ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

NEUMANN JÁNOS
INFORMATIKAI KAR

SZAKDOLGOZAT

OE-NIK	Hallgató neve:	**Gadácsi Ákos**
2022	Hallgató törzskönyvi száma:	**T/007025/FI12904/N**

Óbudai Egyetem
Neumann János Informatikai Kar
Szoftvertervezés és -fejlesztés Intézet

SZAKDOLGOZAT FELADATLAP

Hallgató neve: **Gadácsi Ákos**
Törzskönyvi száma: T/007025/FI12904/N
Neptun kódja: [redacted]

A dolgozat címe:

Közoktatási könyvigénylést végző adminisztációs webalkalmazás ASP.NET alapokon
Adminstrative webapplication for book distribution in public education with ASP.NET

Intézményi konzulens: Sipos Miklós László
Külső konzulens:

Beadási határidő: 2022. május 15.

A záróvizsga tárgyai: Szoftvertechnológia és grafikus felhasználói interfész tervezése
Szoftvertervezés és -fejlesztés specializáció

A feladat

A feladat egy webalkalmazás készítése iskolai adminisztrációs feladatok ellátására, az aktuális új technológiák felhasználásával. Vizsgálja meg, hogy mely technológiák szükségesek egy webes rendszer készítéséhez, elemezze azokat a felhasználhatóság oldaláról. Az adott technológia melletti döntését indokolja. A rendszer legyen képes egy felület biztosítására, ahol a közoktatásban történő könyvigényléseket lehet végezni, ehhez diákok új évfolyamba való lépése legyen szerver időhöz kötötten automatizálva. A feladatban a kivételes esetek is legyenek manuálisan modósithatók (pl. ha adott diák év közben valamilyen váltást eszközöl). A webszervert készítse el ASP.NET segítségével, és használjon fel ezzel jól együttműködő modulokat (pl. Identity). Végezzen továbbá elemzést a tesztelési lehetőségekről és készítse is el az egyes teszt típusokat és teszteseteket.

A dolgozatnak tartalmaznia kell:

- a megoldandó feladat pontos leírását és részletes elemzését,
- magas szintű szerkezeti mintát az alkalmazásról,
- az egyes funkciók megvalósításához felhasználható lehetséges technológiákat,
- az elkészített rendszer tervét, blokkdiagramját,
- a megvalósítás menetét, felmerülő problémákat és az azokra adott megoldást,
- a tesztelési tervet, a tesztelés módszertanát és a tesztelés eredményeit,
- az elkészült szoftver felhasználási lehetőségeit.

ÓBUDAI EGYETEM * Neumann János Informatikai Kar * 1 Budapest

P.H.

.....
Dr. Vámossy Zoltán
intézetigazgató

A szakdolgozat elévülésének határideje: **2024. május 15.**
(OE TVSz 55.§ szerint)

A dolgozatot beadásra alkalmasnak tartom:

.....
külső konzulens

.....
intézményi konzulens

HALLGATÓI NYILATKOZAT

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

Budapest, 2022.05.13.

.....
hallgató aláírása

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar
Neumann Faculty of Informatics

KONZULTÁCIÓS NAPLÓ

Hallgató neve:	Neptun kód:	Tagozat:
GADÁCSI ÁKOS	[redacted]	NAPPALI

Telefon: Levelezési cím (pl: lakcím):

[redacted]

Szakdolgozat címe magyarul:

KÖZOKTATÁSI KÖNYVIGÉNYLÉST VÉGZŐ ADMINISZTRÁCIÓS WEBALKALMAZÁS ASP.NET ALAPOKON

Szakdolgozat címe angolul:

ADMINISTRATIVE WEB APPLICATION FOR BOOK DISTRIBUTION IN PUBLIC EDUCATION WITH ASP.NET

Intézményi konzulens:	Külső konzulens:
SIPOS MIKLÓS LÁSZLÓ	---

Alk.	Dátum	Tartalom	Aláírás
1.	2021.09.10.	Félév teendőinek, menetrendjének és mérföldköveinek ismertetése.	
2.	2021.10.18.	Hasonló rendszerek elemzése.	
3.	2021.11.22.	Irodalomkutatás, annak összegzése valamint konzekvenciák meghatározása.	
4.	2021.12.06.	Tervezés, követelmények specifikációja. Véglegesítés. Leadással kapcsolatos teendők egyeztetése.	

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

A hallgató a Szakdolgozat I. (BSc) tantárgy követelményét teljesítette, beszámolóra bocsátható.

Budapest, 2021.12.06.

Sipos Miklós

intézményi konzulens

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

KONZULTÁCIÓS NAPLÓ

Hallgató neve: GADÁCSI ÁKOS

Neptun Kód: [redacted]

Tagozat: NAPPALI TAGOZAT

Telefon: [redacted]

Levelezési cím (pl.: lakcím): [redacted]

Szakdolgozat címe magyarul:

KÖZOKTATÁSI KÖNYVIGÉNYLÉST VÉGZŐ ADMINISZTRÁCIÓS WEBALKALMAZÁS ASP.NET ALAPOKON

Szakdolgozat címe angolul:

ADMINISTRATIVE WEB APPLICATION FOR BOOK DISTRIBUTION IN PUBLIC EDUCATION WITH ASP.NET

Intézményi konzulens: SIPOS MIKLÓS LÁSZLÓ

Külső konzulens: -

Alk.	Dátum	Tartalom	Aláírás
1.	2022.03.04.	Fejlesztés tervezett menetének revíziója, szakmai egyeztetés, fejlesztési teendők megkezdése.	
2.	2022.04.01.	Fejlesztés állapotának és haladásának áttekintése. Technikai problémák és módosítások egyeztetése.	
3.	2022.04.15.	Felmerülő problémák egyeztetése és megoldása. Tesztelési fázis megindítása.	
4.	2022.05.06.	Dokumentáció véglegesítése. Leadással kapcsolatos teendők egyeztetése.	

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

A hallgató a Szakdolgozat II. tantárgy követelményét teljesítette, védésre bocsátható.

.....

Intézményi konzulens

Budapest, 2022.05.10.

Absztrakt

A közoktatásban egy adminisztrációs problémákkal és időigényes megvalósítással teli feladat az adott évre történő tankönyvek kiosztása minden évfolyam és diák számára.

Ezt rendszerint tanárok, iskola titkárok, könyvtárosok vagy akár diákok végzik valamilyen táblakezelő szoftver segítségével vagy akár papír alapon, írott formában. Nekik egyesével kell végig menniük a rendelési listán, hogy melyik diák milyen könyveket igényel.

Ezekkel a fájlokkal kifejezetten az a probléma, hogy különböző strukturált felépítést alkalmaznak, Sokszor különböző számítógépen a történik az eltárolásuk, valamint így elvesznek. Nem mindig lehet megfelelően visszakövetni egy adott év rendelését, mivel nem egy centralizált módon történik ezen adatok tárolása.

Több döntő tényező van, például az, hogy egy adott diák angolt vagy németet tanul. Előfordulhat, hogy egy adott tanár különböző könyvkiadó könyveit preferálja, bonyolítva evvel a rendelést. Utána kell nézniük mi az, ami egyáltalán az igényelhető listán szerepel, vagy külön bonyolult módon be kell rendelniük, esetleg több listát kell összenézni.

A célom egy olyan web alkalmazás fejlesztése, ami képes ezt a feladatott könnyebben, esetleg pár folyamatot automatikus módon megvalósítani. Az alkalmazásban évfolyam, osztály és név szerint lehet szűrni arra, hogy ki milyen tankönyveket igényelt az adott évben.

A szoftver az évfolyamot sikeresen teljesítő diákokat automatikusan lépteti a következő osztályra év végén. A speciális eseteket, mint például: egy diák elmegy az iskolából, osztályt vált vagy nem sikerül teljesíteni az adott évet, manuálisan kell kezelni a tanárának, a titkárságnak vagy az adminisztrátornak, aki ezt a feladatott ellátja.

A dolgozatban kitérek a különböző fejlesztői rendszerek kutatására és összehasonlítom a különböző szerver oldali keretrendszereket és jelenlegi piaci szereplőket.

A fejlesztés során elkészítettem egy olyan modern felhasználói felülettel rendelkező, könnyen áttekinthető webes platformot. Mely könnyen alkalmazható egy rendszergazda által üzemeltettet is iskolai magánhálózaton. Az alkalmazás bejelentkezéshez kötött, ezért csak megfelelő hitelesítéssel rendelkező felhasználók érhetik el.

Véleményem szerint ezzel a központosítót rendszerrel ellehet végezni az általam felvetett problémát.

Abstract

In public education, a task full of administrative problems and time-consuming implementation is to distribute textbooks for that year to all grades and students.

This is usually done by teachers, school secretaries, librarians or even students using a spreadsheet software or even on a written basis. They need to go through the order list one by one to see which students need which books.

The problem with these files in particular is that they have a different structure, are often stored on different computers, and thus get lost. It is not always possible to adequately trace the order of a given year, as there is no centralised way of storing this data.

There are several decisive factors, such as whether a particular student may be learning English or German, which publisher's books do individual teachers prefer, and which ones are at all on the list of available books or should be ordered separately.

My goal is to develop a web application that can accomplish this task more easily, you can automatically complete a few processes. In the app, you can filter by year, class and name to see who has requested which books.

Preference can be defined in advance for a given student, such as what language the student is learning at which level at which teacher.

The software automatically promotes students who successfully complete the grade to the next grade at the end of the year. Special cases, such as: a student leaving school, changing class, or failing to complete a particular year, must be handled manually by their teacher, and the secretary must approve the change.

In the dissertation, I cover the research of different development systems and compare different server-side frameworks and market players.

In the course of the development, I created an easy-to-use web platform with a modern user interface. It can be easily deployed on a private school network hosted by an administrator. The application requires a login and can only be accessed by users with appropriate authentication.

In my opinion, this centralisation system can solve the problem I have raised.

Tartalomjegyzék

1. Bevezetés

Általános iskolákban probléma a tankönyvek rendelésének kezelése. Sokszor mai napig egy táblázatkezelő formátumokban vagy akár papír alapon vezetik azt, hogy melyik diák milyen tankönyvet kér. Több döntő indok van például az, hogy egy adott tanuló esetleg angolt vagy németet tanul, az, hogy egy tanár melyik könyvkiadó könyveit preferálja. Ezen a téren is van változás, sokszor most már a legtöbb könyv használatát előre központilag meghatározzák.

Cél egy olyan központi adatbázissal rendelkező alkalmazást készíteni, melyben ezeket a könyvrendelési problémákat egy helyen lehet vezérelni a tanároknak, tanítóknak, titkárságnak vagy esetleg az iskolai könyvtárasoknak. Az alkalmazásomban évfolyam, osztály szerint lehet szűrni arra, hogy ki milyen könyveket igényelt.

Adott diákokhoz előre lehet definiálni preferenciát, mint például azt, hogy az adott diák milyen nyelvet milyen szinten tanul melyik tanárnál. Esetleg hátrányos helyzetű diák könyvtári könyveket kap és így mentesül a könyvek költségei alól.

Az adott feladatban a célom az, hogy egy olyan webalkalmazást alkossak, ami képes ezt az adminisztratív problémákat leegyszerűsíteni. Ebben a rendszerben a tanár feladata a tanév végét követően meghatározni azt, hogy melyik diák léphet tovább a következő évre, ezt követően pedig a szükséges megadni a diák által igényelt különböző könyveket.

Az adott diák összesített könyvrendelési igényeit a rendszer továbbítja a diák felé, így a szülei/gondviselője felügyeletével lehetőséget kap ellenőrizni, hogy az oktatója megfelelően alakította a ki a következő évre tervezett könyveit.

A rendszer a felvett diákokat automatikusan lépteti a következő osztályra év végén. A speciális eseteket, mint például egy diák elmegy az iskolából, osztályt vált vagy nem sikerül teljesíteni az adott évet manuálisan kell kezelni a tanárának.

A fő szempontja a rendszeremnek, hogy lehető legtöbb feladat automatizáltan történjen és ha bármilyen manuális feladat adódna akkor arról az illetékest értesítse. Amennyiben a felhasználó a webes felületen van bejelentkezve az értesítés a webes felületen keresztül történik.

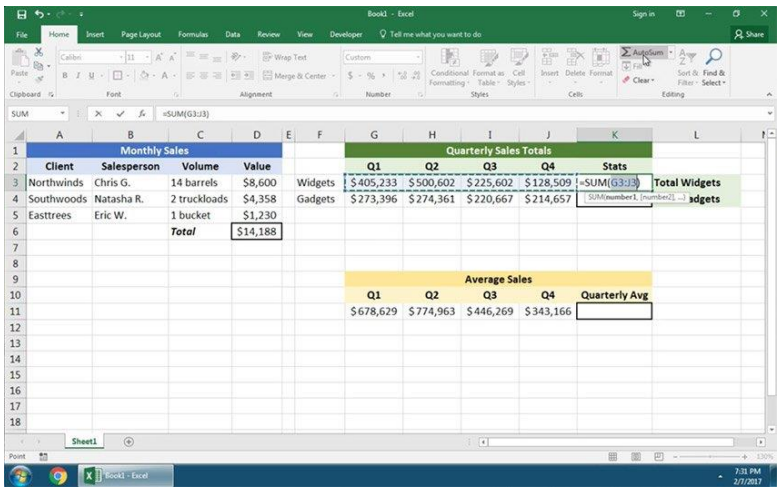
2. Irodalomkutatás

2.1 Hasonló rendszerek

Ennek a dolgozatnak a fő szempontja az, hogy egy olyan adminisztrációs rendszert hozzak létre, amit eddig egy táblázatkezelő szoftverben végeztek el, valamilyen minta alapján. Így én ebben a feladatban három különböző rendszert látok, amiknek az elemeit szeretném a szoftverembe átvenni. Ez egy előbb említett táblázatkezelő alkalmazás, mint a Microsoft Office Excel, egy iskolákban használatos a tanároknak és titkároknak is egyszerű felhasználói felülettel (User Interface-el) megvalósított naplózó alkalmazás, mint például a Kréta. Valamint kitérek arra, amit én a web alkalmazásom fő részének látok, hogy az általam készített program, valójában egy disztribúciós leltáralkalmazás. Végül elemzem a jelenleg működő könyvellátónak a weboldalát.

2.1.1 Excel

A Microsoft Excel a Microsoft által fejlesztett táblázatkezelő program. (1.ábra mutatja.) Az Excel segítségével munkafüzeteknek nevezett számolótáblák hozhatók létre, amelyeken a felhasználó formázott adatokat helyezhet el. Az adatok alapján végezhetőek számítások, készíthető rájuk alapozva számos típusú diagram, illetve kimutatás. Az Excel támogatja képességei kiterjesztését a beépített programozási interfész segítségével, melyben VBA nyelven írhatóak makrók, illetve készíthetőek akár felhasználói felülettel is rendelkező kisebb-nagyobb funkcionalitással bíró modulok.



1.ábra
Excel tábla példa

A táblázatkezelő egy olyan számítógépes program, amellyel egy táblázatban tárolt adatokon műveletek végezhetők. A táblázat sorokból és oszlopokból áll, egy sor és egy oszlop metszete egy cellát határoz meg. A cellában érték vagy kifejezés állhat, amelynek az értéke más cellák értékeitől és/vagy külső értékektől (dátum stb.) függ.

A Microsoft Excel alapvető dokumentuma a munkafüzet. A munkafüzetek munkalapokra tagolódnak, ezeket pedig számokkal jelölt sorok és betűkkel vagy betűkapcsolatokkal jelölt oszlopok tagolják. A sorok és oszlopok metszéspontjaiban elhelyezkedő cellák szolgálnak az adatbevitelre. Az adatokat tároló cellák sor- és oszlopcímükkel megcímezhetőek, és így értékük más cellákban végzett számításokban operandusként, illetve függvény paramétereként vehet részt. Az Excel beépített függvénytárral rendelkezik, amelyben az általában elvárható statisztikai, szöveg, matematikai és egyéb műveletek kapnak helyet, illetve a beépített programozási felület lehetővé teszi saját függvények készítését is.

2.1.2 Kréta

A Kréta elektronikus napló a legkorszerűbb informatikai technológiákat ötvöző elektronikus iskolai osztálynapló, amelyet a köznevelésben és a szakképzésben is hatékonyan lehet használni, az általános iskoláktól kezdve egészen a különböző szakképző intézményekig. Az e-napló a szaktanároknak, pedagógusoknak, osztályfőnököknek segít a naplóvezetéshez szükséges adminisztrációs feladatok gyors és hatékony végrehajtásához. (Ahogy a 2.ábra mutatja)

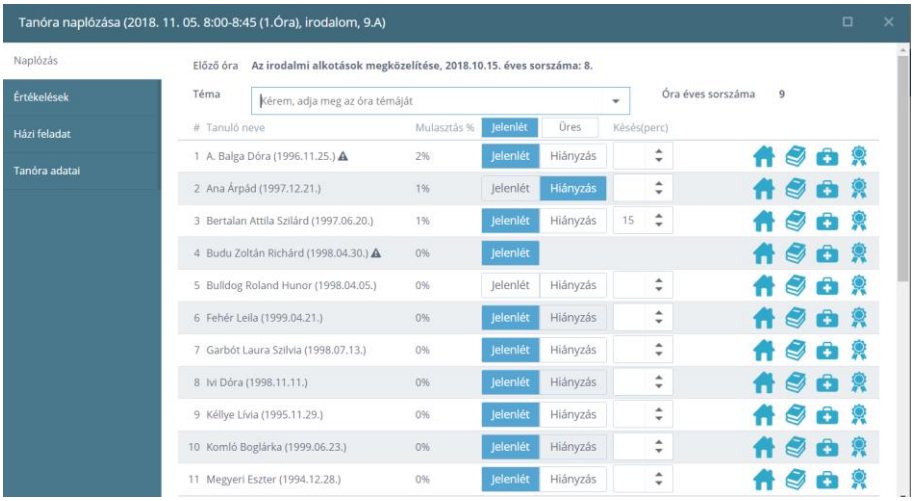
A Kréta szoftver nagy teljesítményű dokumentumgeneráló funkcióval rendelkezik, amely lehetőséget ad a különböző iskolai nyomtatványok, papír alapú dokumentumok előállítására.

A rendszer felhasználó-kezelése biztosítja, hogy az arra jogosultak a saját szerepköreikben (szaktanár, osztályfőnök, igazgató) használható funkciókat érjék el.

