ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

NEUMANN JÁNOS
INFORMATIKAI KAR

SZAKDOLGOZAT

DIPLOMAMUNKA

OE-NIK	Hallgató neve:	**Kertész Gábor Mihály**
2022	Hallgató törzskönyvi száma:	**T/006249/FI12904/N**

Óbudai Egyetem
Neumann János Informatikai Kar
Szoftvertervezés és -fejlesztés Intézet

SZAKDOLGOZAT FELADATLAP

Hallgató neve: **Kertész Gábor Mihály**
Törzskönyvi száma: T/006249/FI12904/N
Neptun kódja: [redacted]

A dolgozat címe:

Webalkalmazás fejlesztése számonkérések támogatásához az informatikai felsőoktatásban
Web Application development for aiding exam taking in computer science related higher education

Intézményi konzulens: Kovács András
Külső konzulens:

Beadási határidő: 2021. december 15.

A záróvizsga tárgyai: Számítógép architektúrák
Szoftvertervezés és –fejlesztés specializáció

A feladat

Ahogy arra a COVID-19 vírus okozta világjárvány is rámutatott, napjainkban az oktatásnak egyre gyakrabban kell támaszkodnia a digitális eszközökre feladatai elvégzéséhez. Az informatikai felsőoktatás ezen belül egy különleges igényekkel rendelkező terület, hiszen belátható, hogy a programozás-orientált feladatok révén a vizsgáztatás hagyományos módszerekkel nehézkes, vagy egyáltalán nem kivitelezhető. A hallgató feladata egy olyan könnyen megtanulható és kezelhető webalkalmazás fejlesztése, mely számos lehetőséget biztosít arra, hogy gördülékenyebbé tegye a számonkérések folyamatát az informatikai felsőoktatás szereplői számára. A rendszerben legyen lehetőség tesztek és beadandó feladatok létrehozására, utóbbi esetén a rendszer legyen képes automatizált javításra (pl. unit tesztek futtatása, kulcsszavak keresése, programkód fordítása, kimenetek összevetése).

A dolgozatnak tartalmaznia kell:

- a szakirodalom részletes áttekintését és értékelését,
- a megoldandó feladat pontos leírását és részletes elemzését,
- a meglévő rendszerek elemzését,
- az alkalmazni kívánt technológiák elemzését, döntések indoklását,
- a rendszertervet,
- a megvalósítás pontos leírását,
- a tesztelési tervet, teszteredményeket, a fejlesztés során felmerült problémák elemzését,
- az elkészült rendszer felhasználási és továbbfejlesztési lehetőségeit.

ÓBUDAI EGYETEM * Neumann János Informatikai Kar * 1 Budapest

PH.

.....
Dr. Vámossy Zoltán
intézetigazgató

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

A dolgozatot beadásra alkalmasnak tartom:

.....
külső konzulens

.....
intézményi konzulens

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

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. 08

hallgató aláírása

KONZULTÁCIÓS NAPLÓ

Hallgató neve:	Neptun kód:	Tagozat:
Kertész Gábor Mihály	[redacted]	NAPPALI

Telefon: Levelezési cím (pl: lakcím):

[redacted]

Szakdolgozat címe magyarul:

Webalkalmazás fejlesztése számonkérések támogatásához az informatikai felsőoktatásban

Szakdolgozat címe angolul:

Web Application development for aiding exam taking in computer science related higher education

Intézményi konzulens: Külső konzulens:

Kovács András

Kérjük, hogy az adatokat nyomtatott nagybetűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2021. 02. 20	Tématervezés	
2.	2021. 03. 10	Irodalmi források ellenőrzése	
3.	2021. 04. 18	Tervezés átbeszélése	
4.	2021. 05. 10	Prezentációs próba	

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

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

Budapest, 2021. Május 14.

intézményi konzulens

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

KONZULTÁCIÓS NAPLÓ

Hallgató neve:	Neptun Kód:	Tagozat:
KERTÉSZ GÁBOR MIHÁLY	[redacted]	NAPPALI

Telefon: [redacted]

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

Szakdolgozat címe magyarul:

WEBALKALMAZÁS FEJLESZTÉSE SZÁMONKÉRÉSEK TÁMOGATÁSÁHOZ AZ INFORMATIKAI FELSŐOKTATÁSBAN

Szakdolgozat címe angolul:

WEB APPLICATION DEVELOPMENT FOR AIDING EXAM TAKING IN COMPUTER SCIENCE RELATED HIGHER EDUCATION

Intézményi konzulens: KOVÁCS ANDRÁS

Külső konzulens:

Kérjük, hogy az adatokat nyomtatott nagy betűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2022.02.15.	IMPLEMENTÁCIÓ TERVEZÉS	
2.	2022.03.25.	IMPLEMENTÁCIÓ ÁTTEKINTÉSE	
3.	2022.04.10.	TESZTELÉSEK KIÉRTÉKELÉSE	
4.	2022.05.10.	VÉGSŐ ELLENŐRZÉS, LEZÁRÁS	

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. május 10.

TARTALOM

1. BEVEZETÉS

A mai modernizálódó világban a technológia térnyerése szinte minden területen megfékezhetetlennek látszik. Ez a trend abszolút uralkodóvá vált a kereskedelemben, a kommunikációban, a médiában, és egyéb, a mindennapjainkat meghatározó területeken. Érdemes azonban észrevennünk, hogy van egy bizonyos szektor, mely látszólagos ellenállással viseltetik ezekkel a digitalizációs folyamatokkal szemben, ez pedig nem más, mint az oktatás területe. [1]

Az oktatás az alap-, közép-, és felsőfokú intézmények szintjén egyaránt egészen a mai napig a klasszikus nevelési metodika mentén szerveződik: az új ismeretanyagok átadásának módja, hogy a tanár egy térben és időben helyezkedik el a diákjaival, órát tart. Ennek során magyaráz, a tantervben szereplő tananyagot retorikai- és egyéb segédeszközök (tábla, kivetítő) segítségével mutatja be. Egy-egy téma lezárásaként, a diákok által sikeresen elsajátított ismeretanyag mértékét felmérendő számonkérést szervez: ez jellemzően dolgozat, beadandó feladat, vagy felsőoktatás esetén zárthelyi dolgozatok, kollokviumok formájában történik. Ez a visszakérdezés az esetek nagyon jelentős részében papíralapúan történik. A további tanári teendők innentől fogva egyértelműek, függetlenül attól, hogy a számonkérés kompetencia-alapú, vagy egzakt ismeretekre építő jellegű volt: ki kell értékelni a válaszokat, meg kell határozni az eredményeket.

Érdemes az imént ismertetett procedúrát végigkövetve analógiát vonnunk más területek munkafolyamataival. A fentiek alapján például az orvosi területet úgy képzelhetjük el, mintha még mindig így zajlana a munkavégzés: új beteg érkezése esetén különböző papíralapú dokumentumokon eltárolják annak adatait, elmondása alapján a medikai múltját, majd ezeket a papírokat a kezelés végeztével egy karton mélyére süllyesztik. Természetesen érezhető, hogy ez egy nagyon korszerűtlen hozzáállás, azonban az oktatást digitális szemszögből vizsgálva éppen ezt az archaikus jelleget figyelhetjük meg.

A problémás területet alapvetően két részre bonthatjuk:

- Ismeretanyagok átadása, klasszikus értelemben vett oktatás
- Számonkérés

Ezek közül az előbbit modernizálni hivatott oktatási platformok száma folyamatosan nő, viszont a számonkérések digitalizálására létrehozott megoldások tárháza viszonylag szegényes, mondhatjuk, hogy a terület gyerekcipőben jár. A kutatómunkám ezt hivatott orvosolni egy, a számonkérésekre, és azok kiértékelésére specializálódott oktatástámogató szoftver fejlesztésének segítségével. [2]

2. A SZAKDOLGOZAT SORÁN MEGVALÓSÍTANDÓ RENDSZER

A következőkben szeretném felvázolni, hogy milyen probléma az, amire megoldást szeretnék adni a szakdolgozatommal, ezután pedig olyan rendszereket szeretnék bemutatni, melyek működésük, feladatkörük szempontjából valamelyes kapcsolódnak a digitális számonkérések világához.

2.1. A PROBLÉMA FELVETÉSE

A technológiai fejlődést teljesen természetes folyamatok idézték elő a legtöbb területen. Amikor egy téma szakértői észrevették, hogy vannak bizonyos munkafolyamatok, melyek könnyedén automatizálhatók, ekkor informatikusok bevonásával olyan megoldásokat hoztak létre, melyek ezeket az automatizációkat megoldják. Könnyű észrevenni, hogy a mi problémakörünkben is rengeteg ilyen monoton, tanári intuíciót abszolút nélkülöző feladat van.

A hagyományos módszerekkel összeállított tesztek létrehozása körülményes feladat egy oktató számára, hiszen jellemzően valamilyen szövegszerkesztő program segítségével kell mindezt megtennie. Akár elektronikus, akár nyomtatott formában szeretné a diákjaihoz eljuttatni a teszteket, szükséges, hogy biztosítson megfelelő mennyiségű helyet a kitöltéshez. Ez teljesen pazarló megközelítés, egy digitális felületen a kérdéshez tartozó válaszlehetőség kialakítása dinamikusan történik, és nem igényli a tanár plusz munkáját.

Hasznos lenne tehát egy olyan felület, ahol a tanárok kérdéssorokat tudnának összeállítani, azonban törekednünk kell rá, hogy ne érezzék korlátozva magukat akkor, amikor ezt az alternatív utat választják. Lehetőséget kell tehát biztosítani minden klasszikus számonkérési mód megadására ezen a felületen. Talán az egyik legnagyobb probléma az ilyen rendszerek esetén a kérdések feltevése során megmutatkozó diverzitás hiánya, a bezártságérzet, a korlátolt számú opció.

Vannak olyan esetek, amikor az oktatási infrastruktúra egyszerűen nem teszi lehetővé, hogy egy digitális formájú tesztet írassanak a diákokkal az oktatók. Ekkor felmerülhet az igénye egy olyan funkciónak, melynek segítségével arra lenne lehetősége a tanároknak, hogy a már elkészített teszteket kinyomtatva tegyék diákok elé, így felgyorsítva a munkafolyamatot.

Problémát jelenthet továbbá a tesztek, lapok helyes kiértékelése. Rengeteg olyan kurzus létezik a közép- és felsőoktatásban egyaránt, ahol az egzakt válaszok kiértékelése nem feltétlen azok tartalmi összetétele miatt válik nehézkessé egy számítógépes program számára, hanem éppen a válaszok formulázása során fellépő ambiguitás jelent gondot. El tudjuk képzelni például azokat a helyzeteket, ahol a kiértékelés „hibás” eredménnyel jelölt meg egy olyan kérdést, melyre a diák által adott válasz csupán azért nem egyezik

az elvárttal, mert esetleg a helyes megfejtés egy szinonimáját adta meg, vagy éppen egy rövidített idegennyelvi alakot használt.

2.2. HASONLÓ RENDSZEREK KERESÉSE

A következőkben olyan rendszereket szeretnék bemutatni és elemezni, melyek az előző pontban felvetett problémák legalább egy részére megoldást nyújtanak. Az alábbi elearning, tehát valamilyen formában az online, illetve számítógépes oktatást lehetővé tevő rendszerek egyaránt széleskörű funkcionalitással bírnak, azonban szeretném megmutatni, hogy miért nem fedik le teljeskörűen az általam felvázolt problémakör problémáit.

