Ez a több a többhöz kapcsolat feloldását és ezáltal a későbbiekben a rendszer gördülékenyebb használatát teszi lehetővé. A diákok tábla az alábbi attribútumok halmazából tevődik össze:

id	egész szám, amely a sorok azonosítására szolgál, azaz elsődleges kulcs;
diak_nev	karakterlánc, ami a hallgatók neveit tárolja, nem vehet fel üres értéket;
kepzesi_forma	felsorolás, ami azt tárolja, hogy az adott diák nappali- vagy levelező tagozaton folytat-e tanulmányokat az adott szakon, nem vehet fel üres értéket;

kezdes_ev	dátum, ami a felvétel időpontját tárolja, üres értéket nem vehet fel;
email	karakterlánc, ami az adott hallgató intézményi email címét tárolja, és a későbbi azonosítást szolgálja, csak egyedi értékeket vehet fel és nem lehet üres;
jelszo	karakterlánc, ami a diák jelszavát tárolja, segítségével hozzá tud férni a saját felhasználói fiókjához, nem lehet üres.

Megjegyzés: Hogy automatikusan frissüljön a diákok évfolyama, ezért a kezdes_ev alakja mindig aktuális év+'-09-01' (tehát mindig szeptember elsején fog változni az adott diák évfolyama).

Mint ahogy már feljebb is említettük, a diákok és a szakok között kapcsolat áll fent. A diák mindig egy szakhoz tartozik (ha egy diák pl. két szakon tanul egyszerre, akkor kétszer kerül bele a táblába egyedi intézményi email címmel, azaz két különböző diáknak számít a fent leírtaknak megfelelően), viszont egy szakhoz több hallgató is tartozhat vagy esetleg egy sem (pl., ha nincs elég jelentkező és nem indul el a szak), ezért a kapcsolat egy a többhöz (5. ábra).



5. ábra. A szakok és diákok közötti kapcsolat (Saját kép, 2022)

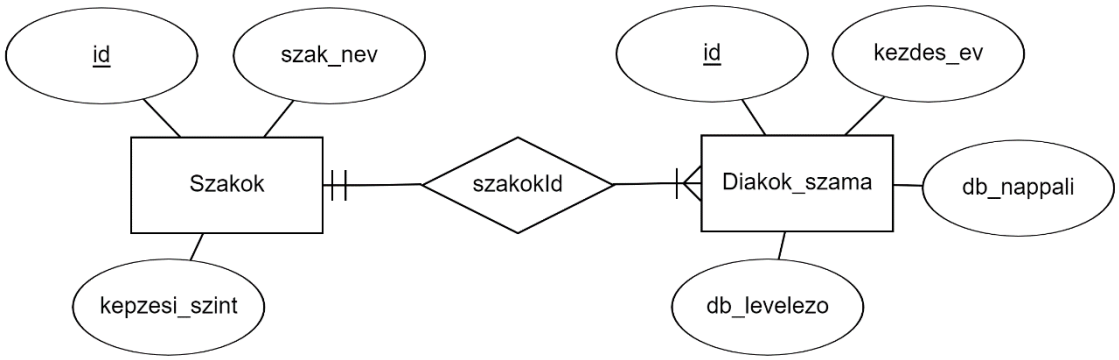
Diákok száma

A Diákok száma egyed a II. RFKMF-án aktuálisan tanulmányt folytató diákoknak a számát tárolja, minden szakra és évfolyamra külön-külön. A tábla megfelelő cellájának értéke (szakokId és kezdes_ev alapján) minden regisztráció alkalmával növekszik, vagy ha esetleg kitörlünk egy diákot a rendszerből, akkor csökkeni fog (*Megjegyzés*: ezt két trigger segítségével oldottuk meg, lásd: 42. old). A levelezős és nappalis diákoknak a számát külön-külön attribútumokban tároljuk, így a későbbiek során könnyebben kérdezhetünk le különböző statisztikai adatokat (pl. hány levelezős diák tanul az intézményben). Ennek megfelelően a tábla az alábbi oszlopokkal fog rendelkezni:

id	egész szám, amely a sorok azonosítására szolgál, azaz elsődleges kulcs;
db_nappali	egész szám, ami az adott évfolyam, adott szakján tanulmányt folytató nappalis diákoknak a számát tárolja;
db_levelezo	egész szám, ami az adott évfolyam, adott szakján tanulmányt folytató levelezős diákoknak a számát tárolja;
kezdes_ev	dátum, ami a diákok felvételének az időpontját tárolja, üres értéket nem vehet fel.

A megszorítások meghatározásánál fontos még arra is figyelni a Diákok száma tábla esetében, hogy a szakokId és kezdes_ev párosnak minden esetben egyedinek kell lennie. Ez megakadályozza a későbbiekben az ismétlődő adatok felvételének a lehetőségét.

A diákok száma és szakok reláció között kapcsolat áll fent. A diákok száma tábla egy adott sorához mindig csak egy szak tartozik (ezt a feljebb definiált megszorítás is biztosítja), viszont egy szakhoz több (különböző kezdési éveket tartalmazó) diákok száma tábla bejegyzés is tartozhat (*Megjegyzés:* akkor is tárolunk adatot, ha eddig még nem indult el a szak, hiába meg volt hirdetve; vagy ha éppen csak az adott évben nem indult el a csoport). Ezért a két tábla közötti kapcsolat egy a többhöz (6. ábra).



6. ábra. A diákok száma és szak közötti kapcsolat (Saját kép, 2022)

Mintatantervek

A Mintatantervek egyed a II. RFKMF különböző szakjainak a mintatanterveit fogja tárolni. Amikor mintatantervet készítünk, akkor gyakorlatilag összekötjük a szakokat a tantárgyakkal (*Megjegyzés*: ez több a többhöz kapcsolat lenne, ezért van szükségünk a mintatanterv kapcsolótáblára). Azonban figyelnünk kell arra, hogy egy tantárgyat a mintantervben megfelelően elhelyezni csak úgy lehet, ha a szak-tantárgy párosításhoz hozzácsatoljuk a félévet is, mivel nem mindegy az, hogy egy tantárgyat csak egy- vagy

akár több féléven keresztül is tanulja-e a diák. Tehát ez alapján már kijelenthetjük, hogy ebben az esetben egy adott szak adott félévében egy tantárgy csak egyszer fog szerepelni. Továbbá a táblában egyszerre jelen lehet egy adott mintatanterv több változata is, mivel ezek a programok időről időre módosulnak. Azért, hogy ezt kezelni tudjuk érdemes egy kezdési évet is társítanunk a szak-tantárgy-félév hármashoz, mivel így tudjuk a későbbiekben egyértelműen eldönteni, hogy az adott diákra melyik mintatanterv is vonatkozik konkrétan. Valamint a tábla tartalmazni fogja az adott tantárgy oktatására vonatkozó adminisztrációs és oktatási adatokat is, az adott szak, kezdési év, félév hármason belül (kredit, szeminárium óraszám, önálló munka stb.). Az előbbiek alapján a tábla (vagy kapcsolt tábla) attribútumai:

id	egész szám, amely a sorok azonosítására szolgál, azaz elsődleges kulcs;
kezdes_ev	dátum, ami a mintatanterv érvényességének a kezdeti időpontját tárolja a diák kezdési évéhez igazítva, üres értéket nem vehet fel;
felev	felsorolás, ami azt mutatja meg, hogy az adott tantárgyat a mintatanterv aktuális pontján melyik félévben, vagy félévekben tanítják, és nem vehet fel üres értéket;
kredit	egész szám, a tantárgy kreditértékét tárolja a mintatanterv aktuális pontján, nem lehet üres;
tipus	felsorolás, ami azt tárolja, hogy a mintatanterv aktuális pontján az adott tantárgy vizsgával vagy beszámolóval zárul-e, nem vehet fel üres értékeket;
eloadas	egész szám, ami a mintatanterv aktuális pontján az előadások óraszámát tárolja, értéke nem lehet üres;
laboratoriumi	egész szám, ami a mintatanterv aktuális pontján a laboratóriumi foglalkozások óraszámát tárolja, értéke nem lehet üres;
gyakorlati	egész szám, ami a mintatanterv aktuális pontján a gyakorlati foglalkozások óraszámát tárolja, értéke nem lehet üres;
onallo_munka	egész szám, ami a tantárgy tanulása során önálló munkára fordítandó óraszámot tárolja a mintatanterv aktuális pontján, értéke nem lehet üres.

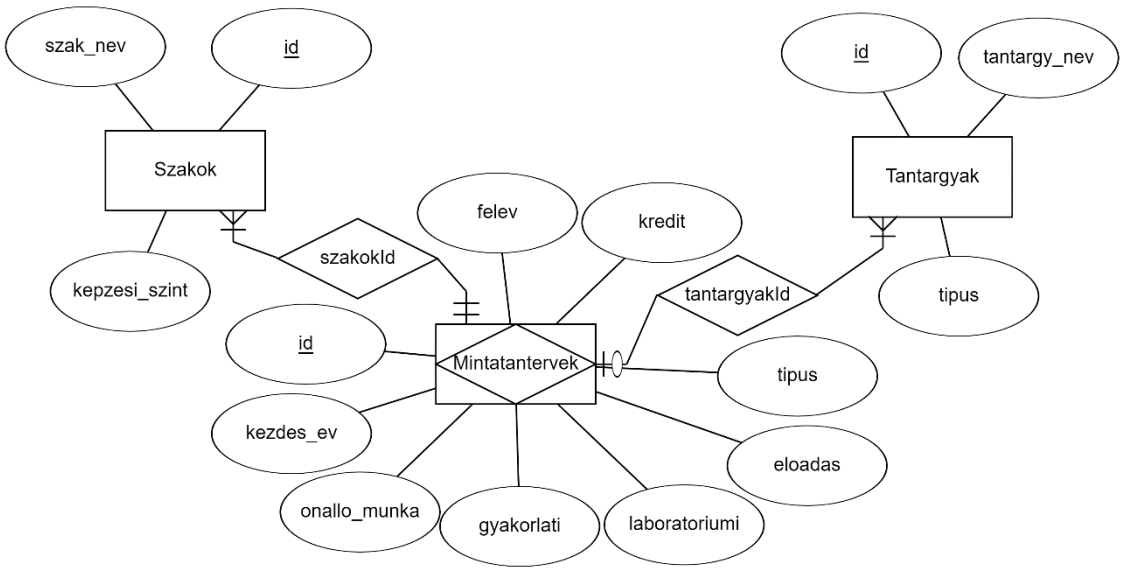
A megszorítások meghatározásánál fontos még arra is figyelni, hogy a szakokId, felev, kezdes_ev és tantargyakId négyesnek minden esetben egyedinek kell lennie, azaz egy adott évi programban, egy szaknak az adott félévén belül egy tantárgy csak egyszer

szerepelhet. Ez megakadályozza a későbbiekben az ismétlődő adatok felvételének a lehetőségét.

A mintatantervek és a szakok reláció között kapcsolat van. A fent megfogalmazott megszorítások és követelmények következményeként egy szakhoz az adott tantárgy, kezdési év, félév hármason belül egyetlen mintatanterv sor tartozik. Viszont adott tantárgy, kezdési év, félév hármas által meghatározott mintatanterv sorhoz több szak is tartozhat (pl. Pedagógiai, 2021/2022 évfolyam, 1 félév: Matematika szak, Biológia szak stb.). Tehát a kapcsolat egy a többhöz (7. ábra).

Továbbá a mintantervek és a tantárgyak reláció között is kapcsolat áll fent. Egy tantárgyhoz az adott szak, kezdési év, félév hármason belül csakis egyetlen mintatanterv sor kapcsolódik, vagy egy sem (pl. nem tanulja jelenleg senki az aktuális tantárgyat). De egy szak, kezdési év, félév hármas által meghatározott mintatanterv sorhoz akár több tantárgy is kapcsolódhat (pl. Matematika szak, 2021/2022 évfolyam, 1 félév: Pedagógia, Lineáris algebra stb.)

Tehát a mintatanterv relációt egy speciális kapcsolt táblának is fel lehet fogni a tantárgyak és szakok között (7. ábra).



7. ábra. A tantárgy, mintatanterv és szak közötti kapcsolat (Saját kép, 2022)

Eredmények

Az Eredmények egyed a II.RFKMF hallgatóinak a vizsga- és beszámoló eredményeit fogja tárolni. Tehát ez gyakorlatilag egy online leckekönyv, amiben diákokat, tantárgyakat és tanárokat kötünk össze.

Vizsgáljuk meg először a diákok és tantárgyak kapcsolatát. Egy tantárgyat tanulhat több diák is vagy egy sem, viszont egy diák egyszerre több tantárgyat is tanul, tehát a kapcsolat több a többhöz. Ezt a kapcsolatot az eredmények kapcsolótábla segítségével tudjuk létrehozni. Viszont, mivel egy tantárgyat a diák akár több féléven keresztül is tanulhat, ezért a kapcsolótáblában az egyértelműség biztosítása érdekében egy félév oszlopot is kell tárolnunk. Továbbá mivel egy virtuális leckekönyvről van szó, ezért jegybeírás is történik. Így egy adott diák, tantárgy, félév hármashoz a legrosszabb esetben 3 beírás is tartozhat, így az egyértelműség biztosítása miatt tárolni fogjuk a kapcsolt táblában az eredmény beírásának a dátumát is. Tehát egy eredmények tábla beírást egyértelműen a diák, tantárgy és dátum hármas fog meghatározni. Viszont szükséges a félévet továbbra is tárolni, mivel egy félévben a diák egy tantárgyból maximum háromszor vizsgázhat, és így ez az adat a program helyes működésének szintén hozzá fog járulni.

Az reláció helyes kialakításának az érdekében fontos még megvizsgálni mélyebben az előbb említett pótvizsgázás következményét is. Ha egy diák megbukik egy tantárgyból pótvizsga- vagy pótbeszámoló díjat kell befizetnie, és csak ezután vizsgázhat megint. Így a táblában tárolnunk kell azt is, hogy a diák vizsgához van-e engedve az adott félévben az adott tárgyból az adott időpontban vagy sem. Az első vizsgánál ez alapértelmezetten engedélyezve lesz, második- vagy harmadik esetben viszont alapértelmezetten nem, amit a tanulmányi osztály tud majd átállítani, ha az adott diák befizette a pótvizsga vagy pótbeszámoló díját.

Most pedig vizsgáljuk meg a diákok és tanárok kapcsolatát. Egy diákot több tanár is taníthat, és egy tanár több diákot is tanít, azaz osztályoz egyidejűleg. Így a kapcsolat megint több a többhöz, és ezt a kapcsolatot is az eredmény kapcsolótáblán keresztül fogjuk létrehozni, amit a tantárgy és dátum oszlopok fognak egyértelműen meghatározni.

És végül a tanárok és tantárgyak kapcsolatát tekintsük át. Egy tantárgyat akár több különböző tanár is oktathat az adott időpontban a különböző diákoknak (pl. Idegen nyelv: angol és német), és egy tanár több tantárgyat is oktat, ezért ebben az esetben is több a

többhöz kapcsolatról van szó. Ezt a kapcsolatot is az eredmény kapcsolótáblán keresztül valósítjuk meg, amit a diák és dátum oszlopok fognak egyértelműen meghatározni. Viszont mivel vannak olyan tantárgyak, ahol a szemináriumot (gyakorlatot) és az előadást két különböző tanár is oktatja az adott diákcsoporton (vagy szakon) belül, ezért az eredmények tábla egy sorában két tanár (szeminárium- és előadás tanár) is meg fog jelenni, és mindkét tanár tud majd osztályozni, a rendszer gördülékeny használatának biztosítása érdekében. A tábla attribútumai a papír alapú leckekönyvhöz hasonlóak lesznek:

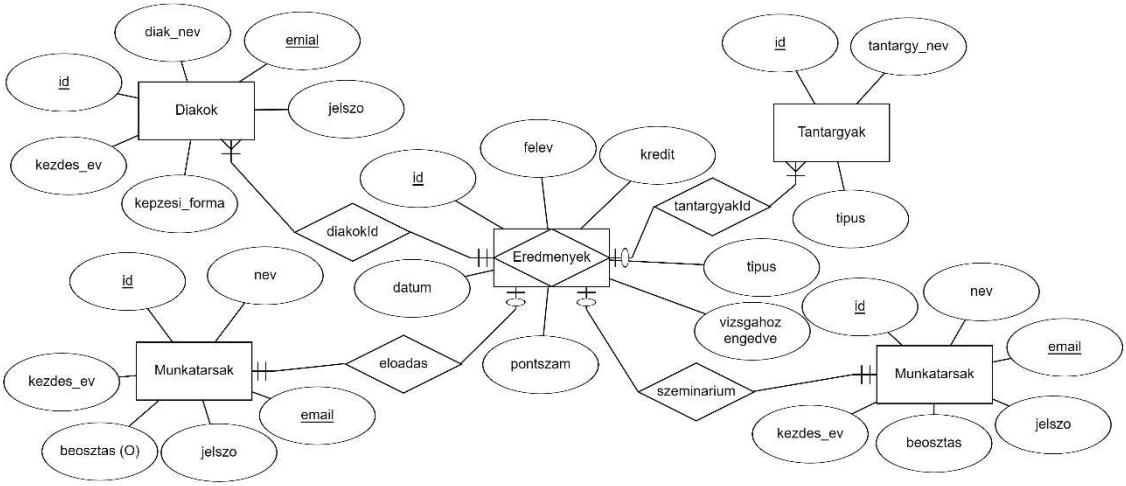
id	egész szám, amely a sorok azonosítására szolgál, azaz elsődleges kulcs;
felev	felsorolás, ami azt mutatja meg, hogy az adott tantárgyból a diák melyik félévben, vagy félévekben kap osztályzatot, és nem vehet fel üres értéket;
kredit	egész szám, a tantárgy kreditértékét tárolja az adott félévben, nem lehet üres;
tipus	felsorolás, ami azt tárolja, hogy az adott félévben az adott tantárgy vizsgával vagy beszámolóval zárul-e, nem vehet fel üres értékeket;
vizsgahoz_engedve	logikai érték, ami azt mutatja, hogy a hallgató vizsgázhat-e az adott féléven az adott tantárgyból az adott időpontban (pótvizsga esetén van jelentősége), értéke nem lehet üres;
pontszam	egész szám, ami a diák adott tantárgy, adott félévben, adott időpontban elért eredményét tárolja;
datum	dátum, ami azt mutatja meg, hogy mikor vizsgázott a diák az adott tantárgyból az adott félévben.

A megszorítások meghatározásánál fontos még arra is figyelni, hogy a diakokId, tantargyakId, félév és datum négyesnek minden esetben egyedinek kell lennie, azaz egy diáknak egy tantárgyból, egy adott félévben jegybeírás az adott dátumon belül csak egyszer szerepelhet. Ez megakadályozza a későbbiekben az ismétlődő adatok felvételének a lehetőségét.

Az eredmények és a diákok reláció között kapcsolat áll fent. A fent megfogalmazott megszorítások következményeként egy diákhoz a tantárgy és dátum pároson belül egyetlen eredmény sor tartozik. Viszont adott tantárgy, dátum páros által meghatározott eredmény sorhoz több diák is tartozhat (pl. Pedagógia, 2021-01-18: Dudás János, Szabó Ádám stb.). Tehát a kapcsolat egy a többhöz.

Továbbá az eredmények és tantárgyak reláció között is kapcsolat áll fent. Egy tantárgyhoz a diák, dátum pároson belül egyetlen eredmény sor tartozik, vagy egy sem (nem szerepel a tantárgy jelenleg a tantervben). Viszont adott diák, dátum páros által meghatározott sorhoz több tantárgy is tartozhat (pl. két pótbeszámoló van ugyanazon a napon). Tehát a kapcsolat egy a többhöz.

És az eredmények és munkatársak relációk között is kapcsolat van. Egy munkatárshoz (tanárhoz) az adott diák, tantárgy, dátum hármas által meghatározott eredmény sor tartozik, vagy egy sem (pl. tanulmányi osztály munkatárs), és a diák, tantárgy, dátum hármas is egy tanárt (vagy tanár páros) határoz meg, ezért a kapcsolat egy az egyhez.



8. ábra. Az eredmények diákok, tantárgyak és munkatársak kapcsolata (Saját kép, 2022)

Tarifikációs változók

Az Tarifikációs változok egyed az aktuális tantervi program egy adott pontján szereplő tantárgy helyes óraterhelésének a kiszámításában (tarifikációjának) fog segítséget nyújtani. Ezekre a változókra azért van szükség, mert a képzési program egy adott pontján szereplő tantárgy óraterhelése a tantárgyat tanuló diákok számától is függ. Közvetlen kapcsolata más táblával nem lesz. A tábla a következő attribútumokból épül fel:

id	egész szám, amely a sorok azonosítására szolgál, azaz elsődleges kulcs;
onallo_munka	numerikus érték, önálló munkához tartozó óraszám-szorzó;
dolgozatjavitas_ eloadas	numerikus érték, nappali tagozaton az előadáshoz tartozó dolgozatjavítós óraszám-szorzó;

dolgozatjavitas_ szeminarium	numerikus érték, nappali tagozaton a szemináriumhoz tartozó dolgozatjavítós óraszám-szorzó;
dolgozatjavitas_ levelezo	numerikus érték, levelező tagozaton a dolgozatjavításhoz tartozó óraszám-szorzó;
vizsga_ konzultacio	numerikus érték, vizsga előtti konzultációra adott óraszám az adott csoportban;
beszamolo	numerikus érték, beszámolóra adott óraszám az adott csoportban;
vizsga	numerikus érték, vizsgához tartozó óraszám-szorzó;
bsc2	numerikus érték, 2. évfolyamos évfolyammunkára adott éves óraszám (egy diákra);
bsc3	numerikus érték, 3. évfolyamos évfolyammunkára adott éves óraszám (egy diákra);
bsc4	numerikus érték, 4. évfolyamos szakdolgozatra adott éves óraszám (egy diákra);
msc1	numerikus érték, 1. évfolyamos diplomamunkára adott éves óraszám (egy diákra);
msc2	numerikus érték, 2. évfolyamos diplomamunkára adott éves óraszám (egy diákra).