Az e-Napló megalkotása elsősorban az oktatási intézmények vezetőinek, tanárainak igényeit tükrözi az alábbi alapvető elvek szerint:

- bonyolult menüstruktúrától mentes, könnyen használható kezelőfelület a hatékonyság érdekében;
- paraméterezett, testre szabható funkciók, hogy minden intézmény csak a szükségleteinek megfelelő adatokat kezeljen;
- igazodik az adminisztrációs rendszer adattartalmának feltöltöttségéhez, nem szükséges minden adat megadása a használathoz;
- hatékony integrálási lehetőségek különböző iskolai szoftverrendszerekhez,
- egyszerű, letisztult felhasználói felület,
- egyesíti a hagyományos papír alapú és az elektronikus osztálynaplók előnyeit,
- használatával a papír alapú napló kiváltható,

- az intézmény infrastruktúrájának függvényében bárhonnan elérhető a szoftver egy böngészőprogram használatával,
- tetszőleges infokommunikációs eszközön használható. [1]



2.ábra
Kreta Felület

2.1.3 Inventoria Inventory Software

Inventoria a NHC Software által kifejlesztett, egy részben ingyenes leltár és raktár alkalmazás, ami segítséget nyújt a kis és közép szintű raktárvállalkozásoknak a mindennapi menedzselésben és raktáruk készleteinek feltöltésében. (Ahogy a 3. ábra mutatja.)

Rendelések kezelése:

- Létre tudd hozni rendeléseket és közvetlenül tudd e-mail-t írni a szállítóknak
- Adatbázist tart fenn vevőkről és beszállítókról
- Ha alacsony a raktáron lévő árú akkor küld értesítést az újra rendelés szükségességéről
- Frissíti a tárgyak darabszámát amikor egy rendelés történik

Leltári jelentések:

- Figyeli a leltár állapotát, költségeit és átlagait
- Raktáradat külön tudod kategorizálni fajta és helyszín között
- Eladás esetén elmenti a vásárlási történetet

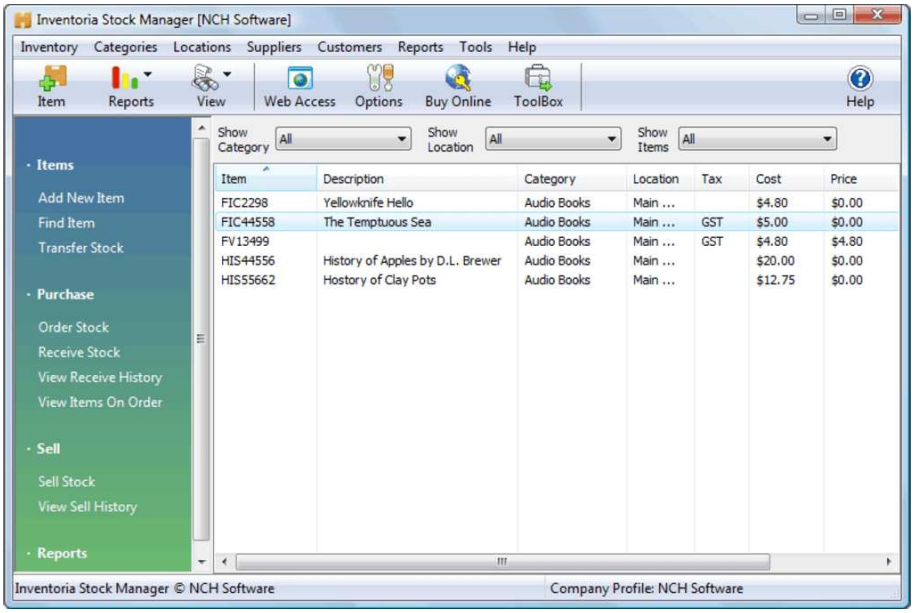
Tárgyak kezelése:

- Lehetséges teljes leltár fájlból való feltöltésére
- Barcode alapján is lehet tárgyat hozzáadni
- A tárgyak leírásába lehet URL-t vagy képeket csatolni

- Összeköthető más üzleti programokkal, hogy több vállalkozásnak a leltári adatait lehessen kezelni

Felhasználók és helyszínek kezelése:

- Képes az árukat egy-egy lokációk között küldeni
- Különböző hozzáférési szinteket lehet megszabni a felhasználóknak



3. ábra
Inventoria Inventory Software kezdőoldal

2.1.4 Kello (Könyvtárellátó Nonprofit Kft.)

A Könyvtárellátó Nonprofit Kft. (Kello) ezen a néven 2007 óta tevékenykedik. Önálló állami vállalatként azonban már 1991 óta működik, 1994-ben alakult át közhasznú társasággá. Állami cégként az Emberi Erőforrások Minisztériuma Köznevelésért Felelős Államtitkárságának szakmai felügyelete alatt áll. A 2013/2014-es tanévtől bevezetett állami tankönyvforgalmazás új rendszere szerint az állam az országos tankönyvellátást a Könyvtárellátón keresztül biztosítja. Alapító okirata szerinti alapvető célkitűzése: oktatási és kulturális tevékenység végzése. Alapítása óta a Könyvtárellátó rendszeresen részt vállal a kulturális és oktatási tárca, a Nemzeti Kulturális Alap, illetve egyéb szervezetek magyar könyvtárakat és iskolákat, közgyűjteményi és közművelődési dolgozókat, határon túli magyar könyvtárakat és pedagógusokat támogató közhasznú és közcélú programjainak lebonyolításában.

A Kello feladata a tankönyvek országos megrendelése, beszerzése és az iskolák számára történő eljuttatásának megszervezése. A tankönyvellátási rendszer hatékony működése érdekében a Kello elektronikus információs rendszert alkalmaz. A társaság 2013-2014-es években megújította arculatát, majd a következő években bővítette webes szolgáltatásait is.

A közhasznúság jegyében – hozzáadott értékként - szerkeszti, kiadja és havonta két alkalommal a könyvtáraknak megküldi az Új Könyvek című állománygyarapítási tájékoztatót, amely teljességre törekedve ismerteti a magyar nyelven megjelent könyvújdonságokat. E kiadvány 2014 októberében ünnepelte megjelenésének 50. évfordulóját.

Pedagógusok oktatási - nevelési munkájának támogatására működteti a Modern Iskola magazint, amelyben aktuális szakmai hírek és információk mellett a szülők számára is nagyon hasznos és érdekes cikkeket közöl.

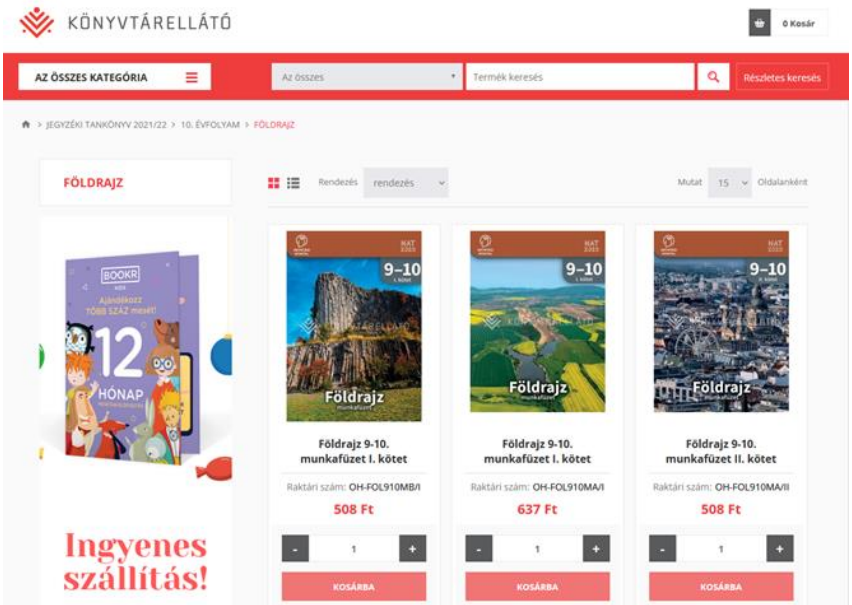
Az olvasáskultúra támogatásának céljából az interneten elérhető Online Könyvkultúra Magazint tart fenn, amelynek révén tájékoztatja partnereit és a széles nyilvánosságot a hazai és nemzetközi könyves közélet eseményeiről, újdonságairól, érdekességeiről. A magazinban több mint húszezer cikk található, köztük az Új Könyvek szerkesztősége által írt recenziók. Az internetes magazin sikere kapcsán 2015 decemberétől, negyedévente már nyomtatott formában is megjelenik a Könyvkultúra Magazin.

A nyomtatott változatot az Új Könyvek soron következő számával együtt, ingyenes juttatja el valamennyi hazai könyvtárba.

A társaság körülbelül 600 könyvkiadóval és több száz könyvkiadással is foglalkozó egyéb szakmai szervezettel áll kapcsolatban.

Évente több ezer újonnan megjelenő könyvet, tankönyvet és tanítási segédletet, s több mint 600 magyar folyóiratot forgalmaz.

Webáruházat működtet, ahonnan a beérkező új könyvek azonnal megrendelhetők, továbbá a tankönyvek széles kínálata is megtalálható a felületen az általános iskolától a felsőoktatásig. (4. ábra is jellemzi) [2]



4. ábra
Könyvtárellátó webáruháza

A Kello lehetőséget nyújt iskoláknak az éves vagy évközi tankönyvrendelések lebonyolításában. Egy előre meghatározott rendelési időszakban lehet a Kello által előre kiadott felhasználónévvel vagy jelszóval rendeléseket leadni diákok számára. Amennyiben nem rendelkezünk regisztrált fiokkal van mód a rendkívül tartalmas webárúházban a nekünk kellő könyvet megvásárolni.

Kello legfontosabb feladata a különböző könyvtárak állománygyarapítása és ehhez segít az erre kialakított szakmai honlap, amit szintén csak elfogadott regisztrációval lehet elérni.

A szakmai honlapon teljességre törekvően tájékoztatást adnak a magyar nyelvű könyvekről, és válogatva egyéb kiadványokról. A könyvtárak által megrendelt kiadványokat a kiadóktól beszerzik, elkészítik a szabványos bibliográfiai leírást és a könyvtárak igényei szerint könyvtári szereléssel szállítják. A mindennapi munka megkönnyítése érdekében könyvtári felszereléseket és nyomtatványokat is forgalmaznak. Mindezeken túl megrendelőlapon több mint 600 folyóiratból is válogathatnak Partnereik.

2.1.5 Hasonló rendszerek összegzése

	Excel	**Kréta**	**Inventoria Inventory Software**	**Kello**
Rendszer	Asztali alkalmazás, Webes felületen is elérhető	PHP-ban fejlesztett Webes adminisztrációs felület	Windows Formban megirt asztali leltár alkalmazás	Blazor-ban fejlesztett Webes áruház felület
Költsége	Diákoknak ingyenes, egyébként fizetős	Ingyenes, de csak meghatározott célközönségnek elérhető	Ingyenes próba üzemód, de fizetés ellenében rendelkezik plusz szolgáltatásokkal	Ingyenes, de bizonyos csak külön hozzáféréssel engedélyezett
Pozitív	• Microsoft által támogatott rendszer • Rengeteg előre legenerált sablon áll rendelkezésünkre	• A célközönség számára jól átlátható • Felváltja a régi papíros napló megoldást	• Bizonyos ideig ingyenesen elérhető • Legtöbb leltár szoftver közül a legolcsóbb	• Webáruháza modern és letisztult • Rendelkezik külön könyvtáros felülettel
Negatív	• Inkonzisztensek a táblák • Sokszor nincs rendesen egységesítve	• Elavult technologiában íródott	• Régi átláthatatlan dizájn • Rengetek felesleges funkció	• Tanároknak a könyvrendeléseit manuálisan kezeli

1.tábla
Hasonló rendszerek tábla

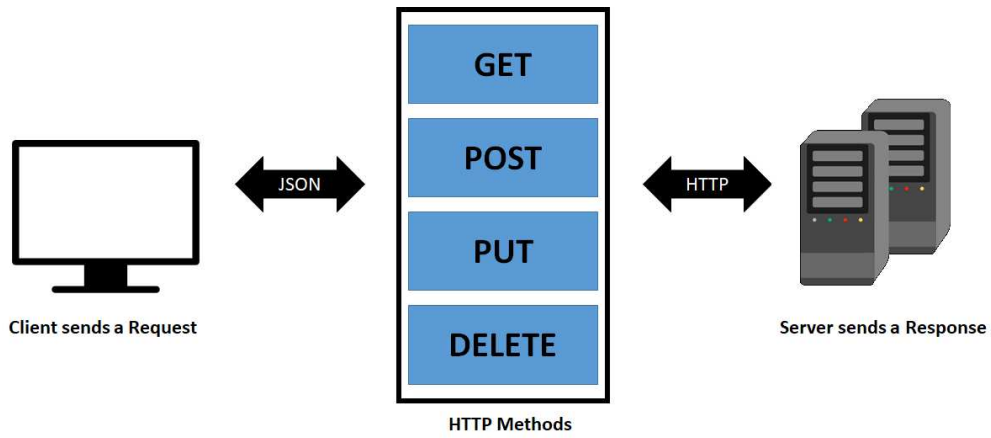
2.2 Application Programming Interface

API egy rövidítése az Application Programming Interface-nek, ez egy olyan szoftveres csatorna, ami összeköt kettő applikációt, hogy egymással tudjanak kommunikálni. Például amikor egy alkalmazást használunk a mobiltelefonunkon, az alkalmazás csatlakozik az internethez, és adatokat küld a szervernek. A szerver először lekéri az adatokat, értelmezi, elvégzi a szükséges műveleteket és visszaküldi a telefonra. Az alkalmazás ezután értelmezi az adatokat, és olvasható módon megjeleníti a kívánt információkat. A telefon adatai sohasem kerülnek ki teljesen a szerverre, és a szerver információi sem kerülnek át a telefonra. Ehelyett mindegyik kis adatcsomagokkal kommunikál, és csak azt osztja meg, ami szükséges.

2.2.1 REST

A projectemhez, hogy fennálljon a kliens és szerver közötti kapcsolat elengedhetetlen, hogy API-knak a használatát és különböző kapcsolati protokollokat ismertessek

A REST architektúrális megszabásoknak halmaza, nem pedig protokoll vagy szabvány. Amikor a kliens oldal kérést küld a RESTful API-n keresztül, a forrás állapotának reprezentációját továbbítja a kérelmezőnek vagy a végpontnak. Ezt az információt vagy ábrázolást a JSON fájlon keresztül többféle formátum egyikében szállítják. A JSON a legáltalánosabban használt fájlformátum, mivel a neve ellenére nyelv-agnosztikus, valamint ember és gép által is olvasható. (Ahogy az 5. ábra jellemzi.)



5. ábra
REST API működése

A fejlécek és a paraméterek szintén fontosak a HTTP kérések szempontjából, mivel fontos azonosító információkat tartalmaznak a kérelem metaadatairól, jogosultságáról, egységes erőforrás-azonosítójáról (URI), cookie-król stb. Vannak kérésfejlécek és válaszfejlécek, amelyek mindegyike saját HTTP-kapcsolati információval és státusz kódokkal rendelkezik.

Azért, hogy az API RESTful legyen megkell felelnie több kritériumnak. Az egyik ilyen a kliens és szerver oldali architektúrának HTTP kéréseken keresztül kell történnie. A kliens nem tárolhat felhasználó adatokat. Cacheable adatok, amelyek leegyszerűsítik az kliens szerver interakciókat. Egységes interfész a komponensek között, így az információ szabványos formában kerül továbbításra és az a képesség, hogy kérésre futtatható kódot küldjön a szerverről a kliensnek, kibővítve a kliens funkcionalitását.

REST API egy gyorsabb és könnyebb fajtája a kliens-szerver oldali kommunikációnak, mint az ezt megelőző SOAP technológia, ami XML fájlokat használ és minden üzenet küldéshez tartozik egy transaction compliance, ami lelassítja ezt a folyamatot.

2.2.2 SOAP

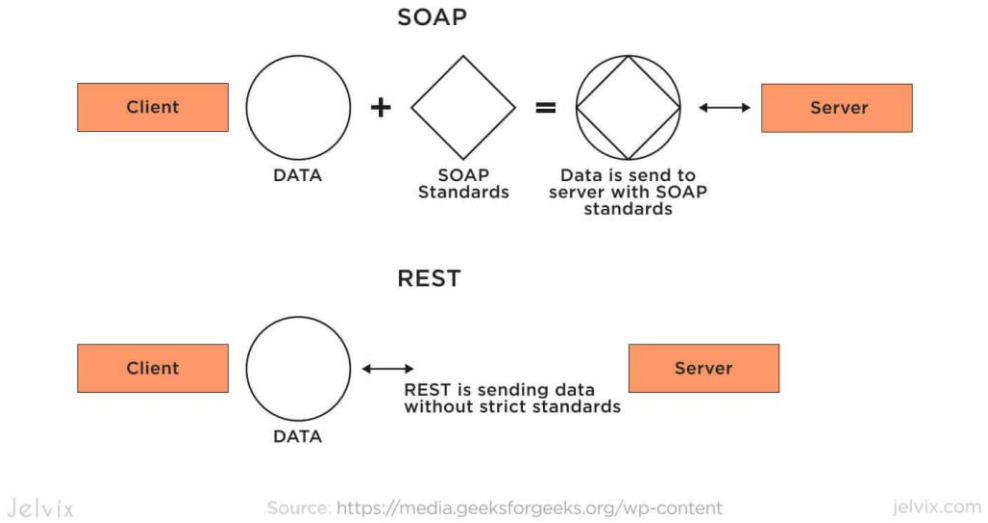
A SOAP (korábbi nevén Egyszerű objektumhozzáférési protokoll) egy egyszerű adattovábbító protokoll decentralizált, osztott környezetben való használatra. Egy SOAP üzenet egy küldő és egy fogadó közti adatátvitel. Több SOAP üzenet használatával kérés-válasz rendszerek is létrehozhatók. (6.ábra jellemzi)

A SOAP független a szállító rétegtől, de ezt a szerepet leggyakrabban a HTTP tölti be, hogy a már meglévő internetes infrastruktúrán futhasson. A SOAP lehetővé teszi a kikeresett webszolgáltatások használatát azáltal, hogy az útvonalkezelés számára üzenetútvonalat biztosít. A SOAP protokollt használják a webszolgáltatások UDDI lekérdezéseinél. A munkaterület a SOAP 1.1 verzióját támogatja.

A SOAP XML-alapú protokoll, ami minden üzenethez három részt határoz meg:

- **Boríték.** A boríték meghatározás egy keretrendszert az üzenet tartalmának és feldolgozási módjának a leírásához. A SOAP üzenetek egy borítékból állnak, ami bármennyi fejlécet tartalmazhat (vagy akár egyet sem) és pontosan egy törzset. A boríték az XML dokumentum legfelső eleme, ami tárolót biztosít a vezérlőinformáció, a címzés és maga az üzenet számára. A fejlécek tartalmazzák az összes vezérlő információt, például a szolgáltatási minőség jellemzőket. A törzs tartalmazza az üzenetazonosítást és annak a paramétereit. A fejléc és a törzs is a boríték leszármazott elemei.
- **Kódolási szabályok.** A kódolási szabályok készlete írja le az alkalmazások által definiált adattípusok példányait. A kódolási szabályok határozzák meg a sorosítási mechanizmusokat, amik az alkalmazások által definiált adattípusok cseréjéhez használható. A SOAP egyrészt egy XSD-alapú, programozási nyelvtől független adattípus sémát határoz meg, másrészt pedig kódolási szabályokat minden, e modell szerint meghatározott adattípushoz. A SOAP kódolás nem felel meg a WS-I szabványnak, így a webszolgáltatásoknál a literál (vagyis kódolás nélküli) használata javasolt, a WS-I szabványnak való megfeleléskor pedig ez kötelező is.