2.2.1. MOODLE

A moodle rendszerének bemutatása

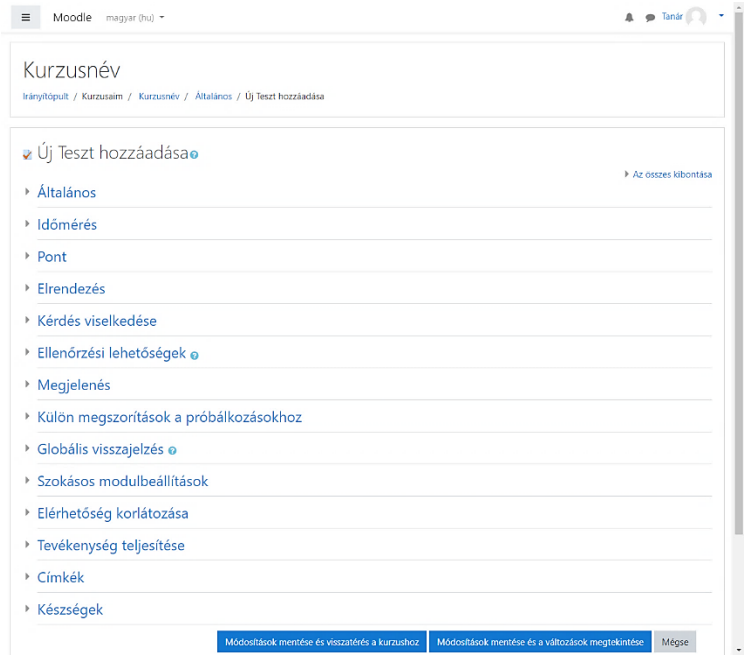
Az első kiválasztott rendszer a Moodle. Ez egy nyílt forráskódú, ingyenes licenc alatt terjesztett, PHP nyelven íródott elearning keretrendszer. Nagyon széleskörű funkcionalitással bír, gyakorlatilag lehetőség van a rendszerben arra, hogy egy komplett oktatási kurzust levezényeljünk. Ennek természetesen része a számonkérés is, én éppen az ezt lehetővé tevő **tesztmodulra** fogok fókuszálni a következőkben. [3]

Számonkérés létrehozására lehetősége van az arra jogosult, szerkesztő tanároknak a kurzus felületén. Ez a kurzusoldal tetején található beállítások ikonra való kattintás, majd a *Szerkesztés bekapcsolása* menüpont kiválasztása után válik elérhetővé. Ekkor a kurzuson belüli megfelelő helyen a *Tevékenység vagy tananyag beszúrása* feliratra kattintunk, majd a listából a *Teszt* opciót választva érkezünk meg a számonkérés szerkesztésének felületére. Ez az út talán kicsit bonyolultnak, feleslegesen hosszúnak tűnhet, ez éppen abból fakad, hogy a Moodle egy komplex rendszer, rengeteg lehetőséggel, ahol ráadásul nem a számonkérésekre fektették a hangsúlyt a fejlesztés során.

Mire is képes ez a felület? A Moodle saját segítségének tanúsága szerint a tesztek alkalmasak:

- Kurzusvizsgához
- Szövegértési feladatokhoz vagy téma összegzéshez
- Gyakorló tesztként korábbi vizsgák anyaga alapján
- Azonnali ismeret-ellenőrzéshez
- Önellenőrzésre

Emellett megtudhatjuk, hogy az egyes próbálkozások - az esszékérdések kivételével - automatikusan kiértékelésre kerülnek, az eredmény pedig bekerül az osztályozónaplóba. [4]



2.1 ábra: A Moodle tesztszerkesztő felülete

Tekintsük át, hogy milyen lehetőségeink vannak a teszt létrehozása során. Ehhez a Moodle szerkesztő felülete használható, mely a 2.1 ábrán is szerepel. Az olyan trivialitásokon túl, mint a teszt neve, leírása, megadhatunk például időkeretet, melynél különböző lehetőségeink vannak annak szabályozásra, hogy mi történjen, ha kifut a diák az időből:

- Elvesszük a vezérlést tőle, a leadás automatikusan megtörténik
- Türelmi időt adunk, a megnyitott kérdés még befejezhető, de továbbiak megválaszolására nincs mód
- A próbálkozásokat már a határidő előtt le kell adni, különben figyelmen kívül lesznek hagyva

Lehetőségünk van továbbá a próbálkozások számát beállítani. Az alapbeállítás nagyon optimista hozzáállást mutat, talán inkább az önellenőrző jellegű tesztekre van berendezkedve, és kevésbé a számonkérésekre, hiszen korlátlan számú kitöltést biztosít, valamint a kiértékelést is úgy határozza meg, hogy a legmagasabb pontszámú próbálkozást számítja. Ezek természetesen állíthatók, de talán nekünk erre nem is lenne szükségünk.

A kérdéseken belüli válaszlehetőségek összekeverése is egy hasznos beállítás lehet, ez a klasszikus másolás problémáját oldja meg: a diákoknak nem is kell olyan közelségben lenni a társukhoz, hogy el tudják olvasni a válaszlehetőségeket, elég csupán, ha látják,

hogy hányadik opciót jelölte társuk. Ha összekeverjük a válaszokat, ez természetesen jelentős mértékben visszaszorítható. Arra is van lehetőségünk, hogy a kérdések feltevésének sorrendje is véletlenszerűen történjen.

Ellenőrzési lehetőségeket tekintve azt mondhatjuk, hogy a rendszer itt is sokrétű lehetőséget biztosít, hiszen bekapcsolhatjuk például, hogy ne csak a kérdéssor kitöltése után kapjanak visszajelzést a diákok, hanem már az egyes kérdések után is láthassák, helyesen válaszoltak-e.

Egyéb megszorítási lehetőségek között szerepel, hogy beállíthatjuk, milyen hálózati címekről tölthető ki a kérdéssor. Ez a – elsőre igen marginálisnak ható –lehetőség kifejezetten hasznos lehet például egy vizsga íratásakor, így biztosítva, hogy biztosan minden résztvevő egy teremben, esetleg alhálózaton tartózkodik. A jelszóval való védés kissé súlytalannak tűnhet az előbbi opció fényében, de ilyenre is van lehetőség. Ezek megfontolandó lehetőségek, lévén a biztonsági szempontok igen fontosak, ha például a felsőoktatásban való felhasználásra gondolunk. Emellett az implementálásuk is viszonylag egyszerűnek tekinthető.

A mi szempontunkból talán az egyik legjelentősebb funkció a kérdések kezelése. A Moodle rendszere ebben is rendkívül flexibilis, a leggyakoribb kérdéstípusok közül szinte mindent megtalálunk a lehetőségek között:

- Feleletválasztós
- Igaz – hamis (egyszerű feleletválasztós)
- Párosítás
- Kiegészítendő kérdés (szövegdobozzal vagy lenyíló menüből)
- Egy számjegyes (véletlenszerű számokkal is)
- Több számjegyes
- Esszé
- Beépített válaszos (speciális kóddal hozható létre)
- Egyszerű számításos
- Elhúzás szövegbe
- Elhúzás képre
- Véletlenszerűen kiválasztott kiegészítős kérdések párosítással

A kérdéssor formázását is beállíthatjuk, viszonylag gyakori megközelítés az oktatók részéről, hogy egy oldalra egy kérdés kerül, így nehezebben terelődik el a diák figyelme. Látható, hogy rendkívül széles a lehetőségek tárháza, egyszerűbb és komplikáltabb kérdéssor összeállítható segítségükkel. Ami mindenképpen pozitívumként említhető, hogy a rendszer támogatja a tudományterület-specifikus beépülő modulok hozzáadását is. A kérdések hozzáadása - a korábbiakhoz hasonlóan – viszonylag körülményesen zajlik, minden létrehozott kérdés után vissza kell lépnünk a teszt oldalára, majd itt új kérdést beszúrni. Ez azért nem a legszerencsésebb tervezés, mivel a leggyakoribb

használati eset az, hogy egy kérdés után egy következőt is be szeretnénk szúrni, nem pedig visszalépnénk a kvíz főoldalára.

A rendszerbe már betöltött kérdések szigorúan egy felhasználóhoz való hozzárendelése rendkívül pazarló megközelítés volna. Teljesen életszerű, hogy az egyik oktató által már eltárolt kérdéseket egy másik oktató saját oktatási célra szeretné felhasználni. Ez a megosztás lehetséges a Moodle-ben, de nem alapértelmezett beállítás, és a konfigurációja, valamint a használata is igen körülményes. [5]

A tesztek kiértékelése automatikusan történik (az esszékérdéseket kivéve, itt elvárja a rendszer a tanári intuíciót), egy egyszerű összegzést kapunk az eredményekről. Ezt a 2.2 ábra mutatja be: [6]

		First name / Surname	Email address	State	Started on	Completed	Time taken	Grade/100.00	Q. 1 /11.11	Q. 2 /11.11	Q. 3 /11.11	Q. 4 /11.11	Q. 5 /11.11	Q. 6 /22.22	Q. 7 /11.11	Q. 8 /11.11
☐		Jerry Hart Review attempt	jerryhart159@example.com	Finished	17 October 2012 9:52 PM	17 October 2012 9:53 PM	48 secs	44.44	✗ 0.00	✓ 11.11	✓ 11.11	✗ 0.00	✗ 0.00	✓ 11.11	✗ 0.00	✓ 11.11
☐		Carol Warren Review attempt	carolwarren213@example.com	Finished	17 October 2012 9:54 PM	17 October 2012 9:54 PM	42 secs	51.85	✓ 11.11	✓ 11.11	✓ 7.41	✓ 11.11	✗ 0.00	✗ 0.00	✗ 0.00	✓ 11.11
☐		David Stewart Review attempt	davidstewart157@example.com	Finished	17 October 2012 9:55 PM	17 October 2012 9:56 PM	46 secs	70.37	✓ 11.11	✓ 11.11	✓ 3.70	✗ 0.00	✓ 11.11	✓ 11.11	✓ 11.11	✓ 11.11
		Overall average						55.56 (3)	7.41 (3)	11.11 (3)	7.41 (3)	3.70 (3)	3.70 (3)	7.41 (3)	3.70 (3)	11.11 (3)

2.2 ábra: Moodle rendszerben kiértékelt teszt eredménye

A Moodle rendszerének értékelése

Mint láthattuk, a Moodle egy rendkívül komplex rendszer, mely nem elsősorban fókuszál a számonkérésekre, csupán biztosít ilyen lehetőséget is.

A keretrendszer tesztrendszerének gyengeségei talán éppen ebből az összetettségből fakadnak. A felület használata sok esetben nehézkes, rengeteg almenün keresztül vezet el az út egy-egy funkcióhoz. Az egzakt válaszok ellenőrzése előre megadott minták alapján történhet csak, bár legalább több válaszlehetőség, valamint a kis- és nagybetűk megkülönböztetésének beállítására van lehetőségünk. Mivel a rendszer egyben kezeli a különböző kérdésfajtákat, így egy csak igaz-hamis alapú kérdéssor exportálására semmilyen formában sincs lehetőségünk.

Pozitívumként említhetjük, hogy a rendszer, lévén régóta a piacon van, nagyon stabil, determinisztikus működést produkál. Kérdések feltevésére rengeteg formában lehetőségünk van. Nem elhanyagolható a biztonságosság szempontja sem: a Moodle nagyon jól teljesít ebből a szemszögből nézve az egyszerű, de hatékony beállítási opcióival. Az értékelő rendszer is teljesértékű, jól használható statisztikákat, kimutatásokat is kaphatunk a diákok teljesítményéről. A diákok irányába tett visszajelzések módja is jól konfigurálható. A kérdésbank - bár alapértelmezetten mindig az aktuális kurzusra vonatkoztatott hatáskörrel működik csak – egy kiváló ötlet, ennek

egy könnyebben átlátható változata talán kijelenthető, hogy alapkövetelmény bármilyen hasonló célra rendeltetett, modern rendszerben.

Összességében elmondható, hogy a rendszer több pozitív tulajdonsággal bír, mint negatívval. Mindenképpen vannak jó megoldásai a Moodle-nek, melyek beépítését érdemes lenne megfontolni a saját rendszerünk esetén is.

A fentiek összefoglalásaként tekintsük át, hogyan értékelhető a Moodle néhány egyszerű szempont alapján:

Kezelhetőség	Bonyolult, sokszor nehézkes, lassabb munkavégzést eredményez
Bővíthetőség, testreszabhatóság	Jó, támogatja tudományterület-specifikus beépülő modulok telepítését
Ár	Ingyenes
Lehetőségek diverzitása	Rengeteg kérdésfajta, igen széleskörű felhasználhatóság
Legjobb egyedi funkció	Lehetőség IP-cím alapján történő szűrésre a tesztek kitöltésekor
Dizájn, vizualitás	Egységes, letisztult, de a sok funkció miatt helyenként átláthatatlan

2.1 táblázat: A Moodle értékelése különböző szempontok alapján

2.2.2. REDMENTA

A Redmenta rendszerének bemutatása

A Redmenta egy kifejezetten feladatlap-készítésre specializálódott alkalmazás. Lévén, hogy ingyenes, és magyar fejlesztésű rendszer, nagy népszerűségnek örvend hazánkban. A fejlesztők saját bevallása szerint a motivációt az alkalmazás elkészítéséhez az adta, hogy egy, a nagyobb, bonyolult keretrendszereknél egyszerűbb megoldással szolgáljanak. Azt, hogy mennyire jól használható a Redmenta, és milyen megoldásokkal operál, az alábbiakban részletezem. [7]

Mint korábban említettem, a Redmenta szolgáltatásai ingyenesen igénybe vehetők, ehhez csupán egy regisztrációra van szükségünk. A regisztrációs, valamint a tesztlapok áttekintésére szolgáló felület egyaránt jól átlátható, könnyen kezelhető. Belépés után az *Új feladatlap létrehozása* gombra kattintással van lehetőségünk tesztet összeállítani. Az

ezt követő felületen a már létrehozott kérdések megtekintése, valamint újak hozzáadása mellett a tesztlap alapbeállításait is van lehetőségünk módosítani.