Tantárgy tarifikációk

A Tantárgy tarifikációk egyed fogja tárolni az óraterhelés kiszámításához szükséges alapadatokat a II. RFKMF aktuális tanévi programjának megfelelően. Valamint a reláció segítségével össze tudjuk kötni a tantárgyakat és a munkatársakat (tanárokat) és ezáltal egy adott tanár tantárgyak oktatásából származó óraterhelését is ki tudjuk számítani, tehát egyfajta speciális kapcsolt táblának is lehet tekinteni. Mivel vannak olyan tantárgyak, ahol a szemináriumot (gyakorlatot) és az előadást két különböző tanárnak is kioszthatjuk egy adott tantárgyon belül, ezért az tantárgy tarifikáció tábla egy sorában két tanár is meg fog jelenni, így biztosítani tudjuk a későbbiek során a rendszer gördülékeny működését. A tábla az alábbi attribútumokból áll:

id	egész szám, amely a sorok azonosítására szolgál, azaz elsődleges kulcs;
felev	felsorolás, a tantárgy helyét mutatja a diákok programjában;

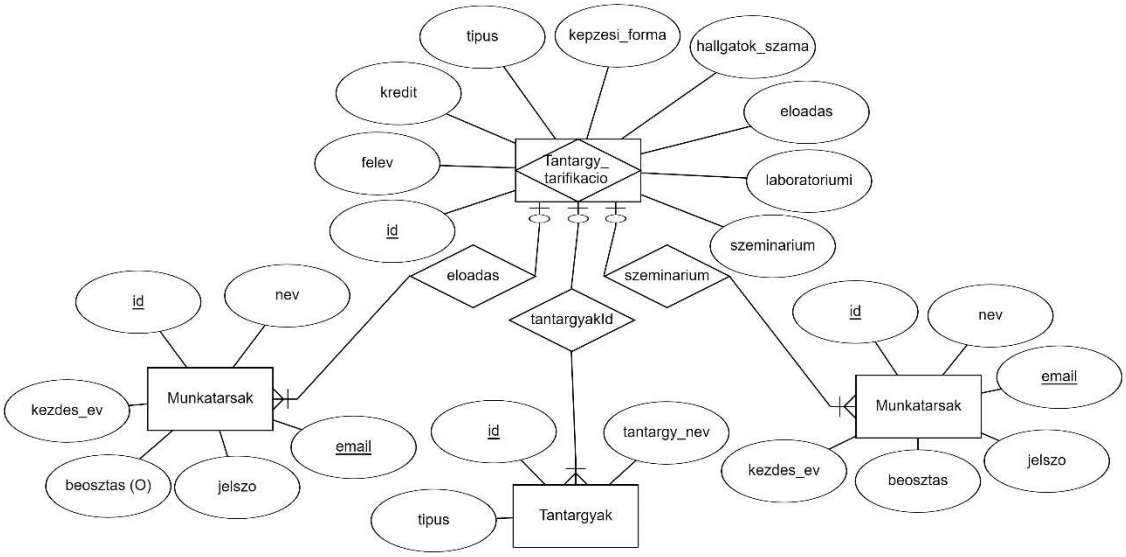
kredit	egész szám, a tantárgy kreditértékét tárolja az adott félévben, nem lehet üres;
tipus	felsorolás, ami azt tárolja, hogy az adott félévben az adott tantárgy vizsgával vagy beszámolóval zárul-e, nem vehet fel üres értékeket;
kepzesi_forma	felsorolás, azt tárolja, hogy a tantárgyat nappali- vagy levelező képzési formán részt vevő diákok hallgatják-e;
hallgatok_szama	egész szám, ami azt tárolja, hogy hány diák hallgatja a tantárgyat a program adott pontján, üres értéket nem vehet fel;
eloadas	egész szám, a program adott pontján a tantárgyra fordított előadás-kontaktórák számát tárolja;
szeminarium	egész szám, a program adott pontján a tantárgyra fordított szeminárium-kontaktórák számát tárolja;
laboratoriumi	egész szám, a program adott pontján a tantárgyra fordított laboratóriumi-kontaktórák számát tárolja.

A megszorítások meghatározásánál fontos még arra is figyelni, hogy a tantargyakId, felev, kepzesi_forma, munkatarsakId és szeminarium_munkatarsakId ötösnek minden esetben egyedinek kell lennie. Azért kerül a megszorításba bele a két tanár azonosító, mivel néhány tantárgy oktatása esetén fennállhat olyan eset, hogy egy adott tantárgyat ugyanazon a szakon belül esetleg több tanár is oktathatja egymástól függetlenül (pl. az Idegen nyelv tanítása során lehet a fele csoport angolt fog tanulni a másik fele németet). Ez megakadályozza a későbbiekben az ismétlődő adatok felvételének a lehetőségét.

A tantárgy tarifikációk és tantárgyak reláció között kapcsolat van. Egy tantárgyhoz az adott félév, tanár (vagy szeminárium- és előadás tanár), képzési forma hármason belül egyetlen tantárgy tarifikációs sor tartozik, vagy egy sem (ha éppen nincs a programban a tantárgy). Viszont adott félév, tanár (vagy szeminárium- és előadás tanár), képzési forma hármas által meghatározott tantárgy tarifikációs sorhoz több tantárgy is tartozhat (pl. Nagy Csaba (vagy Nagy Csaba és Kiss Lilla), 1 félév, nappali: Pedagógia, Szociológia stb.). Tehát a kapcsolat egy a többhöz (9. ábra).

Továbbá a tantárgy tarifikációk és munkatársak reláció között is kapcsolat van. Egy munkatárshoz (vagy szeminárium- és előadás munkatárs párhoz) az adott félév, tantárgy, képzési forma hármason belül egyetlen tantárgy tarifikációs sor tartozik, vagy

egy sem (ha a munkatárs nem tanár). Viszont a félév, tantárgy, képzési forma hármas által meghatározott tantárgy tarifikációs sorhoz több munkatárs (vagy szeminárium- és előadás munkatárs pár) is tartozhat (pl. 1. félév, nappali, Idegen nyelv: Kiss Anna (vagy Kiss Anna és Fekete Pál), Fekete Irén (vagy Nagy Irma, Asztalos Károly stb.) stb.). Tehát a kapcsolat egy a többhöz (9. ábra).



9. ábra. A tantárgyak, tantárgy tarifikációk és munkatársak kapcsolata (Saját kép, 2022)

Gyakorlat tarifikációk

A Gyakorlat tarifikációk egyed fogja tárolni a különböző gyakorlatok (pedagógiai gyakorlat, terepgyakorlat) óraterhelésének a kiszámításához szükséges alapadatokat a II. RFKMF aktuális tanévi programjának megfelelően. Valamint a reláció segítségével össze tudjuk kötni a gyakorlatokat és a munkatársakat (tanárokat) és ezáltal egy adott tanár gyakorlati óraterhelését is ki tudjuk számítani. Azért van szükség a tábla létrehozására, mivel az általános tantárgyakkal ellentétben a gyakorlatok egy szakon belül konkrét óraterheléssel bírnak és ezek az értékek nem függenek olyan adatoktól mint a diákok száma (csak annyi a kikötés, hogy aktív legyen a szak). Az előbbiek alapján a tábla az alábbi attribútumokból áll:

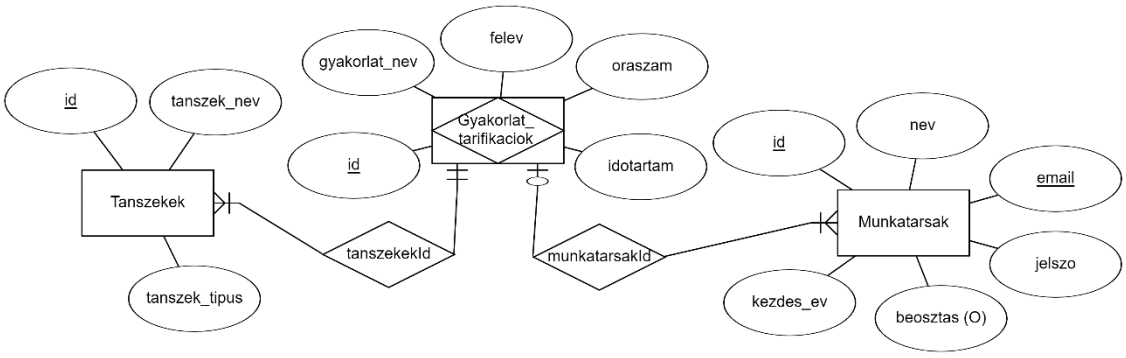
id	egész szám, amely a sorok azonosítására szolgál, azaz elsődleges kulcs;
gyakorlat_nev	karakterlánc, amit a gyakorlat nevét tárolja, nem vehet fel üres értéket;
felev	felsorolás, a gyakorlat helyét tarolja az adott szak programjában;

oraszam	egész szám, a program adott pontján a gyakorlatra fordított óraszámot tárolja, nem vehet fel üres értéket;
idotartam	egész szám, amit a gyakorlat hosszát tárolja hetekben, üres értéket nem vehet fel.

A megszorítások meghatározásánál fontos még arra is figyelni, hogy a gyakorlat_nev, felev és tanszekekId hármasnak minden esetben egyedinek kell lennie. Azért kerül bele a megszorításba a félév, mivel egy adott pedagógiai gyakorlat több félévben is szerepelhet, különböző hossz- és óraterhelés értékekkel (pl. Pedagógiai gyakorlat (Matematika és Informatika tanszék): 5. félév – 2 hét, 7 félév – 4 hét stb.). Ez megakadályozza a későbbiekben az ismétlődő adatok felvételének a lehetőségét.

A gyakorlat tarifikációk és tanszékek reláció között kapcsolat van. Egy tanszékhez az adott félév, gyakorlat név pároson belül egyetlen gyakorlat tarifikációs sor tartozik. Viszont adott félév és gyakorlat név által meghatározott gyakorlat tarifikációs sorhoz több tanszék is tartozhat (pl. 1. félév, Pedagógiai gyakorlat: Biológia és Kémia Tanszék, Matematika és Informatika tanszék stb.). Tehát a kapcsolat egy a többhöz (10. ábra).

Továbbá a gyakorlat tarifikáció és munkatársak reláció között is kapcsolat van. Egy munkatárshoz az adott gyakorlat név, félév pároson belül egyetlen gyakorlat tarifikációs sor tartozik, vagy egy sem (ha a tanár nem vezet gyakorlatot). Viszont adott félév, gyakorlat név páros által meghatározott gyakorlat tarifikációs sorhoz több munkatárs is tartozhat (pl. Pedagógiai gyakorlat, 1. félév: Kiss Anna (Matematika és Informatika tanszék), Fekete Irén (Biológia és Kémia Tanszék stb.). Tehát a kapcsolat egy a többhöz (10. ábra).



10. ábra. A gyakorlat tarifikációk, tanszékek és munkatársak kapcsolata (Saját kép, 2022)

Szakdolgozat és évfolyammunka tarifikációk

A Szakdolgozat évfolyammunka tarifikációk egyed fogja tárolni az évfolyam-munkák, szakdolgozatok és diplomamunkák óraterhelésének a kiszámításához szükséges alapadatokat a II. RFKMF aktuális tanévi programjának megfelelően. A relációban gyakorlatilag a munkatársakat (tanárokat) és a szakokat kötjük össze a Szakdolgozat évfolyammunka tarifikációk kapcsolt tábla segítségével. A tábla az alábbi attribútumokból áll:

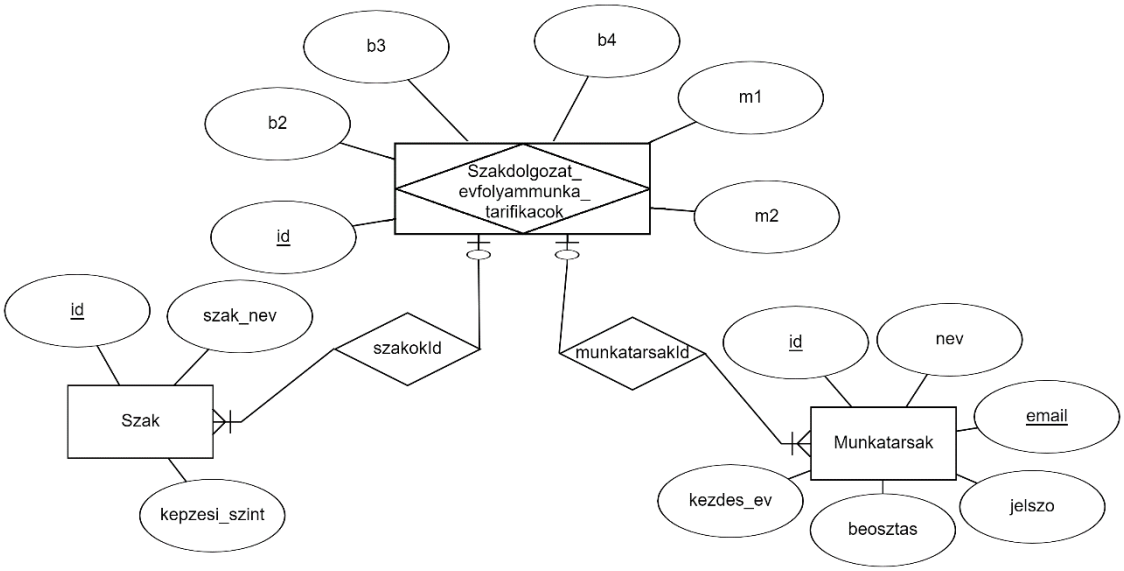
id	egész szám, amely a sorok azonosítására szolgál, azaz elsődleges kulcs;
b2	egész szám, ami az adott BSc-s szakon az adott tanárnál 2. évfolyamon évfolyammunkát írók számát tárolja, nem lehet üres;
b3	egész szám, ami az adott BSc-s szakon az adott tanárnál 3. évfolyamon évfolyammunkát írók számát tárolja, nem lehet üres;
b4	egész szám, ami az adott BSc-s szakon az adott tanárnál 4. évfolyamon szakdolgozatot írók számát tárolja, nem lehet üres;
m1	egész szám, ami az adott MSc-s szakon az adott tanárnál 1. évfolyamon diplomamunkát írók számát tárolja, nem lehet üres;
m2	egész szám, ami az adott MSc-s szakon az adott tanárnál 2. évfolyamon diplomamunkát írók számát tárolja, nem lehet üres;

A megszorítások meghatározásánál fontos még arra is figyelni, hogy a munkatarsakId, szakokId párosnak minden esetben egyedinek kell lennie. Ez megakadályozza a későbbiekben az ismétlődő adatok felvételének a lehetőségét.

A szakdolgozat és évfolyammunka tarifikációk és szakok reláció között kapcsolat van. Egy szakhoz az adott munkatárson belül egyetlen szakdolgozat és évfolyammunka tarifikációs sor tartozik, vagy egy sem (ha nincs a szakon diák). Viszont adott munkatárs által meghatározott szakdolgozat és évfolyammunka sorhoz több szak is tartozhat (pl. Nagy Áron: Matematika szak, Informatika szak stb.). Tehát a kapcsolat egy a többhöz (11. ábra).

Továbbá a szakdolgozat és évfolyammunka tarifikáció és munkatársak reláció között is kapcsolat van. Egy munkatárshoz az adott szakon belül egyetlen szakdolgozat és évfolyammunka tarifikációs sor tartozik, vagy egy sem (ha a tanárnál az adott szakon nem írnak se évfolyammunkát, se szakdolgozatot). Viszont adott szak által meghatározott

szakdolgozat és évfolyammunka tarifikációs sorhoz több munkatárs is tartozhat (pl. Matematika szak: Kiss Anna, Mészáros Irén stb.). Tehát a kapcsolat egy a többhöz (11. ábra).



11. ábra. Szakdolgozat és évfolyammunka tarifikációk, szakok és munkatársak kapcsolata (Saját kép, 2022)

2.1.2. Az adatbázis kapcsolatrendszere

Az `abrakoczi` adatbázis tábláinak a kapcsolatrendszerét az 12. ábra mutatja. A diagramot a https://dbdiagram.io/home weboldal segítségével készítettük el.

2.1.3. Az adatbázis sémája MySQL-ben

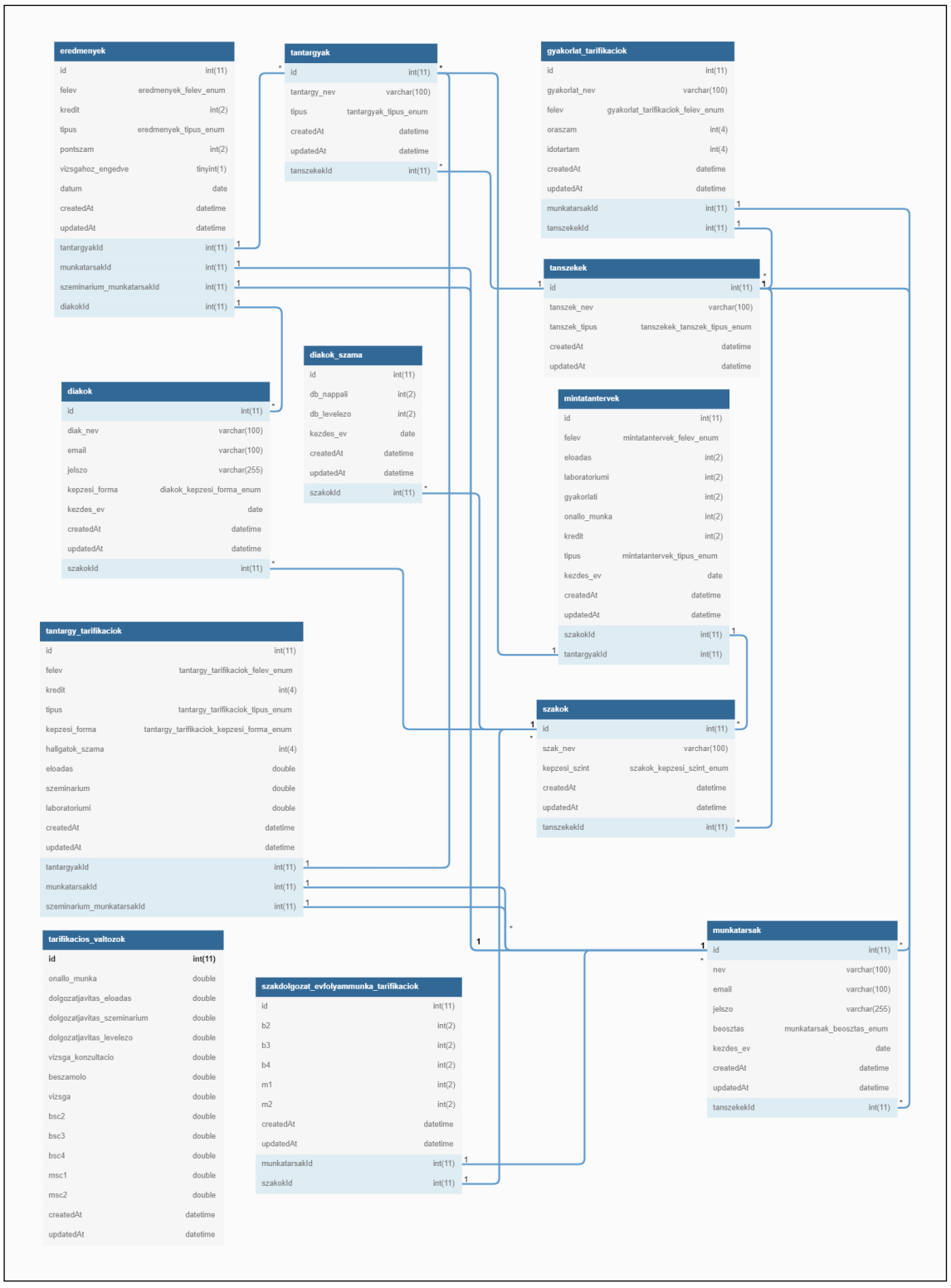
Az adatbázist `abrakoczi` néven hoztuk létre az alábbi MySQL utasítás segítségével:

```
CREATE DATABASE `abrakoczi`/*!40100 DEFAULT CHARACTER SET utf8mb4 */
```

A táblák létrehozásának sorrendje az idegenkulcs hivatkozások miatt nem választható meg tetszőlegesen. Az alábbiakban megadott sorrend egy lehetséges helyes létrehozási sorrendet tükröz.

A kódokat az XAMPP Control Panel v3.3.0 phpMyAdmin[1] programjának a segítségével készítettük el.