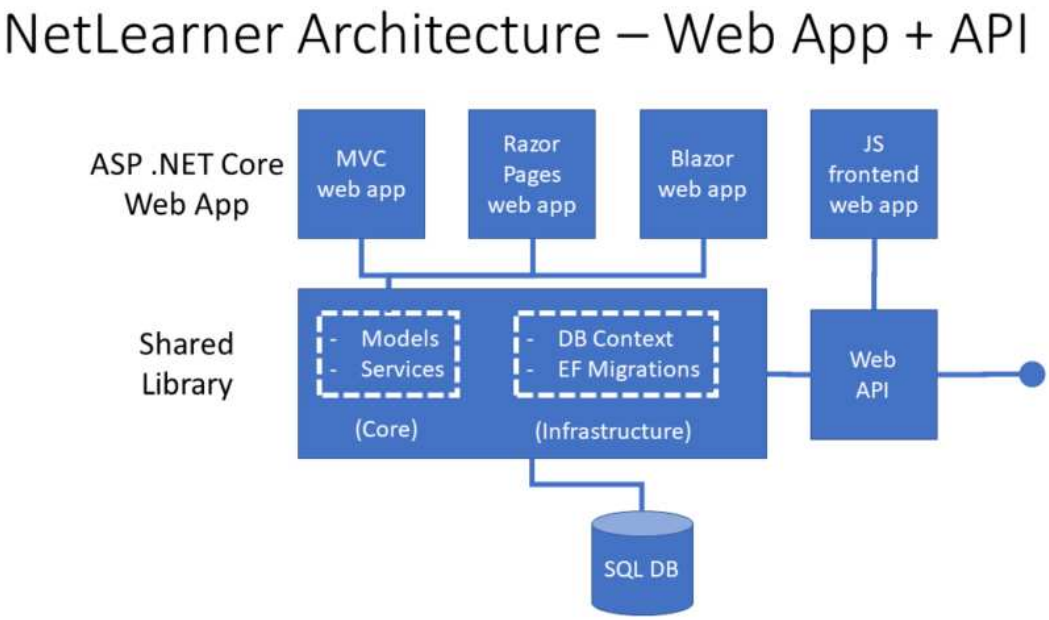
- **Kommunikációstílusok.** A kommunikáció lehet távoli eljáráshívás (RPC) vagy dokumentum-központú (Dokumentum) formátumú. A következőkben ezekről olvashat. [3]



6.ábra
SOAP és REST összehasonlítása

2.3 ASP.NET Core

Az ASP.NET Core egy cross-platformos, nagy teljesítményű, nyílt forráskódú keretrendszer modern, felhőalapú, internetkapcsolattal rendelkező alkalmazások készítéséhez. Tartalmazza az MVC keretrendszert, amely immár egyetlen webes programozási keretrendszerben egyesíti az MVC és a Web API szolgáltatásait. (Ahogy a 7.ábra jellemzi)



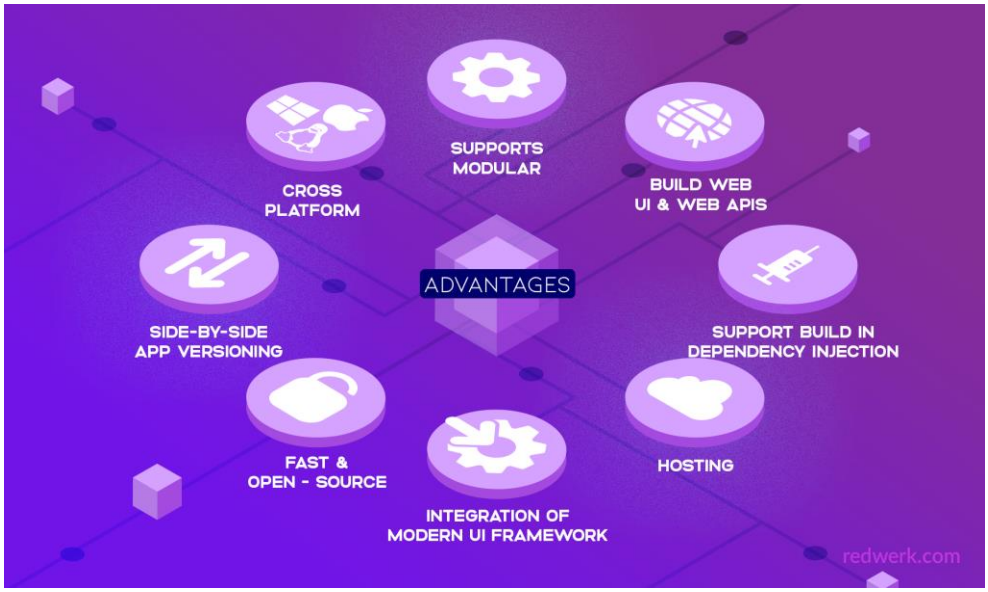
7.ábra
ASP.NET architektúrájának felépítése

2.3.1 ASP.NET Core előnyei

Az ASP.NET Framework-höz képest rengeteg új architektúrális különbsége van, mint például, hogy sokkal modulárisabb és már nem a System.Web.dll-en alapul, hanem jól konstruált NuGet csomagokon. Ezeknek a segítségével sokkal könnyebb optimalizálni a projectet.

További fejlődések a korábbi keretrendszerhez képest:

- Cross-platform alkalmazások építése és futtatása Windows-on, Mac-en és Linux-on.
- támogatja az alkalmazások valódi egymás melletti verziónálását.
- Új eszközök, amik leegyszerűsíti a modern webfejlesztést.
- Egyszerre igazított verem webes felhasználói felülethez és API-khoz.
- Felhőképes környezetalapú konfiguráció.
- Lehetőség az IIS-en vagy a saját folyamatban való tárolásra. (Ahogy 8. ábra jellemzi)



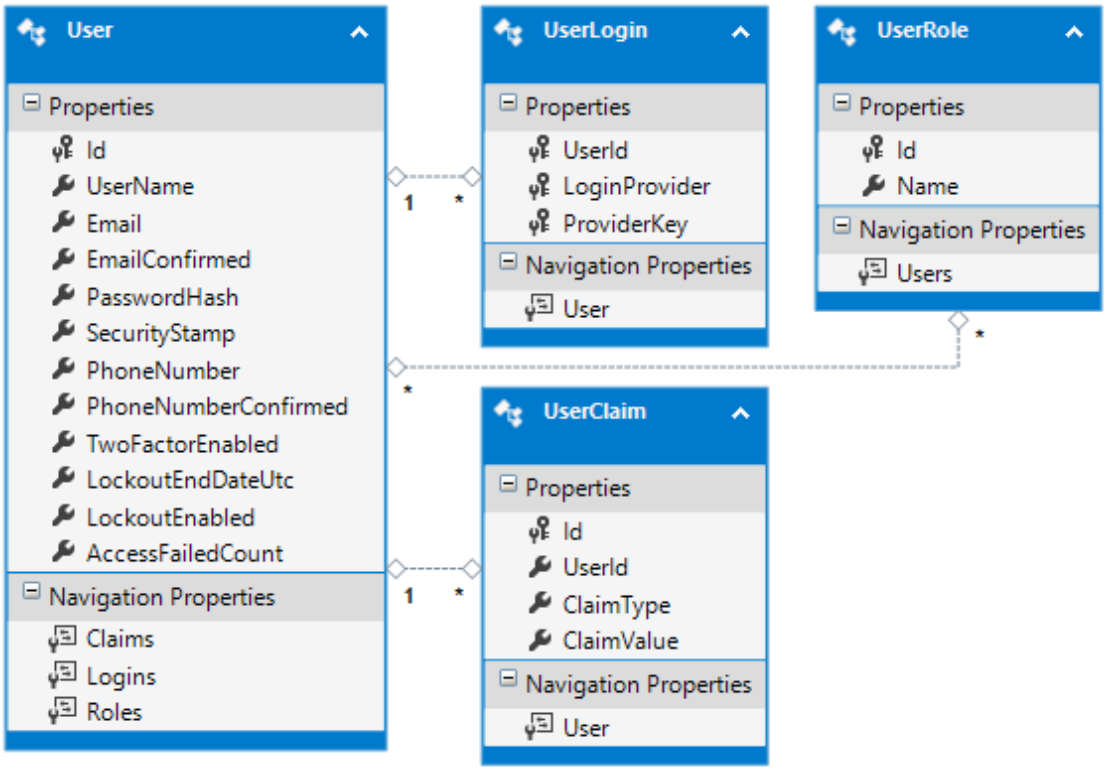
8. ábra
ASP.NET Core előnyei

2.3.2 ASP.NET Core Identity

Az IdentityServert a rugalmasságra tervezték, és ennek egy része lehetővé teszi, hogy bármilyen adatbázist használhasson az azon tárolni szánt felhasználóik és adataik számára. Amennyiben új felhasználói adatbázist szeretnénk létrehozni, akkor az ASP.NET Core Identity az egyik legjobb választási lehetőség.

Az ASP.NET Core Identity egy tagsági rendszer (membership system), amely lehetővé teszi, hogy bejelentkezési funkciókat adjon az alkalmazáshoz. A felhasználók

létrehozhatnak egy fiókot, valamint bejelentkezhetnek felhasználói névvel és jelszóval, vagy használhatnak külső bejelentkezési szolgáltatókat, például Facebook, Google, Microsoft-fiók, Twitter és egyebeket bejelentkezési módokat. (Ahogy 9. ábra is mutatja.)



9. ábra
Identity Tábla felépítése

Az ASP.NET Core Identity konfigurálható úgy, hogy SQL Server adatbázist használjon a felhasználónevek, jelszavak és profiladatok tárolására. Alternatív megoldásként használhatja saját állandó tárolóját az adatok egy másik állandó tárolóban, például az Azure Table Storage-ban való tárolására.

2.3.3 Entity Framework Core

Az Entity Framework (EF) Core a népszerű Entity Framework adatelérési technológia könnyű, bővíthető, nyílt forráskódú és platformok közötti legnépszerűbb változata.

Az EF Core segítségével az adatokhoz való hozzáférés egy modell segítségével történik. A modell entitásosztályokból és egy kontextusobjektumból áll, amely egy munkamenetet képvisel az adatbázissal. A kontextus objektum lehetővé teszi az adatok lekérdezését és mentését.

Az EF Core egy modern objektum-relációs leképző .NET-hez. Támogatja a LINQ-lekérdezéseket, a változáskövetést, a frissítéseket és a sémaáttelepítést. Az EF Core egy szolgáltatói beépülő API-n keresztül működik együtt az SQL Serverrel, az Azure SQL

Database-szel, az SQLite-tal, az Azure Cosmos DB-vel, a MySQL-lel, a PostgreSQL-lel és más adatbázisokkal.

Entity Framework Objektum-relációs leképzés:

- Lehetővé teszi a .NET-fejlesztők számára, hogy .NET-objektumokat használó adatbázisokkal dolgozzanak.
- Kiküszöböli a legtöbb SQL kódot, amelyet általában meg kell írni.
- Az alapul szolgáló adatbázist kiszolgáló középszintű vagy magasabb szintű ismerete elengedhetetlen a nagy teljesítményű éles alkalmazásokban az adatok tervezéséhez, hibakereséséhez, profiljához és migrálásához. Például az elsődleges és idegen kulcsok, megkötések, indexek, normalizálás, DML és DDL utasítások, adattípusok, profilalkotás ismerete.
- Egyes funkciók natív használata nem skálázható jól. Például több gyűjtemény tartalmazza a lusta betöltés (Lazy-Loading) lehetséges használatát, a nem indexelt oszlopokra vonatkozó feltételes lekérdezéseket, az Storage által generált értékekkel rendelkező tömeges frissítéseket és beillesztéseket, az egyidejűség kezelésének hiányát, a nem megfelelő gyorsítótár sorrendjét.
- Biztonsági áttekintés, például kapcsolati karakterláncok és egyéb nem publikus adatok kezelése, hogy adatbázishoz szükséges engedélyek le legyenek tiltva nem telepítési műveletekhez és nem szerverről érkező SQL bemenetek ellenőrzése, valamint érzékeny adatok titkosítása
- Alkalmazások telepítése és áttelepítése. Tervezze meg, hogy az áttelepítéseket miként alkalmazzák a telepítés során. Az alkalmazás indításakor történő végrehajtása párhuzamossági problémákkal járhat, és a normál működéshez szükségesnél magasabb engedélyeket igényel. Az átállás során fellépő végzetes hibákból való felépülés megkönnyítése érdekében szakaszolást szükséges használni. [4]

2.3.4 EF 6 és EF Core 6 összehasonlítása

Az Entity Framework 6 (EF6) egy objektum-relációs leképző, amelyet .NET-keretrendszerhez terveztek, de támogatja a .NET Core-t. Az EF6 egy stabil, támogatott termék, de már nem fejlesztik aktívan.

Az EF Core olyan új funkciókat kínál, amelyek nincsenek implementálva az EF6-ban. Jelenleg azonban nem minden EF6 funkció van megvalósítva az EF Core-ban.

Ezek a szolgáltatások vagy az EF6 mögöttes Entitásadat-modelltől függenek, vagy összetett jellemzők, amelyek viszonylag alacsony megtérüléssel rendelkeznek. EF Core sok olyan dolgot tesz lehetővé, ami az EF6-ban nem lehetséges, fordítva viszont az EF Core nem támogatja az EF6 összes funkcióját.

2.4 Angular

Az Angular egy TypeScript-re épülő fejlesztői és nyílt forrású webalkalmazás platform, amit ugyan az a régi Angular Google csapat készített, mint az AngularJS-t. Angular alapvetően egy kliens oldali rendszerként használjuk, ami egy Node.js runtime environment. Platformként az Angular a következőket tartalmazza:

- Komponens alapú keretrendszer méretezhető webalkalmazások készítéséhez
- Jól integrált könyvtárak gyűjteménye, amelyek sokféle funkciót lefednek, beleértve az átirányítási vezérlést, az űrlapkezelést, az ügyfél-szerver kommunikációt és még sok mást.
- Fejlesztői eszközök készlete a kód fejlesztéséhez, összeállításához, teszteléséhez és frissítéséhez (10.ábra jellemzi)

Az Angular-t úgy tervezték, hogy a frissítés a lehető legegyszerűbb legyen, ezért minimális erőfeszítéssel használja ki a legújabb fejlesztéseket.

10.ábra
Angular infrastruktúra

Egyik legfőbb különbsége az Angular-nak más kliens oldali framework-kel szembe, hogy alapból TypeScript-re épül. A TypeScript egy erősen típusos programozási nyelv, amely JavaScript-re épül, így bármilyen léptékben jobb eszközöket biztosít. (Ahogy a 11. ábra jellemzi)

11. ábra
TypeScript előnyei

A TypeScript további szintaxist ad a JavaScripthez, hogy támogassa a fejlesztővel való szorosabb integrációt. Fogja meg a hibákat korán a szerkesztésben. A TypeScript kód JavaScriptté konvertálóik, amely bárhol fut, ahol a JavaScript fut: böngészőben, Node.js fájlban és az alkalmazásokban. Megérti a JavaScriptet, és típuskövetkeztetést használ, hogy további kód nélkül kiegészítő eszközöket biztosítson. [5]

2.4.1 Angular felépítése

Az komponensek az Angular alkalmazások fő építőelemei. Amik a következők:

- Egy HTML osztály, amely deklarálja, hogy mi jelenik meg az oldalon
- Typescript osztály, amely meghatározza a funkcionalitást
- CSS osztály, amely meghatározza, hogy az összetevő hogyan használható a sablonban

A legjobb módja, hogy ilyen komponenseket csináljunk, hogy az Angular CLI -t használjuk. Aminek a parancsa a ng generate component <component-name>.

Az Angular alkatrészmodellje erős beágyazást és intuitív alkalmazási struktúrát kínál. Az összetevők emellett egyszerűvé teszik az Unit teszt alkalmazását, és javíthatják a kód általános olvashatóságát.

Minden komponensnek van egy HTML-sablonja, amely deklarálja, hogy az adott komponens hogyan jelenik meg. Ezt a sablont soron belül vagy fájl elérési útja alapján határozhatja meg.

Az Angular kiterjeszti a HTML-t további szintaxissal, amely lehetővé teszi dinamikus értékek beszúrását az összetevőből. Az Angular automatikusan frissíti a megjelenített DOM-ot, amikor az összetevő állapota megváltozik. Ennek a szolgáltatásnak az egyik alkalmazása a dinamikus szöveg beszúrása, amint az a következő példában látható.

A sablon szintaktikai útmutatói megmutatják, hogyan vezérelheti az UX/UI-t az adatok koordinálásával az osztály és a sablon között.

E mellett lehetőségünk van alkalmazásaink HTML szókincsét bővíteni speciális Angular szintaxissal a sablonokban. Az Angular például segít a DOM értékek dinamikus lekérésében és beállításában olyan szolgáltatásokkal, mint a beépített sablonfüggvények, változók, eseményfigyelés és adat-összerendelés.

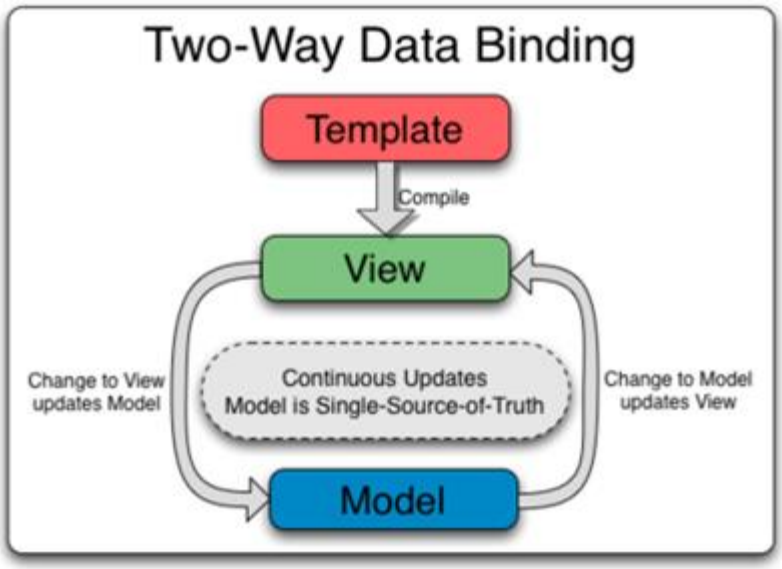
Szinte minden HTML szintaxis érvényes sablonszintaxis. Mivel azonban az Angular sablon egy teljes weboldal része, és nem a teljes oldal, nem kell olyan elemeket beillesztenie, mint a <html> vagy <body>, és kizárólag az az oldal, amelyet fejlesztünk. [6]

A TypeScript osztály felelős főképpen a logikáért. Ebben helyezkedik különböző funkcionalitást ellátó metódusok, amik például a html bemeneti elemek lenyomásakor vagy az oldal betöltésekor történik. Nehezebb vagy esetleg akad olyan logikai művelet, amit más komponens is használ, akkor azt célszerű egy service be kiszervezni. Ezt az ng generate service <service> paranccsal lehet megvalósítani.