A kérdések létrehozása, valamint a teszthez hozzáadása egy erre dedikált felületen történik. Ez talán egy kevésbé tájékozott felhasználó számára túlzsúfolt hatást kelthet a rengeteg funkciójával, azonban ennek a tervezésnek éppen ez az előnye: egyszerre láthatjuk a már betöltött kérdéseket, hozhatunk létre újakat, vagy szúrhatunk be meglévőket a globális kérdésbankból.

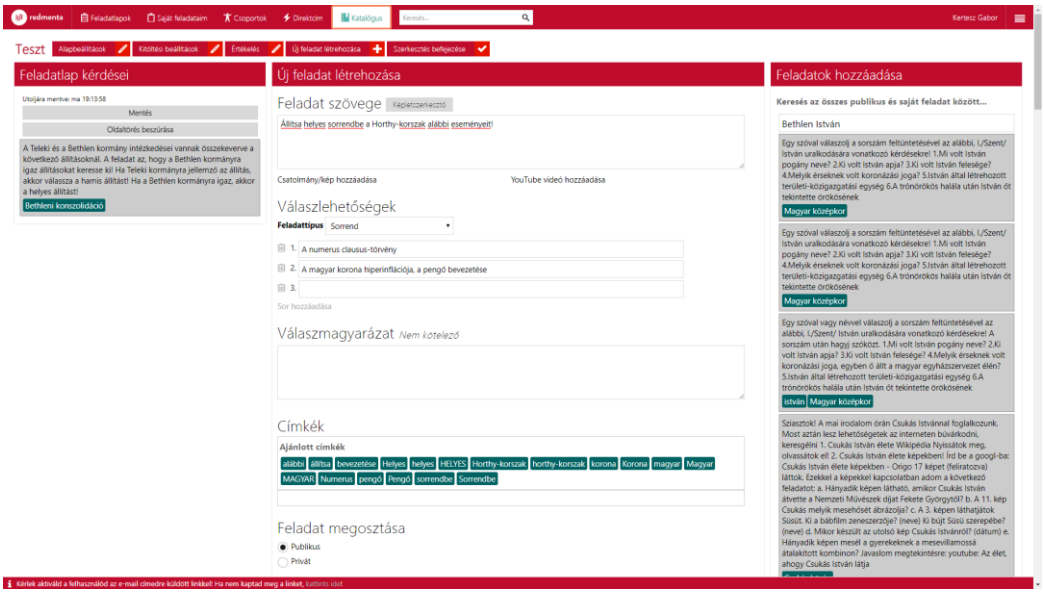
Ami a saját feladatok létrehozását illeti, azt mondhatjuk, hogy a Redmenta megtartja az eddig látott egyszerű megközelítést. A feladat szövegébe az alapvető multimédiás elemek mellett YouTube videót is beágyazhatunk, valamint hatalmas pozitívum, hogy a szövegdobozt felszerelték egy *Képletszerkesztő* funkcióval is, mely a természettudományos jellegű kérdések feltevését egyszerűsíti le jelentősen.

Válaszlehetőségek terén a rendszer a lehető legalapvetőbbekre szorítkozik. A következő lehetőségek biztosítottak:

- Feleletválasztós (egy jó válasszal)
- Feleletválasztós (több jó válasszal)
- Igaz – Hamis
- Kifejtős kérdés
- Rövid válasz (jellemzően egyszavas)
- Párosítás
- Sorrendbe helyezés

A válaszlehetőségek bekérése egyszerű HTML formos formában van megoldva, így jól átlátható. Lehetőségünk van válaszmagyarázatot is megadni a válaszok után.

Külön kiemelendő a Redmenta **címkerendszere**. Minden kérdés létrehozásakor lehetőségünk van megadni a kérdés kapcsán releváns címkéket, melyek – amennyiben a feladatot publikusan közzétesszük – segítségül szolgálnak a kérdés tematikájának azonosításában. A rendszer nagyon egyszerű módon tesz automatikus ajánlásokat is, ez gyakorlatilag abban merül ki, hogy a kérdés leírásában és a válaszlehetőségekben megtalálható szavakat különböző formában felsorolja a motor. Nincs tehát semmilyen kontextus-érzékeny elemzés, de ha kézzel megfelelően beállítjuk a paraméterként szolgáló címkéket, akkor ezek után mindenki számára könnyen kereshető, és akár használható is a feladatunk.



2.3 ábra: A Redmenta szerkesztői felülete

A 2.3-as ábrán látható a korábban tárgyalt szerkesztői felület. Megfigyelhető, hogy a paneles, oszlopokba szervezett, modern megjelenítés jól használható felületet eredményez, éppen a megfelelő komplexitást nyújtva.

A tesztek kiértékelése teljesen automatikusan zajlik, kivéve a kifejtős válaszokat.

A biztonságosság szempontjából alapvető beállítási lehetőségeink vannak. Különböző láthatósági szintekkel szabályozhatjuk a hozzáférést. A kategóriák a következőképpen lettek kialakítva a Redmentában:

- Privát
- Csak azoknak, akiknek a Redmenta felületén meg lett osztva
- Bárkinek, aki regisztrált és ismeri a direktcímet[1]
- Bárkinek, aki regisztrált felhasználó
- Bárkinek a világon

A redmenta rendszerének értékelése

A Redmenta használata során valóban érezhető a fejlesztők részéről tett erőfeszítés arra, hogy egy egyszerűbb, átláthatóbb rendszert alkossanak, azonban ez helyenként a funkcionalitás rovására megy. A feltehető kérdések fajtái egy kissé korlátozottak, könnyen elképzelhető, hogy egy tanár szimplán azért nem ezt a platformot választja, mivel azt érzi, hogy a diverzitás hiánya miatt nem tudna teljesértékű számonkérést

[1]A direktcím a Redmenta rendszerében a tesztek egyedi azonosítására alkalmas karakterlánc, melyet mi magunk alakíthatunk ki. A tesztek elérése is ezen keresztül történik, a konkrét érték egy URL paraméterként helyettesítődik be a megosztható linkbe.

összeállítani. Emellett nincs lehetőségünk a kérdéssor exportálására sem, tehát csak online tölthetőek ki a tesztek, ez szűkíti a felhasználhatóság körét.

Vannak azonban nagyon jó megoldásai a Redmentának. Itt említhető a címkerendszer, mellyel nagyon könnyen és gyorsan visszakereshetővé válnak a már létrehozott kérdések, valamint a letisztult, átlátható, és emellett megfelelő funkcionalitást biztosító kezelőfelületet is pozitívumként értékelhetjük.

Összességében tehát az látható, hogy a Redmenta gyors és hatékony munkavégzést tesz lehetővé, némi funkcionális deficit árán. Lehetséges, hogy a mi rendszerünk működése néhány ponton ennél mélyebb tervezést igényel majd, de a Redmenta jó pár megoldásának újrafelhasználása mindenképpen megfontolandó. A Moodle rendszerénél már látott szempontok alapján a Redmenta a következőképpen értékelhető:

Kezelhetőség	Könnyen kezelhető, kevés tanulást igényel
Bővíthetőség, testreszabhatóság	Viszonylag gyenge, nincs lehetőségünk személyre szabni a számonkéréseket
Ár	Ingyenes
Lehetőségek diverzitása	Csak alapvető kérdésfajták, de van lehetőség matematikai képletek beszúrására
Legjobb egyedi funkció	Kérdések címkézhetősége, kereshető kérdésbank
Dizájn, vizualitás	Egyszerű, jól átlátható

2.2 táblázat: A Redmenta értékelése különböző szempontok alapján

2.2.3. SOCRATIVE

A Socrative rendszerének bemutatása

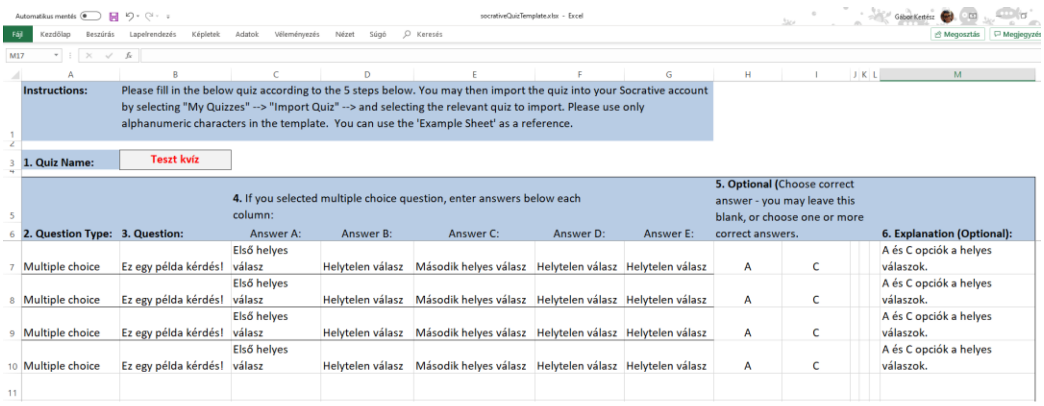
A Socrative egy több platformon is elérhető, online kérdőív készítő alkalmazás. Tanárként regisztrációra van szükség, ezt követően válik elérhetővé a tesztkészítés. [8]

A rendszer különlegessége a multiplatformitásában rejlik. Lévén, hogy az alkalmazás elérhető webes, mobilos (iOS és Android egyaránt) felületen is, rendkívül jó flexibilitást nyújt a számonkérések lebonyolításához.

A Socrative csak angol nyelven érhető el, ez mindenképpen hátrányt jelenthet a mindennapi használatában. Bejelentkezés után a menüsávban található *Quizzes* fülre

kattintva kezdhetünk neki a tesztek létrehozásának. Itt rögtön lehetőségünk nyílik kiválasztani, hogy teljesen új kérdéssort szeretnénk összeállítani, vagy egy már meglévőt használnánk fel.

Az utóbbi lehetőséget választva egészen szokatlan megközelítéssel találkozhatunk. Egyrészt lehetőségünk van velünk mások által egy azonosító segítségével megosztott kérdéssor betöltésére, azonban lehetséges komplett kérdéssor importálása is. Ez a következőképpen zajlik: egy Microsoft Excel formátumú táblázatot tudunk letölteni az elhelyezett hivatkozás segítségével, mely tartalmaz egy munkalapot, ami sablonként szolgál. Az táblázat sorainak kitöltésével tudjuk a kérdéseket és a hozzájuk tartozó válaszlehetőségeket megadni. A webes feldolgozás gyorsan és pontosan történik, a fájl betallózása után rögtön is láthatjuk a kérdéseinket a weblapon. Érezhető ugyanakkor, hogy rendkívül lecsökken a lehetőségeink száma, amennyiben ezt a fajta feladatlap-összeállítási megközelítést választjuk, cserébe viszont a nyers adatok megadása, majd a rendszer által elvégzett automatizált beolvasás rendkívül meggyorsítja a munkavégzést. Nincs szükség arra, hogy egy webes felület bonyolult oldalai között navigáljunk, egy új kérdés hozzáadásához csupán a következő sorba kell lépnünk. [9]



Instructions:	Please fill in the below quiz according to the 5 steps below. You may then import the quiz into your Socrative account by selecting "My Quizzes" --> "Import Quiz" --> and selecting the relevant quiz to import. Please use only alphanumeric characters in the template. You can use the 'Example Sheet' as a reference.								
1. Quiz Name:	Teszt kvíz								
		4. If you selected multiple choice question, enter answers below each column:					5. Optional (Choose correct answer - you may leave this blank, or choose one or more correct answers.		
2. Question Type:	3. Question:	Answer A:	Answer B:	Answer C:	Answer D:	Answer E:			6. Explanation (Optional):
Multiple choice	Ez egy példa kérdés!	Első helyes válasz	Helytelen válasz	Második helyes válasz	Helytelen válasz	Helytelen válasz	A	C	A és C opciók a helyes válaszok.
Multiple choice	Ez egy példa kérdés!	Első helyes válasz	Helytelen válasz	Második helyes válasz	Helytelen válasz	Helytelen válasz	A	C	A és C opciók a helyes válaszok.
Multiple choice	Ez egy példa kérdés!	Első helyes válasz	Helytelen válasz	Második helyes válasz	Helytelen válasz	Helytelen válasz	A	C	A és C opciók a helyes válaszok.
Multiple choice	Ez egy példa kérdés!	Első helyes válasz	Helytelen válasz	Második helyes válasz	Helytelen válasz	Helytelen válasz	A	C	A és C opciók a helyes válaszok.