[1] Letölthető: https://www.apachefriends.org/hu/download.html



12. ábra. `Az abrakoczi` adatbázis kapcsolatrendszere (Saját kép, 2022)

A ***Tanszékek*** relációt létrehozó MySQL kód:

```
CREATE TABLE `tanszekek` (
  `id` int(11) NOT NULL,
  `tanszek_nev` varchar(100) NOT NULL,
  `tanszek_tipus` enum('tanszék','tanszéki csoport') NOT NULL,
  `createdAt` datetime NOT NULL,
```

```
  `updatedAt` datetime NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
ALTER TABLE `tanszekek`
  ADD PRIMARY KEY (`id`),
  ADD UNIQUE KEY `tanszek_nev` (`tanszek_nev`);
ALTER TABLE `tanszekek`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;
COMMIT;
```

A ***Szakok*** relációt létrehozó MySQL kód:

```
CREATE TABLE `szakok` (
  `id` int(11) NOT NULL,
  `szak_nev` varchar(100) NOT NULL,
  `kepzesi_szint` enum('BA','BSc','MA','MSc',) NOT NULL,
  `createdAt` datetime NOT NULL,
  `updatedAt` datetime NOT NULL,
  `tanszekekId` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
ALTER TABLE `szakok`
  ADD PRIMARY KEY (`id`),
  ADD UNIQUE KEY `szakKepzesiSzint` (`szak_nev`,`kepzesi_szint`),
  ADD KEY `tanszekekId` (`tanszekekId`);
ALTER TABLE `szakok`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;
ALTER TABLE `szakok`
  ADD CONSTRAINT `szakok_ibfk_1` FOREIGN KEY (`tanszekekId`)
REFERENCES `tanszekek` (`id`) ON DELETE SET NULL ON UPDATE CASCADE;
COMMIT;
```

A ***Tantárgyak*** relációt létrehozó MySQL kód:

```
CREATE TABLE `tantargyak` (
  `id` int(11) NOT NULL,
  `tantargy_nev` varchar(100) NOT NULL,
  `tipus` enum('általános felkészítés','általános felkészítés
              osztott','szakmai felkészítés','szakmai felkészítés
              osztott','szabadon választható','szabadon választható
              osztott' , 'gyakorlat', 'fakultatív'),
  `createdAt` datetime NOT NULL,
  `updatedAt` datetime NOT NULL,
  `tanszekekId` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
ALTER TABLE `tantargyak`
  ADD PRIMARY KEY (`id`),
  ADD UNIQUE KEY `nevTipusTanszek`
                (`tantargy_nev`,`tipus`,`tanszekekId`),
  ADD KEY `tanszekekId` (`tanszekekId`);
ALTER TABLE `tantargyak`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;
ALTER TABLE `tantargyak`
  ADD CONSTRAINT `tantargyak_ibfk_1` FOREIGN KEY (`tanszekekId`)
REFERENCES `tanszekek` (`id`) ON DELETE SET NULL ON UPDATE CASCADE;
COMMIT;
```

A ***Munkatársak*** relációt létrehozó MySQL kód:

```
CREATE TABLE `munkatarsak` (
  `id` int(11) NOT NULL,
  `nev` varchar(100) NOT NULL,
  `email` varchar(100) NOT NULL,
  `jelszo` varchar(255) NOT NULL,
```

```
  `beosztas`
enum('professzor','docens','adjunktus','oktató','tudományos
      munkatárs','kutató','asszisztens','tanársegéd','laboráns','gya-
      kornok') DEFAULT NULL,
  `kezdes_ev` date NOT NULL,
  `createdAt` datetime NOT NULL,
  `updatedAt` datetime NOT NULL,
  `tanszekekId` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
ALTER TABLE `munkatarsak`
  ADD PRIMARY KEY (`id`),
  ADD UNIQUE KEY `email` (`email`),
  ADD KEY `tanszekekId` (`tanszekekId`);
ALTER TABLE `munkatarsak`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;
ALTER TABLE `munkatarsak`
  ADD CONSTRAINT `munkatarsak_ibfk_1` FOREIGN KEY (`tanszekekId`)
REFERENCES `tanszekek` (`id`) ON DELETE SET NULL ON UPDATE CASCADE;
COMMIT;
```

A ***Diákok*** relációt létrehozó MySQL kód:

```
CREATE TABLE `diakok` (
  `id` int(11) NOT NULL,
  `diak_nev` varchar(100) NOT NULL,
  `email` varchar(100) NOT NULL,
  `jelszo` varchar(255) NOT NULL,
  `kepzesi_forma` enum('nappali','levelező') NOT NULL,
  `kezdes_ev` date NOT NULL,
  `createdAt` datetime NOT NULL,
  `updatedAt` datetime NOT NULL,
  `szakokId` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
-- Eseményindítók `diakok`
DELIMITER $$
CREATE TRIGGER `diakokSzamaCsokkentes` AFTER DELETE ON `diakok` FOR
EACH ROW BEGIN
             IF OLD.kepzesi_forma LIKE "nappali" THEN
                 UPDATE diakok_szama SET diakok_szama.db_nappali =
diakok_szama.db_nappali - 1 WHERE diakok_szama.szakokId = OLD.szakokId
AND  diakok_szama.kezdes_ev = OLD.kezdes_ev;
             ELSEIF OLD.kepzesi_forma LIKE "levelező" THEN
              UPDATE diakok_szama SET diakok_szama.db_levelezo =
diakok_szama.db_levelezo - 1 WHERE diakok_szama.szakokId =
OLD.szakokId AND  diakok_szama.kezdes_ev = OLD.kezdes_ev;
             END IF;
         END
$$
DELIMITER ;
DELIMITER $$
CREATE TRIGGER `diakokSzamaNoveles` AFTER INSERT ON `diakok` FOR EACH
ROW BEGIN
             IF NEW.kepzesi_forma LIKE "nappali" THEN
                 UPDATE diakok_szama SET diakok_szama.db_nappali =
diakok_szama.db_nappali + 1 WHERE diakok_szama.szakokId = NEW.szakokId
AND diakok_szama.kezdes_ev = NEW.kezdes_ev;
             ELSEIF NEW.kepzesi_forma LIKE "levelező" THEN
              UPDATE diakok_szama SET diakok_szama.db_levelezo =
diakok_szama.db_levelezo + 1 WHERE diakok_szama.szakokId =
NEW.szakokId AND diakok_szama.kezdes_ev = NEW.kezdes_ev;
             END IF;
```

```
        END
$$
DELIMITER ;
ALTER TABLE `diakok`
  ADD PRIMARY KEY (`id`),
  ADD UNIQUE KEY `email` (`email`),
  ADD KEY `szakokId` (`szakokId`);
ALTER TABLE `diakok`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;
ALTER TABLE `diakok`
  ADD CONSTRAINT `diakok_ibfk_1` FOREIGN KEY (`szakokId`) REFERENCES
`szakok` (`id`) ON DELETE SET NULL ON UPDATE CASCADE;
COMMIT;
```

A ***Diákok száma*** relációt létrehozó MySQL kód:

```
CREATE TABLE `diakok_szama` (
  `id` int(11) NOT NULL,
  `db_nappali` int(2) NOT NULL DEFAULT 0,
  `db_levelezo` int(2) NOT NULL DEFAULT 0,
  `kezdes_ev` date NOT NULL,
  `createdAt` datetime NOT NULL,
  `updatedAt` datetime NOT NULL,
  `szakokId` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
ALTER TABLE `diakok_szama`
  ADD PRIMARY KEY (`id`),
  ADD UNIQUE KEY `szakokKezdesEv` (`szakokId`,`kezdes_ev`);
ALTER TABLE `diakok_szama`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;
ALTER TABLE `diakok_szama`
  ADD CONSTRAINT `diakok_szama_ibfk_1` FOREIGN KEY (`szakokId`)
REFERENCES `szakok` (`id`) ON DELETE SET NULL ON UPDATE CASCADE;
COMMIT;
```

A ***Mintatantervek*** relációt létrehozó MySQL kód:

```
CREATE TABLE `mintatantervek` (
  `id` int(11) NOT NULL,
  `felev` enum('1','2','3','4','5','6','7','8') NOT NULL,
  `eloadas` int(2) NOT NULL DEFAULT 0,
  `laboratoriumi` int(2) NOT NULL DEFAULT 0,
  `gyakorlati` int(2) NOT NULL DEFAULT 0,
  `onallo_munka` int(2) NOT NULL DEFAULT 0,
  `kredit` int(2) NOT NULL DEFAULT 0,
  `tipus` enum('beszámoló','vizsga') NOT NULL,
  `kezdes_ev` date NOT NULL,
  `createdAt` datetime NOT NULL,
  `updatedAt` datetime NOT NULL,
  `szakokId` int(11) DEFAULT NULL,
  `tantargyakId` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
ALTER TABLE `mintatantervek`
  ADD PRIMARY KEY (`id`),
  ADD UNIQUE KEY `szakTantargyFelevKezdesEv`
(`szakokId`,`tantargyakId`,`felev`,`kezdes_ev`),
  ADD KEY `tantargyakId` (`tantargyakId`);
ALTER TABLE `mintatantervek`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;
ALTER TABLE `mintatantervek`
  ADD CONSTRAINT `mintatantervek_ibfk_1` FOREIGN KEY (`szakokId`)
REFERENCES `szakok` (`id`) ON DELETE SET NULL ON UPDATE CASCADE,
```

```
  ADD CONSTRAINT `mintatantervek_ibfk_2` FOREIGN KEY (`tantargyakId`)
REFERENCES `tantargyak` (`id`) ON DELETE SET NULL ON UPDATE CASCADE;
COMMIT;
```

Az ***Eredmények*** relációt létrehozó MySQL kód:

```
CREATE TABLE `eredmenyek` (
  `id` int(11) NOT NULL,
  `felev` enum('1','2','3','4','5','6','7','8') NOT NULL,
  `kredit` int(2) NOT NULL,
  `tipus` enum('beszámoló','vizsga') NOT NULL,
  `pontszam` int(2) DEFAULT NULL,
  `vizsgahoz_engedve` tinyint(1) DEFAULT 1,
  `datum` date NOT NULL,
  `createdAt` datetime NOT NULL,
  `updatedAt` datetime NOT NULL,
  `tantargyakId` int(11) DEFAULT NULL,
  `munkatarsakId` int(11) DEFAULT NULL,
  `szeminarium_munkatarsakId` int(11) DEFAULT NULL,
  `diakokId` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
ALTER TABLE `eredmenyek`
  ADD PRIMARY KEY (`id`),
  ADD UNIQUE KEY `diakTantargyFelevDatum`
(`diakokId`,`tantargyakId`,`felev`,``datum`),
  ADD KEY `tantargyakId` (`tantargyakId`),
  ADD KEY `munkatarsakId` (`munkatarsakId`),
  ADD KEY `szeminarium_munkatarsakId` (`szeminarium_munkatarsakId`);
ALTER TABLE `eredmenyek`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;
ALTER TABLE `eredmenyek`
  ADD CONSTRAINT `eredmenyek_ibfk_1` FOREIGN KEY (`tantargyakId`)
REFERENCES `tantargyak` (`id`) ON DELETE SET NULL ON UPDATE CASCADE,
  ADD CONSTRAINT `eredmenyek_ibfk_2` FOREIGN KEY (`munkatarsakId`)
REFERENCES `munkatarsak` (`id`) ON DELETE SET NULL ON UPDATE CASCADE,
  ADD CONSTRAINT `eredmenyek_ibfk_3` FOREIGN KEY
(`szeminarium_munkatarsakId`) REFERENCES `munkatarsak` (`id`) ON
DELETE SET NULL ON UPDATE CASCADE,
  ADD CONSTRAINT `eredmenyek_ibfk_4` FOREIGN KEY (`diakokId`)
REFERENCES `diakok` (`id`) ON DELETE SET NULL ON UPDATE CASCADE;
COMMIT;
```

A ***Tarifikációs változók*** relációt létrehozó MySQL kód:

```
CREATE TABLE `tarifikacios_valtozok` (
  `id` int(11) NOT NULL,
  `onallo_munka` double NOT NULL,
  `dolgozatjavitas_eloadas` double NOT NULL,
  `dolgozatjavitas_szeminarium` double NOT NULL,
  `dolgozatjavitas_levelezo` double NOT NULL,
  `vizsga_konzultacio` double NOT NULL,
  `beszamolo` double NOT NULL,
  `vizsga` double NOT NULL,
  `bsc2` double NOT NULL,
  `bsc3` double NOT NULL,
  `bsc4` double NOT NULL,
  `msc1` double NOT NULL,
  `msc2` double NOT NULL,
  `createdAt` datetime NOT NULL,
  `updatedAt` datetime NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
ALTER TABLE `tarifikacios_valtozok`
```

```
  ADD PRIMARY KEY (`id`);
ALTER TABLE `tarifikacios_valtozok`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;
COMMIT;
```

A ***Tantárgy Tarifikációk*** relációt létrehozó MySQL kód:

```
CREATE TABLE `tantargy_tarifikaciok` (
  `id` int(11) NOT NULL,
  `felev` enum('1','2','3','4','5','6','7','8') NOT NULL,
  `kredit` int(4) NOT NULL,
  `tipus` enum('beszámoló','vizsga') NOT NULL,
  `kepzesi_forma` enum('nappali','levelező') NOT NULL,
  `hallgatok_szama` int(4) NOT NULL DEFAULT 0,
  `eloadas` double NOT NULL DEFAULT 0,
  `szeminarium` double NOT NULL DEFAULT 0,
  `laboratoriumi` double NOT NULL DEFAULT 0,
  `createdAt` datetime NOT NULL,
  `updatedAt` datetime NOT NULL,
  `tantargyakId` int(11) DEFAULT NULL,
  `munkatarsakId` int(11) DEFAULT NULL,
  `szeminarium_munkatarsakId` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
ALTER TABLE `tantargy_tarifikaciok`
  ADD PRIMARY KEY (`id`),
  ADD UNIQUE KEY `egyedi`
(`tantargyakId`,`felev`,`kepzesi_forma`,`munkatarsakId`,`szeminarium_m
unkatarsakId`),
  ADD KEY `munkatarsakId` (`munkatarsakId`),
  ADD KEY `szeminarium_munkatarsakId` (`szeminarium_munkatarsakId`);
ALTER TABLE `tantargy_tarifikaciok`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;
ALTER TABLE `tantargy_tarifikaciok`
  ADD CONSTRAINT `tantargy_tarifikaciok_ibfk_1` FOREIGN KEY
(`tantargyakId`) REFERENCES `tantargyak` (`id`) ON DELETE SET NULL ON
UPDATE CASCADE,
  ADD CONSTRAINT `tantargy_tarifikaciok_ibfk_2` FOREIGN KEY
(`munkatarsakId`) REFERENCES `munkatarsak` (`id`) ON DELETE SET NULL
ON UPDATE CASCADE,
  ADD CONSTRAINT `tantargy_tarifikaciok_ibfk_3` FOREIGN KEY
(`szeminarium_munkatarsakId`) REFERENCES `munkatarsak` (`id`) ON
DELETE SET NULL ON UPDATE CASCADE;
COMMIT;
```

A ***Gyakorlat tarifikációk*** relációt létrehozó MySQL kód:

```
CREATE TABLE `gyakorlat_tarifikaciok` (
  `id` int(11) NOT NULL,
  `gyakorlat_nev` varchar(100) NOT NULL,
  `felev` enum('1','2','3','4','5','6','7','8') NOT NULL,
  `oraszam` int(4) NOT NULL,
  `idotartam` int(4) NOT NULL DEFAULT 0,
  `createdAt` datetime NOT NULL,
  `updatedAt` datetime NOT NULL,
  `munkatarsakId` int(11) DEFAULT NULL,
  `tanszekekId` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
ALTER TABLE `gyakorlat_tarifikaciok`
  ADD PRIMARY KEY (`id`),
  ADD UNIQUE KEY `tanszekFelevGyakorlat`
(`tanszekekId`,`felev`,`gyakorlat_nev`),
  ADD KEY `munkatarsakId` (`munkatarsakId`);
```

```
ALTER TABLE `gyakorlat_tarifikaciok`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;
ALTER TABLE `gyakorlat_tarifikaciok`
  ADD CONSTRAINT `gyakorlat_tarifikaciok_ibfk_1` FOREIGN KEY
(`munkatarsakId`) REFERENCES `munkatarsak` (`id`) ON DELETE SET NULL
ON UPDATE CASCADE,
  ADD CONSTRAINT `gyakorlat_tarifikaciok_ibfk_2` FOREIGN KEY
(`tanszekekId`) REFERENCES `tanszekek` (`id`) ON DELETE SET NULL ON
UPDATE CASCADE;
COMMIT;
```

A ***Szakdolgozat és évfolyammunka tarifikációk*** relációt létrehozó MySQL kód:

```
CREATE TABLE `szakdolgozat_evfolyammunka_tarifikaciok` (
  `id` int(11) NOT NULL,
  `b2` int(2) NOT NULL DEFAULT 0,
  `b3` int(2) NOT NULL DEFAULT 0,
  `b4` int(2) NOT NULL DEFAULT 0,
  `m1` int(2) NOT NULL DEFAULT 0,
  `m2` int(2) NOT NULL DEFAULT 0,
  `createdAt` datetime NOT NULL,
  `updatedAt` datetime NOT NULL,
  `munkatarsakId` int(11) DEFAULT NULL,
  `szakokId` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
ALTER TABLE `szakdolgozat_evfolyammunka_tarifikaciok`
  ADD PRIMARY KEY (`id`),
  ADD UNIQUE KEY `munkatarsSzak` (`munkatarsakId`,`szakokId`),
  ADD KEY `szakokId` (`szakokId`);
ALTER TABLE `szakdolgozat_evfolyammunka_tarifikaciok`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;
ALTER TABLE `szakdolgozat_evfolyammunka_tarifikaciok`
  ADD CONSTRAINT `szakdolgozat_evfolyammunka_tarifikaciok_ibfk_1`
FOREIGN KEY (`munkatarsakId`) REFERENCES `munkatarsak` (`id`) ON
DELETE SET NULL ON UPDATE CASCADE,
  ADD CONSTRAINT `szakdolgozat_evfolyammunka_tarifikaciok_ibfk_2`
FOREIGN KEY (`szakokId`) REFERENCES `szakok` (`id`) ON DELETE SET NULL
ON UPDATE CASCADE;
COMMIT;
```

2.2. Az információs rendszer szerveroldali komponenseinek létrehozása NodeJS, ExpressJS, Sequelize környezetben

Miután megterveztük az adatbázist a következő lépés a szerveroldali környezet (back-end) kialakítása, hogy az információs rendszerben tárolt adatokkal különböző műveleteket tudjunk elvégezni. Ehhez a már említett NodeJS, ExpressJS szerveroldali alkalmazásokat fogjuk használni, valamint az adatok manipulálását MySQL parancsokkal a Sequelize modulon keresztül valósítjuk meg.

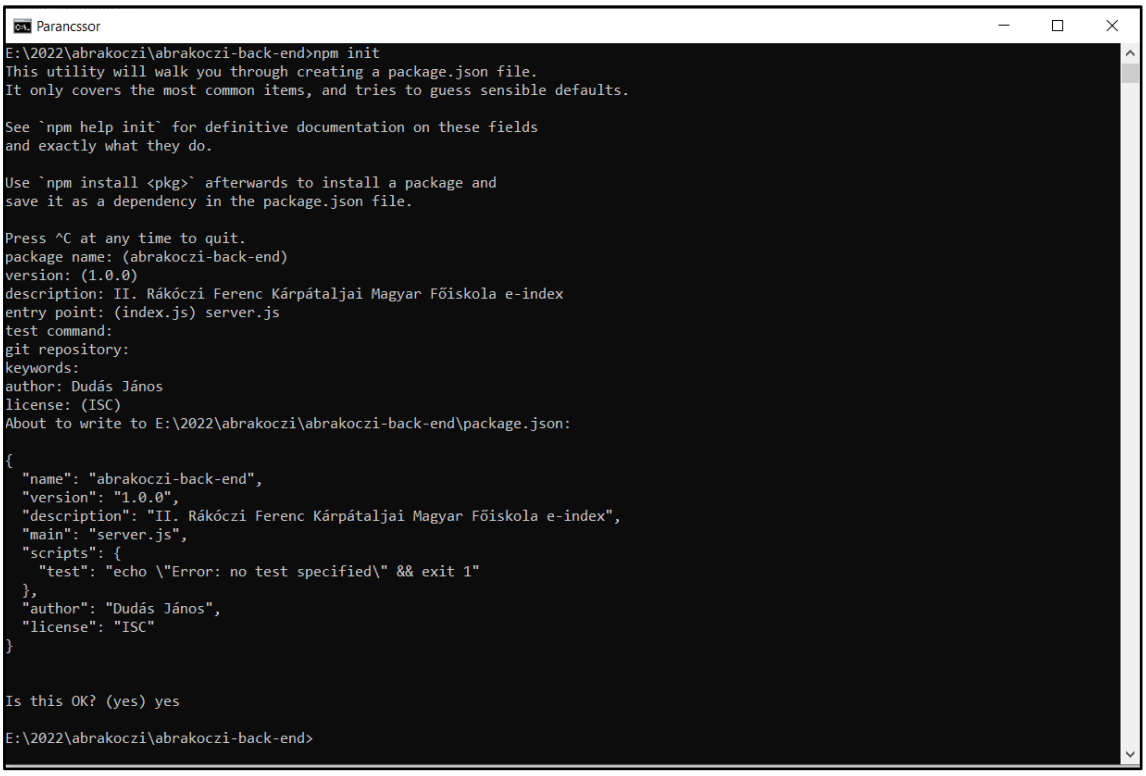
Ahhoz, hogy használni tudjuk ezeket a programokat, telepítenünk[2] kell a NodeJS környezetet. A webes platformunk fejlesztése során a 14.17.3 verziót fogjuk használni. A

[2] Letöltés: https://nodejs.org/en/download/

szerveroldali keretrendszer alapvető beállításait és moduljainak a telepítését oktatóanyag alapján fogjuk elkészíteni [7].

2.2.1. Modulok telepítése, alapvető beállítások

Miután sikeresen telepítettük a NodeJS környezetet a következő lépésben létrehoztunk egy mappát, ahol a platform működéséhez szükséges fájlokat fogjuk tárolni. Aztán parancssorban a létrehozott könyvtárba navigáltunk és beírtuk az npm init parancsot, és a segédprogram segítségével létrehoztuk a működéshez szükséges *package.json* fájlt az 13. ábran látható műveletsor segítségével. Aztán telepítettük az express, sequelize, mysql2, cors modulokat a `npm install express sequelize mysql2 cors --save` parancs segítségével.



13. ábra. Segédprogram a *package.js* létrehozásához (Saját kép, 2022)

A telepítés után a *package.json* fájl tartalma:

```
{
  "name": "abrakoczi-back-end",
  "version": "1.0.0",
  "description": "II. Rákóczi Ferenc Kárpátaljai Magyar Főiskola",
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Dudás János",
```

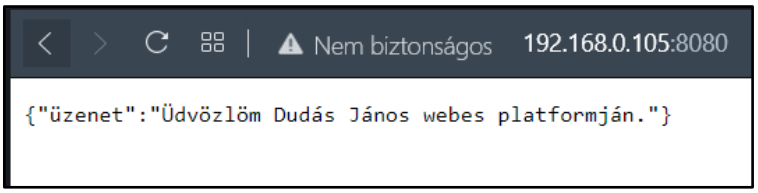
```
  "license": "ISC",
  "dependencies": {
    "cors": "^2.8.5",
    "express": "^4.17.1",
    "mysql2": "^2.2.5",
    "sequelize": "^6.6.5"
  }
}
```

2.2.2. Az ExpressJS webszerver telepítése és beállítása

A gyökérkönyvtárban létrehoztuk a *server.js* fájlt, ahol meghívtuk az express és cors modulokat.

```
const express = require("express");
const cors = require("cors");
const app = express();
var corsOptions = {
  origin: "http://192.168.0.105:4200"
};
app.use(cors(corsOptions));
app.use(express.json());
app.use(express.urlencoded({ extended: true }));
app.get("/", (req, res) => {
  res.json({ üzenet: "Üdvözlöm Dudás János webes platformján." });
});
const PORT = process.env.PORT || 8080;
app.listen(PORT, () => {
  console.log(`A szerver a ${PORT} porton keresztül fut.`);
});
```

Hogy a szerver működését tesztelni tudjuk, a parancssorban beléptünk a szerveroldali keretrendszer (back-end) gyökérkönyvtárába és kiadtuk a `node server.js` parancsot és a böngészőben a 8080-as porton keresztül megnyitottuk a *192.168.0.105* weboldalt, ahol a 14. ábra látható üzenet jelent meg.



14. ábra. ExpressJS webszerver működése a 8080-as porton (Saját kép, 2022)

2.2.3. Sequelize és a MySQL adatbázis beállítása

Az *app* könyvtáron belül létrehoztuk a *config* mappát és ebben a *db.config.js* fájlt, amiben beállítottuk az adatbázis és a webszerver kommunikációjához szükséges alapadatokat, többek között a felhasználó nevet, az adatbázis jelszavát és az adatbázis nevét:

```
module.exports = {
  HOST: "localhost",
  FELHASZNALO: "root",
  JELSZO: "root",
  AB: "abrakoczi",
  nyelv: "mysql",
  pool: {
    max: 5,
    min: 0,
    acquire: 30000,
    idle: 10000
  }
};
```

Miután ezzel megvoltunk létrehoztuk a Sequelize modellek inicializálásához szükséges fájlokat az *app/models* könyvtárban. A modellek és vezérlők létrehozásának folyamatát a Tanszékek egyeden keresztül fogjuk bemutatni.