Azt, hogy automatikus milyen stílusleíró nyelv legyen alapértelmezett komponens generálásnál azt a projektek megalkotását követő környezeti beállításokba lehet megtenni, de későbbiekben is van rá lehetőség. Alapvetően ez a komponens rész felel a megjelenésért.

2.4.2 Kétoldalú adatkötés

A kétoldalú adatkötés az Angular, egyik leghasznosabb tulajdonsága a többi kliens oldali keretrendszerrel szemben. Ez az adatkötés segít a komponensnek a nézethez eljuttatni a szükséges adatot, vagy esetleg a nézetben található adatokat a komponensnek. Az alapértelmezetten AngularJS-ben volt benne, a legújabb verziókban az ngModel segítségével lehet elérni. (Ahogy 12. ábra mutatja.)

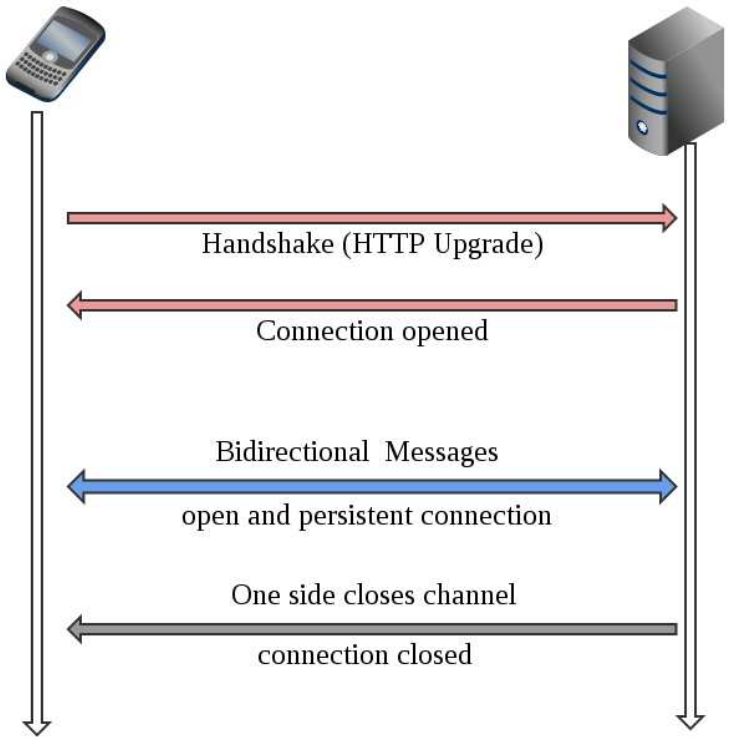


12. ábra
Két oldalú adatkötés példa

2.5 WebSocket

Alapvetően a WebSocket egy olyan webes technológia, ami kétirányú csatornák segítségével teszi lehetővé a kommunikációt a TCP protokollon keresztül. Megalkotása főleg a WebSocket-et megelőző az előtti Comet programozási technikára épül. Fő tulajdonsága, hogy az alkalmazás a Böngésző és a szerver között tudjon két irányban kommunikálni. Ezeket, hogy végrehajtsuk úgynevezett kézfogásokon keresztül kell validálnia a kapcsolatot. Ez úgy történik, hogy a kliens oldal elküld egy kézfogási kérelmet, amire a szerver oldal válaszol. Miután kiépül a kapcsolat teljesen duplex módon lehet kommunikálni a két fő rész között. (Ahogy a 13. ábra jellemzi.)

Ezeknek a fő létjogosultsága, hogy valamilyen értesítés vagy szinkronizálás a frontend és backend között.



13. ábra
WebSocket működése

2.5.1 WebSocket és a REST között

A WebSocket egy TCP-kapcsolaton keresztüli kommunikációs protokoll, amely pontból pontba kommunikációs rendszert biztosít. A protokoll szabványosítása lehetővé tette a használatát, ami nagyon fontos volt az adatok böngészőből a szerverre és a szerverről történő átvitelére. A REST, azaz a reprezentatív állapotátvitel meghatározza a webszolgáltatások létrehozásához használandó megszorítások készletét. Ez a REST-végpontok webalkalmazásban történő HTTP használatával történő létrehozásának egyik alapvető követelménye.

A főbb különbségek a WebSocket és REST között:

- A WebSocket egy alacsony szintű protokoll, amely a socket és a port fogalmán alapul, amelyek az alapul szolgáló szállítási mechanizmusok, míg a REST a CRUD mechanizmuson alapul.
- A WebSocket megköveteli az IP-cím és a port adatainak használatát, amelyek minden alkalmazás esetében alacsonyabb szintű adatok, míg a RESTful alkalmazásnak igéken és HTTP alapúan kell megterveznie a működést.
- A WebSocket kétirányú jellegű, azaz mindkét irányban lehetséges a működés a kliensről a szerverre és fordítva, míg a REST egyirányú megközelítést követ.
- A WebSocket megközelítés ideális a valós idejű skálázható alkalmazásokhoz, míg a REST jobban megfelel a sok kérést igénylő teendőknek.

- A WebSocket egy állapottartó protokoll, míg a REST állapot nélküli protokollon alapul, vagyis a kliensnek nem kell tudnia a szerverről, és a szervernek a kliensről se.
- A WebSocket ideális olyan helyzetekben, ahol a nagy terhelés a játék részét képezi, azaz a valós idejű méretezhető csevegőalkalmazás, míg a REST jobban illeszkedik az alkalmi kommunikációhoz egy tipikus GET kérés forgatókönyvében, amikor RESTful API-at hívnak.
- A WebSocket jobban működik, ha a kliens-szerver ugyanazon a TCP-kapcsolaton keresztül kommunikál a WebSocket kapcsolat teljes élettartama alatt, míg HTTP-kérés esetén új TCP-kapcsolat indul.
- A WebSocket kommunikáció lehetővé teszi a kliens és a szerver számára, hogy egymástól függetlenül beszéljenek, míg a REST-alapú megközelítésben vagy a kliens beszél a klienssel, vagy a szerver beszél az ügyféllel bármikor.
- A WebSocket kommunikációs költsége alacsonyabb, míg a REST-alapú kommunikáció költsége viszonylag magasabb.

	WebSocket	**REST**
HTTP	A HTTP használata a kezdeti kapcsolat során történik.	A HTTP egy általános protokoll a RESTful webszolgáltatásokban.
Kommunikáció	Kétirányú.	Egyirányú.
Koncepció	Socket alapú koncepció.	Erőforrás-alapú koncepció, nem parancsok.
Függőség	Támaszkodjon az IP-címre és a port számra.	A HTTP protokollon alapul, és HTTP metódusokat használ az adatok továbbítására.
Költsége	Költsége relatív kevés.	Költsége nagyobb, mint a WebSocket-é
Teljesítmény	Jobban működik nagyobb terhelés alatt.	Alkalmi kommunikációra kiváló.
Protokollok	A WebSocket egy állapotalapú protokoll.	A REST a HTTP-n alapul, amely egy állapot nélküli protokoll.

2.tábla
WebSocket és REST API összehasonlítás

2.5.2 WebSocket API

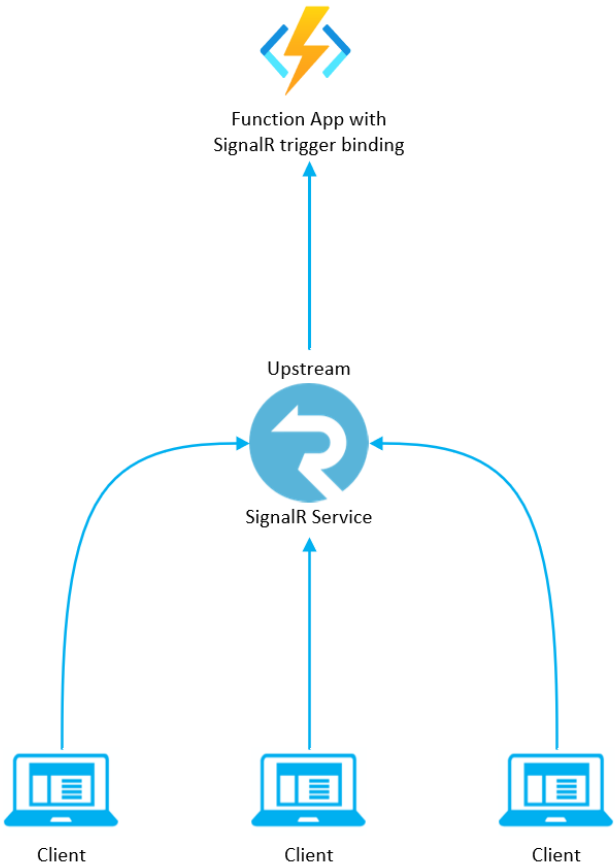
Ahhoz, hogy ezt a technologiát a kódunkba is alkalmazzuk rendszerint egy WebSocket API-t használunk. Ez a rendszer egy hármas interface felépítést követ az alapvető WebSocket, ami ugye a kliens és szerver kapcsolatfelépítéséért és kommunikációjáért felel. A CloseEvent, ami akkor következik be miután a kapcsolat lezárul. Valamint MessageEvent, ami az az esemény, ami akkor keletkezik amikor egy üzenet érkezik.

Ennek a WebSocket-nek nyelvtől és framework-től is függ, hogy milyen megvalósítással oldjuk meg a WebSocket protokollnak az egyik leghíresebb API-k a SignalR, a AsyncAPI és a Gorilla WebSocket.

2.5.3 SignalR

A SignalR egy nyílt forráskódú szoftverkönyvtár a Microsoft ASP.NET számára, amely lehetővé teszi, hogy a szerver aszinkron értesítéseket küldjön az kliens oldali webalkalmazásoknak. A könyvtár szerveroldali és kliensoldali JavaScript összetevőket tartalmaz. (14. ábra)

A SignalR egy egyszerű és tiszta API-t kínál olyan valós idejű webes alkalmazások írásához, ahol a szervernek folyamatosan adatokat kell továbbítania az ügyfeleknek. Gyakori alkalmazások a chat, hírfolyam, értesítések, többjátékos játékok. [7]



14. ábra
SignalR működése

A SignalR számos küldési lehetőséget kihasznál, és automatikusan kiválasztja a legjobb elérhető küldési lehetőséget, figyelembe véve a kliens és a szerver képességeit. A SignalR kihasználja a WebSocket, amely lehetővé teszi a kétirányú kommunikációt a böngésző és a szerver között. A SignalR a burkolat alatt használja a WebSocket-et, amikor elérhető, és ha nem képes egy másik technikához és technológiákhoz, miközben az alkalmazás kódja változatlan marad.

2.6 Összegzés

Irodalomkutatásom során végigevettem az általam legfontosabbnak tartott rendszerek, fejlesztői környezetek alapvető ismertetőjeleit, lényegesebb információit. A fejezetben átvettem különböző keretrendszereknek specifikációit, hogy miért pont azt használjuk és miben különböznek elődjeiknél.

Először a hasonló rendszerek témakörben vettem végig az általam gondolt, az én projektem megvalósításához elengedhetetlen témákat. Mivel a feladatomban kigondolásom szerint lesz kliens és szerver oldal, ezért az első fejezetemben ennek a legismertebb variánsairól írtam, az API-król és ennek a régebbi SOAP protokoll által megvalósított módjáról, valamint az újabb és legtöbb ma használt keretrendszerben alkalmazott REST protokoll megfelelési követelményéről.

Az következő fejezetben kitértem az általam választott szerver oldali keretrendszer specifikusságára és az abban megvalósított adatbázis szolgáltatásra. Ebben összehasonlítottam az álltalános .Net Entity Framework-öt a több platformon is használható .Net Entity Framework Core-os változattal.

Ezután logikusan következő kliens oldali fejezetben kifejtettem előnyét a véleményem szerinti legjobban strukturált JavaScript Framework-nek, az Angular-nak. Ebben leírtam a komponensek felépítését és a főbb aspektusait a TypeScript nyelvnek, majd kitértem az Angular legfontosabb tulajdonságára, a kétoldalú adatkötésre.

Következő fontos téma az értesítés megvalósítása a projektemben, erre a legjobb protokoll és rendszer, amit találtam a WebSocket és az ezt megvalósító Microsoft-os SignalR volt.

Miután ezeket a témákat feldolgoztam arra jutottam, hogy a projektem elkészítéséhez elengedhetetlen ezeknek a témának az ismerete. Ezen okból a feladatom során a Kello-ból kinyert adatokból fogok dolgozni, az előző fejezetekben ismertettet témák szempontjaik szerint.

A szoftver az előbb leírtak alapján fog készülni.

3. Követelmény specifikáció

Alapvetően a probléma amit szeretnék megoldani az az, hogy legyen egy olyan webes alkalmazás, ami képes egy régebbi táblázatos könyvkiosztás ellenőrzést lecserélni egy 21. századi webes felületre. Ennek megfelelően szükséges egy korszerű és akár a privát szektorban is gyakran használt kliens oldali keretrendszer. Ebben a webes megjelenítéstől el kell várni, hogy letisztult, átlátható és felhasználó barát legyen. Ez a program legfőképpen olyan tanároknak és titkároknak készül, akik nem a nehéz és átláthatatlan rendszerekhez szoktak hozzá, ezért megkövetelhető, hogy könnyen elsajátítható és egyszerű funkcionalitásokkal legyen használható.

Ahhoz, hogy az oldal gyorsasága és élvezhető használata fennálljon, el kell különíteni minden nagyobb erőforrást és logikát alkalmazó műveleteket kliens oldalról a szerver oldalra. Mivel az oldalon szenzitív információk is vannak, mint például cím, név és telefonszám, ezért szükség van kliens oldalon egy bejelentkezési felületre, valamint szerver oldalon pedig autentikációra, hogy esetleg ne az összes végpontot lehessen elérni.

Fontos, hogy a rendszerünk képes legyen tájékoztatni a tanuló diákokat arról, hogy mikor, pontosan milyen árban, mely könyveket kapja. Ehhez szükséges egy automatikus tájékoztató rendszer, ami számára a kellő információkkal látja el a felhasználót.

Mivel ez a könyvkiosztás egy másik cég rendszerén történik, a rendszernek nem csak a diákokat, hanem a rendelést elvégző személyt is tájékoztatnunk kell, hogy éppen könyvrendelő időszak van.

Az admin felület tulajdonképpen a könyvek felvételét kezeli a rendszerben, valamint a diákok viszonyát az évfolyamhoz, akiknek a különleges eseteiket is meg kell valósítani.

Ezek a következők lehetnek:

- Tanuló elhagyja az intézményt
- Tanuló megbukik vagy évet ismétel
- Esetlegesen könyvtári könyvet kap, vagy nem igényel valamilyen könyvet

Rendszerünk adatait tároló adatbázison szükséges úgy megvalósítani, hogy egy külön felhő alapú rendszeren kiküldve is használható lehessen és minden felületen elérhető legyen.

4. Tervezés

4.1 Rendszer megtervezése

Ebben a fejezetben az általam választott technológiák elemzésével és döntésem más rendszerekkel való különbségek felsorolásával foglalkozok.

4.1.1 Szerver oldali technológiák

	ASP.NET Core	Express.JS	Django
Programozási Nyelv	C# alapú	JavaScript alapú	Python alapú
Előre elkészült modulok	Van, Nuget csomagok	Van, Express middleware modulok	Van, Django Packages
Cross Platform	igen	igen	igen
Dependency injection	igen	igen	igen
Open-Source	igen	igen	igen
Eredeti szerző	Microsoft	OpenJS foundation	Django Software Foundation

3.tábla
Szerver keretrendszerek összehasonlítása

Szerver oldali technológia kiválasztásnál fő szempont számomra az adott rendszerben megszerzett tapasztalat volt. Ennek érdekében, hogy felkészüljek a feladatra megnéztem, hogy mik a legnépszerűbb backend technológiák jelenleg.

Programozási nyelvben mind a háromra más igaz, hogy mind a C# mind a Python C alapú, de paradigma szempontjából Python egy funkcionális nyelv. JavaScript egy Prototype típusú nyelv dinamikus adat típusokkal, ezért egy kezdőbb szintű felhasználónak ideális.

Mind a 3 szerver oldali keretrendszernek vannak előre elkészített moduljai, amik könnyebé teszik a nagyobb rendszerek fejlesztését. Alapvetően az ASP.NET-nek NuGet csomagjait a Microsofton kívül akár harmadik féltől származó csomagokkal is dolgozhatunk.

Az összes népszerűbb keretrendszer esetében beszélhetünk arról, hogy Cross-platformok. Ez például ez azért jelent előnyt, mert ha csak a Windows szerveren lehetne futtatni a kódunkat akkor az elérhető szerverek 96%-at nem használhatnánk mert azokon Linux fut.

A világ 1 millió szerverének 96,3%-a Linuxon fut. Mindössze 1,9% használ Windows-t, és 1,8% - FreeBSD-t. [8]

ASP.NET kivételével minden másik keretrendszer fejlesztői Foundation-ön keresztül jönnek létre. Ezeknek az az előnye, hogy sokkal jobban hallgatnak a közösség visszajelzésére. Ezzel szemben az ASP.NET-et a Microsoft fejleszti. Az ő előnyük pedig az óriási pénzügyi hátterük miatti erőforrásuk. Például, ha egy kép feltöltő NuGet csomagot szeretnének csinálni, 1 nap alatt tudnak emberi erőforrást vagy ezzel foglalkozó céget alkalmazni.

4.1.2 Kliens oldali technológiák

	Angular	React.JS	Vue.JS
Framework	igen	nem	igen
Virtual DOM	nem	igen	igen
Fő nyelv	TypeScript	Jsx	JavaScript
Előre elkészült modulok	Van	Nincs	Van
Két oldalú adatkötés	Van	nincs	van
Eredeti szerző	Google	Facebook	Evan You

4.tábla
Kliens keretrendszerek összehasonlítása

Kliens oldali framework kiválasztásánál a főbb szempontom a fejlesztő barát technológia volt, mivel ez számomra a strukturáltságán és előre megírt jól dokumentált moduljai miatt az Angular-ra esett a választásom.