2.4 ábra: A Socrative által használt Excel-táblázat

Minden rendszer tervezése esetén fontos problémakör a célplatform kiválasztásának kérdése. Nem elegendő egy jó ötlet, vagy akár jó mérnöki megoldások tömkelege, ha hibásan választunk közeget a megvalósításunk számára, szinte biztosra vehető, hogy alkalmazásunk sikere töredéke lesz csupán a benne rejlő potenciálnak. Érdemes átgondolnunk, hogyha egy olyan ideális oktatás-támogató rendszer fejlesztésén dolgozunk, mely elsősorban a számonkéréseket hivatott modernizálni, akkor milyen lehetőségeink vannak. Miközben a klasszikus asztali számítógépek már visszaszorulni látszanak sok területen, addig az okostelefonok és táblagépek piaca aranykorát éli. Szinte biztosra vehető például az, hogy amennyiben egy átlagos középiskolai osztályban arra kérnénk a diákokat, vegyék elő okostelefonjaikat, akkor nagyon kevés diáknak okozna ez problémát. Nem nehéz azt sem belátni, hogy 30 fő esetén már pusztán infrastrukturális

okokból is sokkal egyszerűbben megoldható egy digitális számonkérés mobileszközök segítségével, mint személyi számítógépeken keresztül.

A rendszer felhasználói felülete barátságos, könnyen kezelhető. Ez el is várható az alkalmazástól, lévén, hogy viszonylag kevés funkciót kellett elhelyezni a grafikus felületen az alkalmazás készítőinek. A program nagy előnye, hogy mobilos felületen is rendelkezésre áll, ehhez két külön alkalmazást biztosít, egyet a tanároknak, és egy másikat a tanulók számára. Ez talán nem a legszerencsésebb megközelítés, hiszen így két, nagyon hasonló architekturális felépítésű alkalmazást kell szimultán karbantartania a fejlesztőknek. A két program funkcionalitását egy közös alkalmazásban implementálva, valamint egy egyszerű autentikációs lépést beépítve - mely során a felhasználó eldöntheti, hogy tanári vagy diák minőségben kívánja használni az applikációt – kiküszöbölhető lett volna a probléma. Egyébiránt elmondható, hogy a felület és a működés, mellyel a hordozható eszközökön találkozunk, nagyban hasonlít a weben látottra, annak szinte összes elemét sikeresen átültették a fejlesztők a mobilos platformra.

A Socrative rendszerének értékelése

A Socrative erősségei láthatóan a multiplatformitásban rejlenek. Az alkalmazás azonban csak angolul érhető el, és itt is szűkösek a rendelkezésre álló lehetőségek, nagyon általános felhasználásra alkalmas csupán a rendszer. Tekintsük át, hogy a korábban felállított szempontrendszerbe miként illeszkedik a Socrative:

Kezelhetőség	Könnyen kezelhető, de idegennyelvi ismeretek szükségesek a használatához
Bővíthetőség, testreszabhatóság	Szűk, kevéssé rugalmas
Ár	Ingyenes
Lehetőségek diverzitása	Csak alapvető kérdésfajták
Legjobb egyedi funkció	Microsoft Excel állományból történő kérdésfeltöltés
Dizájn, vizualitás	Modern

2.3 táblázat: A Socrative értékelése különböző szempontok alapján

2.3. REFLEXIÓ AZ ISMERTETT RENDSZEREKRE

A korábbiakban látott rendszerek kiválasztása során törekedtem arra, hogy egy minél szélesebb szoftverpalettát mutassak be. Így rengeteg jó fejlesztői megközelítéssel találkozhattunk, ezekhez hasonló megoldások implementálása mindenképpen megfontolandó a mi rendszerünk esetén is.

Fontos azonban észrevennünk, hogy a korábban bemutatott alkalmazások nagyon általános felhasználási célúak. Bár néhol látható igyekezet arra a fejlesztők részéről, hogy a rendszerüket specializálhatóvá tegyék, feltehetőleg kevés mozgásterük volt ebben a tekintetben: valószínűsíthetjük, hogy a különböző projektekben már portfolió-menedzsmenti szinten döntöttek arról, hogy a termékeknek általánosan felhasználhatónak kell lenniük. Ez a mi esetünkben problémát jelent, hiszen, ha az **informatikai felsőoktatásra** fókuszálunk, akkor kijelenthető, hogy nem találunk olyan rendszert a piacon, mely képes lenne biztosítani, hogy a területen szerzett ismereteket kielégítően felmérő számonkéréseket hozzunk létre.

Feleletválasztós kérdésekből álló tesztekkel az ismeretanyag egy része valóban lefedhető, sőt, esetenként akár hasznos is lehet egy egyszerű, gyorsan kitölthető kérdéssor kiadása – gondoljunk a felsőoktatási köznyelvben „beugró” néven elterjedt számonkérésre. Feltételezhetjük, hogy kisebb-nagyobb nehézségek árán, de a korábban vizsgált rendszerek mindegyikében hatékonyan elkészíthető, valamint értékelhető egy ilyen teszt, azonban abban az esetben, mikor egyszerre 100, 200, vagy esetenként akár több hallgatót kell ilyen módon vizsgáztatni, figyelembe kell vennünk, hogy pusztán infrastrukturális okokból szerencsésebb volna a klasszikus, papíralapú számonkérés metodikájához folyamodni. Ennek az elősegítője egy olyan funkció implementálása, mely segítségével a digitálisan összeállított kérdéssort valamilyen nyomtatható formátumba tudjuk alakítani, majd az oktató ki tudja azt nyomtatni, és a diákok által kitöltött lapokat könnyedén ki tudja értékelni.

Ezenfelül viszont azt is be kell látnunk, hogy az informatika tudományterületén belül szinte minden alterületen jelentkezik igény programozási ismeretekre. Éppen ezért bármelyik informatikához köthető alapképzés tanmenetét áttekintve azt láthatjuk, hogy abban szerepelnek olyan tárgyak, melyek keretében a hallgatók valamilyen – jellemzően magas szintű – programozási nyelvvel ismerkednek meg. Még a nem kifejezetten fejlesztői irányt megcélzó képzések, mint például a gazdaságinformatikus, vagy villamosmérnöki alapszakok is igen komoly programozási ismeretekkel ruházzák fel a hallgatókat, nem beszélve a szakmai törzsanyagon kívül eső szoftverfejlesztői szakirányokon induló kurzusokról, vagy az egészen programozás-orientált programtervező-informatikus alapképzésről.

Az itt szerzett tudásról természetesen számot is kell adjanak a hallgatók, azonban könnyen belátható, hogy szoftvertervezési és -fejlesztési alapismereteket szinte képtelenség adekvát módon számonkérni hagyományos módszerekkel, ahol a

lehetőségek jellemzően kimerülnek a feleletválasztós, esetleg a kiegészítős kérdésekben. Sok esetben kielégítő lehet pszeudókódokat visszakérdezni, hiszen ebből könnyen látható, hogy a hallgató milyen szintű rálátással bír az adott problémára, azonban ezeknek a képzéseknek a szerves részét képzi a gyakorlati oktatás is. A programozás laborokon jellemzően zárthelyi dolgozatok formájában oldanak meg komplex feladatokat a hallgatók, azonban könnyen azt érezhetik, hogy az elméleti tudásanyag és a gyakorlatban látottak igen gyakran eltérnek egymástól. Mivel az elméleti tesztek során nem lehet megoldani a konkrét programnyelvi implementáció számonkérését, és a gyakorlati zárthelyiken pedig a kisebb, egyszerűbb problémák helyett jellemzően a nagyobb, összetettebb feladatokkal kell foglalkozzanak a hallgatók, így valójában a kétfajta tudásanyag tényleges ötvözése csak nagyon ritkán valósul meg. Felmerül tehát az igény egy olyan rendszerre, ahol a klasszikus kérdéstípusok mellett, az oktatónak lehetősége van kisebb programozási feladatokra szegmentált gyakorlati anyagot is beszúrni a tesztbe. Utóbbi megközelítésnek feltehetőleg pszichológiailag igazolható pozitív hatása is van: gondoljunk arra, hogy egy hallgató valószínűleg eredményesebben ad megoldást egy, a *gyorsrendezés* implementációját igénylő problémára, amennyiben azt egy dedikált gyakorlati kérdés keretében kell megtennie, semmint egy egyébként is kihívást okozó, összetett elméleti zárthelyi dolgozat utolsó feladataként.

Érdemes lehet a korábban ismertetett szempontrendszer *legjobb egyedi funkció* sorát is figyelembe venni. Ezek a megoldások a következők voltak:

Termék	Legjobb egyedi funkció
Moodle	Lehetőség IP-cím alapján történő szűrésre a tesztek kitöltésekor
Redmenta	Kérdések címkézhetősége, kereshető kérdésbank
Socrative	Microsoft Excel állományból történő kérdésfeltöltés

3.1 táblázat: A legjobb egyedi funkciók listája

3. SPECIFIKÁCIÓ

A fent leírtak, valamint a korábban bemutatott szempontrendszer értékelése után az alábbi rendszerspecifikáció állítható össze:

- A felhasználói felület legyen reszponzív, tehát olvasható és kezelhető a különböző méretű eszközökről való elérés esetén is, így biztosítva a mobilos használatot
- A fő felhasználási mód a számonkérések készítése, valamint levezénylése lesz, és a vizuális megjelenítésnek is ezt kell támogatnia, egyszerűvé tennie
- Szükséges, hogy az oktatók és a diákok eltérő felhasználási módjának megfelelően szét tudjuk választani a működést, tehát be kell építeni egy azonosítási lépcsőt
- Legyen lehetőség a kérdéssorok jelszóval való védelmére, valamint a kitöltés IP-cím szerinti korlátozására
- A kérdések legyenek címkézhetők, és igény esetén az új kérdéseket lehessen hozzáadni egy központi kérdésbankhoz, mely kereshető
- Támogassa a program a kérdések Microsoft Excel formátumú táblázatból való betöltését
- Széles kérdéstípus skáláról lehessen kérdésfajtát választani
- Ahhoz, hogy programozási feladatok legyenek beszúrhatók a tesztekbe, szükség van arra, hogy fel tudjuk dolgozni a programkódot. Mivel a korábban említett képzések során a legtöbbet használt forrásnyelv a C#, így az ezen a nyelven íródott kód fordítására lesz szükségünk. Szerencsére, ehhez rendelkezésre áll a Microsoft által fejlesztett, nyílt forráskódú *.NET Compiler Platform*, vagy ismertebb nevén a **Roslyn API**. Természetesen más fordítókkal is bővíthető lehet később a program, ehhez a megfelelő absztrakciós szintek használata szükséges a fejlesztés során.
- A rendszerben legyen lehetőség tesztek és beadandó feladatok létrehozására, utóbbi esetén a rendszer legyen képes automatizált javításra, kulcsszavak keresésére, programkód fordításra, kimenetek összevetésére

4. RENDSZERTERV

Az általam elkészíteni kívánt rendszer több platformon is elérhető kell legyen. A fő felhasználás az asztali, webes felületen keresztül zajlik, a mobilos elérés pedig megoldható ugyanennek a felületnek az egyszerűsítésével. Ez utóbbi az asztali variáns néhány szolgáltatását biztosítja csupán, azokat, melyek kényelmesen kezelhetőek kisebb kijelzőméret mellett is.