Az *app/models/index.js* fájlban fog történni a modelleket inicializálása. A fájl tartalma ennek megfelelően a következő:

```
const abKonfiguracio = require("../config/db.config.js");
const Sequelize = require("sequelize");
const sequelize = new Sequelize(abKonfiguracio.AB,
abKonfiguracio.FELHASZNALO, abKonfiguracio.JELSZO, {
  host: abKonfiguracio.HOST,
  dialect: abKonfiguracio.nyelv,
  operatorsAliases: false,
  pool: {
    max: abKonfiguracio.pool.max,
    min: abKonfiguracio.pool.min,
    acquire: abKonfiguracio.pool.acquire,
    idle: abKonfiguracio.pool.idle }
});
const db = {};
db.Sequelize = Sequelize;
db.sequelize = sequelize;
db.tanszekek = require("./tanszekek.model.js")(sequelize, Sequelize);
module.exports = db;
```

A következő lépésben pedig a *server.js* fájlban meghívtuk a modelleket és mivel még fejlesztő fázisban vagyunk, ezért engedélyeztük az adatbázis eldobását és újra szinkronizálását, amit a modellek végső kialakítása után kapcsoltunk ki:

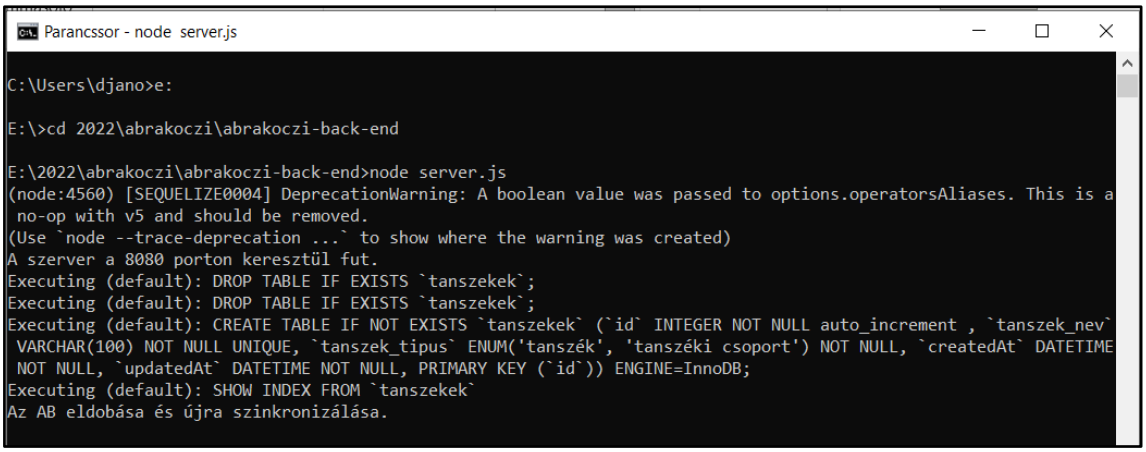
```
const db = require("./app/models");
db.sequelize.sync({ force: true }).then(() => {
 console.log("Az AB eldobása és újra szinkronizálása.");
 //initial();
 });
```

És végül létrehoztuk a Tanszékek egyed modelljét a Sequelize modulban, a *tanszekek.model.js* fájlban:

```
module.exports = (sequelize, Sequelize) => {
    const Tanszekek = sequelize.define("tanszekek", {
      tanszek_nev: {
        type: Sequelize.STRING(100),
        allowNull: false,
        unique: true,
      },
```

```
      tanszek_tipus: {
        type: Sequelize.ENUM('tanszék','tanszéki csoport'),
        allowNull: false
      }
    }, {
        freezeTableName: true,
    });
    return Tanszekek;
  };
```

A Tanszékek modellben kikapcsoltuk a Sequelize modulnak azt a funkcióját, hogy a táblák neveit automatikusan többes számba tegye, mivel magyar modellneveket (táblaneveket) fogunk használni. Végül beírtuk a `node server.js` parancsot és a 15. ábran látható MySQL kód futott le a parancssorban, ami létrehozta a Tanszékek táblát.



15. ábra. Tanszékek reláció létrehozása NodeJS, ExpressJS, Sequelize segítségével (Saját kép, 2022)

2.2.4. Vezérlők létrehozása

Az *app/controllers* könyvtárban létrehoztuk a *tanszekek.controller.js* fájlt, amiben a Tanszékek táblában tárolt adatok kezeléséhez szükséges alapvető függvényeket fogjuk definiálni. Ezek a függvények a létrehozás, az olvasás, a frissítés és a törlés, azaz a tartós tárolás négy alapvető műveletét (CRUD) fogják megvalósítani.

A fájl szerkezete:

```
const db = require("../models");
const Tanszekek = db.tanszekek;
const Op = db.Sequelize.Op;
// az oldalak számolásához
const oldalszamozas = (oldal, meret) => {
    const limit = meret ? +meret : 3;
    const offset = oldal ? oldal * limit : 0;
    return { limit, offset };
};
```

```
const oldalszamozasiAdatok = (data, oldal, limit) => {
    const { count: osszesElem, rows: tanszekek } = data;
    const jelenlegiOldal = oldal ? +oldal : 0;
    const osszesOldal = Math.ceil(osszesElem / limit);

    return { osszesElem, tanszekek, osszesOldal, jelenlegiOldal };
};
// Új tanszékek létrehozása és mentése
exports.letrehozas = (req, res) => {
};
// Minden tanszék listázása az adatbázisból lapozással
exports.tanszekListazasLapozassal = (req, res) => {
};
// Tanszék keresése azonosítója alapján
exports.egyTanszekListazasa = (req, res) => {
};
// Tanszék módosítása azonosítója alapján
exports.frissites = (req, res) => {
};
// Tanszék törlése azonosítója alapján
exports.torles = (req, res) => {
};
// Minden tanszék törlése az adatbázisból.
exports.mindenTorlese = (req, res) => {
};
```

Miután ezzel megvoltunk megírtuk a fenti szerkezetben bemutatott függvényeket a Tanszékek relációban tárolt adatoknak megfelelően.

Új tanszék létrehozása

A *letrehozas()* függvény segítségével fog megvalósulni az adatok felvétele a Tanszékek táblába. A függvény ellenőrizni fogja, hogy üres adatokat ne tudjuk bevinni a tanszek_nev és a tanszek_tipus oszlopokba (a táblában lévő megkötés ezt egyébként sem engedné). Ha sikeres volt az új sor létrehozása, visszaküldi az új tanszék adatait, ha esetleg valamilyen hiba lép fel a létrehozás közben hibaüzenetet kapunk.

```
exports.letrehozas = (req, res) => {
  // Kérés validálása
  if (!req.body.tanszek_nev || !req.body.tanszek_tipus) {
    res.status(400).send({
      message: "A mező tartalma nem lehet üres!"
    });
    return;
  }
  // Tanszék létrehozása
  const tanszekek = {
    tanszek_nev: req.body.tanszek_nev,
    tanszek_tipus: req.body.tanszek_tipus
  };
  // Tanszék mentése az adatbázisba
  Tanszekek.create(tanszekek)
    .then(data => {
      res.send(data);
    })
    .catch(err => {
      res.status(500).send({
```

```
        message:
          err.message || "Hiba lépett fel a tanszék létrehozása közben."
      }); }); };
```

Tanszékek listázása szűrési lehetőségekkel

A *tanszekListazasLapozassal()* függvény segítségével az adatbázisban lévő tanszékeket tudjuk kilistázni, valamint ezek között az adatok között keresni, egész pontosan a tanszék nevére tudunk rákeresni.

```
exports.tanszekListazasLapozassal = (req, res) => {
  const { oldal, meret, tanszek_nev } = req.query;
  var condition = tanszek_nev ? { tanszek_nev: { [Op.like]:
                                `%${tanszek_nev}%` } } : null;
  const { limit, offset } = oldalSzamozas(oldal, meret);
  Tanszekek.findAndCountAll({
    where: condition, limit, offset,
    attributes: { exclude: ['createdAt', 'updatedAt'] },
    order: ['tanszek_nev'],
  })
    .then(data => {
      const response = oldalszamozasiAdatok(data, oldal, limit);
      res.send(response);
    })
    .catch(err => {
      res.status(500).send({
       message:
         err.message || "Hiba lépett fel a tanszékek keresése közben."
      }); }); };
```

Tanszék keresése elsődleges kulcs alapján

Az *egyTanszekListazasa()* függvény segítségével adott *id*-val rendelkező tanszéket tudunk megkeresni. Az adatok frissítésénél lesz majd jelentősége.

```
exports.egyTanszekListazasa = (req, res) => {
  const id = req.params.id;
  Tanszekek.findByPk(id, {
    attributes: { exclude: ['createdAt', 'updatedAt', 'email',
                                                    'jelszo']},
  })
    .then(data => {
      res.send(data);
    })
    .catch(err => {
      res.status(500).send({
       message: "Hiba lépett fel a tanszék keresése közben, a tanszék
                 azonosítója: " + id
      }); }); };
```

Tanszék frissítése elsődleges kulcs alapján

A *frissites()* függvény segítségével tudjuk majd a létező tanszékek adatait frissíteni. Az azonosító (*id*) alapján fogjuk a frissítést elvégezni, így biztosítva a hibás adatok bevitelének lehetőségét. Ebben az esetben sem engedünk üres tanszek_nev és tanszek_tipus mezőket beírni.

```
exports.frissites = (req, res) => {
  if (!req.body.tanszek_nev || !req.body.tanszek_tipus) {
    res.status(400).send({
      message: "A mező tartalma nem lehet üres!"
    });
    return;
  }
  const id = req.params.id;
  Tanszekek.update(req.body, {
    where: { id: id }
  })
    .then(num => {
      if (num == 1) {
        res.send({
          message: "A tanszék sikeresen frissítve lett."
        });
      } else {
       res.send({
         message: `Nem lehet a tanszéket frissíteni, azonosító: ${id}.
                   Lehetséges, hogy a tanszék nem található vagy üres a
                   lekérdezés mező!`
        }); }
    })
    .catch(err => {
      res.status(500).send({
        message: "Hiba lépett fel a tanszék frissítése közben,
                  azonosító: " + id
      }); }); };
```

Tanszék törlése elsődleges kulcs alapján

A *torles()* függvény segítségével adott azonosítóval rendelkező tanszéket tudunk majd törölni az adatbázisből.

```
exports.torles = (req, res) => {
  const id = req.params.id;
  Tanszekek.destroy({
    where: { id: id }
  })
    .then(num => {
      if (num == 1) {
        res.send({
          message: "A tanszék sikeresen törölve lett!"
        });
      } else {
        res.send({
          message: `Nem sikerült a tanszék törlése, azonosító: ${id}.
                    Lehetséges, hogy a tanszék nem található! `
        }); }
    })
    .catch(err => {
     res.status(500).send({
      message: "Hiba lépett fel a tanszék törlése közben, azonosító:"
                + id
      }); }); };
```

Minden tanszék törlése az adatbázisból

A *mindenTorlese()* függvény segítésével tudjuk majd a Tanszékek adatbázisból az összes adatot egyszerre eltávolítani.

```
exports.mindenTorlese = (req, res) => {
  Tanszekek.destroy({
    where: {},
    truncate: false
  })
    .then(nums => {
      res.send({ message: `${nums} tanszék sikeresen törölve!` });
    })
    .catch(err => {
      res.status(500).send({
        message:
          err.message || "Hiba lépett fel a tanszékek törlése közben."
      }); }); };
```

A következő lépésben pedig be kell állítanunk azokat az eseményeket, amelyek ezeket a függvények aktiválni fogják.

2.2.5. Útvonalak beállítása

Amikor egy kliens HTTP-kérést (GET, POST, PUT, DELETE) küld egy végpontra, akkor az útvonalak beállításával meg kell határoznunk, hogy a szerver hogyan válaszoljon. Tehát gyakorlatilag az előbb létrehozott függvényeknek a futtatását bizonyos útvonalak aktiválásához fogjuk kötni. Ezt pedig az *app/routes* könyvtáron belül, *tanszekek.routes.js* néven létrehozott fájlban fogjuk beállítani:

```
const vezerlo = require("../controllers/tanszekek.controller");
module.exports = function (app) {
    var utvonal = require("express").Router();

    utvonal.post("/", vezerlo.letrehozas);

    utvonal.get("/", vezerlo.tanszekListazasLapozassal);

    utvonal.get("/lista", vezerlo.tanszekListazasLapozasNelkul);

    utvonal.get("/:id", vezerlo.egyTanszekListazasa);

    utvonal.put("/:id", vezerlo.frissites);

    utvonal.delete("/:id", vezerlo.torles);

    utvonal.delete("/", vezerlo.mindenTorlese);

    app.use('/api/tanszek', utvonal);
};
```

A *server.js* fájlban is meg kellett hívnunk a Tanszékek táblához létrehozott útvonalválasztó vezérlő fájlt:

```
require("./app/routes/tanszekek.routes")(app);
```

Miután ezt megcsináltuk a `node server.js` parancs segítségével elindítottuk a szervert és leteszteltük a program működését Postman[3] program segítségével. Ellenőriztük a létrehozás, az olvasás, a frissítés és a törlés, azaz a tartós tárolás négy alapvető műveletét (CRUD) a 1. mellékleten látható módon.

Végül az adatbázis minden egyes relációjához elkészítettük a modelleket, a vezérlőket és a vezérlő függvényeket aktiváló útvonalakat, a fenti lépéseknek megfelelően. Valamint a függvények létrehozása során figyelembe vettük a táblák eltérő tulajdonságait, valamint a lehetséges felhasználói igényeket.

2.3. Az információs rendszer kliensoldali komponenseinek létrehozása Angular 12 CLI környezetben

Miután létrehoztuk az információs rendszer adatainak kezeléséhez szükséges szerveroldali modelleket, vezérlőket és útvonalakat, a következő lépés a kliensoldali környezet (front-end) kialakítása, hogy az információs rendszerben tárolt adatokkal az adminisztrációs folyamat szereplői felhasználóbarát, webes platform segítségével tudjanak dolgozni. Ehhez a már említett Angular CLI kliensoldali keretrendszert v12.1.4 verzióját fogjuk használni, és ennek beállításait és moduljainak a telepítését oktatóanyag alapján fogjuk elkészíteni [3].

2.3.1. Angular CLI telepítése, alapvető beállítások

Első lépéként a parancssorban beléptünk a projektünk könyvtárába, kiadtuk az `ng new abrakoczi-frond-end` parancsot, aminek a segítségével létrehoztunk és inicializáltunk egy új Angular CLI alkalmazást abrakoczi-front-end néven. A telepítés néhány percet vett igénybe és a telepítés végén nem kaptunk hibaüzenetet. Aztán pedig létrehoztunk a Tanszékek reláció kezeléséhez szükséges komponenst, modellt és szolgáltatást a következő parancsok segítségével:

- `ng g s services/tanszekek` – az útvonalakat fogja kezelni;

[3] Letöltés: https://www.postman.com

- `ng g class models/tanszekek --type=model` – a Tanszékek reláció modelljét fogja tárolni;
- `ng g c components/tanszekek-lista` – a komponensek vezérléséért fog felelni.

Az *app.modules.ts* fájlban importáltuk a FormsModule, HttpClientModule, NgbModule, NgxPrintModule, NgxPaginationModule, NgSelectModule nevű modulokat. A FormsModule exportálja a szükséges szolgáltatókat és direktívákat a sablon vezérelt űrlapokhoz. A HttpClientModule szolgáltatási modul pedig lehetővé teszi számunkra, hogy HTTP-kéréseket hajtsunk végre, és könnyen kezeljük ezeket a kéréseket és válaszaikat. Az NgbModule a felugró ablakokat kezeli, az NgxPrintModule a nyomtatás megvalósítását biztosítja, az NgxPaginationModule modul az adatok oldalakra bontásáért felel és az NgSelectModule modul pedig a legördülő listákat fogja kezelni.

```
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { HttpClientModule } from '@angular/common/http';
import { BrowserModule } from '@angular/platform-browser';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { TanszekekListaComponent } from './components/tanszekek-
                                        lista/tanszekek-lista.component';
import { NgbModule } from '@ng-bootstrap/ng-bootstrap';
import { NgxPrintModule } from 'ngx-print';
import { NgxPaginationModule } from 'ngx-pagination';
import { NgSelectModule } from '@ng-select/ng-select';
@NgModule({
  declarations: [
    AppComponent,
    TanszekekListaComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    FormsModule,
    HttpClientModule,
    NgbModule,
    NgxPrintModule,
    NgxPaginationModule,
    NgSelectModule,
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Az *app-routing.module.ts* fájlban pedig beállítottuk azt az útvonalat amivel a tanszékek vezérlőt tudjuk majd aktiválni, azaz a tanszékek menüpontot elérni.

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { TanszekekListaComponent } from './components/tanszekek-
                                        lista/tanszekek-lista.component';
```

```
const routes: Routes = [
  { path: '', redirectTo: '', pathMatch: 'full' },
  //Tanszékek útvonal
  { path: 'tansz/tanszek', component: TanszekekListaComponent },];
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Az *index.html* fájlban importáltuk a Bootstrap 4.4.0 ingyenes és nyílt forráskódú CSS-keretrendszert, valamint magyarra állítottuk az oldal nyelvét.

```
<!doctype html>
<html lang="hu">
<head>
  <meta charset="utf-8">
  <title> II. RFKMF E-index</title>
  <base href="/">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="assets/favicon.ico">
  <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com
          /bootstrap/4.4.0/css/bootstrap.min.css"/>
</head>
<body>
  <app-root></app-root>
</body>
</html>
```

Az `src/app.component.html` fájlban pedig beállítottuk az információs rendszer webes felületének a navigálósávját (navbar):

```
<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/
      bootstrap/4.4.0/css/bootstrap.min.css">
<link rel="stylesheet" href="https://use.fontawesome.com/releases
      /v5.7.0/css/all.css">
<div id="app" style="min-height: calc(100vh)">
  <nav class="navbar  navbar-expand-xl navbar-dark bg-primary">
    <a href="#" class="navbar-brand"><img src="/assets/iskola.jpg"
       alt="Logó" style="width:40px;"></a>
    <ul class="navbar-nav mr-auto" routerLinkActive="active">
      <li class="nav-item">
        <a href="/" class="nav-link" routerLink="">Kezdőlap </a>
      </li>
      <!--Tanszékek menüpont-->
      <li class="nav-item">
        <a routerLink="tansz/tanszek" class="nav-link">Tanszékek</a>
      </li>
    </ul>
  </nav>
  <div class="container-fluid mt-1">
    <div class="row justify-content-md-center">
      <div class="col-md-11">
        <router-outlet></router-outlet>
      </div>
    </div>
  </div>
</div>
```

2.3.2. Modellek, útvonalak és vezérlők beállítása

A *models/tanszekek.model.ts* fájlban definiáltuk a Tanszékek modell attribútumait:

```
export class Tanszekek {
    id?: any;
    tanszek_nev?: string;
    tanszek_tipus?: string;
}
```

Aztán pedig a *services/tanszekek.service.ts* fájlban beállítottuk azokat az útvonalakat, amiket az egyes függvények aktiválni fognak a platform működése közben.

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
import { Tanszekek } from '../models/tanszekek.model';
const baseUrl = 'http://192.168.0.105:8080/api/tanszek';
@Injectable({
  providedIn: 'root'
})
export class TanszekekService {
  constructor(private http: HttpClient) { }
  tanszekListazasLapozassal(params: any): Observable<any> {
    return this.http.get<any>(baseUrl, { params });
  }
  egytanszekListazasa(id: any): Observable<Tanszekek> {
    return this.http.get(`${baseUrl}/${id}`);
  }
  letrehozas(data: any): Observable<any> {
    return this.http.post(baseUrl, data);
  }
  frissites(id: any, data: any): Observable<any> {
    return this.http.put(`${baseUrl}/${id}`, data);
  }
  torles(id: any): Observable<any> {
    return this.http.delete(`${baseUrl}/${id}`);
  }
  mindenTorlese(): Observable<any> {
    return this.http.delete(baseUrl);
  }
}
```

Ez gyakorlatilag az Angular HTTPClient szolgáltatást használja a HTTP-kérések küldésére. Láthatjuk, hogy a függvények tartalmazzák a létrehozás, az olvasás, a frissítés és a törlés, azaz a tartós tárolás négy alapvető műveletét. Tehát ez a szolgáltatás fogja összekapcsolni a szerveroldali keretrendszert a kliensoldali keretrendszerrel az útvonalak, azaz a megfelelő csomópontok segítségével.

Végül pedig létrehoztuk a Tanszékek modell kezeléséhez szükséges komponenseket és vezérlőket a *components/tanszekek-lista.component.ts* fájlban, valamint a komponenshez tartozó weboldalt is kialakítottuk a *components/tanszekek-lista.components.html* fájl segítségével. Gyakorlatilag ebben a két fájlban tároljuk a létrehozás, olvasás, frissítés és törlés megvalósításához szükséges függvényeket és

komponenseket. Viszont az olvashatóság megkönnyítése érdekében a különböző funkciók külön-külön kerülnek bemutatásra, de a platform végleges változatában minden alapműveletet egy *components* mappában, azon belül pedig egy-egy TypeScript és HTML fájlban tárolunk.