Alapvetően a Framework-ök és a libary-k között eléggé átláthatatlan sokszor a különbség, de ha mégis meg kéne fogalmazni, akkor a főbb szempont, hogy a fejlesztés során előre megírt modulokat használhatok, vagy harmadik fél által generált kód részletet alkalmazhatok.

A könyvtár lényegében egy sor hívható függvény, amelyek manapság általában osztályokba vannak szervezve. Minden egyes hívás elvégez valamilyen munkát, és visszaadja a vezérlést az ügyfélnek.

A keretrendszer egyfajta absztrakt tervezést testesít meg, több viselkedés beépítésével. A használatához a saját viselkedést a keretrendszer különböző helyein kell elhelyezni, vagy alosztályozással, vagy saját osztályok beillesztésével. A keretrendszer kódja ezután ezeken a pontokon hívja az Ön kódját. - Martin Fowler által adott definíció [9]

A Virtual DOM előnye a többboldali web alkalmazásoknál jön ki, mert ilyenkor nem egy új HTML dokumentumra váltunk, hanem az előző oldal különbségéhez képest renderelünk újra mindent.

Másik főbb szempontom a típusosság megőrzése volt, igaz ma már mind a Vue és a React is támogatja a Typescript-et, mégis alapértelmezetten az Angularban van ez a lehetőség meg. Ez kijavítja véleményem szerinti legnagyobb problémát JavaScripttel a típusosság hiányát.

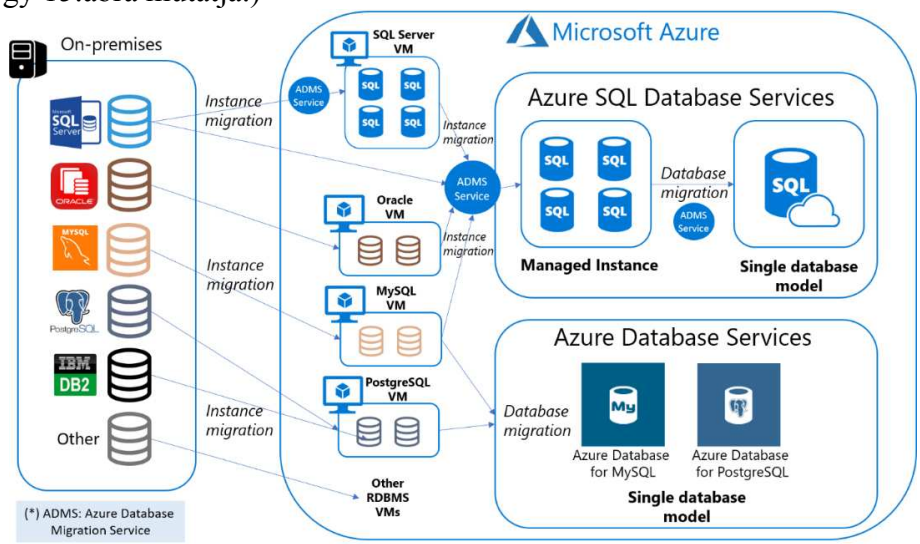
A kétirányú kötés lehetőséget biztosít az alkalmazás összetevőinek az adatok megosztására. A kétirányú kötés segítségével eseményeket hallgathat és értékeket frissíthet egyszerre a szülői és a gyermek komponensek között. [10]

Alapvetően a kétoldalú adatkötés egy eléggé anti-pattern fejlesztési mód, viszont a lehetőség, hogy te egyszerre lehet a hierarchiában föl és lefelé is hivatkozni. Ez kliens oldalon, azért lehet előny, mert így akár egy egyoldali alkalmazáson is lehetséges egymásba ágyazott komponenseken keresztül hivatkozni a másikban lévő adattagokra.

4.1.3 Felhő alapú adatbázis

Az általam választott adatbázis ellátó rendszer az Azure SQL adatbázis szolgáltatása, mivel támogatják a diákokat egy kedvezményes árakkal és jól megírt dokumentációval.

Skálázhatósági lehetőség is nagyon egyszerűen alkalmazható hiszen akár a minimum 2 gigás adatforgalmunkat is több terrásra tudjuk növelni, ha rendszerünknek szüksége lenne rá. (Ahogy 15.ábra mutatja.)



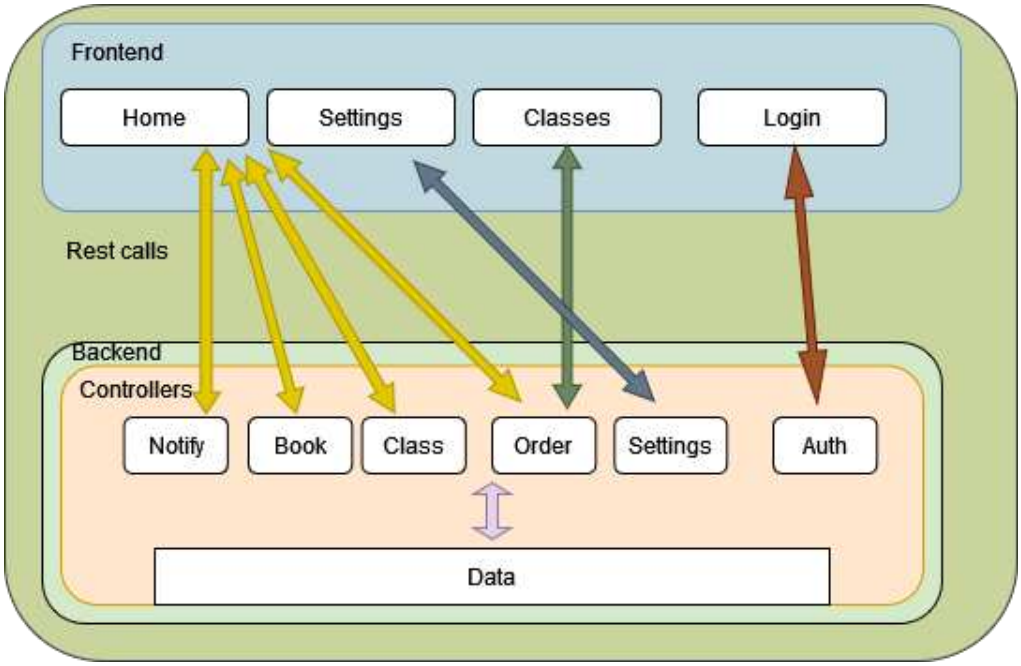
15.ábra
Microsoft Azure SQL adatbázis működése

Az Azure-ban az adatbázis-kiszolgálókat közvetlenül IaaS VM-ekre migrálhatja (tiszta lift és shift), vagy további előnyökkel az Azure SQL Database-re is áttérhet. Az Azure SQL Database a menedzselt példány és a teljes adatbázis-szolgáltatás (DBaaS) lehetőségeket kínálja. A 3-1. ábra mutatja az Azure-ban elérhető többféle relációs adatbázis-migrációs útvonalat. [11]

Lehetőségünk nyílik még akár különböző SQL nyelvekből migration segítségével, ez az Azure rendszerében úgy történik meg, hogy különböző virtuális meghajtókon átalakítjuk az Azure által használt Single database model-lé.

4.2 Rendszerterv

Ebben a fejezetben a rendszerem főbb részit fogom részletezni, hogy milyen felépítésben fog létrejönni. (Ahogy a 16. ábra jellemzi.)



16. ábra
Rendszerterv ábra

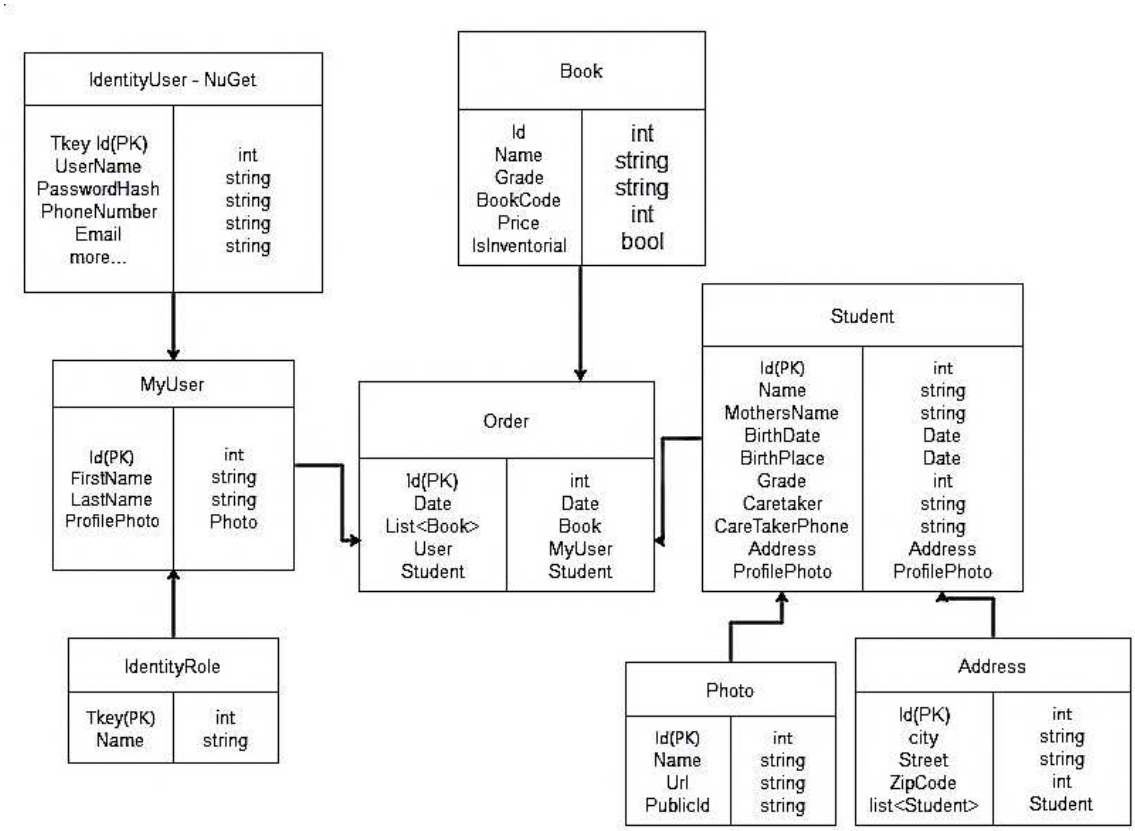
A rendszerembe alapvetően különálló szerver komponensek, backend és frontend van, amik Rest hívásokkal kommunikálnak. Valamint esetleges adat módosítások során a szerver oldali komponensünk a felhő szolgáltatásunkkal kommunikál. Ezeket az adat módosításokat az Entity Framework végzi el, hogy C#-os adattáblából egy általunk választott úgymond „Adat fordítóval” elvégezzük és elküldjük a felhőszolgáltatás SQL adatbázisnak egy adott nyelvre lefordított SQL scriptet.

Ez a tervezési megközelítés leginkább az MVVMC-re hasonlít mivel a kliens oldalunknak és a szerver oldalunk is megvalósít logikát, de ezek elvannak különítve böngészőben megvalósított könnyű logikára és szerver oldalon lévő „nehéz” logikára. Ez a megoldás a skálázhatóság miatt is hasznos, mert ha esetleg nagyobb rendszerünk lesz már külön szerver komponenseken van UI és a Business Logic.

4.2.1 Adatbázisom felépítése

Az adatbázisom kezdetleges tervét az általam kigondolt különálló táblákból készítettem el. (17. ábra) Táblák:

- MyUser: egy IdentityUserből leszármaztatott entitás.
- Order: rendeléshez szükséges adattokkal rendelkezik
- Book: könyvek adatait tartalmazza
- Student: diákok 4T és adatai és esetleges UI szempontjából kellő adatok
- Address: Diákok Címeit tartalmazza
- Photo: 3. külső képtároló rendszerhez szükséges adattagok



17. ábra
Entitás Kapcsolati diagram

4.2.2 Funkciólista

Ebben a fejezetben az általam kigondolt végpontok listáját írom le. Ezeket Publikus és védett végpontokra különítjük el.

Nyílt elérhetőségű végpontok:

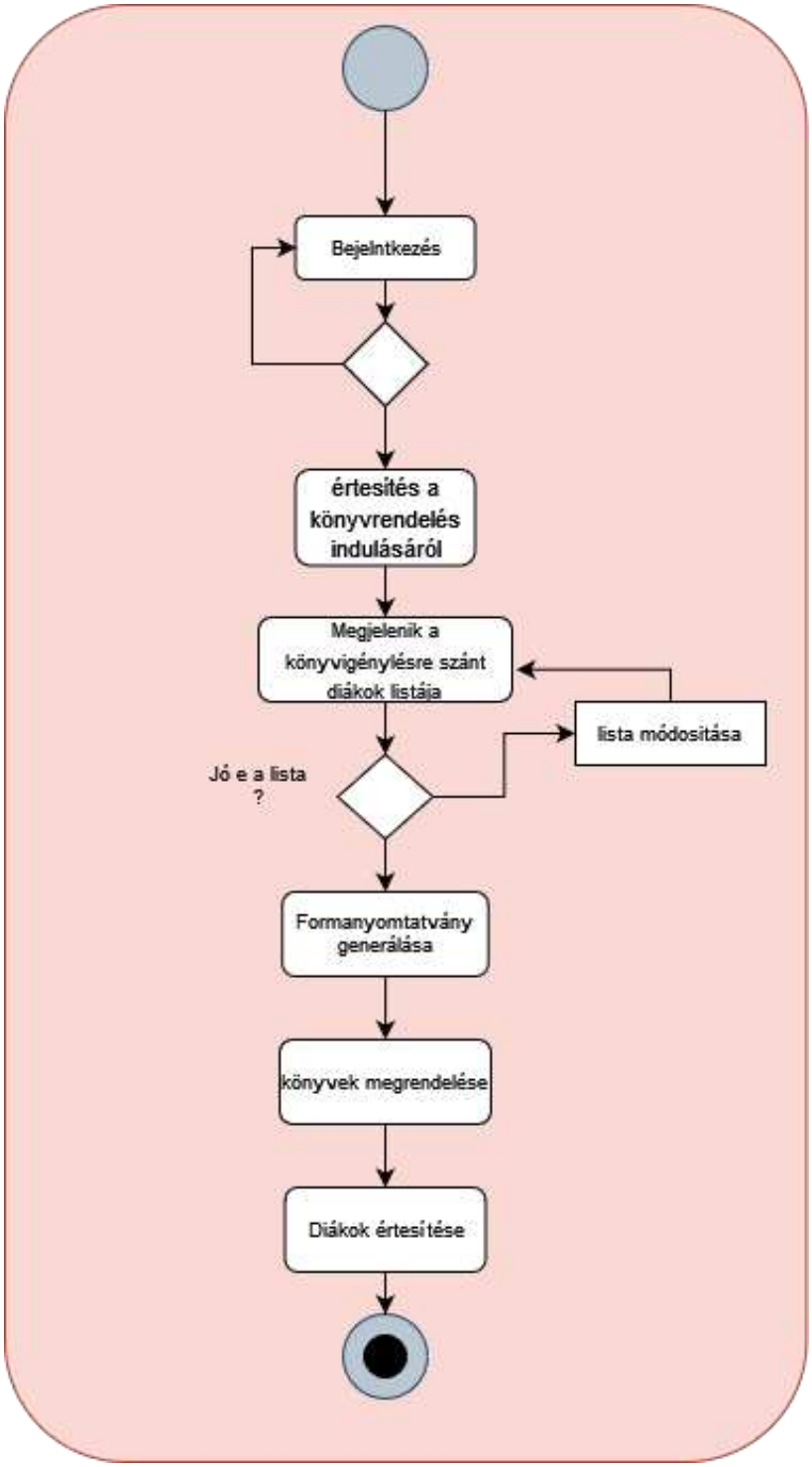
- `[POST]`
  `Api/Auth/LogIn:`
  A Felhasználó bejelentkezése és az autentikált végpontok eléréséhez szükséges token visszakapásához szükséges
- `[PUT]`
  `Api/Auth/Register:`
  a Felhasználó az adatbázisba történő regisztrációjához szükséges végpont

Végpontok, Amiket a Felhasználó érhet el:

- `[PUT]`
  `Api/User/ModifyUser:`
  Felhasználó adatainak módosításához szükséges végpont
- [`GET`]
  `Api/User/GetUser/{id}:`
  Egy felhasználó lekérés egy adott azonosító által
- [`POST`]
  `Api/User/NotifyUser:`
  A felhasználó értesítése a könyvfoglalás időszakáról
- `[POST]`
  `Api/Order/CreateOrder:`
  Rendelés kiírásához szükséges végpont
- `[POST]`
  `Api/Order/SendOrder:`
  Rendelés elküldéséhez szükséges végpont
- `[PUT]`
  `Api/Order/ModifyOrder:`
  Rendelésadatainak módosításához szükséges végpont
- `[DELETE]`
  `Api/Order/DeleteOrder:`
  Rendelések törléséhez szükséges végpont
- `[POST]`
  `Api/Book/AddBook:`
  Könyv a leltár hozzáadásához szükséges végpont
- `[DELETE]`
  `Api/Book/RemoveBook:`
  Könyv törléséhez szükséges végpont a leltárból

- [PUT]
  Api/Book/ModifyBook:
  User adatainak módosításához szükséges végpont
- [GET]
  Api/Book/GetBook/{id}:
  Egy könyv lekéréséhez szükséges végpont egy adott azonosító által.
- [GET]
  Api/Book/GetBooks:
  Könyvek kilistázásához szükséges végpont.
- [POST]
  Api/Student/AddStudent:
  Diák hozzáadásához szükséges végpont.
- [DELETE]
  Api/Student/RemoveStudent:
  Diák törléséhez szükséges végpont.
- [PUT]
  Api/Student/ModifyStudent:
  Diák adatainak módosításához szükséges végpont.
- [GET]
  Api/Student/GetStudent/{id}:
  Egy diáklekérése egy adott azonosító által
- [GET]
  Api/Student/GetStudents:
  Összes Diák lekérése

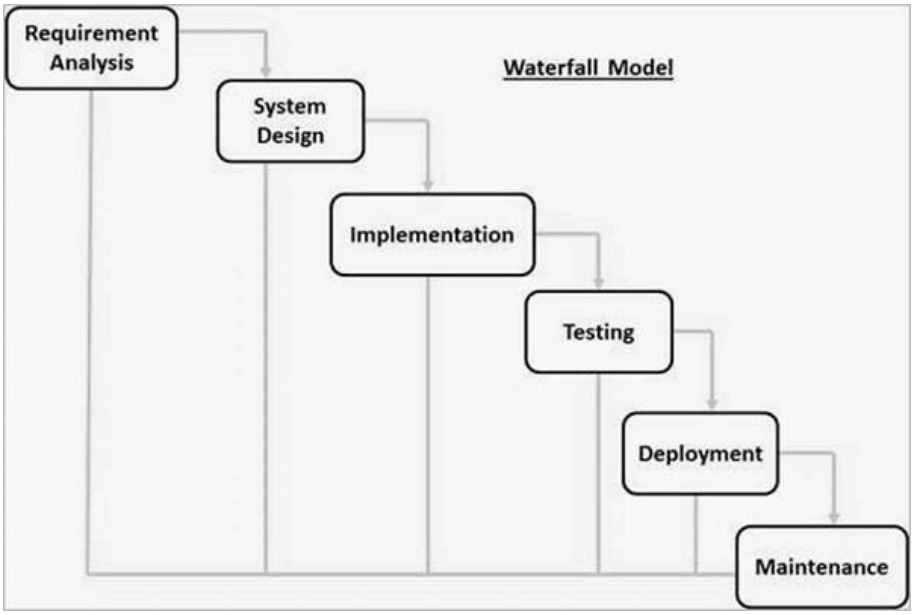
Rendszeremet egy folyamat ábrázolásán keresztül bemutatom, hogy lehet egy felhasználónak egy könyvet megrendelnie és azon belül mikor és hol és kinek történik meg az értesítési folyamat. (Ahogy a 18. ábra ábrázolja.)