4.1. A WEBALKALMAZÁS TERVE

A rendszer alapját egy központi alkalmazás képezi. Ennek a feladata a felhasználók authentikálása, lehetővé tenni a tesztek létrehozását, a kérdésbank eléréséhez szükséges interfészt biztosítani, a számonkéréseket levezényelni, azokat kiértékelni. Mindezt persze olyan módon, hogy az egyes folyamatok egy igényes felületen irányíthatóak, áttekinthetőek legyenek.

A fentiek megvalósításához a webalapú megközelítésre esett a választásom. Ennek oka teljesen a mai alkalmazásfejlesztési trendekben keresendő: míg az klasszikus asztali szolgáltatások visszaszorulóban vannak, a webes megoldások virágkorukat élik. Az ilyen alkalmazások jól skálázhaztóak, széles körben, sok eszközökön, és könnyen felhasználhatóak – a használatukhoz gyakorlatilag egy modern böngészőre van csak szükségünk. A hálózati infrastruktúra is éppen ehhez a trendhez illeszkedett, és rohamosan fejlődött az elmúlt évtized során – gondoljunk csak arra, hogy mennyivel elérhetőbbek lakossági szinten is a mobilinternet szolgáltatások, illetve mennyivel jobbak a vezetéknélküli internetezési viszonyok például az egyetemi campusokon. Mivel tehát a hálózati és böngészési feltételek is szélesebb körben adottak, így kézenfekvő a webalapú megoldás választása. [10]

A technológia fent említett népszerűsége révén rettentően széles az implementációs lehetőségek skálája. Az alábbiakban a különböző megvalósítási lehetőségek összehasonlításának tárgyalása olvasható.

Technológia	Forrásnyelv	Jellemzők
Angular	Javascript	Rendkívül flexibilis, sok fejlesztési célra használható keretrendszer. Szerveroldalon is implementálható, de az erősségei elsősorban a kliensoldali, tehát front-end fejlesztésben rejlenek.

Spring	Java	A JVM ökoszisztémába kiválóan illeszkedik, rendkívül elterjedt. A függőségek kezelését sokrétű absztrakciós lehetőségei révén nagyban megkönnyíti.
ASP.NET Core	C#	Elsősorban a szerveroldali feladatokat kiválóan ellátó keretrendszer, mely kiegészülve olyan front-end megoldásokkal, mint a Razor könnyen használható robusztus, de jól tesztelhető, erőforráshatékony webes alkalmazások fejlesztésére.
Laravel	PHP	Elsősorban üzleti felhasználási környezetben elterjedt keretrendszer, melynek erőssége a könnyű fejleszthetőségben rejlik. Szintén MVC tervezési mintát követ.

4.1 táblázat: Néhány népszerű webes keretrendszer és jellemzőik

Bár rengeteg opció áll rendelkezésre, elsősorban a felhasználási esetünk alapján kell megoldást választanunk. A Javascript és Typescript nyelvekre építő keretrendszerek rendkívül népszerűek, és valóban szép webes felületek készíthetőek a segítségükkel, azonban nekünk elsősorban az intenzív szerveroldali igényeket kiszolgálni képes implementáció választására kell törekednünk. A Spring és a Laravel kiforrott technológiák, melyek ráadásul két, egyaránt robusztus és elterjedt programnyelvre építenek – míg a Spring Java-alapú, addig a Laravel segítségével PHP-ban tudunk fejleszteni. Ennek a két technológiának az előnye éppen ebben rejlik, hiszen azzal, hogy széleskörben használtak, nagy felhasználói bázissal, és rengeteg referenciával rendelkeznek. [11]

A választásom azonban mégis az ASP.NET Core keretrendszer használatára esett. Ennek oka, hogy ez is rendkívül népszerű, és olyan módon tartozik a Microsoft ökoszisztémájába, hogy emellett nyílt-forráskódú projekt is. A technológia mellett szól továbbá, hogy kiváló erőforrás-hatékonysággal rendelkezik, tökéletesen illeszkedik a felhőalapú virtualizációs trendekhez, és a mára igazán kiforrott, komplex programnyelv, a C# körül összpontosul. A kliensoldali megjelenítés, tehát tulajdonképpen a HTML nézet generálása is megoldható a technológia segítségével, ehhez egyrészt használható az MVC mintára épülő verzió, valamint a Razor Pages, mely egy jobban centralizált, újabb megközelítés. Emellett érdemes megemlíteni, hogy a mikroszervíz architektúra irányába történő orientáció is lehetséges, hiszen, ha csak egy API végpontként tekintünk a webes alkalmazásra, akkor az könnyen integrálható például egy Javascript-alapú kliensoldali applikációval. Természetesen a megfelelő architekturális rétegzés könnyíti az átállást, tehát elvárás az üzleti logika leválasztása a megjelenítésről.

A karbantarthatóság, későbbi fejleszthetőség szempontjából is jó választás lehet az ASP.NET Core. Mivel a .NET Core a Microsoft nyílt forráskódú .NET keretrendszer implementációja, így gyakorlatilag bármilyen platformon fejleszthetünk a technológia segítségével. [12] [13]

Ami a frontendet, vagyis az alkalmazásnak a felhasználóval történő interakciókat biztosító részét illeti, itt több választási lehetőség mutatkozik. Az utóbbi években ipari sztenderddé vált az a megközelítés, miszerint a webes felhasználói felület működtetéséért felelős kliens oldali alkalmazás valamilyen Javascriptes alapokon nyugvó keretrendszert használ, azonban a dolgozat írásának idején elérhetővé vált egy új alternatíva is. Ez a megoldás nem más, mint a szintén a Microsoft égisze alatt fejlődő, és hasonlóan nyílt forráskódú Blazor, mely kiválóan integrálható az ASP.NET Core-al. Ennek a megközelítésnek az előnye, hogy komplex, harmadik fél, vagy egy közösség által fejlesztett, karbantartott, bonyolult Javascriptes keretrendszerek használata nélkül is hasonlóan reszponzív és modern webes felületek készíthetők. Ennek megvalósításához egyrészt épít a ténylegesen böngészőben futtatható forráskódra – amihez a Javascript interpreter helyett egy új, szintén a böngészőkbe integrált környezetet, WebAssemblyt használ -, másrészt pedig készíthetők teljesen szerveroldali alkalmazások is. A Blazor elnevezés is utal a megoldás gyökereire: a korábbi ASP-s MVC mintán alapuló alkalmazások *View* rétegéből ismert népszerű Razor Pages megjelenítést kombinálták a WebAssembly nyújtotta lehetőségekkel.

A két Blazoros megközelítés közül az utóbbihoz szükség van egy WebSocket alapú kommunikátorra. Ehhez a Microsoft oltalma alá tartozó, valós idejű üzenetküldő implementációt, a SignalR-t használja a Blazor, annak érdekében, hogy megtartsa a *single-page application,* tehát a frissítés, látszólagos újratöltés nélkül működő oldal jellegét. A SignalR a kétirányú kommunikációt biztosítja a kliens és a szerver között, tehát egyrészt feldolgozza a felhasználói felületen keletkező eseményeket, valamint gondoskodik róla, hogy a háttérben történő változásoknak megfelelően lehetőség legyen frissítéseket is eszközölni azon. Ennek a megközelítésnek az ára, hogy SignalR kommunikációs csatornáin rengeteg adat vándorol a felület és szerver között, így például lassabb kapcsolat esetén időbe telhet, mire az alkalmazás betöltődik. Ha a kliens-oldali Blazor implementációt tekintjük, akkor azt láthatjuk, hogy a szűk keresztmetszetet ez esetben a WebAssembly meglétének igénye jelenti a böngészőben. Azonban ezt leszámítva rengeteg előnye van ennek a verziónak. Például a Big Data világából ismert az a törvényszerűség, hogy jelentős hatékonyság-növekedés érhető el, amennyiben nem az adatokat, hanem az az azokat manipuláló programkódot mozgatjuk, lévén, hogy a programkód mérete kilobyte-os léptékkel leírható a legtöbb esetben. Természetesen a megjelenítendő, nyers adatok eléréséhez itt is a Web API irányába intézett kérésekre lesz szükségünk, azonban így kisebb csomagok közlekednek HTTP kéréseken keresztül, hiszen nem a szerveroldalon kerül legenerálásra az egész oldal. [14]

Tekintettel arra, hogy a dolgozatom írásának idején a WebAssembly 1.0-ás verziója a négy fő böngésző (Google Chrome, Mozilla Firefox, Safari, Microsoft Edge) legfrissebb frissítésének mindegyikében szállításra került, így a választásom természeteszen a kliens-oldali Blazor használatára esett. [15]

A képet tovább árnyalja, hogy az említett két verzió mellett a redmondi cég további Blazor-alapú megoldások létrehozásán gondolkodik, melyek összeségükben lehetővé tennék, hogy C# alapú, platformfüggetlen, asztali felhasználói felületek legyenek készíthetők a technológia segítségével - így kielégítve az ilyen jellegű, igencsak régen megfogalmazott fejlesztői igényeket. Ezt a fejlődést a rendszer továbbfejlesztésének szempontjából lehet érdemes követni.

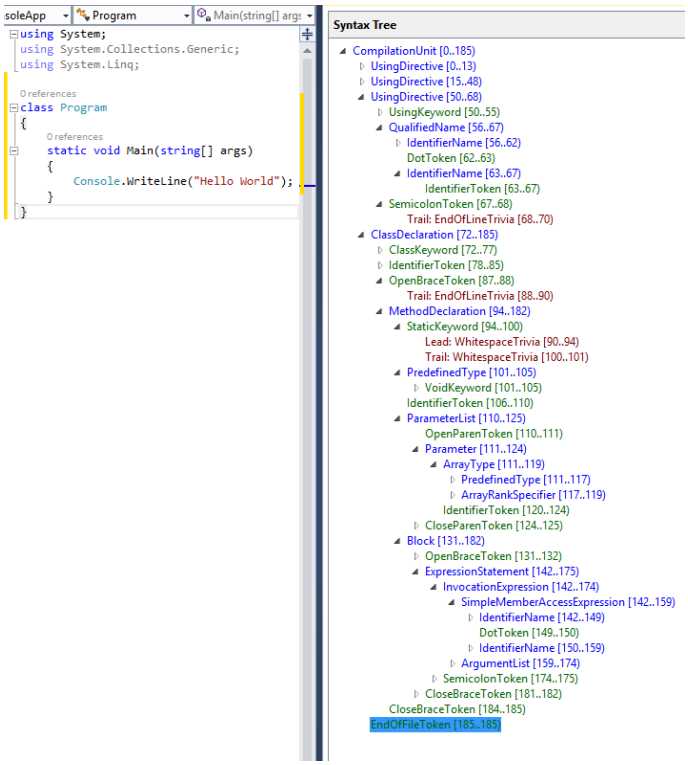
4.2. A FORDÍTÓPROGRAM TERVE

A korábban ismertetett, az informatikai felsőoktatásra specializált igényeknek megfelelően szükség lesz egy modulra az alkalmazásban, mely forráskódelemzést tesz lehetővé. Ez elsősorban egy C# fordító, mely moduláris jellegű fejlesztés esetén később bővíthető lehet más programnyelvekkel is. A megvalósításhoz a Microsoft által fejlesztett, nyílt forráskódú *.NET Compiler Platform,* vagy ismertebb nevén a Roslyn API áll rendelkezésre.