Új tanszék hozzáadása

Új tanszéket űrlap stílusú weboldal kitöltésével tudunk felvenni. Ha valamelyik mezőt üres értékkel szeretnénk elküldeni a szervernek, akkor az hibaüzenetet dob és a weboldal pirossal kiírja a hiba okát és a hibásan kitöltött mező alatt is megjelenik a hiba leírása piros kiemeléssel. A *tanszekek-lista.component.ts* fájl tartalma ennek megfelelően:

```
import { Component, OnInit } from '@angular/core';
import { Tanszekek } from 'src/app/models/tanszekek.model';
import { TanszekekService } from 'src/app/services/tanszekek.service';
@Component({
  selector: 'app-tanszekek-lista',
  templateUrl: './tanszekek-lista.component.html',
  styleUrls: ['./tanszekek-lista.component.css']
})
export class TanszekekListaComponent implements OnInit {
  hiba = '';
  tanszek: Tanszekek = {
    tanszek_nev: '',
    tanszek_tipus: ''
  };
  constructor(
    private tanszekService: TanszekekService,
  ) { }
  ngOnInit(): void {
  }
  tanszekMentese(): void {
    const data = {
      tanszek_nev: this.tanszek.tanszek_nev,
      tanszek_tipus: this.tanszek.tanszek_tipus
    };
    this.tanszekService.letrehozas(data)
      .subscribe(
        response => {
          console.log(response);
        },
        error => {
          this.hiba = error.error.message;
          console.log(error);
        });
  }
  ujTanszek(): void {
    this.tanszek = {
      tanszek_nev: '',
      tanszek_tipus: '',
    };
  }
}
```

A platform, új tanszék felvételét lehetővé tévő oldalának a megjelenítéséért felelős *tanszekek-lista.component.html* fájl az alábbi szerkezettel rendelkezik:

```
<div class="table-responsive">
<div class="edit-form">
<div class="card card-container">
 <h2 class="container text-center mb-3">Tanszék hozzáadása</h2>
 <div class="submit-form">
  <div class="form-group">
    <label for="tanszek_nev">Tanszék neve</label>
    <input #tanszek_nev="ngModel" placeholder="Tanszék neve"
           type="text" class="form-control" id="tanszek_nev" required
           [(ngModel)]="tanszek.tanszek_nev" name="tanszek_nev" />
    <div class="alert-danger"  *ngIf="tanszek_nev.errors  &&
         mentes_hiba">A tanszék nevének a megadása kötelező </div>
   </div>
  <div class="form-group">
    <label for="tanszek_tipus">Tanszék típusa:</label>
    <ng-select #tanszek_tipus="ngModel" class="form-control" bindValue
               ="tanszek.tanszek_tipus"[clearable]="true"
              [searchable]="false" id="tanszek_tipus" required
              [(ngModel)]=tanszek.tanszek_tipus"
              name="tanszek_tipus">
      <ng-option value="">--Tanszék típusa--</ng-option>
      <ng-option value="tanszék">tanszék</ng-option>
      <ng-option value="tanszéki csoport">tanszéki csoport</ng-option>
     </ng-select>
     <div class="alert-danger" *ngIf="tanszek_tipus.errors &&
          mentes_hiba">A tanszék típusának a megadása kötelező
     </div>
   </div>
   <button (click)="mentes_hiba=''; tanszekMentese()"
           class="btn btn-success btn-block">Mentés <i class="fa fa-
           floppy-o"></i>
   </button>
   <div class="d-flex justify-content-md-center mt-4" *ngIf="hiba">
     <p class="alert alert-danger">{{ mentes_hiba }}</p>
   </div>
 </div>
</div>
</div>
</div>
```

Tanszékek listázása

Van lehetőségünk a rendszerben szereplő tanszékeket megtekintésére, valamint a tanszékek nevére rákeresni, és ezeket az adatokat ki is tudjuk nyomtatni. Ennek megfelelően a *tanszekek-lista.component.ts* fájl tartalma:

```
import { Component, OnInit } from '@angular/core';
import { Tanszekek } from 'src/app/models/tanszekek.model';
import { TanszekekService } from 'src/app/services/tanszekek.service';
@Component({
  selector: 'app-tanszekek-lista',
  templateUrl: './tanszekek-lista.component.html',
  styleUrls: ['./tanszekek-lista.component.css']
})
export class TanszekekListaComponent implements OnInit {
```

```
  kereses = false;
  tanszekek?: Tanszekek[] = [];
  tanszek_nev = '';
  oldal = 1;
  elemSzam = 0;
  oldalMeret = 3;
  constructor(
    private tanszekService: TanszekekService,
  ) { }
  ngOnInit(): void {
    this.tanszekListazasaLapozassal();
  }
  parameterekDefinialasa(tanszekNev: string, oldal: number,
                         oldalMeret: number): any {
    let parameterek: any = {};
    if (tanszekNev) { parameterek[`tanszek_nev`] = tanszekNev;}
    if (oldal) { parameterek[`oldal`] = oldal - 1;}
    if (oldalMeret) { parameterek[`meret`] = oldalMeret;}
    return parameterek;
  }
  tanszekListazasaLapozassal(): void {
    this.kereses = true;
    const parameterek = this.parameterekDefinialasa(this.tanszek_nev,
                        this.oldal, this.oldalMeret)
    this.tanszekService.tanszekListazasLapozassal(parameterek)
      .subscribe(
        response => {
          const { tanszekek, osszesElem } = response;
          this.tanszekek = tanszekek;
          this.elemSzam = osszesElem;
          console.log(response);
        },
        error => {
          this.tanszekek = [];
          console.log(error);
        });
  }
  oldalValtas(event: number): void {
    this.oldal = event;
    this.tanszekListazasaLapozassal();
  }
}
```

És a böngészőben a tanszékek lista megjelenítéséért felelős *tanszekek-lista.component.html* fájl szerkezete:

```
<div class="col-md-10 p-3 bg-white border ml-auto mr-auto mb-0 mt-0">
<div class="col-md-12">
 <div class="input-group mb-3 bg-light border p-4">
  <input (keyup.enter)="oldal = 1; tanszekListazasaLapozassal()"
         type="text" class="form-control mt-1 mb-1" placeholder =
         "Név szerinti keresés" [(ngModel)]="tanszek_nev" />
  <button class="btn btn-primary mt-1 mb-1 ml-1" type="button"
         (click)="oldal = 1; tanszekListazasaLapozassal()">
          <i class="fas fa-search"></i> Keresés
  </button>
  <button class="btn btn-danger mt-1 mb-1 ml-1" type="reset"
         (click)="tanszek_nev = ''; oldal = 1;
          tanszekListazasaLapozassal()"> <i class="fas fa-undo"></i>
  </button>
 </div>
```

```
</div>
<div class="container-fluid" *ngIf="tanszekek?.length">
<div class="row">
  <div class="col-md-6 text-left">
  <div class="ne-nyomtassa mt-1 mb-1">
    <pagination-controls previousLabel="Előző" nextLabel="Következő"
            [responsive]="true" class="row justify-content-center"
            (pageChange)="oldalValtas($event)"> </pagination-controls>
  </div>
  </div>
  <div class="col-md-6 text-right">
    <button class="btn btn-light border mt-1 mb-1 ml-1" mat-raised-
           button color="primary"[useExistingCss]="true"
           [printStyle]="{ div: {'padding':'50px'}}"
           printSectionId="print-section" ngxPrint>
           Nyomtatás <i class="fas fa-print"></i>
    </button>
   </div>
</div>
</div>
<div id="print-section">
<div class="table-responsive">
  <table class="table table-bordered table-striped table-hover">
    <thead>
     <tr *ngIf="tanszekek?.length">
      <th> # </th>
      <th> Tanszék neve </th>
      <th> Tanszék típusa </th>
     </tr>
    </thead>
    <tbody>
     <tr class="group-item" *ngFor="let tanszekek of tanszekek! |
         paginate : {itemsPerPage: oldalMeret, currentPage: oldal,
         totalItems: elemSzam } let i = index">
       <td>{{i+1}}</td>
       <td> {{tanszekek.tanszek_nev}} </td>
       <td> {{tanszekek.tanszek_tipus}} </td>
      </tr>
     </tbody>
   </table>
</div>
</div>
<div class="ml-3 text-secondary" *ngIf="!tanszekek?.length &&
      kereses">Nem található a keresésnek megfelelő elem.
</div>
</div>
```

Tanszék adatainak frissítése, tanszék törlése

A rendszerben szereplő tanszékeknek az adatait tudjuk módosítani, és ha már nincs szükségünk egy tanszékre, azt ki is tudjuk törölni. A *tanszekek-lista.component.ts* fájl tartalma ennek megfelelően:

```
import { Component, OnInit } from '@angular/core';
import { Tanszekek } from 'src/app/models/tanszekek.model';
import { TanszekekService } from 'src/app/services/tanszekek.service';
import { ModalDismissReasons, NgbModal } from '@ng-bootstrap/ng-
bootstrap';
@Component({
```

```
  selector: 'app-tanszekek-lista',
  templateUrl: './tanszekek-lista.component.html',
  styleUrls: ['./tanszekek-lista.component.css']
})
export class TanszekekListaComponent implements OnInit {
  ablak_bezaras: string | undefined;
  kereses = false;
  jelenlegiTanszek: Tanszekek = {
    tanszek_nev: '',
    tanszek_tipus: ''
  };
  uzenet = '';
  hiba = '';
  constructor(
    private modalService: NgbModal,
    private tanszekService: TanszekekService,
  ) { }
  ngOnInit(): void { }
  felugroAblak(content: any) {
    this.modalService.open(content, { ariaLabelledBy: 'modal-basic-
                  title', centered: true }).result.then((res) => {
      this.ablak_bezaras = `Closed with: ${res}`; (res) => {
      this.ablak_bezaras = `Dismissed ${this.bezarasIndok(res)}`;
    });
  }
  private bezarasIndok(reason: any): string {
    if (reason === ModalDismissReasons.ESC) {
      return 'by pressing ESC';
    } else if (reason === ModalDismissReasons.BACKDROP_CLICK) {
      return 'by clicking on a backdrop';
    } else {
      return `with: ${reason}`;
    }
  }
  tanszekFrissitese(): void {
    this.tanszekService.frissites(this.jelenlegiTanszek.id,
    this.jelenlegiTanszek)
      .subscribe(
        response => {
          console.log(response);
          this.uzenet = response.message;
        },
        error => {
          this.hiba = error.error.message;
          console.log(error);
        });
  }
  tanszekTorlese(): void {
    this.tanszekService.torles(this.jelenlegiTanszek.id)
      .subscribe(
        response => {
          console.log(response);
          this.modalService.dismissAll();
        },
        error => {
          console.log(error);
        });
  }
}
```

És a szerkesztés űrlap böngészőben történő megjelenítéséért felelős *tanszekek-lista.component.html* szerkezete pedig:

```
<div *ngIf="jelenlegiTanszek.id" class="edit-form">
<div class="card card-container">
<h4 class="container text-center">Szerkesztés</h4>
<div class="submit-form">
 <form>
  <div class="form-group">
  <label for="tanszek_nev">Tanszék neve</label>
   <input #tanszek_nev="ngModel" type="text" class="form-control"
      required id="tanszek_nev" name="tanszek_nev"
      [(ngModel)]="jelenlegiTanszek.tanszek_nev" />
   <div class="alert-danger" *ngIf="tanszek_nev.errors && hiba">
       A szervezeti egység nevének megadása kötelező
   </div>
  </div>
  <div class="form-group">
  <label for="tanszek_tipus">Tanszék típusa:</label>
   <ng-select #tanszek_tipus="ngModel" type="text" required
      [virtualScroll]="true" [clearable]="false" class="form-control"
      id="tanszek_tipus" [(ngModel)]="jelenlegiTanszek.tanszek_tipus"
      name="tanszek_tipus">
     <ng-option value="tanszék">tanszék</ng-option>
     <ng-option value="tanszéki csoport">tanszéki csoport </ng-option>
   </ng-select>
  </div>
 </form>
 <div class="d-flex justify-content-md-center">
  <button class="btn btn-danger btn-block mr-2 mt-2" type="submit"
            (click)="felugroAblak(modalData)">
                    Törlés <i class="fas fa-trash"></i>
  </button>
  <button (click)="hiba = ''; uzenet = ''; tanszekFrissitese()"
           type="submit" class="btn btn-success btn-block mr-2 mt-
           2">Frissítés <i class="fa fa-refresh"></i>
  </button>
 </div>
 <div class="d-flex justify-content-md-center mt-4" *ngIf="hiba">
  <p class="alert alert-danger">{{ hiba }}</p>
 </div>
 <div class="d-flex justify-content-md-center mt-4" *ngIf="uzenet">
  <p class="alert alert-success">{{ uzenet }}</p>
 </div>
</div>
<ng-template #modalData let-modal>
 <div class="modal-header">
  <h4 class="modal-title" id="modal-basic-title"> Törlés
                                                  megerősítése </h4>
 <button type="button" class="close" aria-label="Close"(click) =
     "modal.dismiss('Cross click')"> <span aria-hidden="true">×</span>
 </button>
 </div>
 <div class="modal-body">
  <p><strong>Biztosan törölni szeretné a tanszéket? </strong></p>
  <p>A tanszékhez kapcsolódó adatokat eltávolítjuk adatbázisból. </p>
 </div>
 <div class="modal-footer">
  <button type="button" class="btn btn-danger"
            (click)="tanszekTorlese(); modal.close('Save click')">
```

```
            <i class="fas fa-trash"></i> Törlés
  </button>
  <button type="button" class="btn btn-light border"
            (click)="modal.close('Save click')">Bezárás
  </button>
 </div>
 </ng-template>
</div>
</div>
```

Végül, miután végeztük a modellek, útvonalak, vezérlők és a különböző komponensek beállításával az `ng serve --host 192.168.0.105 --disable-host-check` parancs segítségével teszteltük a platform működését. Ennek a folyamatát a 2. melléklet mutatja be. A platform működésének az ellenőrzését követően elkészítettük a többi táblához is a megfelelő komponenseket, útvonalakat, modelleket és vezérlőket.

2.4. Az információs rendszer adatvédelmi komponenseinek létrehozása JSON Web Token (JWT) alapú technológia segítségével

Miután létrehoztuk és beállítottuk a szerver- és kliensoldali keretrendszereket, utolsó lépésként az adatok védelmét és a jogosultságok kezelését megvalósító komponensek kialakítása következett. Ehhez a már említett JSON Web Token (JWT) alapú technológiát fogjuk használni. Ennek az alapvető beállításait és moduljainak a telepítését oktatóanyag alapján fogjuk elkészíteni [4, 6] (*Megjegyzés:* a dolgozatban csak a munkatársak egyed bejelentkezését fogjuk bemutatni).

2.4.1. JSON Web Token a szerveroldali környezetben

A szerveroldali környezet biztonságos, token alapú azonosításhoz kötött adattovábbításának beállításához első lépésként a bcryptjs, jsonwebtoken modulokat telepítettük az `npm install bcryptjs jsonwebtoken --save` parancs segítségével.

A sikeres telepítést követően kialakítottuk a Szerepkörök egyed modelljét a Sequelize modulban és több a többhöz kapcsolaton keresztül összekapcsoltuk a Munkatársak táblával (így tudjuk biztosítani, hogy pl. egy munkatárs képviselhet a rendszerben egyszerre tanszéki adminisztrátor és tanár szerepkört is). Ennek megfelelően a Szerepkör reláció modellje:

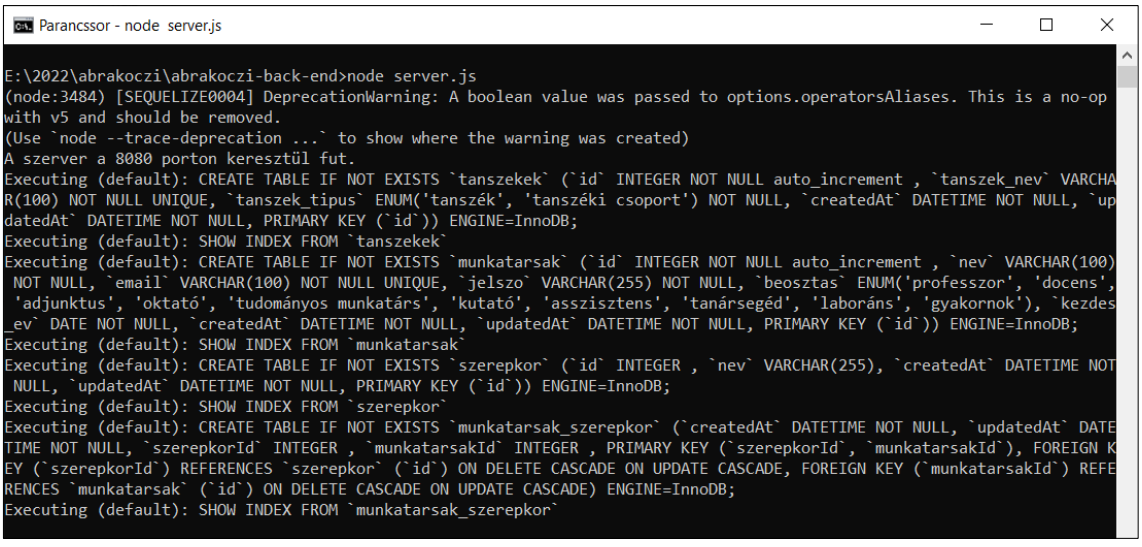
```
module.exports = (sequelize, Sequelize) => {
  const Szerepkor = sequelize.define("szerepkor", {
    id: {
      type: Sequelize.INTEGER,
```

```
      primaryKey: true
    },
    nev: {
      type: Sequelize.STRING
    }
  }, {
      freezeTableName: true
  });
  return Szerepkor;
};
```

Aztán a *models/index.js* fájlban meghívtuk a Szerepkör egyedet, és definiáltuk a Munkatársak és Szerepkörök között lévő több a többhöz kapcsolatot.

```
db.munkatarsak = require("./munkatarsak.model.js")(sequelize,
Sequelize);
db.szerepkor = require("../models/szerepkor.model.js")(sequelize,
Sequelize);
db.munkatarsak.belongsTo(db.tanszekek, {
  foreignKey: "tanszekekId",
  as: "tanszekek",
});
db.szerepkor.belongsToMany(db.munkatarsak, {
  through: "munkatarsak_szerepkor",
  foreignKey: "szerepkorId",
  otherKey: "munkatarsakId"
});
db.munkatarsak.belongsToMany(db.szerepkor, {
  through: "munkatarsak_szerepkor",
  foreignKey: "munkatarsakId",
  otherKey: "szerepkorId"
});
db.SZEREPKOR = ["student", "admin", "department_admin", "boss",
                "teacher", "study_department"];
```

Végül a `node server.js` parancs segítségével létrehoztuk a Szerepkör táblát és a Munkatársak és Szerepkör közötti kapcsolatot az 16. ábranak megfelelően.



16. ábra. Szerepkör létrehozása és összekapcsolása a Munkatársakkal (Saját kép, 2022)

Továbbá, mivel a jsonwebtoken függvények, mint például a *verify()* vagy a *sign()* olyan algoritmust használnak, amelynek titkos kulcsra van szüksége (karakter-láncként) a token kódolásához és dekódolásához, ezért az *app/config* könyvtárban létrehoztuk az *auth.config.js* fájlt az alábbi tartalommal:

```
module.exports = {
    secret: "janos-titkos-kulcsa"};
```

A szerepkörök azonosítása és a token érvényességének az ellenőrzése a *middleware/jwtHitelesites* fájlban fog történni az általunk definiált a *tokenEllenorzes*, *tanszekiAdmin* és *tanar* eljárások segítségével.

```
const jwt = require("jsonwebtoken");
const config = require("../config/auth.config.js");
const db = require("../models");
const Munkatarsak = db.munkatarsak;
tokenEllenorzes = (req, res, next) => {
    let token = req.headers["x-access-token"];
    if (!token) {
        return res.status(403).send({
            message: "Nincs token megadva!"
        }); }
    jwt.verify(token, config.secret, (err, decoded) => {
        if (err) {
            return res.status(401).send({
                message: "Jogosulatlan!"
            }); }
        req.munkatarsakId = decoded.id;
        next();
    }); };
tanar = (req, res, next) => {
    Munkatarsak.findByPk(req.munkatarsakId, {
        attributes: { exclude: ['jelszo'] },
    }).then(munkatarsak => {
        munkatarsak.getSzerepkors().then(szerepkor => {
            for (let i = 0; i < szerepkor.length; i++) {
                if (szerepkor[i].nev === "teacher") {
                    next();
                    return; } }
            res.status(403).send({
                message: "Tanári szerepkör szükséges!"
            });
            return;
        });
    }); };
tanszekiAdmin = (req, res, next) => {
    Munkatarsak.findByPk(req.munkatarsakId, {
        attributes: { exclude: ['jelszo'] },
    }).then(munkatarsak => {
        munkatarsak.getSzerepkors().then(szerepkor => {
            for (let i = 0; i < szerepkor.length; i++) {
                if (szerepkor[i].nev === "department_admin") {
                    next();
                    return; } }
            res.status(403).send({
                message: "Tanszéki adminisztrátor szerepkör
```

```
                                szükséges!"
            });
        });
    }); };
const jwtHitelesites = {
    tokenEllenorzes: tokenEllenorzes,
    tanar: tanar,
    tanszekiAdmin: tanszekiAdmin};
module.exports = jwtHitelesites;
```

A *middleware/index.js* fájlban exportáltuk *jwtHitelesites.js* által definiált eljárásokat.

```
const jwtHitelesites = require("./jwtHitelesites");
module.exports = {
  jwtHitelesites,
};
```

Aztán létrehoztuk a bejelentkezéseket kezelő *jwt-hitelesites.controller.js* vezérlőt és létrehoztuk a *bejelentkezes()* függvényt.

```
const { tanszekek } = require("../models");
const db = require("../models");
const config = require("../config/auth.config");
const Munkatarsak = db.munkatarsak;
var jwt = require("jsonwebtoken");
var bcrypt = require("bcryptjs");
exports.bejelentkezes = (req, res) => {
    Munkatarsak.findOne({
        where: {
            email: req.body.email
        }, include: [{ model: tanszekek, as: "tanszekek" }],
    })
        .then(munkatarsak => {
            if (!munkatarsak) {
                return res.status(404).send({ message: "A felhasználó
                                                    nem található." }); }
            var ervenyesJelszo = bcrypt.compareSync(
                req.body.jelszo,
                munkatarsak.jelszo);
            if (!ervenyesJelszo) {
                return res.status(401).send({
                    accessToken: null,
                    message: "Hibás jelszó!"
                });}
            var token = jwt.sign({ id: munkatarsak.id },
                config.secret, {
                expiresIn: 86400 // 24 óra
            });
            var authorities = [];
            munkatarsak.getSzerepkors().then(szerepkor => {
                for (let i = 0; i < szerepkor.length; i++) {
                    authorities.push("SZEREPKOR_" +
                    szerepkor[i].nev.toUpperCase());}
                res.status(200).send({
                    id: munkatarsak.id,
                    email: munkatarsak.email,
                    szerepkor: authorities,
                    accessToken: token,
                    nev: munkatarsak.nev,
```

```
                    kezdes_ev: munkatarsak.kezdes_ev,
                    tanszekId: munkatarsak.tanszekId,
                    tanszek: munkatarsak.tanszek,
                });
            });
        })
        .catch(err => {
            res.status(500).send({ message: err.message });
        }); };
```

Azt az útvonalat, aminek a segítségével a *bejelentkezes()* függvényt aktiválni fogjuk a *hitelesites.routes.js* fájlban állítottuk be.

```
const controller = require("../controllers/jwt-
                                hitelesites.controller");
module.exports = function(app) {
  app.use(function(req, res, next) {
    res.header(
      "Access-Control-Allow-Headers",
      "x-access-token, Origin, Content-Type, Accept"
    );
    next();
  });
  app.post("/api/hitelesites/bejelentkezes",
controller.bejelentkezes);
};
```

Végül pedig a *tanszekek.routes.js* fájlban az útvonalak aktiválását token alapú hitelesítéshez és szerepkörhöz kötöttük az alábbiaknak megfelelően:

```
const {jwtHitelesites} = require("../middleware");
const vezerlo = require("../controllers/tanszekek.controller");
module.exports = function (app) {
  app.use(function (req, res, next) {
        res.header(
          "Access-Control-Allow-Headers",
          "x-access-token, Origin, Content-Type, Accept"
        );
        next();
      });
  var utvonal = require("express").Router();
  utvonal.post("/", [jwtHitelesites.tokenEllenorzes,
      jwtHitelesites.tanszekiAdmin], vezerlo.letrehozas);

  utvonal.get("/", [jwtHitelesites.tokenEllenorzes,
      jwtHitelesites.tanar], vezerlo.tanszekListazasLapozassal);
};
require("./app/routes/hitelesites.routes")(app);
```

Tehát az információs rendszerben, a fent beállított komponenseknek köszönhetően, saját magunk tudjuk majd beállítani, hogy az egyes útvonalakat, és ezáltal az útvonalakhoz kapcsolódó függvényeket, ki és milyen szerepkör segítségével tudja majd elérni és aktiválni. Végül teszteltük a rendszert a 3. mellékleteknek megfelelően.