18. ábra
Könyvrendelés szekvencia diagramja

4.2.3 Fejlesztés tervezett menete

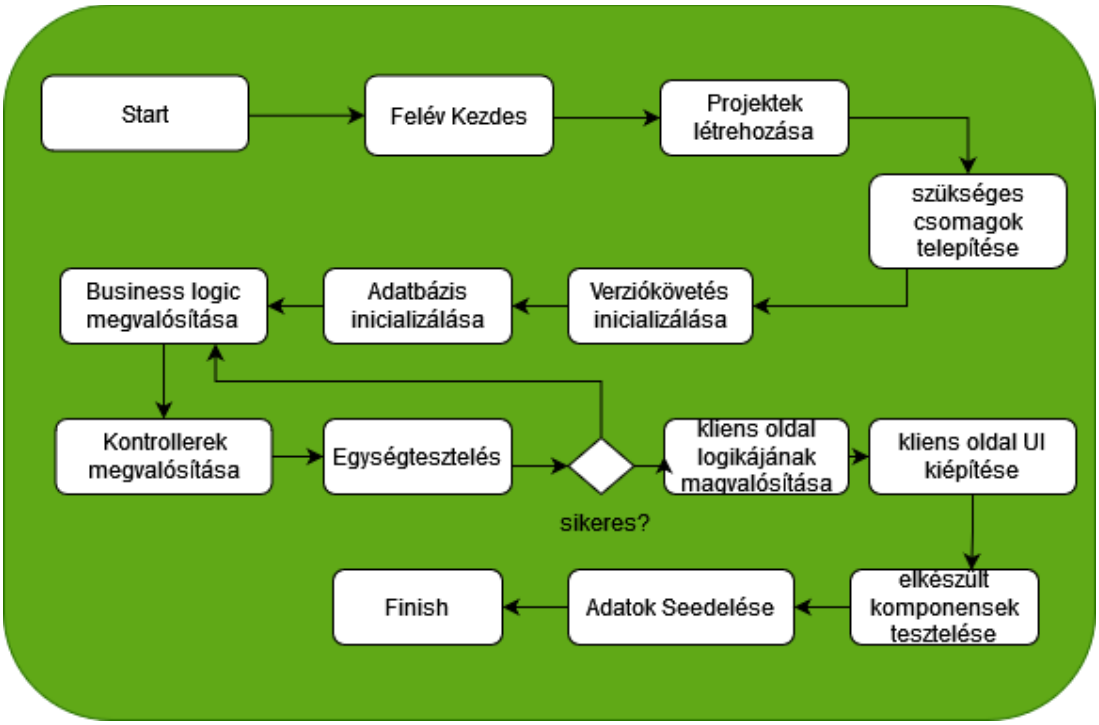
A fejlesztés ebben a feladatban egyedül végzem így az általános csapatokra épített agilis fejlesztési metodológiák nem lesznek szükségesek, mint a SCRUM. Ezért egy sokkal jobban az egyénre tervezett régi metodológiát választottam a vízesés modellt. Itt az időmet főképpen az egymást követő taszkok alkotják majd. (Ahogy a 19. ábra jellemzi.)



19. ábra
Vízesés modell felépítése

A fejlesztésemet egy előre megkövetelt verziókövető rendszerben a GitHub-ban töltöm fel, ez azért fontos, mert esetleg bármilyen lokális hiba vagy törlés esetén is elérhető egy régebbi verzió.

A fejlesztésem először a projektek létrehozásával történik, majd a szükséges NuGet csomagok telepítésével folytatódik. Ezután az előbb említett GitHub repository-ba Feltöltöm az eddigi projektet. Ezután az lokálisan kreálok egy adatbázist majd ezeket a kódomba a táblák alapján migration segítségével inicializálom. Ezután történik a logika és a kontrollerek megírása és ezeket egységteszteléssel ellenőrzőm, ha minden megfelelő akkor megyek tovább a kliens oldalra, ha nem akkor ellenőrzőm a logikámat és a tesztelést, hogy jó-e. Ezután történik a kliens oldal logiája és alap funkciójának implementálása, majd kinézetének megalkotása a követelménybe maghatározott szempontok alapján. Ezután funkciótesztet végzek és ha minden megfelel el mentek adatokat, hogy lehessen velük tesztelni. (Ahogy a 20.ábra mutatja.)



20.ábra
Fejlesztés menetét leíró diagram

5. Fejlesztési Napló

5.1 Fejlesztési napló bevezetés

A webes alkalmazásom elkészítése során törekedtem az általam előirt funkció követelményeket teljesíteni. Funkciók alapvetően Chromium specifikusan készültek, így ügyeltem arra, hogy az általam választott fejlesztési csomagok és keretrendszerek erre alkalmasak legyenek.

Fejlesztés során alkalmaztam az általam ismert területeket, a szerver közeli ASP.NET Core szempontjait és előnyeit figyelembe véve olyan csomagokat kiválasztani, amiket korábban már használtam és kellő tapasztalatom van benne.

Korábbi munkáimból több módszertant felhasználtam, ilyen volt például az e-mail-es levelezésre használt smtp kliens. Másik az értesítés funkció szempontjából megvizsgált a websocketekkel történő kommunikációra használt SignalR. Felmerültek olyan megoldandó feladatok, hogy hogyan lehet Excel fájlba exportálni adatokat, hogy azt később letöltésre tudjon kerülni.

A webalkalmazás fejlesztése során több probléma felmerült, amit leginkább az eredményezett, hogy olyan megoldandó problémába kerültem, ami tervezés során nem merült fel.

Egyik legelső probléma az entitás táblám újragondolása volt, mert előzőleg általam kigondolt profil kép funkciót nem tartottam relevánsnak a projektem során. Valamint több adat taggal volt szükséges kiegészíteni a táblákat, hogy az általam kigondolt funkciókat megfelelően implementálhassam.

Végül pedig olyan probléma is fennált, hogy az általam kigondolt éves évfolyam léptetés automatikus megvalósítása ideálisabb lett volna, ha manuálisan történik, de végül a későbbi további fejlesztés és új funkciók implementálása miatt szükségesnek tartottam a feladat kiírásom szerint előre definiált módon megvalósítani.

5.2 NSwag Service generátor

Az NSwag: The Swagger/OpenAPI toolchain for .NET, ASP.NET Core and TypeScript egy olyan interface generátor segédeszköz, ami segítségével egy gomb lenyomásával képesek vagyunk kontrollerek paraméteri és visszatérési értéke alapján legenerálni egy általunk előre beállított szempontok szerinti TypeScript fájlt. Ez leginkább a nagyobb Enterprise szintű rendszereknél szokás alkalmazni, de a saját szempontomból a monoton service írások helyett, egy sokkal gyorsabb módja az API kommunikáció létrejöttének.

Ennek az API generálásnak több fajta módja lehet, az egyik a minden build-eléskor lefut és úgy generálja le a klienst, hogy az ehhez szükséges alkalmazásunk csproj fáljába beírjuk a NuGet csomag cél futtatható fájlt. Viszont tapasztalataim alapján .Net 6-ra még nem kompatibilis, így az én projektemben ez irreleváns lenne alkalmazni.

Másik módja a generálásnak, hogy a program futtatáskor egy adott időpontban swagger által láttot végpontokat JSON formában lekérjük és az alapján képezünk TypeScript fájlt.

Ennél eggyel jobb megoldás az általam választott, hogy a NSwag stúdió során végig megyünk a különböző opciókon és minden egyes opció után kutatást végzünk, vagy fórumokat olvasunk, hogy az adott projektben az egy releváns opció-e vagy nem. Amennyiben megvannak ezek az opciók, a generate output gomb segítségével ellenőrizhetjük, hogy ez egy működőképes a generált kliens vagy nem. Ezek után, ha mindent leellenőriztünk, generate file gombra kattintva felülírhatjuk a kliens generált fájlt. Majd az eddigi konfigurációt elmentjük egy JSON fájlba, abból a célból hogyha módosítjuk a kontrollereinket, akkor elég csak egyszer a generate file-ra kattintani és nem kell az egész konfigurációt újra megvalósítani.

A legjobb megoldás az lenne, hogy ez az elmentett JSON konfigurációs fájlt parancs sorból futtatjuk, viszont tapasztalataim során szintén ugyan abba a problémába kerültem, hogy a dokumentum írásakor nem kompatibilis még .Net 6-tal.

Egy megoldandó feladat az volt, hogy hogyan lehet az access token-t átadni ennek a generált fájlnak. Választ egy hosszas fórum olvasás után találtam meg. Lényege, hogy generált kontroller klienseket leszármaztatjuk egy ősosztályból. Ebben az ősosztályban megszabjuk, hogy az autorization headerbe, az általunk kapott token kerüljön be. Ennek segítségével jogosultság kezelést tudunk megvalósítani.

5.3 Angular Material

Az Angular Material egy UI komponens könyvtár, aminek segítségével könnyen és gyorsan tudunk felhasználóbarát webes alkalmazásokat létrehozni, ami ezen rendelkezik a mai rendszerfejlesztésből nélkülözhetetlen böngésző függetlenséggel, ezen belül reszponzivitással. Valamint fontos a fejlesztés kontinuitás szempontjából, hogy hosszú távú komponens támogatással rendelkezik.

Számomra azért volt fontos, hogy valamilyen css framework-öt használjak, mert fejlesztés során hátrányos a nagyon részletes munka, ami véleményem szerint a kinézet variálásra jellemző. Én fejlesztés során jól alakítható, de mégis egy olyan komponenskönyvtárral rendelkező framework-öt kerestem, ami könnyen implementálható.

Alapból 4 témával rendelkezik, de könnyen variálható személyre szabottan alakítható, valamint megoldható benne a ma már egyre elterjedtebb sötét téma kialakítása.

Fejlesztés során leginkább a kinézet alakítási kompetencia hiány alakult ki, mint probléma, de az Angular Material jó dokumentációja, valamint előre elkészített példák miatt gyorsan megoldásra találtam.

5.4 SignalR implementáció

A SignalR-nek több hasznos megvalósítása létezik, mint például üzenet küldő alkalmazásoknál, hogy mindig folytonos értesítés vagy üzenet érkezzen anélkül, hogy bármikor frissíteni kéne a felhasználónak az alkalmazást. Az én fő funkcióm szempontjából a könyvrendelési időszakról értesítjük a különböző osztályoknak a tanárait.

A funkció implementálás során el kellett különíteni a szerver és kliens oldali teendőket. A kliens oldalon könnyebb volt a megvalósítás, ott miután kialakítottam a felületet csak azt kellett megoldani, hogy minden esemény hívásakor betöltsük az éppen aktuális listát lefrissítve.

Szerver oldalon nehezebb megvalósítás volt, mert hitelesítést is meg kellett valósítani benne. Itt különböző fórumok böngészése után találtam egy jól implementálható megoldást.

5.5 SMTP Kliens

Az SMTP a Simple Mail Transfer Protocol rövidítése, ami egy szabvány kommunikációs protokoll az e-mailek Interneten történő továbbítására. Az SMTP egy viszonylag egyszerű, szövegalapú protokoll, ahol egy üzenetnek egy vagy több címzettje is lehet. Könnyen tesztelhetjük az SMTP-t a Telnet program segítségével. Az SMTP szolgáltatás a TCP 25-ös portját használja.

Az én SMTP kliens kialakításánál fontos volt, hogy kívülről konfigurálható legyen, mert véleményem szerint ez egy szenzitív adatnak számít, hiszen gondoljunk bele, hogy ha valaki ismeri az szerverem e-mailjét, akkor bárkinek képes a mi e-mailünket felhasználva a saját üzeneteit elküldeni.

Szintén elmondható az, hogy élesítés esetében előrelátó gondolat ez a kiszervezés, mivel Domain bérlés esetében a szolgáltatás része az smtp. Egyfajta aláírásnak tekinthető, ha a weboldal címe és az e-mail küldője egy Domain tartományban található.

Ebben a külső konfigurációban szerepelnie kell egy smtp host-nak, ez jelen esetben Google termékét a Gmail-t használtam. Ezen felül megkell adnunk egy port számot és egy tulajdonos e-mailt, hogy ezeket utána minden e-mail küldésnél felhasználva lehessen értesíteni. Ezen felül egy kell egy smtp secret, amit 2000 napi limit árán bármilyen Gmail fiók által ki lehet váltani.

Ezt az e-mail küldő funkciót én a könyvrendelésnél történő diákok értesítésére használtam. Magába az e-mail tartalmába az adott diáknak a könyvrendelése van benne, valamit az általa hitelesített tanárnak az e-mailje és telefonszáma.

5.6 XLS generálás

Az XLS fájlok generálásánál több szempontot megvizsgáltam, köztük az újra felhasználhatóságát és hogy mennyire komplex műveletek szükségesek magában a generált fájlban. Ezért választásom, hogy végül egy Angular Material tábla átkonvertálása mellett döntöttem.

Szerver oldali megvalósításnál olyan előnyök merültek fel, mint reportok és statisztika gyártás, de mivel az én feladat meghatározásomban nem szerepeltek ilyen kikötések, ezért úgy döntöttem, hogy kliens oldalon használom a XLSX npm csomagot, ahol a Table_to_sheet metódussal könnyen át tudtam konvertálni az Angular Material táblámat.

5.7 Hangfire

A Hangfire NuGet csomagot a visszatérő feladatok ellátására kerestem és használtam. Egyik fő feladatom az volt, hogy az év végi évfolyam emeléseket az osztályoknak egy automatikus rendszeren keresztül történjen.

Hangfire segítségével az adatbázisunkba tárolhatjuk a különböző folytonos feladatainkat, mert ha esetleg leállna az adatbázis, annak újraindításakor is fennálljon a folyamat.

A feladat ütemezésének megadhatunk különböző időpontokat. Az én megoldásomban, minden nap 1x lefut a folyamat és ha megegyezik az időpont a megadottal akkor létrejön az esemény.

Fejlesztés közben felmerült, hogy ez az automatikus folyamat erre a feladatra kicsit magas az erőforrás igénnyel rendelkezik, hiszen egész évben technikailag 1x fut le, ezért egy manuális esemény küldéssel lehet célszerűbb lett volna a megvalósítás. Viszont a később fennálló továbbfejlesztések miatt, a rendszer implementálása több előnnyel járt, mint hátránnyal.

6. Tesztelés

Az alkalmazásom során több szempontból közelítettem meg a tesztelési folyamatokat. Egyik fő szempont a funkcionalitás ellenőrzése. Viszont mivel különböző teszteseteket alkalmaztam, az alkalmazásom ellenőrzésére Unit tesztelést végeztem.

#	Követelmény	Tesztleírás	Mérés és cél	Eredmény
1.	A regisztrált felhasználók képesek legyenek bejelentkezni	Bejelentkezési felületen megadjuk az általunk használt felhasználónevet és jelszót	Bejelentkezési felületről sikeres navigálás a főoldalra	✅
2.	Tanár hozzáadása a rendszerhez	Adminisztrációs felületen tanár hozzáadása majd azzal a felhasználóval bejelentkezés	Az adatbázisunkban megjelenik	✅
3.	Diák hozzáadása a rendszerhez	Adminisztrációs felületen diákot regisztrálunk a rendszerben	A regisztrált tanuló megjelenik az adatbázisban	✅
4.	Könyv hozzáadása a rendszerhez	Adminisztrációs felületen könyvet regisztrálunk a rendszerben	A regisztrált könyv megjelenik az adatbázisban	✅
5.	Osztály hozzáadása a rendszerhez	Adminisztrációs felületen osztályt regisztrálunk a rendszerben	A regisztrált osztály megjelenik az adatbázisban	✅
6.	Tanár hozzárendelése ez osztályhoz	Adminisztrációs felületen hozzárendelünk egy tanárt az osztályhoz	A regisztrált osztálynak megjelenik egy tanár azonosító az adatbázisban	✅
7.	Diák hozzárendelése ez osztályhoz	Adminisztrációs felületen hozzárendelünk egy tanárt az osztályhoz	A regisztrált diáknak megjelenik egy osztály azonosító az adatbázisban	✅
8.	Tanároknak a könyvrendelési időszakról értesítés	Adminisztrációs felületen beállítani a könyvrendelési időszakot és elküldeni azt	Minden tanárnak megjelenik az értesítési felületén az üzenet és az értesítés jelző ablak	✅

9.	Az értesítési mezőben az értesítés törlése	Értesítési felületen lévő értesítés törlése	Az adott értesítés az adatbázisból törlődik	✅
10.	Rendelési felületen minden a tanárhoz hozzárendelt osztály megjelenik	Főoldalon lekérni az összes felhasználóhoz tartozó osztályt	Főoldalon megjelenik az adatbázisban hazarendelt osztályok	✅
11.	Rendeléshez szükséges könyvek táblába betöltése	Rendelési felületen megjelenik könyvek listája	Rendelési felületen a tábla tartalma megegyezik az adatbázis tartalmával	✅
12.	Elindítani az éves osztálylépteő alkalmazás	Adminisztrációs felületen az éves könyvléptetést elindítani	A megadott osztálylépteési időpontban az osztályok évfolyamszáma megnő eggyel.	✅
13.	Leállítani az éves osztálylépteő alkalmazás	Adminisztrációs felületen az éves könyvléptetést leállítani	A megadott osztálylépteési időpontban az osztályok évfolyamszáma nem változik	✅
14.	Rendelési táblázatból könyvrendelési XLS fájlt generálni	Rendelési felületen a Excel exportot megnyomni	A generált Excel fájlt letöltve megegyezik a tábla tartalmával	✅
15.	Rendelés elküldése az osztályban levő diákoknak	A rendelési felületen a rendelés elküldése az osztályban levő diákoknak	Az email megérkezik az adott tanuló e-mailjére	✅
16.	Tanárok osztályában lévő diákok megjelenítése	Az osztály felületen a diákok listázása	Az adatbázisban lévő tanulok osztály azonosítója megegyezik a megjelenített tanulókkal	✅
17.	Osztályban lévő diákok törlése	Osztályban megtalálható diákok törlése	Az adott tanuló osztály azonosítója nulla lesz	✅

5.tábla
Tesztelés tábla

Ezen kívül a fejlesztésem során alkalmaztam ProofOfConcept típusú feladat tesztelést, hogy ne az éles rendszeremen próbáljam ki először a feladat implementációját. Az leginkább smtp-vel történő e-mail küldés volt, aminek kódját a forrásállományok közé feltöltöttem.