A Roslyn API gyakorlatilag a klasszikusan transzparens C# fordítási folyamatot teszi láthatóvá. A fordító a következő modulokból épül fel:

- Lexikális elemző, felbontó modul
- Szemantikai elemző modul
- Köztes nyelvi kódgenerátor

Az első modul feladata, hogy a nyers programkódi bemenetet *linkelje,* tehát összekapcsolja az összes tartalmazó fájlt egy monolitikus szöveggé, majd feldarabolja, a későbbi modulok számára értelmezhető módon formalizálja azt. Ez a folyamat a *tokenizálás,* melynek végeredményeként - a beszélt nyelveinkben ismert lexéma fogalommal analóg - elemi egységek, tokenek jönnek létre. Ezen műveletsor elvégzéséért a Roslyn esetében a **Syntax API** komponens felelős. Az adatreprezentációhoz egy kézenfekvő megközelítést választottak a redmondi cég mérnökei: eredményként gyakorlatilag egy faszerkezetet kapunk, melyre a Microsoft *szintaxis-fa* néven referál. Ez minden részletében reprezentálja a programkód struktúráját, ám semmilyen információtöbbletet nem nyújt arról. A 4.1 ábra egy, a Syntax API által előállított minta kimenetet mutat: [16] [17]



4.1 ábra: Mintakód és a Syntax API által előállított hozzátartozó faszerkezet

A következő, tehát a szemantikai analízist végző modul bemenetként éppen az előbb említett szintaxis-fát kapja meg. A **Semantic API** feladata szimbólumokat (típusokat, névtereket, változókat) keresni a struktúrában. A *binding,* tehát összekötés folyamata során kerülnek összekapcsolásra ezek a kódban található azonosítókkal. A műveletsor végeztével a szemantikai vizsgálat eredménye alapján eldönthető, hogy a kód lefordítható-e hibátlanul, és amennyiben nem, akkor információt nyerhetünk a hibák jellegéről. [18]

Az utolsó lépéshez az **Emit API** kapcsolódik. Ennek feladata, hogy előállítsa az úgynevezett *köztes nyelvi kódot,* ami gyakorlatilag egy platform- és processzor-független utasításkészlet. Ezt a későbbi fordító az aktuális szoftveres és hardveres környezet számára értelmezhető, megfelelő natív kóddá alakítja.

A három API együtt nagyon flexibilis és sokrétű felhasználást tesz lehetővé. Látható, hogy minden fázis külön-külön is kiemelhető, így azok eredményei használhatóak szétválasztva is. A szintaktikai analízis eredményének segítségével választ kaphatunk olyan kérdésekre például, hogy bizonyos kulcsszavak hol találhatóak a programkódban, vagy akár listázhatjuk az összes importált könyvtárat is. Ezen komponens segítségével vizsgálhatunk meg továbbá metódus-szignatúrákat, paraméterlistákat, osztályöröklődési hierarchiákat is. [19]

Az iméntiek kontextusba helyezéséhez a szemantikai analízis áll rendelkezésünkre. A gyakori felhasználási esetek közé tartozik az egy bizonyos szakaszon belüli változók

listázása, konkrét típusok adattagjainak, kapcsolódó metódusainak megjelenítése, a láthatósági szintek vizsgálata.

A Roslyn API előnye, hogy támogatja a LINQ kiterjesztő metódusokat is, továbbá az úgynevezett *query syntax,* tehát a C#-ban található LINQ-alapú lekérdező nyelv is használható arra, hogy információt nyerjünk a programkódról. Ezek segítségével megfelelően komplex ellenőrzések készíthetők egy-egy programozási feladat esetén, a számonkérés készítőjének egészen specifikus kritériumai is könnyedén ellenőrizhetők az API használatával. Emellett természetesen lehetőségünk van a lefuttatott kód kimenetét, eredményét megjeleníteni, vizsgálni, valamint magát a forráskódot visszaadni is. Ez egyszerűbb programozási feladatok ellenőrzése esetén válhat az oktatók hasznára.

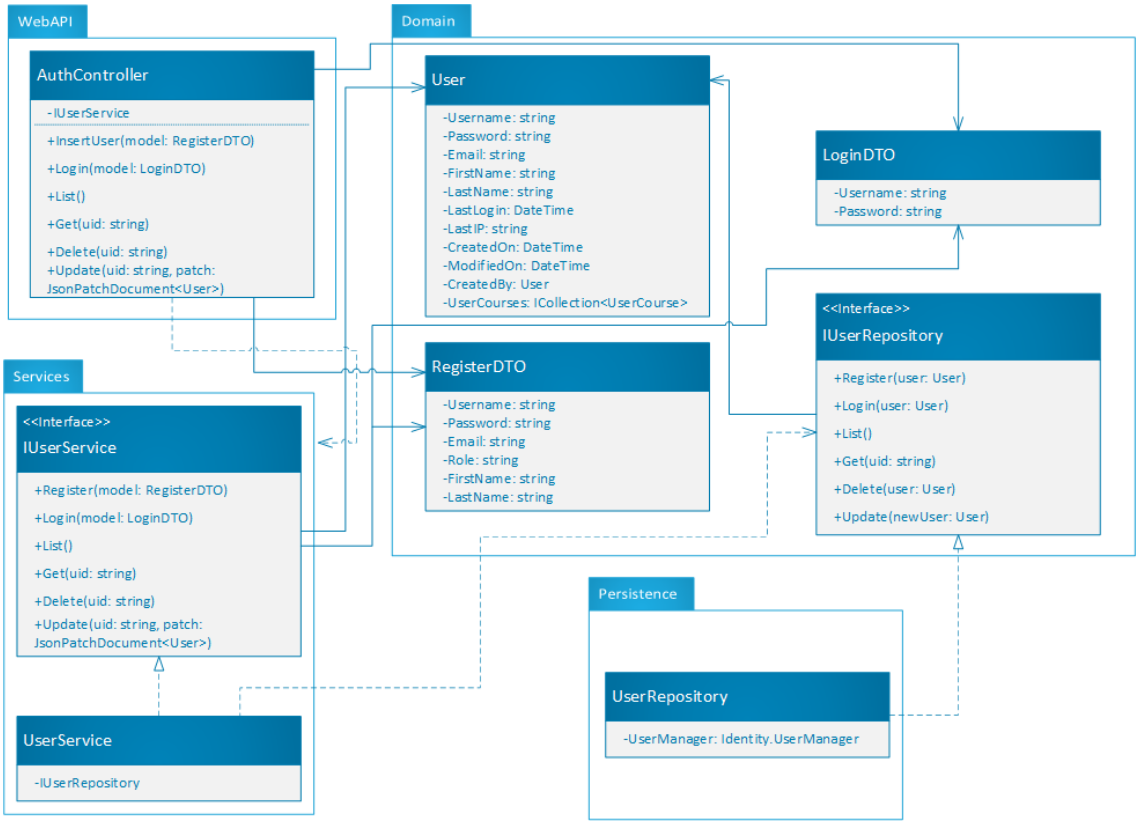
A Roslyn API nyújtotta lehetőségek tárháza hatalmas, egy rendkívül széles körben felhasználható szolgáltatásról beszélünk, melyet a legtöbb esetben diagnosztikai célú, vagy a fejlesztést valamilyen módon támogató (saját integrált fejlesztői környezet, vagy egy meglévő IDE-be beépülő modul) alkalmazások írásához használják a felhasználók. A mi esetünkben azonban egy ennél jócskán egyszerűbb használati esetről beszélhetünk, így célszerű megközelítés lehet egy saját API létrehozása, mely egy *facadeként* funkcionálna a Roslyn API előtt, egy egyszerű interfészt biztosítva, így elrejtve annak komplexitását.

4.3. ARCHITEKTURÁLIS TERVEZÉS

A modern szoftverfejlesztés utóbbi tíz-tizenöt évében alapvető követelménnyé vált, hogy olyan architekturális tervezési minták mentén közelítsük meg a fejlesztést, mint például a SOLID-elvek. Az egészen egyszerűnél nagyobb komplexitású rendszerek esetén minden bizonnyal a legfontosabb tervezési elv, melynek betartására törekednünk kell, a *separation of concerns,* vagyis a funkciók szétválasztása. Ennek megfelelően a feladatunk, hogy a fejlesztés során a felelősségi körök minimalizálásának segítségével megkeressük a logikailag önálló kódrészeket, és ezeket modularizáljuk, önálló egységekbe szervezzük, így elkerülendő, hogy egy hatalmas, kibogozhatatlan kódbázis álljon elő. Utóbbi ismérve lehet, hogy egy folyamat végigkövetése során rengeteg modul között kell ugrálnunk, tehát a kód kohéziója alacsony, az összefüggő részek a szükségesnél nagyobb távolságra találhatóak egymástól, aminek hatására az őket tartalmazó modulok egymással függésben állnak, így mondhatjuk, hogy *tight coupling* áll elő, vagyis szorosan kapcsoltak a komponensek. Mivel egy ilyen rendszer olvashatósága, átláthatósága rossz, ezért kevés rugalmassággal bír egy esetleges bővítés, átszervezés, fejlesztés során, tehát gyakorlatilag a termék élettartama - és ezáltal költséghatékonysága - drasztikusan csökkent a hibás architekturális tervezés nyomán. [20]

A rétegzett tervezés tehát alapvető elvárás, azonban ennek megvalósítására több lehetőség is mutatkozik. Tekintve, hogy az alkalmazásom túlmutat egy egyszerűbb

applikáció követelményein, viszont nem egy hatalmas, nagyvállalat által fejlesztett és karbantartott termék, így egy köztes megoldást választottam. A klasszikus, monolitikus alkalmazásoknál megszokott rétegzést néhány *domain-driven design*, vagyis domain-vezérelt tervezési elemmel egészítettem ki. Ez felkészíti az alkalmazást arra, hogy később átszervezhető legyen monolitikus applikációból egy mikroszervíz architecktúrába. A tervezési sémát az authentikáció kontextusához tartozó osztálydiagrammal mutatom be: [21]

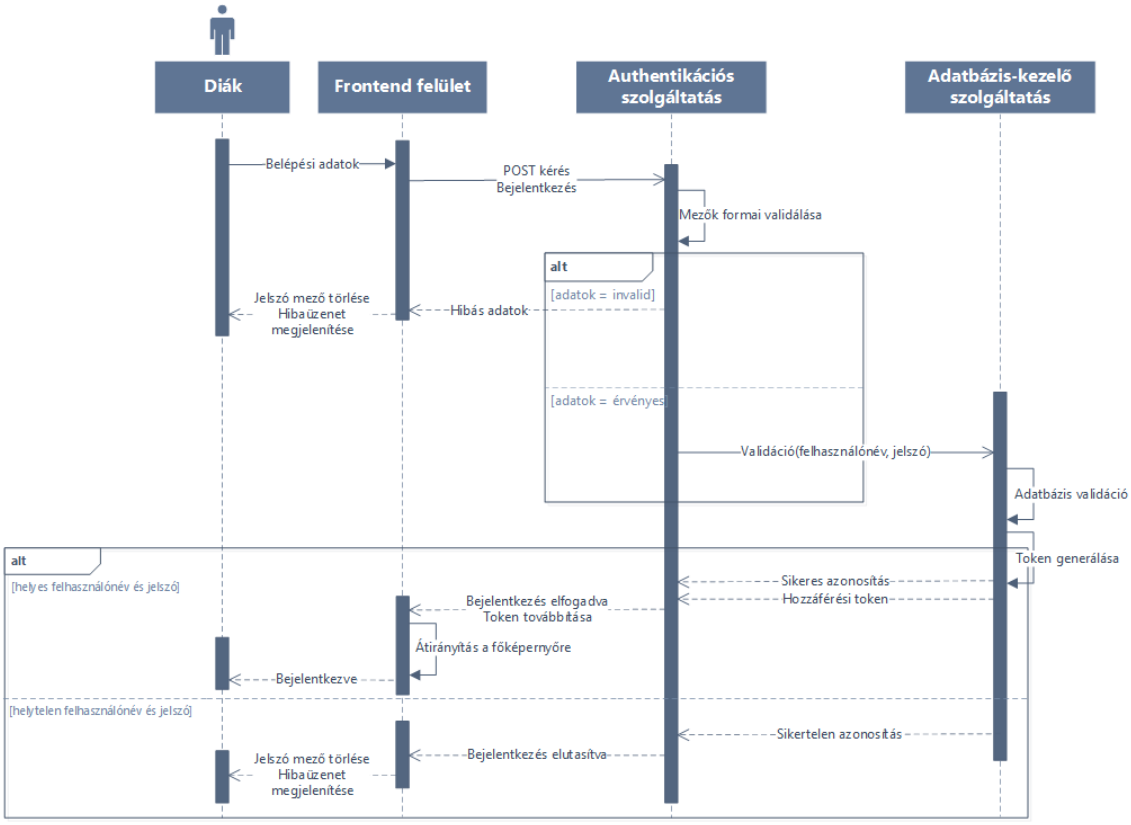


4.2 ábra: Az authentikációs kontextus osztálydiagramja

A 4.2 ábra hivatkozási irányai jól mutatják, hogy az összes komponens a *User*, vagyis a felhasználót modellező adatosztálytól függ, aköré csoportosul. Ez megfelel a domain-driven design azon alapelvének, hogy a domain modell jelenti az alkalmazás magját, amire hagyma-szerűen épül rá a többi komponens. [22]