2.4.2. JSON Web Token a kliensoldali környezetben

A kliensoldal biztonságos, token alapú azonosításhoz kötött adattovábbításának a kialakításához az Angular 12 CLI környezetben az alábbi komponenseket generáltuk le:

- `ng g s _services/hitelesites` – a kliens- és szerveroldali keretrendszer összekötése a bejelentkezés megvalósításához;
- `ng g s _services/token-tarolas` – a felhasználó token alapú azonosításának kezeléséhez szükséges függvényeket fogja tárolni;
- `ng g c bejelentkezes` – a felhasználó bejelentkezését megvalósító komponensek halmaza.

A szerveroldali környezet beállításához a *_services/hitelesites.service.ts* fájlban definiáltuk azt az útvonalat, amit a bejelentkezés során a felhasználó aktiválni fog, mikor megpróbál belépni a saját fiókjába.

```
import { Injectable } from '@angular/core';
import { HttpClient, HttpHeaders } from '@angular/common/http';
import { Observable } from 'rxjs';
const AUTH_API = 'http://localhost:8080/api/hitelesites/';
const httpOptions = {
  headers: new HttpHeaders({ 'Content-Type': 'application/json' })
};
@Injectable({
  providedIn: 'root'
})
export class HitelesitesService {
  constructor(private http: HttpClient) { }
  bejelentkezes(email: string, jelszo: any): Observable<any> {
    return this.http.post(AUTH_API + 'bejelentkezes', {
      email,
      jelszo
    }, httpOptions);
  }
}
```

Aztán a *_services/token-tarolas.service.ts* fájlban beállítottuk a tokenek kezeléséhez szükséges függvényeket.

```
import { Injectable } from '@angular/core';
const TOKEN_KEY = 'auth-token';
const USER_KEY = 'auth-user';
@Injectable({
  providedIn: 'root'
})
export class TokenTarolasService {
  constructor() { }
  kijelentkezes(): void {
    window.sessionStorage.clear();
  }
  public tokenMentes(token: string): void {
    window.sessionStorage.removeItem(TOKEN_KEY);
```

```
    window.sessionStorage.setItem(TOKEN_KEY, token);
  }
  public tokenAtadas(): string | null {
    return window.sessionStorage.getItem(TOKEN_KEY);
  }
  public munkatarsakMentese(munkatarsak: any): void {
    window.sessionStorage.removeItem(USER_KEY);
    window.sessionStorage.setItem(USER_KEY,
JSON.stringify(munkatarsak));
  }
  public munkatarsakAtadasa(): any {
    const munkatarsak = window.sessionStorage.getItem(USER_KEY);
    if (munkatarsak) {
      return JSON.parse(munkatarsak);
    }
    return {};
  }
}
```

Az `app/_helpers` könyvtárban létrehoztuk az *auth.interceptor.ts* fájlt, amiben beállítottuk, hogy a rendszer a HTTP kéréseket ellenőrizze és átalakítsa, mielőtt azokat elküldenénk a szervernek. Ezt a HttpInterceptor *intercept()* függvény valósítja meg.

```
import { HTTP_INTERCEPTORS, HttpEvent } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { HttpInterceptor, HttpHandler, HttpRequest } from
'@angular/common/http';
import { TokenTarolasService } from '../_services/token-
tarolas.service';
import { Observable } from 'rxjs';
const TOKEN_HEADER_KEY = 'x-access-token';
@Injectable()
export class AuthInterceptor implements HttpInterceptor {
  constructor(private token: TokenTarolasService) { }
  intercept(req: HttpRequest<any>, next: HttpHandler):
Observable<HttpEvent<any>> {
    let authReq = req;
    const token = this.token.tokenAtadas();
    if (token != null) {
      authReq = req.clone({ headers: req.headers.set(TOKEN_HEADER_KEY,
token) });
    }
    return next.handle(authReq);
  }
}
export const authInterceptorProviders = [
  { provide: HTTP_INTERCEPTORS, useClass: AuthInterceptor, multi: true
} ];
```

Ezt a szolgáltatást az *app.module.ts* fájlban is meghívtuk az alábbi kódsor segítségével:

```
providers: [authInterceptorProviders],
```

Miután ezzel megvoltunk beállítottuk a bejelentkezési űrlap megfelelő működéséhez szükséges komponenseket. A *bejelentkezes.component.ts* fájl tartalma ennek megfelelően:

```
import { Component, OnInit } from '@angular/core';
import { HitelesitesService } from '../_services/hitelesites.service';
import { TokenTarolasService } from '../_services/token-
                                                    tarolas.service';
@Component({
  selector: 'app-bejelentkezes',
  templateUrl: './bejelentkezes.component.html',
  styleUrls: ['./bejelentkezes.component.css']
})
export class BejelentkezesComponent implements OnInit {
 szovegesMezo?: boolean;
 urlap: any = {
   email: null,
   jelszo: null
 };
 bejelentkezve = false;
 sikertelenBejelentkezes = false;
 munkatarsa: any;
 hibaUzenet = '';
 szerepkor: string[] = [];
 constructor(private hitelesitesService: HitelesitesService, private
                              tokenTarolas: TokenTarolasService) { }
 ngOnInit(): void {
   if (this.tokenTarolas.tokenAtadas()) {
    this.bejelentkezve = true;
    this.szerepkor = this.tokenTarolas.munkatarsakAtadasa().szerepkor;
    this.munkatars = this.tokenTarolas.munkatarsakAtadasa();
    } }
 szovegesMezoValtas() {
    this.szovegesMezo = !this.szovegesMezo;}
 bekuldeskor(): void {
  const { email, jelszo } = this.urlap;
  this.hitelesitesService.bejelentkezes(email, jelszo).subscribe(
   data => {
    this.tokenTarolas.tokenMentes(data.accessToken);
    this.tokenTarolas.munkatarsakMentese(data);
    this.sikertelenBejelentkezes = false;
    this.bejelentkezve = true;
    this.szerepkor = this.tokenTarolas.munkatarsakAtadasa().szerepkor;
    this.oldalFrissitese();
    },
      err => {
        this.hibaUzenet = err.error.message;
        this.sikertelenBejelentkezes = true;
      } ); }
  oldalFrissitese(): void {
    window.location.reload();
  } }
```

Aztán kialakítottuk a böngésző által megjelenített bejelentkezési űrlapot a *bejelentkezes.component.html* fájlban.

```
<div class="card card-container">
<img id="profile-img" src="//ssl.gstatic.com/accounts/ui/
                        qqqavatar_2x.png" class="profile-img-card" />
 <form *ngIf="!bejelentkezve" name="form" (ngSubmit)="f.form.valid &&
                                elkuldesre()" #f="ngForm" novalidate>
   <div class="form-group">
    <label for="email">Email</label>
    <input type="text" class="form-control" name="email"
           [(ngModel)]="urlap.email" required #email="ngModel" />
```

```
    <div class="alert alert-danger" role="alert" *ngIf="email.errors
           && f.submitted"> Email megadása kötelező! </div>
    </div>
    <div class="form-group">
     <label for="jelszo">Jelszó </label>
     <div class="input-group">
     <input [type]="szovegesMezo ? 'text' : 'password'" class="form-
           control" name="jelszo"[(ngModel)]="urlap.jelszo" required
           minlength="6" #password="ngModel" />
      <div class="input-group-append">
       <span class="input-group-text">
       <i class="fa" [ngClass]="{'fa-eye-slash': !szovegesMezo, 'fa-
             eye': szovegesMezo}" click)="szovegesMezoValtas()"></i>
       </span>
       </div>
      </div>
      <div class="alert alert-danger" role="alert"
                              *ngIf="password.errors && f.submitted">
      <div *ngIf="password.errors.required">Jelszó megadása kötelező
      </div>
      <div *ngIf="password.errors.minlength">
           A jelszónak legalább 6 karakter hosszúnak kell lennie
      </div>
     </div>
     </div>
     <div class="form-group">
      <button class="btn btn-primary btn-block"> Bejelenkezés
      </button>
      </div>
     <div class="form-group">
      <div class="alert alert-danger" role="alert" *ngIf="f.submitted
           && sikertelenBejelentkezes">
               A Bejelentkezés sikertelen: {{ hibaUzenet }}
      </div>
     </div>
    </form>
    <div class="alert alert-success" *ngIf="bejelentkezve">
        bejelentkezve, mint {{ munkatars.nev }}!
    </div>
</div>
```

Beállítottuk azt is, hogy a különböző menüpontok a navigálósávban csak bejelentkezetett felhasználónak jelenjen meg. Az *app.component.ts* fájl tartalma ennek megfelelően:

```
import { Component, OnInit  } from '@angular/core';
import { TokenTarolasService } from './_services/token-
tarolas.service';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent implements OnInit {
  menusor = true;
  title = 'abrakoczi-front-end';
  private szerepkor: string[] = [];
  bejelentkezve = false;
  tanszekiAdmin = false;
  constructor(private tokenTarolasService: TokenTarolasService, ) { }
```

```
  ngOnInit(): void {
    this.bejelentkezve = !!this.tokenTarolasService.tokenAtadas();
    if (this.bejelentkezve) {
      const munkatarsak =
this.tokenTarolasService.munkatarsakAtadasa();
      this.szerepkor = munkatarsak.szerepkor;
      this.tanszekiAdmin =
this.szerepkor.includes('SZEREPKOR_DEPARTMENT_ADMIN');
    }
  }
  kijelentkezes(): void {
    this.tokenTarolasService.kijelentkezes();
    window.location.reload();
  }
}
```

És végül a böngészőben megjelenített *app.component.html* komponens kialakítása következett:

```
    <ul class="navbar-nav ml-auto" *ngIf="!bejelentkezve">
      <li class="nav-item">
        <a href="/" class="nav-link"
                    routerLink="bejelentkezes">Bejelentkezés</a>
      </li>
    </ul>
    <ul class="navbar-nav ml-auto" *ngIf="bejelentkezve">
      <li class="nav-item">
        <a href="/" class="nav-link"
                      (click)="kijelentkezes()">Kijelentkezés</a>
      </li>
    </ul>
```

Miután beállítottuk a komponenseket teszteltük a platform működését a 4. mellékleteknek megfelelően. A sikeres ellenőrzést követően mindegy egyes útvonalhoz beállítottuk a megfelelő adatvédelmi- és jogosultsági komponenseket és ezáltal befejeztük az online webes információs rendszer kialakítását.

ÖSSZEFOGLALÁS

Diplomamunkám során online információs rendszer készítésével foglalkoztam az oktatási folyamatok adminisztrálására a II. Rákóczi Ferenc Kárpátaljai Magyar Főiskola példáján. A platform kialakításához felhasznált eszközök: 1.) szerveroldali keretrendszer (back-end): NodeJS, ExpressJS, Sequelize; 2.) kliensoldali keretrendszer (front-end): Angular 12 CLI; 3.) adattárolás és adatfeldolgozás: MySQL adatbáziskezelő rendszer; 4.) adatok védelme és jogosultságok kezelése: JSON Web Token (JWT).

A munka során első lépésként információt gyűjtöttünk az intézmény adminisztrációs folyamatairól és ezek alapján megterveztük a platform mögött működő adatbázist. A folyamat során meghatároztuk az egyedtípusokat és azok tulajdonságait, valamint megállapítottuk az egyedek között fennálló kapcsolatokat. A kitűzött céloknak megfelelően a rendszer gördülékeny működését 12 tábla biztosítja: Tanszékek, Szakok, Tantárgyak, Munkatársak, Diákok, Diákok száma, Mintatantervek, Eredmények Tarifikációs változók, Tantárgy tarifikációk, Gyakorlat tarifikációk, valamint a Szakdolgozat és évfolyammunka tarifikációk.

Miután megterveztük és létrehoztuk az adatbázist következő lépésként a szerveroldali keretrendszer kialakítása következett. Ennek a beállítása során figyelembe vettük a különböző felhasználói igényeket, és úgy írtuk meg az adatok kezelését megvalósító függvényeket, hogy a leendő felhasználóknak minél rugalmasabb lekérdezési és szűrési lehetőségeket tudjunk biztosítani a rendszer segítségével.

A kliensoldali keretrendszert szintén, a szerveroldal kialakítása során követett céloknak megfelelően készítettük el. Valamint úgy alakítottuk ki a felületet, hogy az új adatok hozzáadása és a meglévő adatok szerkesztése felugró ablakok (űrlapok) segítségével történjen, és a változtatások az űrlap elküldése után azonnal megjelenjenek a platformon. Ennek a tanári óraelosztások során kiemelkedő szerep jut.

Végül pedig az adatok védelmét és a megfelelő hozzáférési engedélyek kezelését lehetővé tévő komponenseket alakítottuk ki. A rendszerben 6 szerepkört hoztunk létre, amelyek különböző jogosultsággal rendelkeznek.

A JWT alapú biztonságos adatkezelési technológia lehetővé teszi a platform interneten történő megosztását, ezáltal a távoli elérés biztosítását a diákoknak és a tanároknak egyaránt. Ezáltal az információs rendszer az adminisztrációs folyamatok

megvalósításán túl elektronikus leckekönyvként is funkcionál, ami segítséget nyújt a távoktatás során a diákok szummatív értékelésének kivitelezésében és ezáltal szerepet vállal a hallgatók motivációjának a fenntartásában, az értékelési folyamat hatékony és gyors megvalósításán keresztül.

IRODALOMJEGYZÉK

1. A JavaScript programozási nyelv [Elektronikus forrás]: SIMON ÁKOS SCHAFFER KRISZTIÁN EGRI PÉTER és a többiek. 2014.
   Interneten: http://nyelvek.inf.elte.hu/leirasok/JavaScript/index.php
   [Hozzáférés: 2022.05.10.]
2. AZAT MARDAN: *Express.js Guide – The Comprehensive Book on Express.js*. Leanpub, 2014. 295 p.
   Interneten: https://pepa.holla.cz/wp-content/uploads/2016/08/Express.js-Guide.pdf
   [Hozzáférés: 2022.05.01]
3. BEZKODER [Elektronikus forrás]: *Angular 12 + Node.js Express + MySQL example: CRUD Application*. 2022.
   Interneten: https://www.bezkoder.com/angular-12-node-js-express-mysql/
   [Hozzáférés: 2022.05.13.]
4. BEZKODER [Elektronikus forrás]: *Angular 12 Login and Registration example with JWT & Web Api.* 2021.
   Interneten: https://www.bezkoder.com/angular-12-jwt-auth/
   [Hozzáférés: 2022.05.01.]
5. BEZKODER [Elektronikus forrás]: *Full Stack.* 2022.
   Interneten: https://www.bezkoder.com
   [Hozzáférés: 2022.05.01.]
6. BEZKODER [Elektronikus forrás]: *Node.js Express: JWT example | Token Based Authentication & Authorization.* 2021.
   Interneten: https://www.bezkoder.com/node-js-jwt-authentication-mysql/
   [Hozzáférés: 2022.05.13.]
7. BEZKODER [Elektronikus forrás]: *Node.js Rest APIs example with Express, Sequelize & MySQL*. 2021.
   Interneten: https://www.bezkoder.com/node-js-express-sequelize-mysql/
   [Hozzáférés: 2022.03.13.]
8. BUDA ANDRÁS: *IKT és oktatás, együtt vagy egymás mellett?* Belvedere Meridionale kiadó, 2017. 205 o.
   Interneten: http://real-eod.mtak.hu/9424/1/Buda_Andras_2017_%20Ikt_es_Oktatas.pdf
   [Hozzáférés: 2021.11.26.]
9. CAIO RIBEIRO PEREIRA: *Building APIs with Node.js*. São Vicente - SP, São Paulo, Brazil, 2016. 136 p.

10. CAROL EVANS (2013). *Making sense of assessment feedback in higher education*. Review of Educational Research 83(1), 2013.
11. DAVID FLANAGAN: *JavaScript: The Definitive Guide, 7th Edition*. O'Reilly Media, Inc., 2020. 740 p.
12. Didaktika – Elméleti alapok a tanítás tanuláshoz / szerk. FALUS IVÁN. Nemzeti Tankönyvkiadó, Budapest, 2003. 448 o.
    Interneten: http://biologus2011.weebly.com/uploads/8/9/1/5/8915888/falus-ivan-didaktika.pdf ; https://docplayer.hu/21399618-Didaktika-elmeleti-alapok-a-tanitas-tanulasahoz.html
    [Hozzáférés: 2022.05.06.]
13. ETHAN BROWN: *Web Development with Node and Express*. O'Reilly Media, Inc. 2014. 307 p.
    Interneten: https://www.vanmeegern.de/fileadmin/user_upload/PDF/Web_Development_with_Node_Express.pdf
    [Hozzáférés: 2022.04.05.]
14. IKT innováció / LENGYELNÉ MOLNÁR TÜNDE, KIS-TÓTH LAJOS, ANTAL PÉTER, RACSKÓ RÉKA. Eger, 2015. 141 o.
    Interneten: http://okt.ektf.hu/data/szlahorek/file/kezek/05_ikt_02_27/index.html ; http://p2014-25.palyazat.ektf.hu/public/uploads/5-ikt-innovacio-lengyelne-kis-toth-antal-racsko-isbn_565d5553721a1.pdf
    [Hozzáférés: 2021.09.16]
15. JOHN HATTIE, HELEN TIMPERLEY: *The power of feedback*. Review of Educational Research 77(1), 2007.
16. JOHN L. VIESCAS, MICHAEL J. HERNANDEZ: *SQL-lekérdezések földi halandóknak*. Kiskapu Kft, Budapest, 2009. 560 o.
17. MOLNÁR EDIT KATALIN, VÍGH, TIBOR: *A tantervelmélet és a pedagógiai értékelés alapjai*. Szegedi Tudományegyetem, „Mentor(h)áló 2.0 Program", TÁMOP-4.1.2.B.2-13/1- 2013-0008 projekt, 2013.
    Interneten: http://www.jgypk.hu/mentorhalo/tananyag/Tantervelmlet_s_a_pedaggiai_rtkels_alapjai/index.html
    [Hozzáférés: 2021.10.10.]
18. MOLNÁR GYÖRGY: *Az IKT-val támogatott tanulási környezet követelményei és fejlesztési lehetőségei.* Szakképzési Szemle, Nemzeti Szakképzési és Felnőttképzési Hivatal, Budapest, 2008. 257-278 o.
    Interneten: http://www.mszt.iif.hu/documents/szsz0803-molnar.pdf ; https://www.researchgate.net/publication/288787134_Az_IKT-val_tamogatott_tanulasi_kornyezet_kovetelmenyei_es_fejlesztesi_lehetosegei
    [Hozzáférés: 2021.11.21.]

19. NATE MURRAY, FELIPE COURY, ARI LERNER, CARLOSTABORDA: *ng-book The Complete Guide to Angular*. San Francisco, California, 2020. 628 p.
20. OKAN BULUT, MARIA CUTUMISU, ALEXANDRA AQUILINA, DEEPAK SINGH: *Effects of Digital Score Reporting and Feedback on Students' Learning in Higher Education*, Frontiers in Education 4(65), 2019.
21. POOJA MAHINDRAKAR, UMA PUJERI: *Insights of JSON Web Token*. International Journal of Recent Technology and Engineering 8(6), 2020. 1707-1710 p.
    Interneten: https://research.mitwpu.edu.in/publication/insights-of-json-web-token
    [Hozzáférés: 2022.02.29.]
22. Radványi Tibor: Adatbázisrendszerek. Hallgatói Információs központ, 2014. 221 o.
    Interneten: https://dtk.tankonyvtar.hu/xmlui/bitstream/handle/123456789/3476/2011-0038_22_radvanyi_hu.pdf ;
    https://people.inf.elte.hu/kiss/DB/adatbazisrendszerek.pdf
    [Hozzáférés: 2022.02.24.]
23. RADVÁNYI TIBOR: *Haladó DBMS.* ISBN Kiadó, 2011. 112 o.
    Interneten: http://aries.ektf.hu/~hz/pdf-tamop/pdf-xx/Radvanyi-hdbms-HU.pdf
    [Hozzáférés: 2022.05.10.]
24. SEBASTIÁN E. PEYROTT: *The JWT Handbook*. Auth0 Inc., 2016-2018. 118 p.
    Interneten:
    https://assets.ctfassets.net/2ntc334xpx65/o5J4X472PQUI4ai6cAcqg/13a2611de03b2c8edbd09c3ca14ae86b/jwt-handbook-v0_14_1.pdf
    [Hozzáférés: 2021.12.11.]
25. SEQUELIZE CONTRIBUTORS [Elektronikus forrás]: *Sequelize v6*. 2022.
    Interneten: https://sequelize.org/docs/v6/
    [Hozzáférés: 2022.04.21.]
26. STEFAN DETSCHEW: *Impact of Ict in the Developing Countries on the Economic Growth*. Auflage, 2007. p.28.
27. SUPER-POWERED BY GOOGLE [Elektronikus forrás]: *Introduction to Angular concepts*. Angular Docs, 2010-2021.
    Interneten: https://angular.io/guide/architecture
    [Hozzáférés: 2022.04.22.]
28. SZABÓ BÁLINT: *Adatbázis fejlesztés és üzemeltetés II.* Eszterházy Károly Főiskola, Eger, 2013. 187 o.
    Interneten: https://mek.oszk.hu/14400/14432/pdf/14432.pdf
    [Hozzáférés: 2022.03.01.]
29. Távoktatás a II. Rákóczi Ferenc Kárpátaljai Magyar Főiskolán [Elektronikus forrás].