7. Értékelés

Miután végeztem a fejlesztéssel és leteszteltem a folyamatokat, úgy értékelem, hogy pár megvalósítási különbség ellenében az általam készített program a meghatározott követelményeknek megfelel.

A webalkalmazás képes egy éves könyvrendelést átláthatóan és egyszerűen megvalósítani. A rendszer képes értesíteni a tanárokat a könyvrendelési időszakról. A felhasználóknak lehetőségük van az ő általuk kezelt osztályokat menedzselni és a szükséges változtatásokat megtenni, kivenni a szükségtelen adatokat.

Könyvrendelés az eredetitől annyiban tér el, hogy nem automatikusan követi le a diákok óráit évről évre, hanem a tanárnak lehetősége van a könyvek közül szelektálni, a kelő mennyiséget meghatározni.

Az értesítések nemcsak a könyvrendelési időszaknál történnek, hanem az adott rendelésnek az azokon rajta levő diákoknak is elküldi a teljes könyvlistát, olyan célból hogyha esetleg nem szerepel olyan könyv rajta, amit nekik szükséges legyen lehetőségük visszajelezni, hogy azt módosítani lehessen.

Amit nem sikerült megvalósítani az a diák felület, hogy ott kezelje a különböző rendeléseknek a visszajelzését. Fő indok, hogy nem találtam kellően tartalommal telinek a diákoknak fenntartott részt, de egy esetleg későbbi továbbfejlesztés során elképzelhető, hogy megvalósításra kerül.

8. Továbbfejlesztési lehetőségek

További fejlesztési terveimet többféle szempontból közelítem meg. Legelső az el nem készült ötleteknek a megvalósítása, valamint különböző az iskolai élethez fontos és még nem implementált funkciókkal bővítése.

A nem implementált problémára megoldás az, hogy amikor a könyvek ténylegesen megérkeznek és el kell dönteni, hogy kinek mi jut. Arra egy egyszerű akár egynapos bejelentkezéssel elérhetővé váljanak a megérkező könyveknek a megfelelő elosztása.

Esetleg lehetne, hogy az adott iskolának a fórumát ebbe a rendszerbe integrálni, hiszen rengetek kisebb általános iskolának, nincsen arra forrása, hogy egy külön rendszergazdát fenntartson ilyen feladatokra. Ez kiváltana a jellemzően kevés megtekintéssel rendelkező iskolai faliújságot.

Ötletként felmerül bennem az, hogy ha például valaki hosszabb időre megbetegszik vagy nem képes bemenni az tanulásra szánt intézménybe, hogy nekik egy tanuló videó megosztó részt alkotni, ahol kisebb nagyobb tesztekkel ki lehetne váltani a tudást igazoló dolgozatokat vagy témazárókat. Valamint egy olyan résszel bővíteni, ahol kialakulhatnának online korrepetálási felületek vagy tanulószobák, ahol a prominensebb diákok tudnának oktatni az olyan társaiknak, akik valamilyen lemaradással küzdenek.

9. Összegzés

A közoktatásban egy adminisztrációs problémákkal és időigényes megvalósítással teli feladat az arra az évre történő tankönyvek kiosztása minden évfolyam és diák számára. Erre alkotott webalkalmazásom képes a rendszerint tanárok, iskola titkárok, könyvtárosok vagy akár diákok által valamilyen táblakezelő szoftverrel vagy akár írott papír alapon történő könyvrendelést felváltani az én általam alkotott rendszer. Ahelyett, hogy nekik egyesével kell végig menniük a rendelési listán, hogy melyik diák milyen könyveket igényel.

A célom az volt, hogy egy olyan web alkalmazást fejlesszek, ami képes ezt a könyvelosztási feladatott könnyebben és automatikusan megvalósítani. Az alkalmazásban évfolyam, osztály szerint lehet szűrni arra, hogy ki milyen könyveket igényelt. A szoftver az évfolyamot sikeresen teljesítő diákokat automatikusan lépteti a következő osztályra év végén. A speciális eseteket, mint például: egy diák elmegy az iskolából, osztályt vált vagy nem sikerül teljesíteni az adott évet, manuálisan tudja kezelni a felhasználó.

Véleményem szerint ez egy bármely iskola számára könnyen implementálható könyvrendelési folyamat, amit kis oktatással is el lehet sajátítani.

Irodalomjegyzék

[1] Kreta dokumentáció,
(https://tudasbazis.ekreta.hu),
utoljára megtekintve: 2021.12.05.

[2] Könyvtárellátó Nonprofit Kft. Szolgáltatásainak leírása,
(https://www.kello.hu/szolgaltatasaink),
utoljára megtekintve: 2021.12.05.

[3] SOAP IBM általi meghatározása,
(https://www.ibm.com/docs/hu/rsm/7.5.0?topic=standards-soap),
utoljára megtekintve: 2021.12.05.

[4] EF Core O/RM szabályok,
(https://docs.microsoft.com/hu-hu/ef/core/),
utoljára megtekintve: 2021.12.05.

[5] Angular sablon felépítése,
(https://angular.io/guide/template-syntax),
utoljára megtekintve: 2021.12.05.

[6] TypeScript alapvető előnyei,
(https://www.typescriptlang.org/),
utoljára megtekintve: 2021.12.05.

[7] SignalR hasznossága,
(https://devblogs.microsoft.com/dotnet/signalr-building-real-time-web-applications/), utoljára megtekintve: 2021.12.08

[8] Framework és Libary közti különbségek,
(https://www.sitepoint.com/most-popular-frontend-frameworks-compared/),
utoljára megtekintve: 2021.12.05.

[9] Angular két oldalú adatkötés,
(https://angular.io/guide/two-way-binding),
utoljára megtekintve: 2021.12.05.

[10] Azure SQL adatbázis belső felépítése,
(https://docs.microsoft.com/en-us/dotnet/architecture/modernize-with-azure-containers/migrate-your-relational-databases-to-azure),
utoljára megtekintve: 2021.12.05.

[11] Linux szerverek elterjedése,
(https://hostingtribunal.com/blog/linux-statistics/#gref),
utoljára megtekintve: 2021.12.05.

Ábrajegyzék

Táblajegyzék

Mellékletek

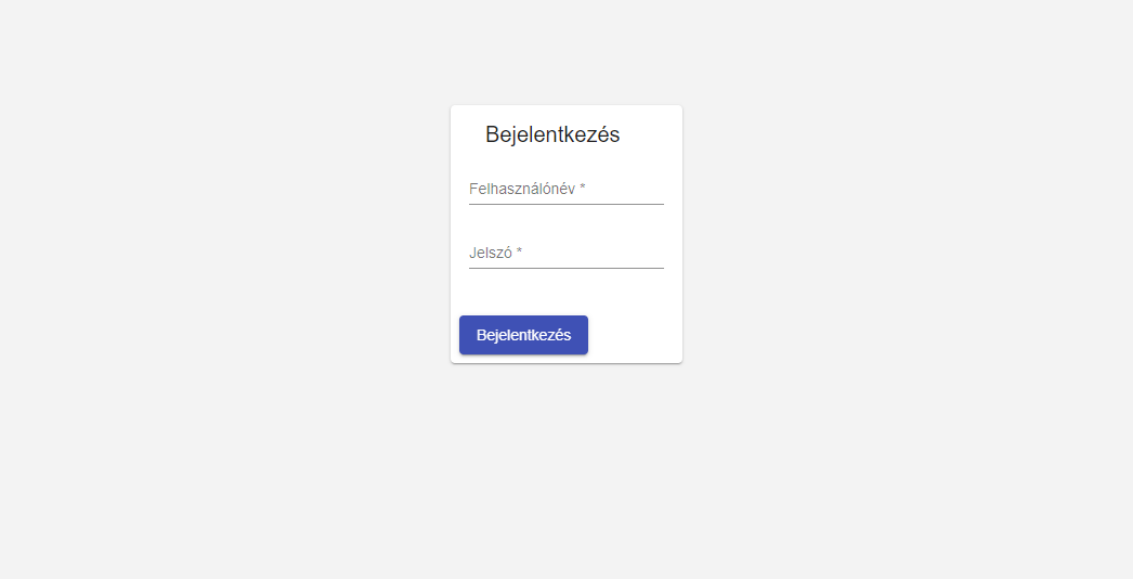
Bejelentkezéshez szükséges információk:

Admin felhasználó:

- Felhasználónév: admin
- Jelszó: verystrongpassw0rd!

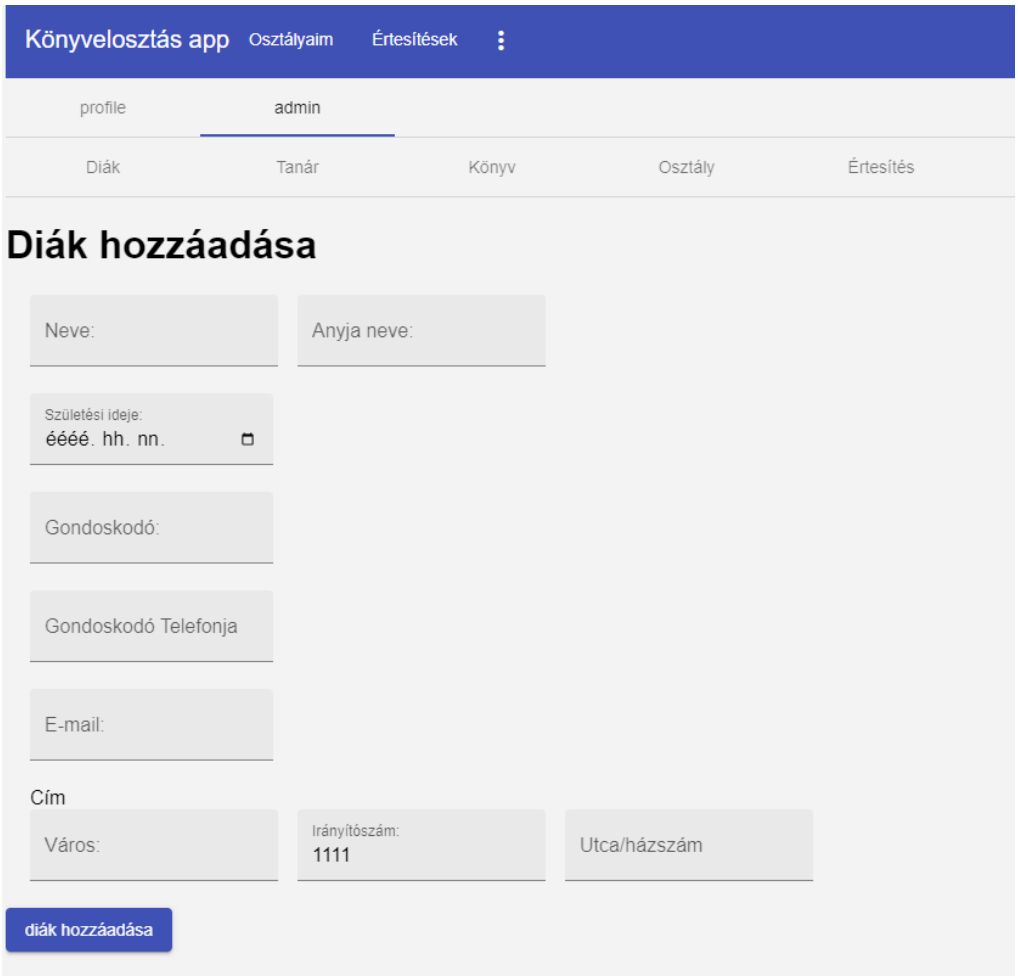
Teszt felhasználó:

- Felhasználónév: testuser
- Jelszó: string

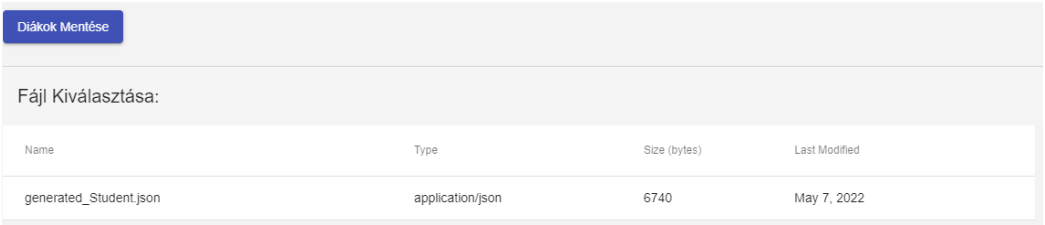


Képen a bejelentkezési felület látszódik, a Bejelentkezés gombra kattintva megjelenik a fő oldal.

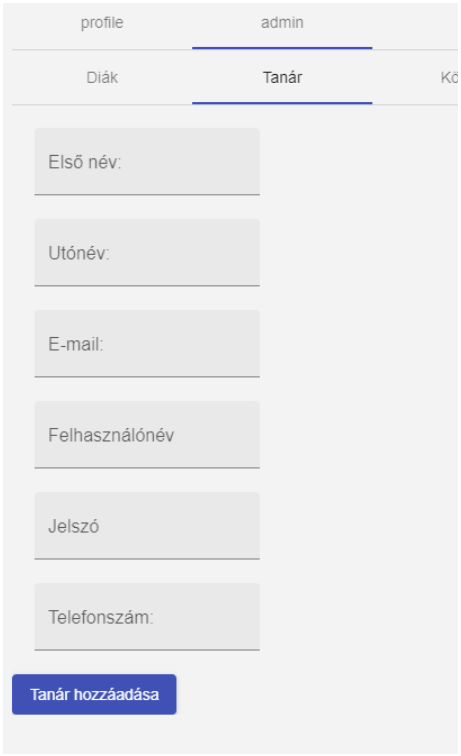
Képen a menüsáv látható. A zöld nyíllal jelzett beállítások gombra kattintva elérhetőek az admin műveletek.



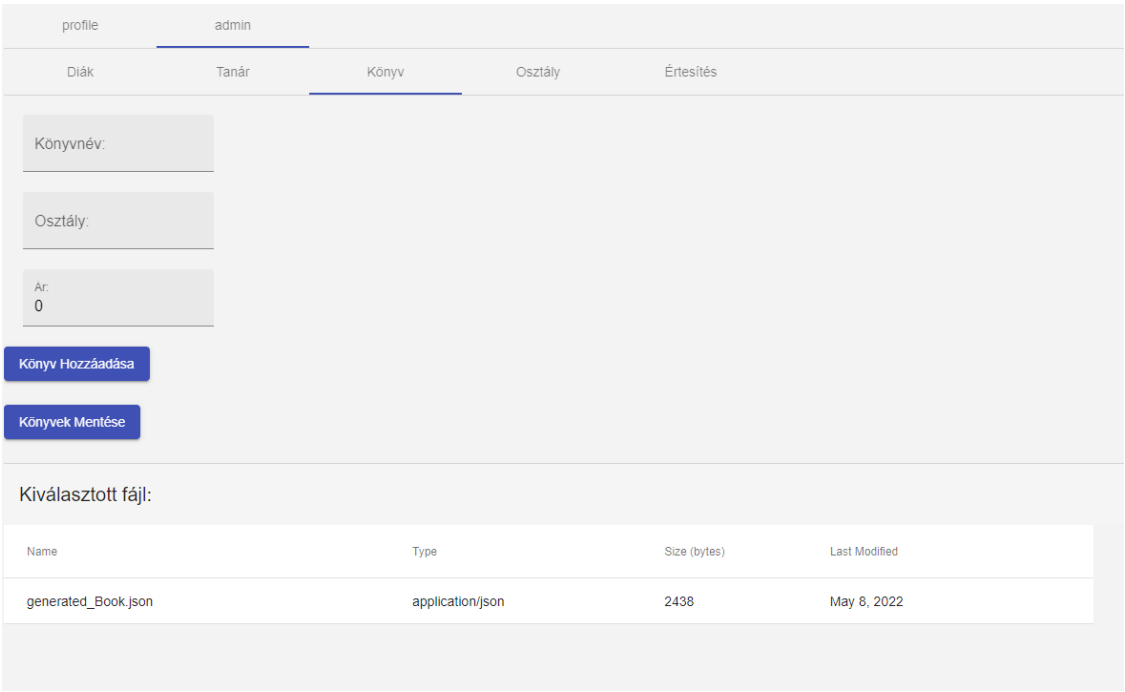
Képen az Admin felület diák hozzáadása látszódik. Itt egyesével van lehetőség hozzáadni a rendszerhez a tanulót.