4.4. BELÉPTETÉS

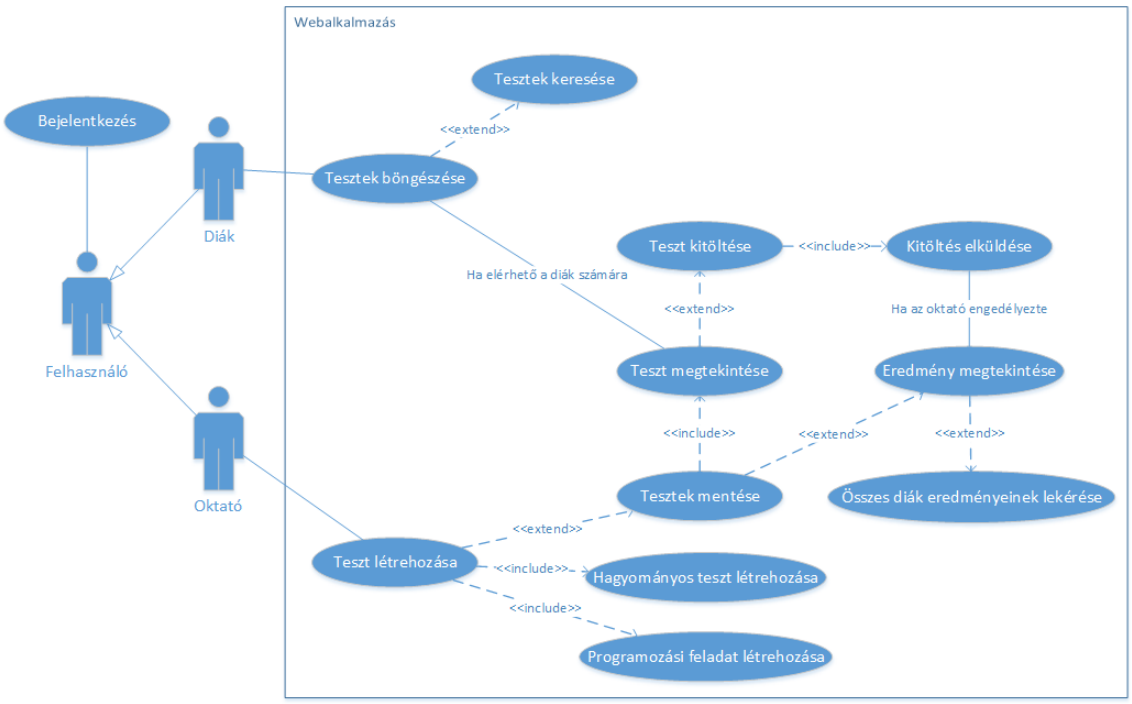
Mivel az alkalmazással szemben alapvető követelmény, hogy különböző csoportba tartozó felhasználókat tudjon kezelni, így az ő biztonságos beléptetésük egy sarokpontját képezi az applikációnak. A temérdek authentikációs megoldás közül az OAuth2.0-ás sztenderdnek megfelelő *bearer token*-es azonosítást választottam, lévén, hogy ennek alapjául a JSON Web Token formátum szolgál, mellyel az ASP kontrollerei könnyedén tudnak dolgozni. A beléptetés folyamatát a következő, 4.3 ábra szemlélteti: [23]



4.3 ábra: A beléptetés folyamata

4.5. HASZNÁLATI ESETEK

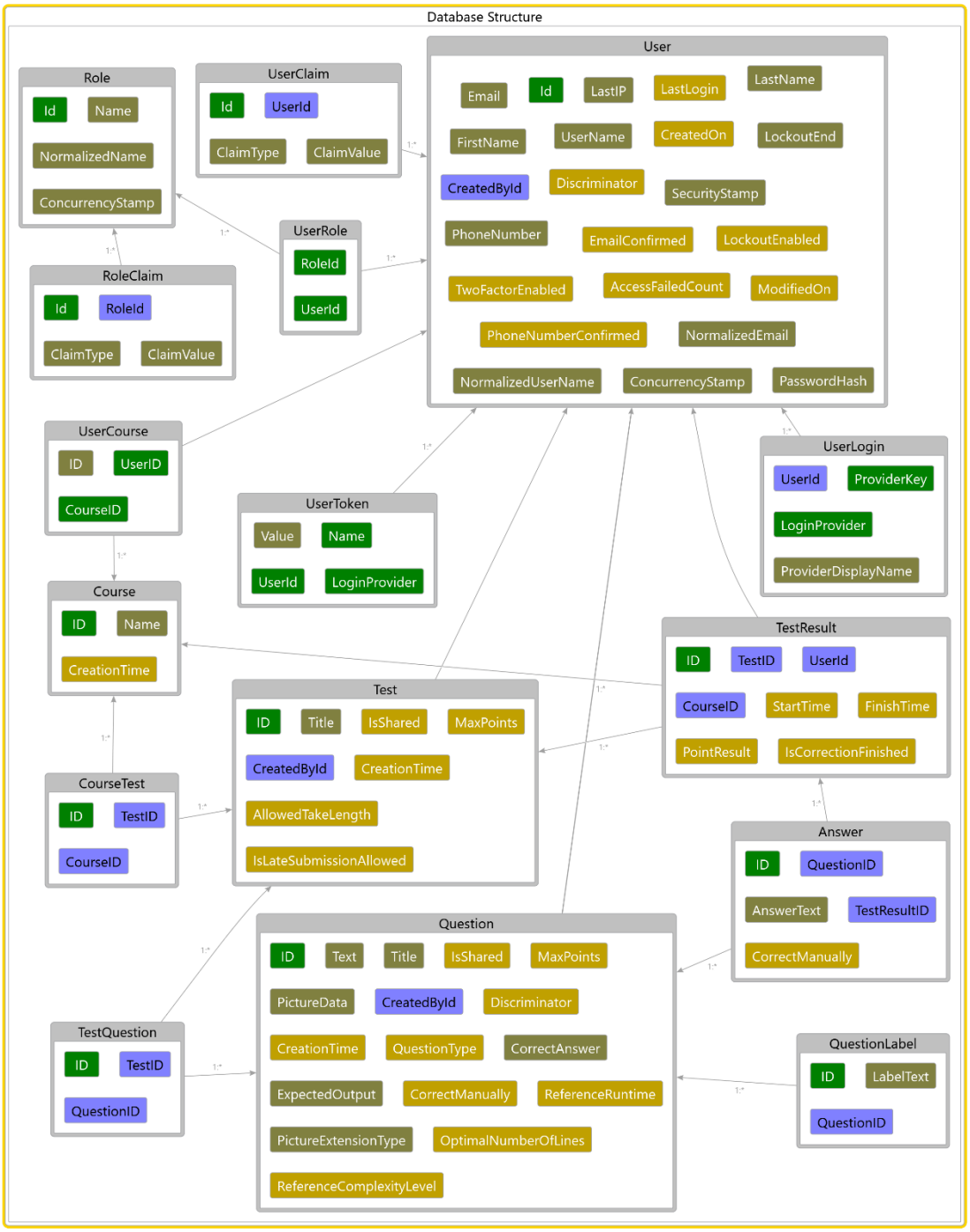
Ahhoz, hogy megfelelő képet kaphassunk arról, hogy az applikációval milyen lépések folyamatán keresztül fognak kommunikálni a felhasználók, definiálnunk kell használati eseteket. Esetünkben szerencsére ezek jól elválnak egymástól, szemléltetésükhöz pedig az alábbi, 4.4 ábra a legalkalmasabb:



4.4 ábra: A rendszer használatieset-diagramja

4.6. AZ ADATBÁZIS TERVE

Az adatok tárolásának elve, modus operandija természetesen kulcskérdést képez minden modern alkalmazás esetén. Ahhoz, hogy a programot hatékony tudjuk összekötni egy adatbázissal, szükségünk lesz egy *object-relational mapping*, vagyis egy objektum-relációs leképezést végző kiszolgálóra. Ennek segítségével a domain modellünkben található objektumokat tudjuk megfeleltetni olyan entitásoknak, amelyek értelmezhetőek az adatbázis-kezelő számára. Mivel az ASP .Net Core legjobban a Microsoft saját készítésű ORM szolgáltatásával, az Entity Framework Core-al tud együttműködni, így nekem is erre esett választásom. Szerencsére az EF Core rengeteg adatbázismotort támogat, így célszerű azt kiválasztani, ami a legtöbb futtató környezetben elérhető lesz. Ennek megfelelően, a rendkívül széles körben elterjedt MySQL adatbázist választottam. Az implementációtól független adatbázisterv a 4.5 ábrán látható: [24]



4.5 ábra: Az adatbázis terve

Az adatbázis mellett szükségünk lesz olyan tárolóra is, ahol a hagyományos SQL alapú adatbázisokba nem, vagy nehezen leképezhető struktúrákat tároljuk. Ilyenek például a fényképek, vagy a mi esetünkben különös fontos programkódok, valamint az azok fordítása során előálló állományok. Ez természetesen a környezettől függ, egy helyi szerveren való futtatás esetén annak háttértárolóját vehetjük igénybe, de ha felhőalapú

megoldásban gondolkodunk, ott is rendelkezésünkre állnak különböző objektum tárolók, melyek éppen az ilyen felhasználást hivatottak kiszolgálni.

5. IMPLEMENTÁCIÓ

Az alábbiakban a szoftver implementációját szeretném bemutatni, fókuszálva annak néhány sarokpontjára.

5.1. A SZERVEROLDALI ALKALMAZÁS

A szerveroldali alkalmazás megalkotásakor a hagyományos, monolitukus alkalmazások esetén megszokott rétegzett architektúra elveit követtem. A kialakult rétegeket, és azok feladatait az alábbi táblázatban foglaltam össze:

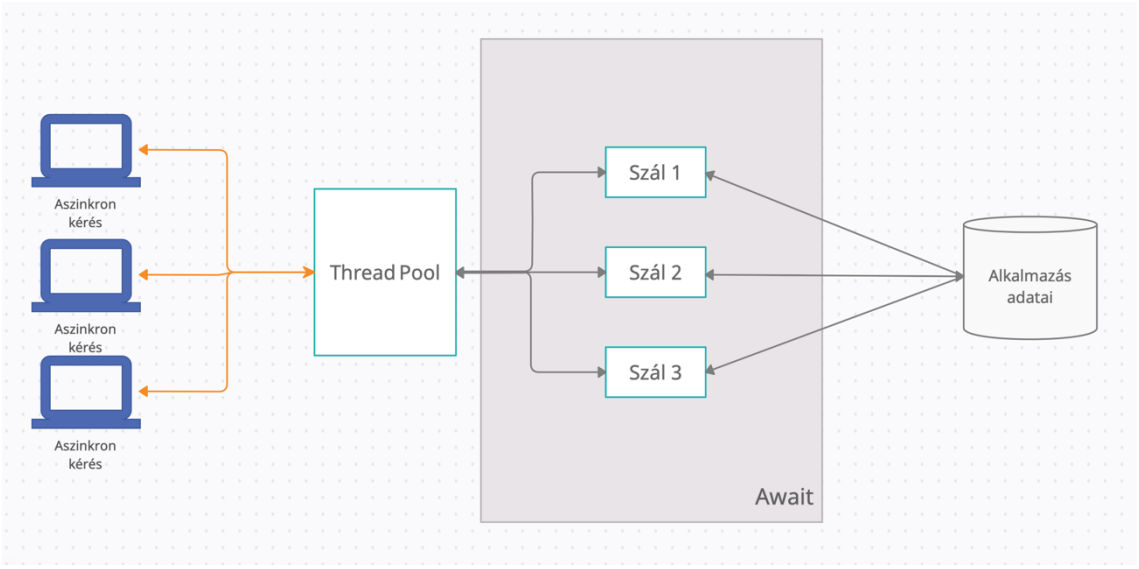
Réteg neve	Réteg feladata
Domain	Tartalmazza az alkalmazásban használt modelleket, adattároló osztályokat, segédmetódusokat, kivételeket
Repository	Biztonságos hozzáférést biztosít CRUD metódusok segítségével az adatbázishoz jól definiált, egyszerű interfészeken keresztül
Services	Az üzleti logikát implementáló osztályok gyűjtőhelye
WebAPI	Ellenőrzött hozzáférést biztosít az alkalmazás szolgáltatásaihoz a weben keresztül, a RESTful API elveit követve