Interneten: https://kmf.uz.ua/hu/rendelet-a-ii-rakoczi-ferenc-karpataljai-magyar-foiskola-tavoktatasi-rendszerenek-tokeletesiteserol/
[Hozzáférés: 2021.09.01.]

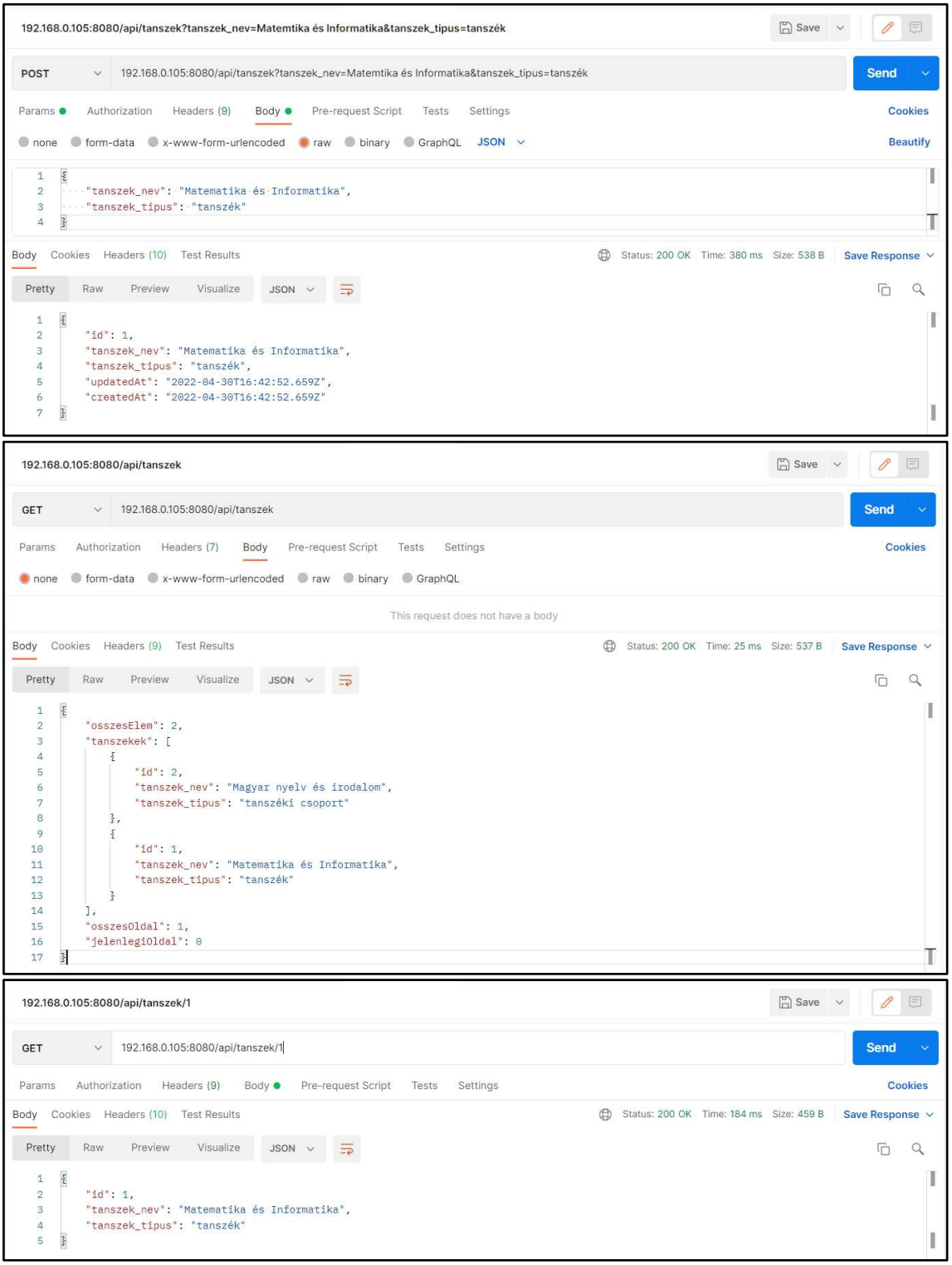
30. TÖRÖK BALÁZS: *Az elektronikus iskolai adminisztráció (1. rész).* Új Pedagógiai Szemle, Infrastruktúra és funkciók, Oktatáskutató és Fejlesztő Intézet Kiadói és Kommunikációs Központ, 2013/3-4. p. 25-42.
Interneten:
https://folyoiratok.oh.gov.hu/uj-pedagogiai-szemle/az-elektronikus-iskolai-adminisztracio-1-resz-0 ;
https://www.epa.oszk.hu/00000/00035/00157/pdf/EPA00035_upsz_2013_03-04_025-042.pdf
[Hozzáférés: 2022.01.11.]
31. VALERIE J. SHUTE: *Focus on formative feedback*. Review of Educational Research 78(1), 2008.

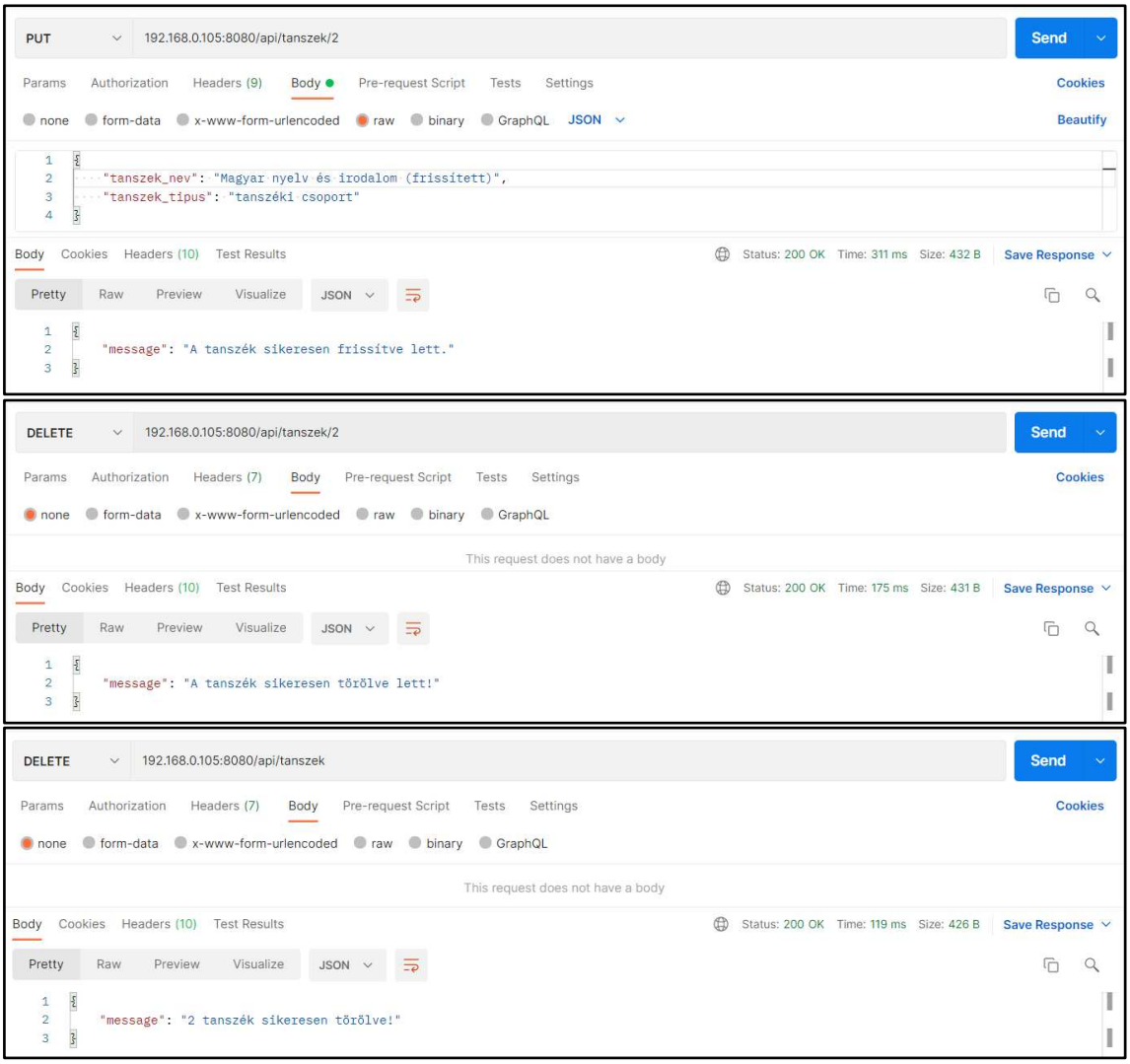
ÁBRAJEGYZÉK

1. ábra. Táblák a relációs adatmodellben (Forrás: Szabó, 2013) 13
2. ábra. A tanszékek és szakok kapcsolata (Saját kép, 2022) 24
3. ábra. A tanszékek és tantárgyak kapcsolata (Saját kép, 2022) 25
4. ábra. A tanszékek és munkatársak kapcsolata (Saját kép, 2022) 26
5. ábra. A szakok és diákok közötti kapcsolat (Saját kép, 2022) 27
6. ábra. A diákok száma és szak közötti kapcsolat (Saját kép, 2022) 28
7. ábra. A tantárgy, mintatanterv és szak közötti kapcsolat (Saját kép, 2022) 30
8. ábra. Az eredmények diákok, tantárgyak és munkatársak kapcsolata (Saját kép, 2022) 33
9. ábra. A tantárgyak, tantárgy tarifikációk és munkatársak kapcsolata (Saját kép, 2022) 36
10. ábra. A gyakorlat tarifikációk, tanszékek és munkatársak kapcsolata (Saját kép, 2022) 37
11. ábra. Szakdolgozat és évfolyammunka tarifikációk, szakok és munkatársak kapcsolata (Saját kép, 2022) 39
12. ábra. Az abrakoczi adatbázis kapcsolatrendszere (Saját kép, 2022) 40
13. ábra. Segédprogram a *package.js* létrehozásához (Saját kép, 2022) 47
14. ábra. ExpressJS webszerver működése a 8080-as porton (Saját kép, 2022) 48
15. ábra. Tanszékek reláció létrehozása NodeJS, ExpressJS, Sequelize segítségével (Saját kép, 2022) 50
16. ábra. Szerepkör létrehozása és összekapcsolása a Munkatársakkal (Saját kép, 2022) 66

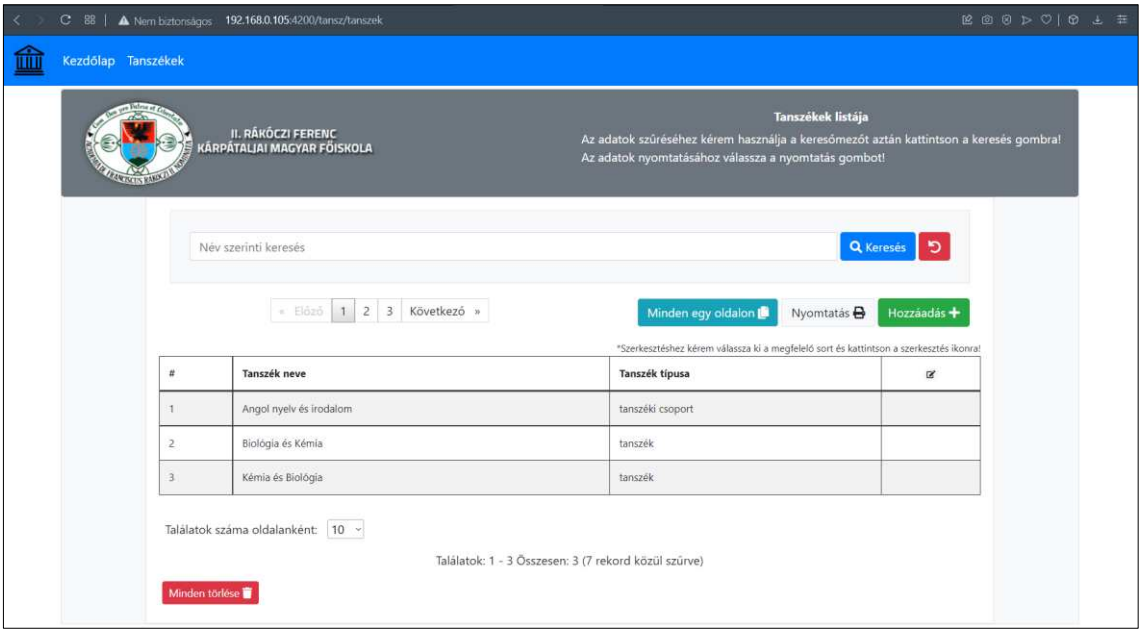
MELLÉKLETEK

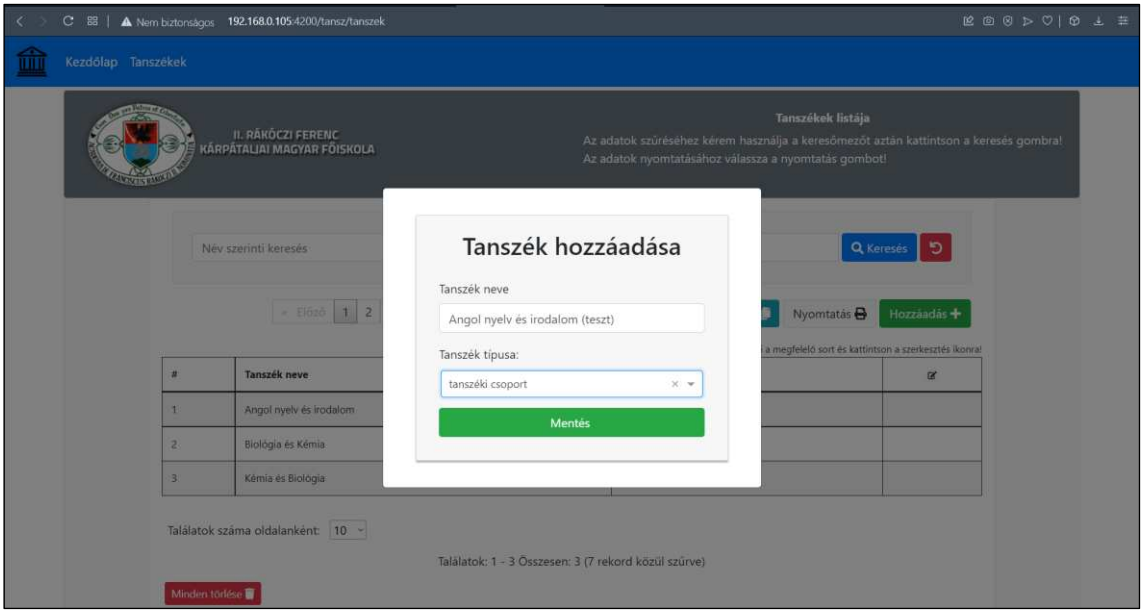
1. melléklet. A szerveroldali környezet tesztelése (Saját képek, 2022)

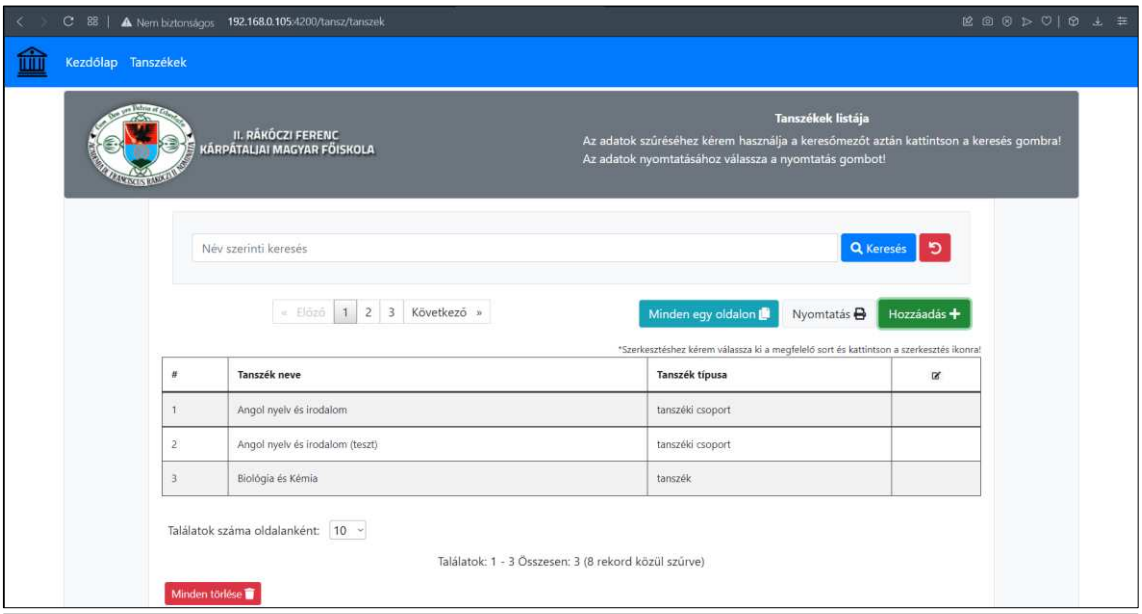


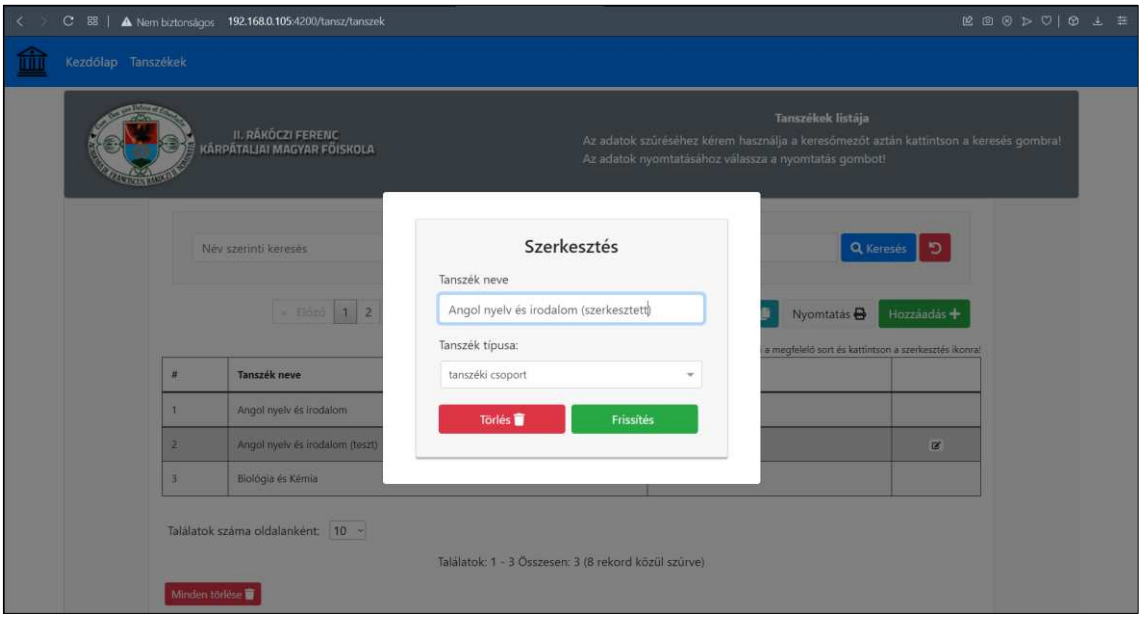


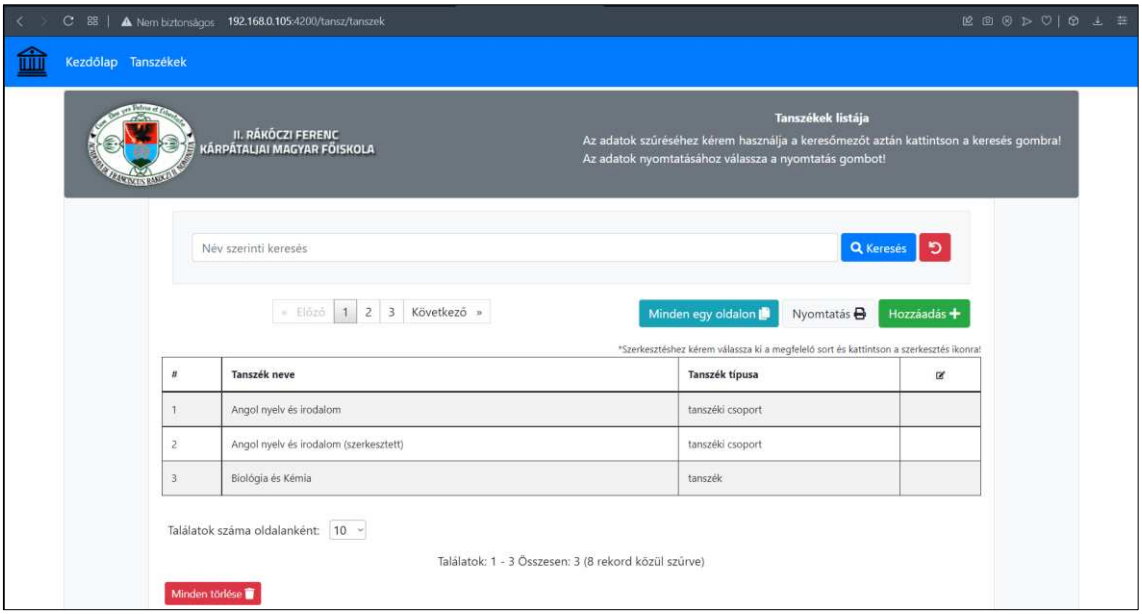
2. melléklet. A kliensoldali környezet tesztelése (Saját képek, 2022)