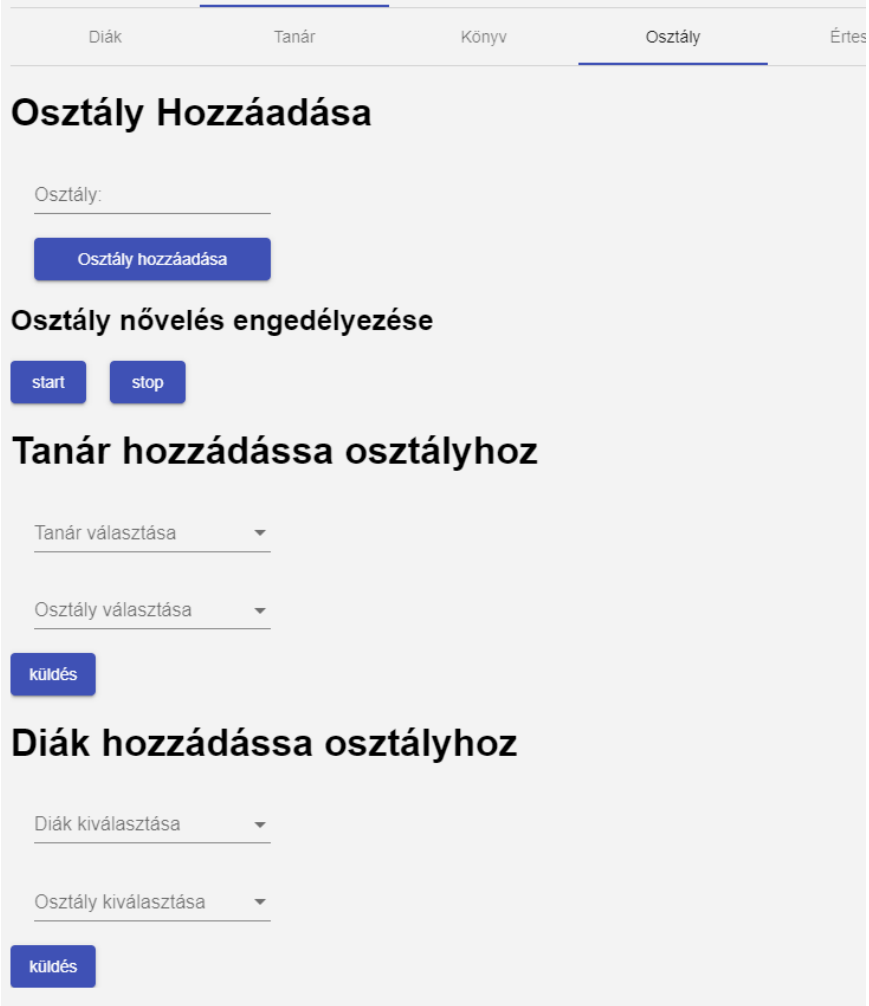
Képen az Admin felület több diák hozzáadása látszódik. Ehhez egy json listát kell feltölteni, amire egy példát `genareted_Student` néven mellékeltem a forráskódok mellett az **Assets** mappában.



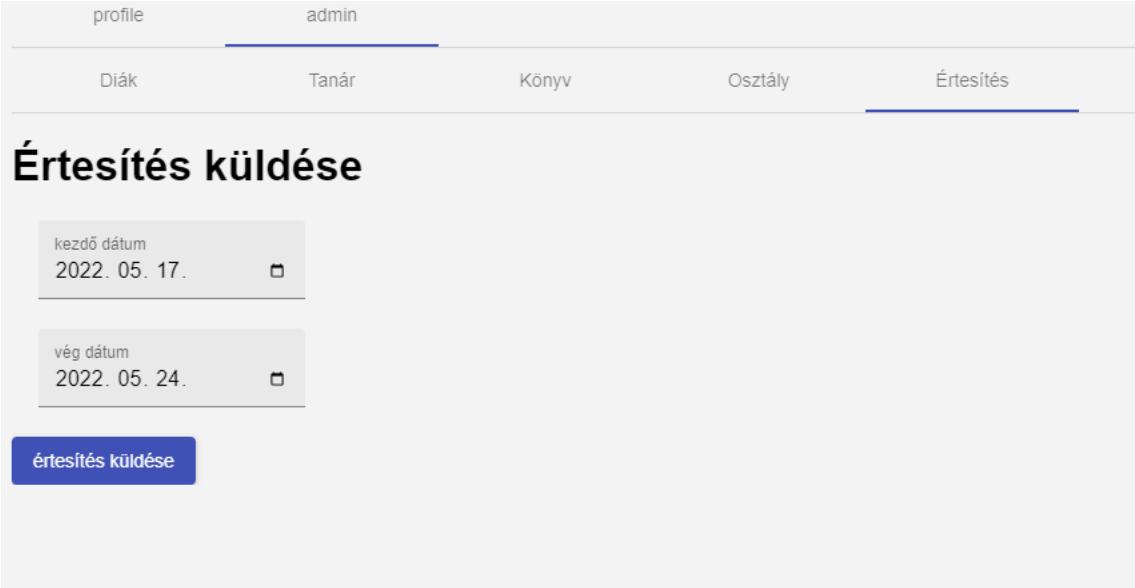
Képen az Admin felület tanár hozzáadása látszódik. Itt egyesével van lehetőség hozzáadni a rendszerhez a tanárokat, akikkel később be lehet lépni.



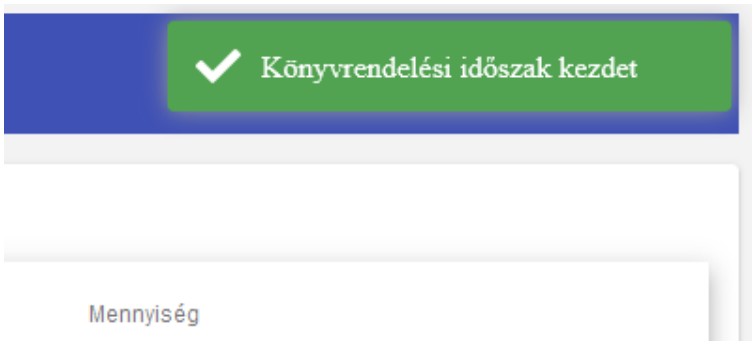
Az alábbi képen az Admin felület könyvek hozzáadása látszódik. Itt egyesével van lehetőség hozzáadni a rendszerhez a könyveket, vagy json formátumban egyszerre többet, amire egy példát `genareted_Book` néven mellékeltem a forráskódok mellett az **Assets** mappában.



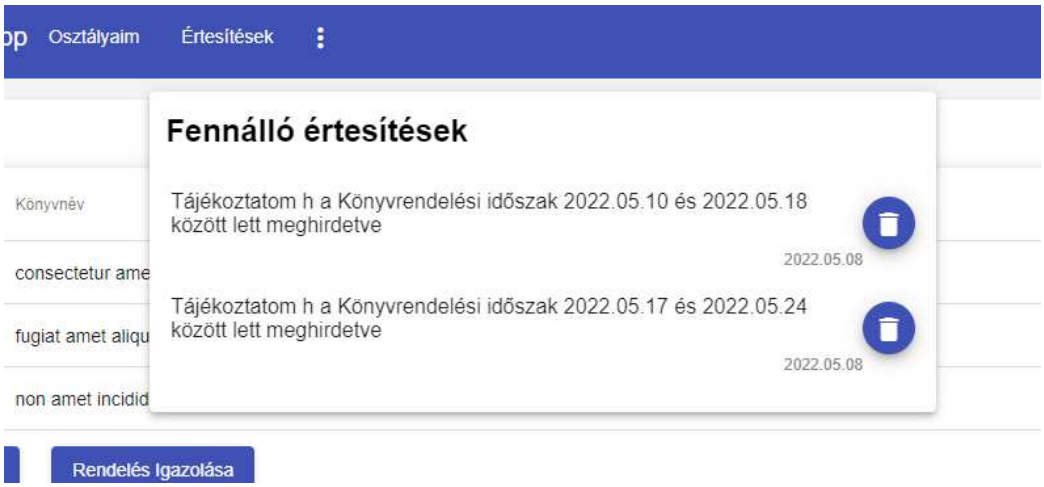
Képen az Admin nézet osztály tab-ja látható. Itt lehetőség van a rendszerhez osztályokat hozzáadni. Alatti lévő részben az éves osztály növelés látható, itt start gombra kattintva elindíthatjuk a folyamatot stop gombra kattintva leállíthatjuk a folyamatot. Az alatta található két form azért felelős, hogy hozzárendelje a tanárokat és a diákokat adott osztályhoz.



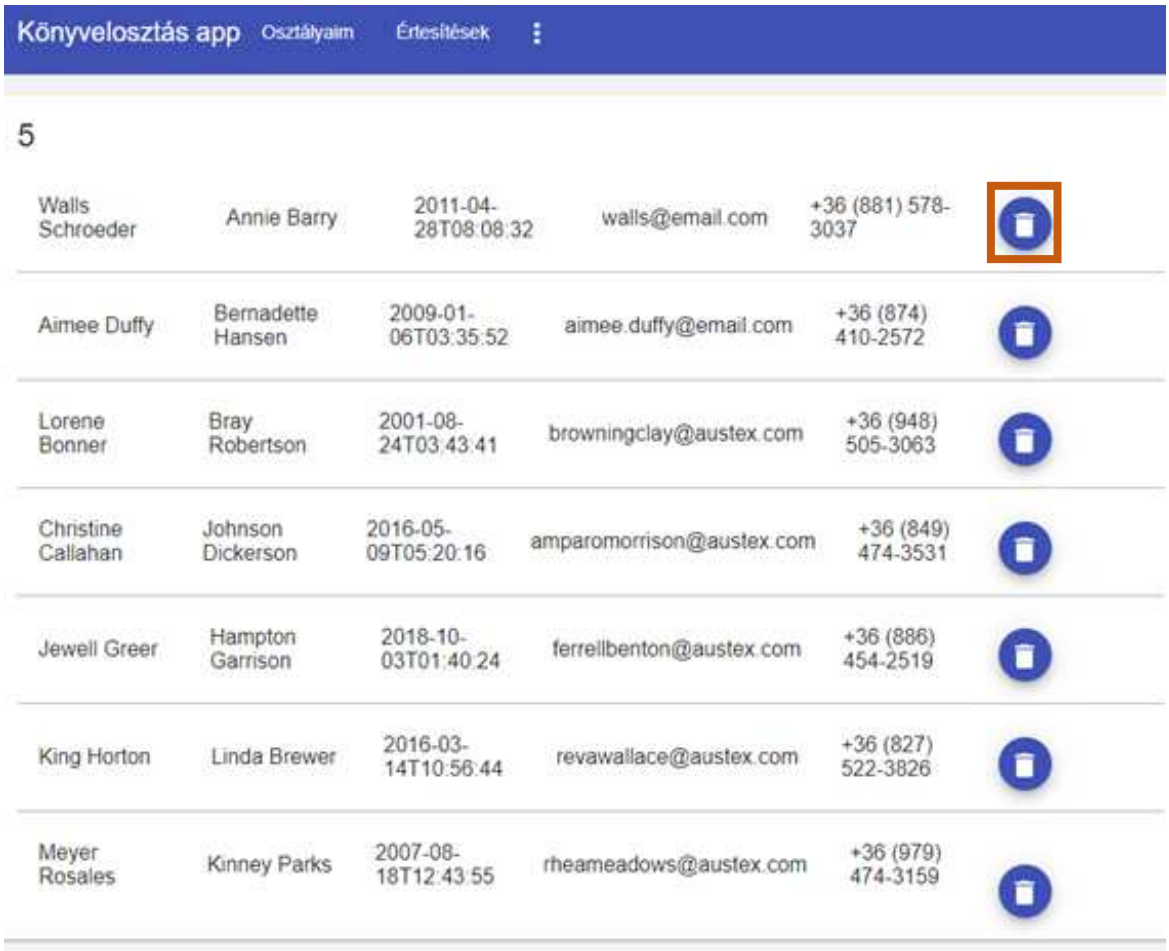
Képen az Admin nézet értesítés tab-ja látható. Itt a form-ot kitöltve értesítéseket lehet kiküldeni a felhasználóknak.



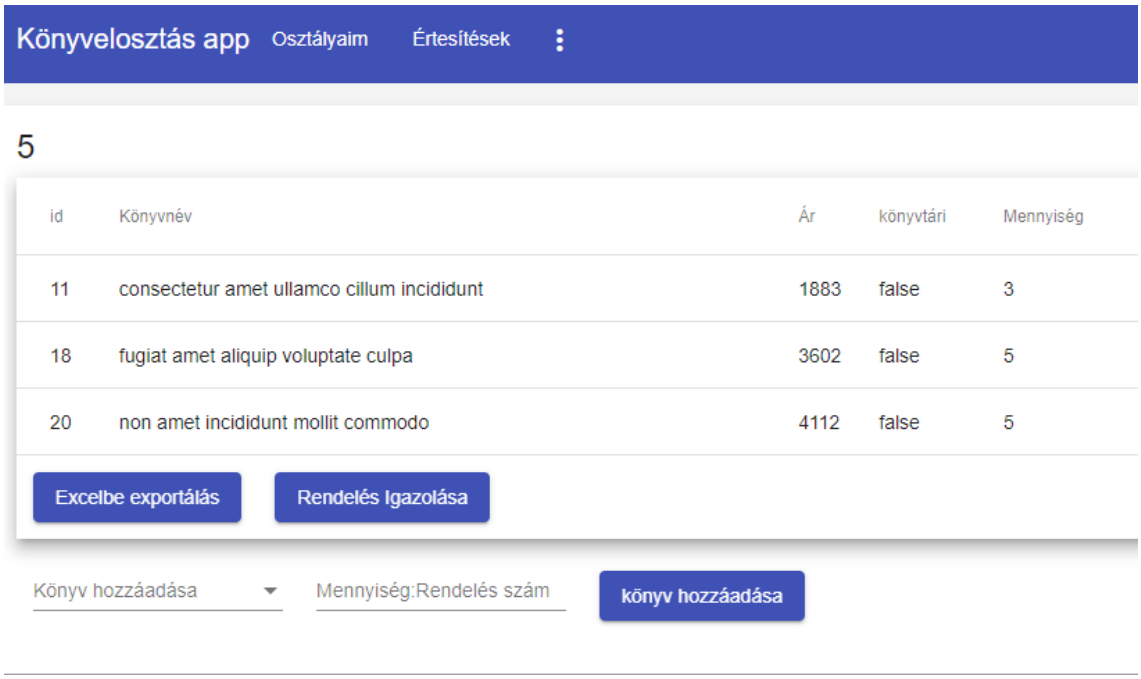
Az alábbi képen a könyvrendelésről látszódik az értesítés, amikor a felhasználó be van jelentkezve.



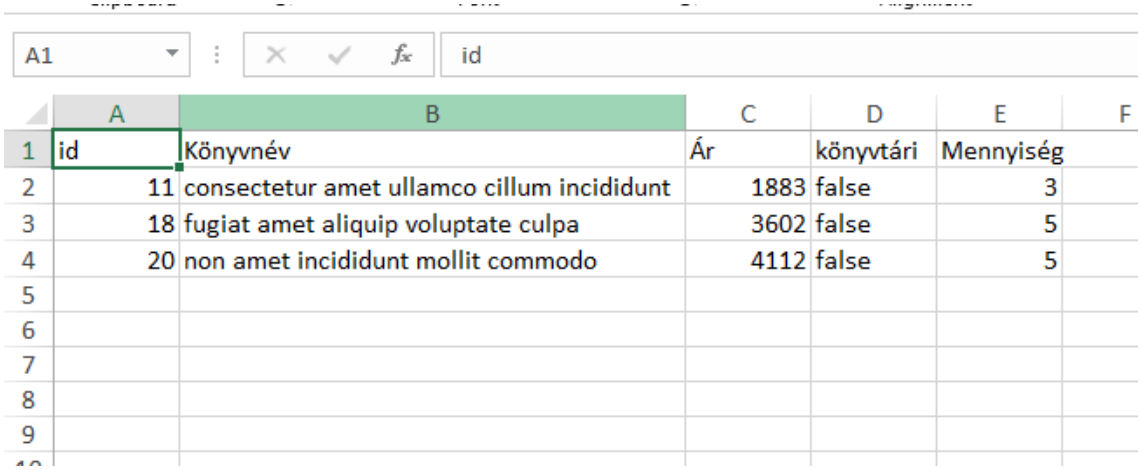
A képen látható, hogy az értesítés továbbá megjelenik az Értesítések fülön tájékoztatva a felhasználót a könyvrendelési időszakról



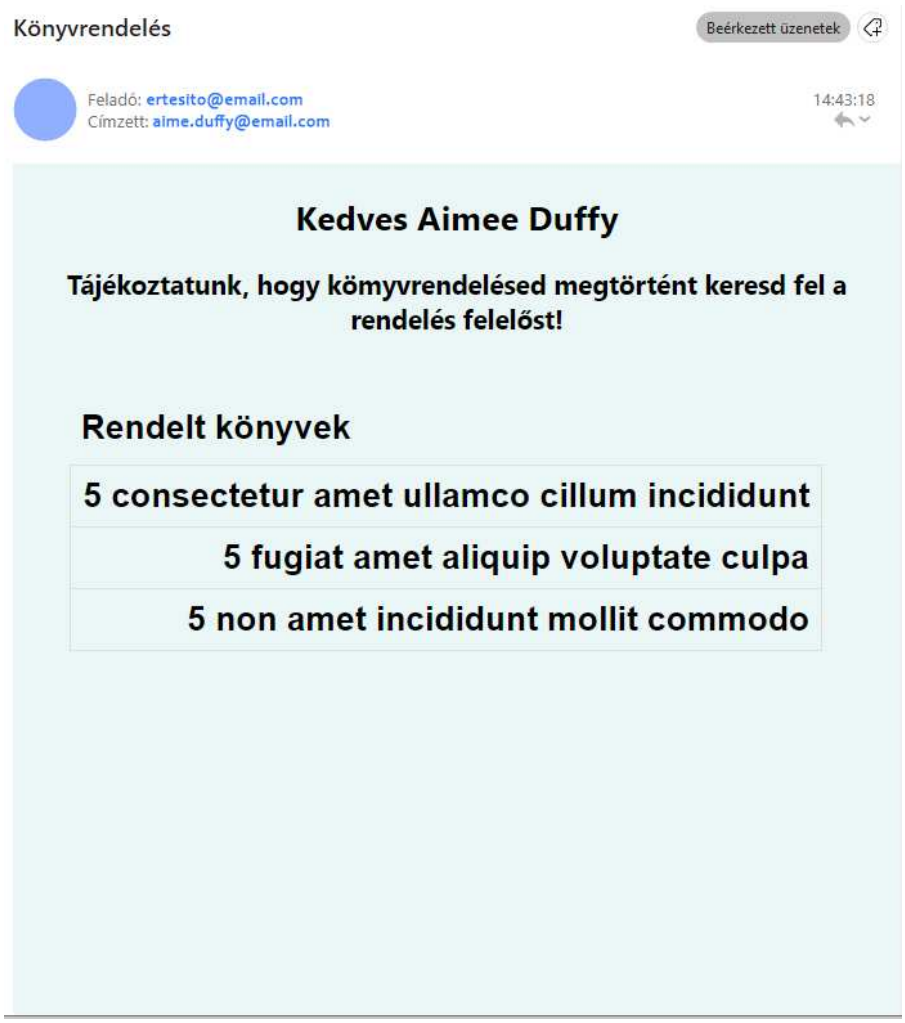
Az alábbi képen az Osztályaim fül alatt felhasználóhoz tartozó osztályok láthatóak. Itt a piros négyzettel jelzet gombra kattintva van lehetősége a felhasználónak a nem hozzá tartozó diákoknak az információit kivenni a listából.



Az alábbi képen a rendelési felület látszódik. Itt lehetősége van a felhasználónak könyveket hozzáadni a rendeléshez. Rendelést XLS fájlba konvertálni, valamint Rendelés igazolása gombra kattintva értesítjük az osztály tanulóit rendelés tartalmáról.



Képen a generált XLS dokumentum látható.



Képen a diáknak a rendelést igazoló e-mail-jét mutatja.