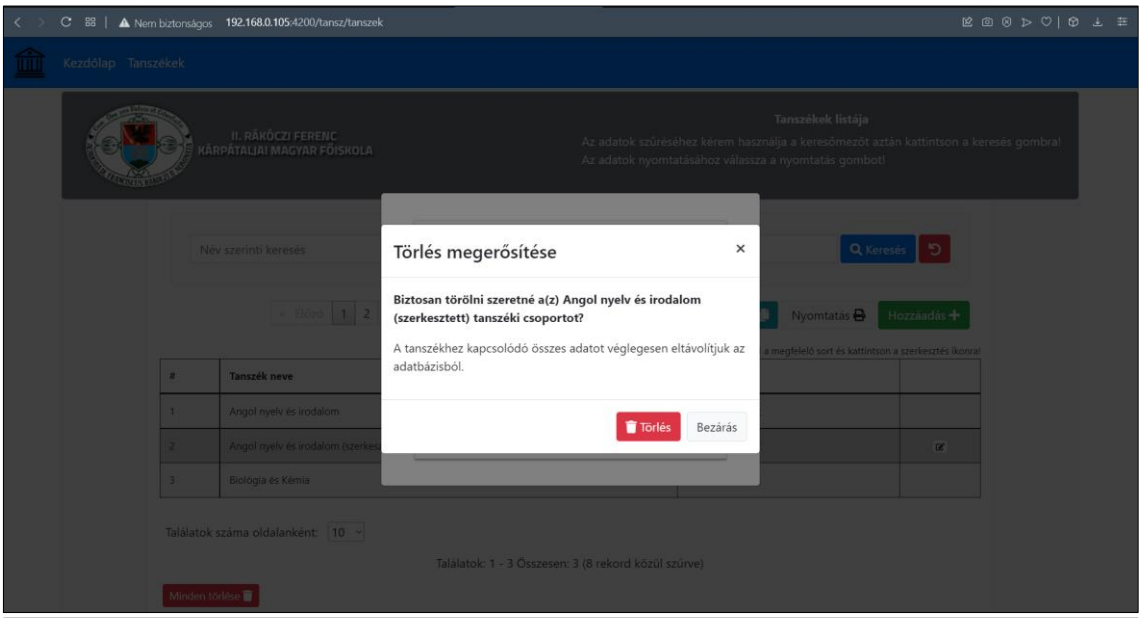


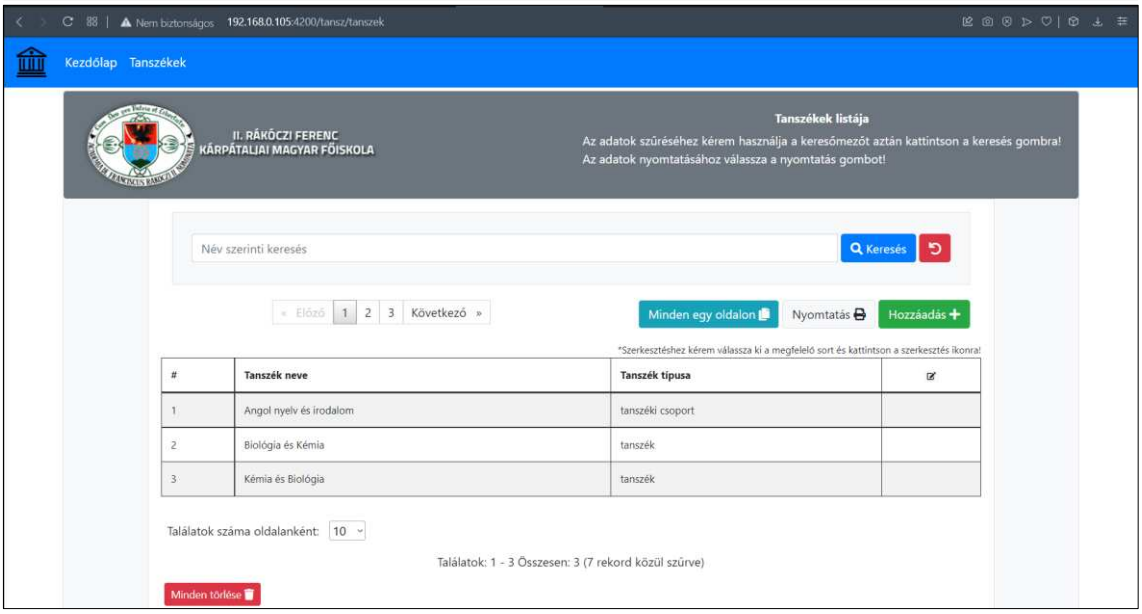

Nem biztonságos 192.168.0.105:4200/tansz/tanszek
Kezdőlap Tanszékek
II. RÁKÓCZI FERENC KÁRPÁTALJAI MAGYAR FŐISKOLA
Tanszékek listája
Az adatok szűréséhez kérem használja a keresőmezőt aztán kattintson a keresés gombra!
Az adatok nyomtatásához válassza a nyomtatás gombot!
Tanszék hozzáadása
Tanszék neve
Angol nyelv és irodalom (teszt)
Tanszék típusa:
tanszéki csoport
Mentés
Név szerinti keresés
Keresés
Nyomtatás
Hozzáadás
Tanszék neve
1 Angol nyelv és irodalom
2 Biológia és Kémia
3 Kémia és Biológia
Találatok száma oldalanként: 10
Találatok: 1 - 3 Összesen: 3 (7 rekord közül szűrve)
Minden törlése


Nem biztonságos 192.168.0.105:4200/tansz/tanszek
Kezdőlap Tanszékek
II. RÁKÓCZI FERENC KÁRPÁTALJAI MAGYAR FŐISKOLA
Tanszékek listája
Az adatok szűréséhez kérem használja a keresőmezőt aztán kattintson a keresés gombra!
Az adatok nyomtatásához válassza a nyomtatás gombot!
Név szerinti keresés
Keresés
« Előző 1 2 3 Következő »
Minden egy oldalon
Nyomtatás
Hozzáadás
*Szerkesztéshez kérem válassza ki a megfelelő sort és kattintson a szerkesztés ikonra!
Tanszék neve Tanszék típusa
1 Angol nyelv és irodalom tanszéki csoport
2 Angol nyelv és irodalom (teszt) tanszéki csoport
3 Biológia és Kémia tanszék
Találatok száma oldalanként: 10
Találatok: 1 - 3 Összesen: 3 (8 rekord közül szűrve)
Minden törlése

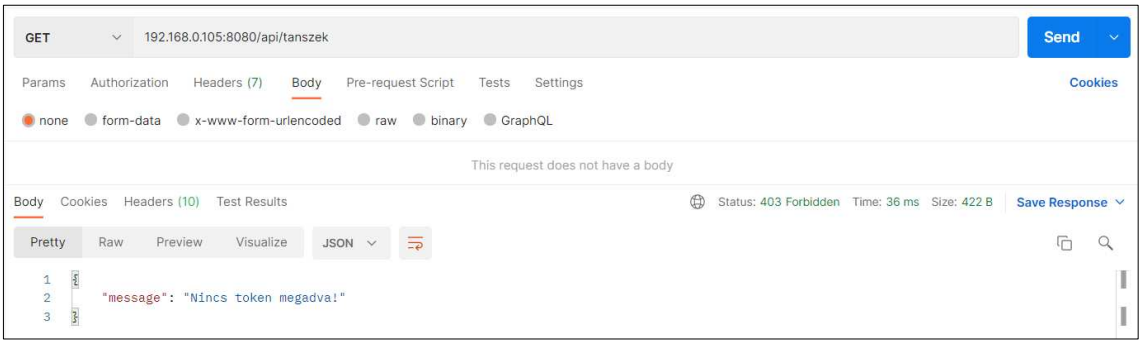

Nem biztonságos 192.168.0.105:4200/tansz/tanszek
Kezdőlap Tanszékek
II. RÁKÓCZI FERENC KÁRPÁTALJAI MAGYAR FŐISKOLA
Tanszékek listája
Az adatok szűréséhez kérem használja a keresőmezőt aztán kattintson a keresés gombra!
Az adatok nyomtatásához válassza a nyomtatás gombot!
Szerkesztés
Tanszék neve
Angol nyelv és irodalom (szerkesztett)
Tanszék típusa:
tanszéki csoport
Törlés
Frissítés
Találatok száma oldalanként: 10
Találatok: 1 - 3 Összesen: 3 (8 rekord közül szűrve)
Minden törlése

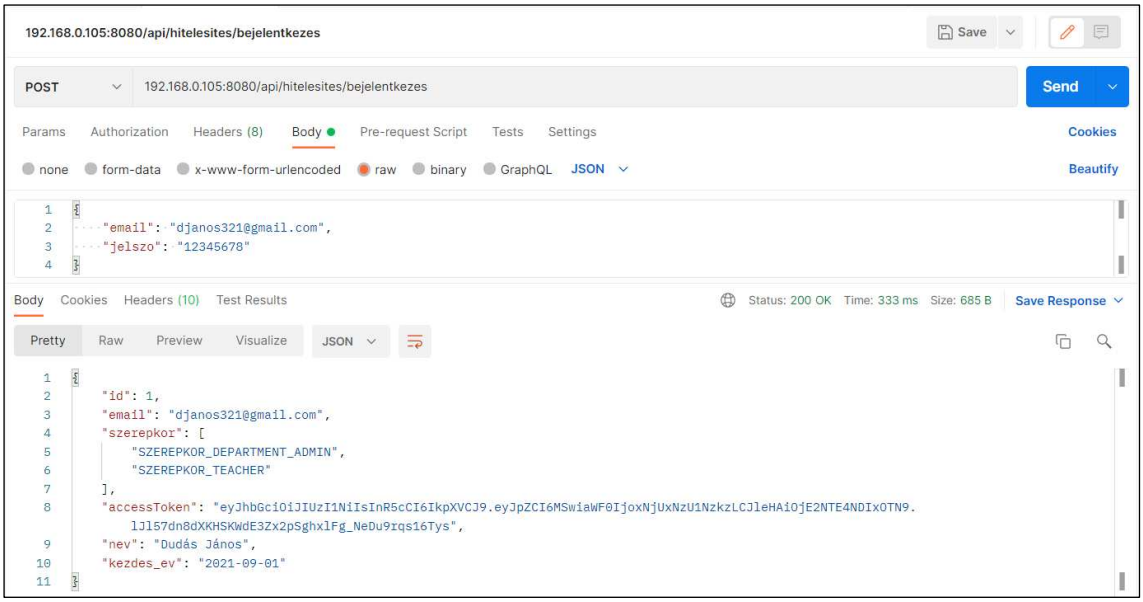

Nem biztonságos 192.168.0.105:4200/tansz/tanszek
Kezdőlap Tanszékek
II. RÁKÓCZI FERENC KÁRPÁTALJAI MAGYAR FŐISKOLA
Tanszékek listája
Az adatok szűréséhez kérem használja a keresőmezőt aztán kattintson a keresés gombra!
Az adatok nyomtatásához válassza a nyomtatás gombot!
Név szerinti keresés
Keresés
« Előző 1 2 3 Következő »
Minden egy oldalon
Nyomtatás
Hozzáadás
*Szerkesztéshez kérem válassza ki a megfelelő sort és kattintson a szerkesztés ikonra!
Tanszék neve Tanszék típusa
1 Angol nyelv és irodalom tanszéki csoport
2 Angol nyelv és irodalom (szerkesztett) tanszéki csoport
3 Biológia és Kémia tanszék
Találatok száma oldalanként: 10
Találatok: 1 - 3 Összesen: 3 (8 rekord közül szűrve)
Minden törlése

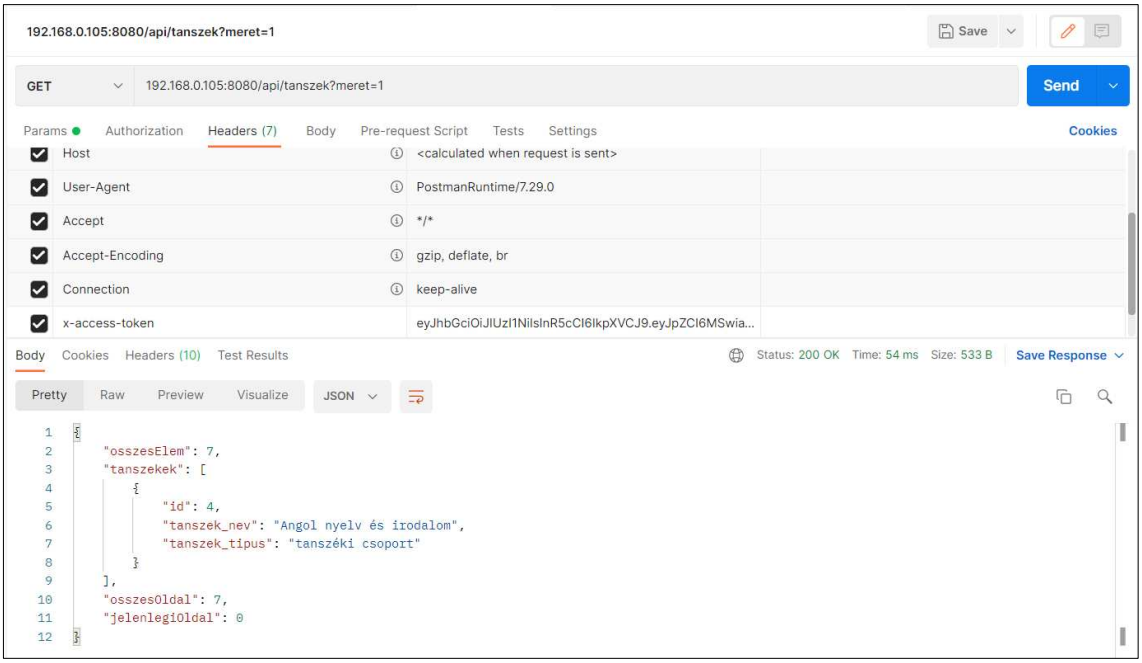

Törlés megerősítése
Biztosan törölni szeretné a(z) Angol nyelv és irodalom (szerkesztett) tanszéki csoportot?
A tanszékhez kapcsolódó összes adatot véglegesen eltávolítjuk az adatbázisból.
Törlés
Bezárás


Nem biztonságos 192.168.0.105:4200/tansz/tanszek
Kezdőlap Tanszékek
II. RÁKÓCZI FERENC KÁRPÁTALJAI MAGYAR FŐISKOLA
Tanszékek listája
Az adatok szűréséhez kérem használja a keresőmezőt aztán kattintson a keresés gombra!
Az adatok nyomtatásához válassza a nyomtatás gombot!
Név szerinti keresés
Keresés
« Előző 1 2 3 Következő »
Minden egy oldalon
Nyomtatás
Hozzáadás
*Szerkesztéshez kérem válassza ki a megfelelő sort és kattintson a szerkesztés ikonra!
Tanszék neve Tanszék típusa
1 Angol nyelv és irodalom tanszéki csoport
2 Biológia és Kémia tanszék
3 Kémia és Biológia tanszék
Találatok száma oldalanként: 10
Találatok: 1 - 3 Összesen: 3 (7 rekord közül szűrve)
Minden törlése

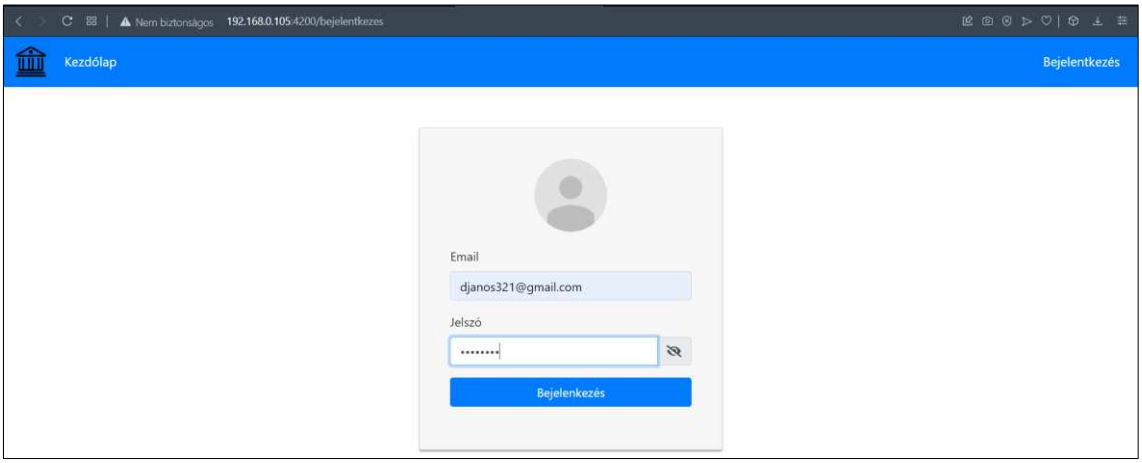
3. melléklet. JWT szerveroldali tesztelése (Saját képek, 2022)

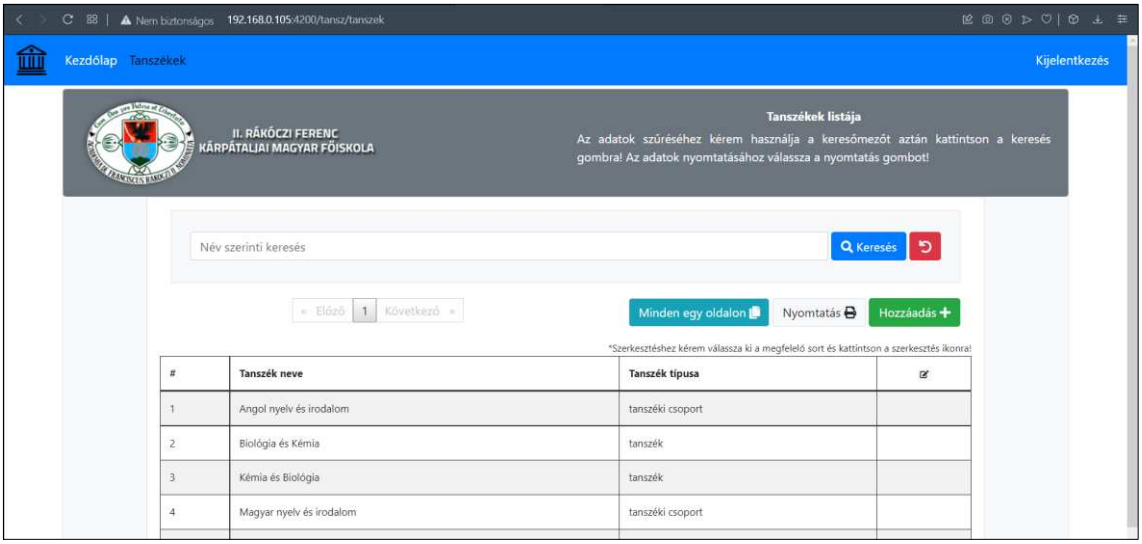






4. melléklet. JWT kliensoldali tesztelése (Saját képek, 2022)





ВИСНОВКИ

Під час виконання дипломної роботи я займався створенням онлайн-інформаційної системи для адміністрування освітніх процесів на прикладі Закарпатського угорського інституту ім. Ференца Ракоці II. Для побудови платформи були використані інструменти: 1.) серверний фреймворк (back-end): NodeJS, ExpressJS, Sequelize; 2.) фреймворк на стороні клієнта (front-end): Angular 12 CLI; 3.) зберігання та обробка даних: система керування базами даних MySQL; 4.) захист даних та керування авторизацією: JSON Web Token (JWT).

На початку роботи ми зібрали інформацію про адміністративні процеси управління навчальним процесом і на основі цього розробили базу даних платформи. Були визначені типи сутностей, їх властивості, а також зв'язки між ними. Відповідно до поставлених цілей безперебійну роботу системи забезпечують 12 таблиць: кафедри, курси, предмети, співробітники, студенти, кількість студентів, зразки навчальних планів, змінні тарифікації, тарифікація предметів, тарифікація практик і дипломних та курсових робіт.

Після проектування та створення бази даних наступним кроком було створення серверної частини. При налаштуванні ми врахували різні потреби користувачів і написали функції, які реалізують управління даними таким чином, щоб забезпечити найбільш гнучкі варіанти запитів і фільтрації для потенційних користувачів за допомогою системи.

Ми також створили фреймворк на стороні клієнта відповідно до цілей, які переслідували під час розробки серверної сторони. Розроблено інтерфейс так, щоб можна було додавати нові дані та редагувати наявні за допомогою спливаючих вікон (форм), а зміни відображалися на платформі, щойно форму надана. Це значно спрощує розподіл навантаження викладачів.

Розроблені також компоненти, які дозволяють захистити дані та керувати відповідними правами доступу. Для цього в системі було створено 6 ролей.

Технологія безпечного керування даними на основі JWT дозволяє використовувати платформу через Інтернет, забезпечуючи віддалений доступ як для студентів, так і для викладачів. Таким чином, крім реалізації адміністративних процесів, інформаційна система також містить електронну залікову книжку, що

спрощує підсумкове оцінювання студентів при дистанційному навчанні і, таким чином, відіграє роль у підтримці мотивації студентів шляхом ефективного та швидкого впровадження процес оцінювання.

Köszönetnyilvánítás

Elsősorban szeretnék köszönetet mondani prof. Holovács József témavezetőmnek, segítőkész és türelmes támogatásáért, hasznos tanácsaiért, valamint alapos munkájáért. Illetve Beregszászi István Tanár Úrnak, aki jó tanácsokkal látott el, ha bármilyen kérdés merült fel a szakdolgozat írása közben.

Továbbá szeretném megköszönni azoknak, akik mindvégig bíztattak és támogattak, hogy ez a szakdolgozat elkészülhessen.

UNICHECK

Ім'я користувача:
Моца Андрій Андрійович

ID перевірки:
1011318103

Дата перевірки:
24.05.2022 14:20:18 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
24.05.2022 15:08:27 EEST

ID користувача:
100006701

Назва документа: **Dudas-Janos-Diplomamunka**

Кількість сторінок: **90** Кількість слів: **18571** Кількість символів: **155155** Розмір файлу: **2.84 MB** ID файлу: **1011204957**

7.82%
Схожість

Найбільша схожість: **1.22%** з Інтернет-джерелом (https://www.bezkoder.com/angular-12-node-js-express-mysql)

6.4% Джерела з Інтернету	434	Сторінка 92
2.54% Джерела з Бібліотеки	123	Сторінка 95

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	1

Nyilatkozat

Alulírott, Dudás János, 014. Középiskolai oktatás (Matematika) képzési program hallgatója, kijelentem, hogy a dolgozatomat a II. Rákóczi Ferenc Kárpátaljai Magyar Főiskolán, a Matematika és Informatika Tanszéken készítettem, 014. Középiskolai oktatás (Matematika) MSc diploma megszerzése végett.

Kijelentem, hogy a dolgozatot más szakon korábban nem védtem meg, saját munkám eredménye, és csak a hivatkozott forrásokat (szakirodalom, eszközök stb.) használtam fel.

Tudomásul veszem, hogy dolgozatomat a II. Rákóczi Ferenc Kárpátaljai Magyar Főiskola könyvtárában a kölcsönözhető könyvek között helyezik el.