5.1 táblázat: A szerveralkalmazás rétegeinek rövid összefoglalása

Mielőtt rátérnék a rétegek konkrét implementációjának tárgyalására, fontos megemlíteni, hogy a rétegek közötti adattovábbítás során, valamint minden időigényes művelet esetén aszinkron működést valósítottam meg. Ennek oka, hogy a fejlődő erőforrások, párhuzamos architektúrák miatt a modern alkalmazásokban alapvető elvárássá vált, hogy párhuzamos kódot készítsenek a fejlesztők. Ez egy olyan optimalizálást jelent, ami nem okoz plusz költséget a program futtatása során, cserébe jelentősen növelni tudja a teljesítményt sok, párhuzamos kérés esetén. Az ASP kontrolleréhez beérkező kérések befutássukkor egy szálhoz kerülnek hozzárendelésre, melyen keresztül processzoridőhöz jutnak majd, és elvégezhetik a műveleteiket. Ezek a szálak egy-egy kérés feldolgozása után újrahasznosításra kerülnek, és visszakerülnek egy váróba. Probléma akkor keletkezik, amikor ebben a *Thread poolban,* tehát az ASP által használt szálak gyűjtőhelyén, elfogynak a szabad szálak. Ez viszonylag hamar előfordulhat, hiszen az előbbiekből eleve korlátolt számú várakozik csak. Bár az ASP rendszere teljesen

önállóan kezeli a párhuzamos kéréseket, az olyan műveletek, mint például az adatbázisból olvasás hosszabb időre blokkolja a szálat, melyen a végrehajtás történik. Ha eközben érkezik egy új kérés, mely számára nem jut szabad erőforrás, az várakozik. Azonban ennek a kérésnek a kiszolgálására nem kerül kiosztásra a korábban említett, blokkolt szál, még akkor sem, ha az mondjuk 3 másodpercig nem is végez tényleges munkát, csupán válaszra vár valamelyik rétegben. Az aszinkron belső működés erre a problémára ad egy bizonyos mértékű megoldást azzal, hogy a várakozó szálak visszakerülnek a thread poolba, hogy az üresjáratuk idején is műveleteket végezhessenek. Természetesen nem tudjuk így sem biztosítani, hogy mindig minden kérés azonnal szálhoz jusson, azonban határozottan jobb kihasználtsági fokot érhetünk a meglévő erőforrásainkkal.

A C# természetesen nyelvi szinten támogatja a párhuzamosságot. Erre leginkább a Taskok használhatók, melyek egy konkurenciát támogató csomagba ágyazzák be a hagyományos utasítás-blokkjainkat: ugyanúgy műveletek hajtódnak végre, melyek végén tetszőleges visszatérési értékkel térhetünk vissza, azonban ezen felül olyan metódusokkal ruházták fel a Taskokat, melyek segítségével monitorozni tudjuk a végrehajtás állapotát, vagy akár le is állíthatjuk azt. Az implementáció során az aszinkron kódot használó metódusokat az *async* kulcsszóval jelöltem meg, ezután pedig a kódblokkban használhatóvá válik az *await* szó, melyet az utasítások elejére kell elhelyezni, ha azt szeretnénk, hogy az adott utasítás lefutását, esetleg eredményét megvárja a végrehajtás. Erre ugyanúgy használható a task.Result() metódus is, azonban nagy különbség van a kettő között: míg az utóbbi blokkolja a szálat, addig az await kulcsszó használatával a szál felszabadításra kerül. Az alábbi, 5.1 ábrán jól látható, hogy a kérések esetén, azok élettartalmát nézve, mely szakaszban kerülnek a szálak felszabadításra:



5.1 ábra: Az aszinkron kérések feldolgozásának módja

Ami a kollekciókat illeti, itt is a C# erősségére támaszkodtam: a yield-return mechanika lehetővé teszi, hogy aszinkron kollekciókat adjunk vissza. Ennek lényege, hogy amint rendelkezésre áll egy elem az adatforrástól, azt azonnal továbbítjuk, nem várjuk meg, hogy az összes entitás megérkezzen. Így tulajdonképpen egy folyamot alakítunk ki a programon belül, ahol egy-egy elem is közlekedhet a rétegek között, amint az rendelkezésre áll. A sarokpontokat természetesen a két végpontja jelenti az alkalmazásnak: a WebAPI-nál történő HTTP válasz, illetve az adatbázisból olvasás. Szerencsére mind a kettő megoldott, hiszen az EF képes folyamszerűen beolvasni adatokat, amennyiben nem használunk olyan metódusokat, mint például a *ToList()* vagy *ToArray(),* mivel ezek a teljes kollekció bufferbe töltésére kényszerítik a keretrendszert. Talán érdekesebb, hogy a WebAPI miként adja vissza az aszinkron kollekció elemeit: az ASP .NET Core 6-os verziója óta képes arra, hogy aszinkron visszaadja egy IAsyncEnumerable kollekció elemeit az API végpontja segítségével is. Ennek feltétele, hogy a beépített System.Text.Json könyvtárat használjuk a JSON szerializáláshoz a kontrollerben.

Az 5.2 ábrán látható a CourseRepository osztály egyik metódusa, mely azért felelős, hogy az adatbázis megfelelő táblájából visszaadja a kurzusokhoz tartozó entitásokat – mindezt a fentieknek megfelelően aszinkron módon, a yield return kulcsszavak használatával. Külön érdemes kiemelni az *Include()* metódus használatát az adattáblán. Az Entity Framework képes a *lazy loadingra,* vagyis a lusta betöltésre, melynek segítségével elérhető, hogy egy betöltött egyeden belül található referencia esetén csak akkor töltődjön be ténylegesen a másik entitás, ha arra a kódban hivatkozás történik. Ez látszólag megkönnyíti a programozó dolgát, hiszen nem kell explicit megmondania, hogy mikor, mi töltődjön be, azonban mégis kerülendő ennek használata, hiszen a teljesítményt jelentősen rontani tudja: a híres N+1 lekérdezés probléma is előállhat. Az alábbi példán keresztül bemutatva, ennek lényege, hogy lazy loading esetén egy darab lekérdezés segítségével megkaphatjuk az összes kurzust (ez az 1 a képletben), majd az ORM annyi új lekérdezést indít, amennyi kurzusunk van (tehát +N darabot), annak érdekében, hogy megkaphassuk a hozzájuk tartozó UserCourses elemet, mely egyébként a kapcsolótábla egyede. Természetesen N darab új lekérdezést indítani teljesen pazarló megközelítés, ezzel szemben az *Include()* használata úgynevezett *eager loadingra,* vagyis mohó betöltésre támaszkodik. Ebben az esetben már korán értesítjük az Entity Frameworköt, hogy szükségünk lesz a hivatkozott egyedekre is, így azokat rögtön betölti, és az eredeti lekérdezés részeként vissza is adja.

```
public async IAsyncEnumerable<Course> GetAllAsync()
        {
            await foreach (var item in db.Courses.Include(x => x.UserCourses).AsAsyncEnumerable())
            {
                yield return item;
            }
        }
```

5.2 ábra: A CourseRepository.cs állomány néhány sora

5.1.1. DOMAIN RÉTEG

A domain az alkalmazás magját képezi. Ahogy arra a tervezés fejezetben is utaltam, a domain-driven designnak megfelelően ez egy olyan réteg, melynek elemeire az alkalmazás összes többi pontja építeni tud. Ennek megfelelően előfordult az implementáció során, hogy logikailag egymáshoz kevésbé kapcsolódó elemek is egymás mellé kerültek ebben a rétegben.

A webalkalmazás implementációját ezzel a réteggel kezdtem, hiszen itt találhatóak az adattárolás szempontjából a legfontosabb elemek, a modell osztályok. Ezek alatt technikailag olyan egyszerű C# osztályokat értek, melyek csupán auto-propertyket tartalmaznak, így az Entity Framework számára értelmezhetőek, és jól leírják a relációkat az alkalmazás entitásai között. Az egy-az-egyhez kapcsolatok kifejezetten egyszerűen leírhatók: elhelyezünk egy-egy referenciát a másik entitásra mindkét osztályunkban, később az EF ezeket a tényleges tábláinkban idegenkulcsokra cseréli majd. Az egy a többhöz típusú adatbázis kapcsolatok is könnyedén modellezhetők: abban az entitásban, melyhez több másik kapcsolódik, elhelyezünk egy, a másik típus elemeiből álló kollekciót, míg a kapcsolat másik oldalán egy konkrét referenciát adunk meg arra a típusra, melynek egyedei közül az egyikhez kapcsolódni fogunk ezzel az entitással. Az Entity Framework egyik nagy előnye, hogy nem kell idegenkulcsokkal vesződnünk, hiszen a tervezők megalkottak egy olyan működést, melyre csak *navigációs tulajdonságok* néven hivatkoznak. Ennek lényege, hogy az imént említett referenciákat életre kelti a keretrendszer, ezek használhatóak lesznek futásidőben is. Tehát az adatbázis elemeit a kódban úgy tudjuk kezelni, mintha azokat mi magunk hoztuk volna létre, az alkalmazáson belül, valójában pedig ezek egy külső adatszolgáltatótól érkeznek. Ennek a működésnek egy hátulütőjével találkoztam a fejlesztés során: amikor a WebAPI-ig jut egy ilyen navigation propertyt tartalmazó entitás, akkor a JSON-be történő szerializáció során természetesen ez is beágyazásra kerül a válaszba, minden saját tulajdonságával együtt. Így viszont körkörös beágyazás alakulhat ki, mivel a másik típus is tartalmazhat hivatkozást az eredeti entitásra. Ez egészen addig megy, míg el nem érjük a HTTP válasz fizikai méretkorlátját, ezután pedig kivétel keletkezik. Megoldást jelentenek az annotációk: ezeket a nyelvi elemeket az EF esetén jól ki tudjuk használni, hogy extra információkat adjunk az egyes tulajdonságokról a framework számára, mintegy

iránymutatást adva azok használatáról. Ebben a konkrét esetben a *[JsonIgnore]* attribútummal dekoráltam azokat a mezőket, melyeket futásidőben használtam, azonban a JSON válaszokban semmiképpen sem szerettem volna viszontlátni.

A kapcsolódási típusok közül egyértelműen több problémát jelent a több-a-többhöz kapcsolatok implementálása. Bár erre a legújabb Entity Framework verzió már sokkal egyszerűbben ad lehetőséget, a fejlesztés kezdetének idején még csak egy bonyolultabb megoldás létezett. Ennek részeként létre kell hoznunk kapcsoló entitásokat, majd a két eredeti típusunkban ilyen kapcsoló entitásokból álló kollekciót helyezünk el, egyet-egyet. Ezután még az adatbázisunk kontextusában (ez az az osztály, ahol elsődlegesen konfigurálni tudjuk, hogy az EF milyen táblákat hozzon létre, illetve egyéb finomhangolásokat is eszközölni tudunk) specifikálnunk kell, hogy ez a több-a-többhöz kapcsolat valóban fennáll. Ennek a módszernek a támogatása továbbra is megmaradt, annak ellenére, hogy az Entity Framework 6. verziója már implicit is el tudja végezni ezeket a műveleteket. Az ok, amiért ez a látszólag bonyolultabb, több konfigurációt igénylő út is ajánlva maradt a fejlesztők által, egyszerű: előnyös lehet, hogy egyedi mezőket is konfigurálni tudunk a kapcsolótáblában. Például, ha rekordonként el szeretnénk tárolni, hogy a kapcsolat mikor jött létre, arra így van lehetőségünk, mondjuk egy dátummező elhelyezésével a kapcsoló entitásban. Ez utóbbiak természetesen szintén a domain rétegben kerültek meghatározásra.

5.1.2. REPOSITORY RÉTEG

A repository réteg feladata, hogy ellenőrzött módon, egyszerű műveletek segítségével biztosítson hozzáférést az adattáblákhoz. Alapvetően CRUD, tehát Create, Read, Update, Delete – vagyis létrehozás, olvasás, frissítés, törlés műveleteket kell megoldania egy repositorynak. Mivel ez minden egyes típus esetén elmondható, így definiáltam egy generikus IRepository<T> interfészt, melyet minden konkrét implementáció használ. A fő feladata ezeknek az osztályoknak, hogy az egyszerű műveletek megvalósításán keresztül „építőköveket” biztosítsanak a szolgáltatások bonyolult logikai műveletei számára. Emellett fontos, hogy az adatbázis-közeli entitásokat elrejtsék, hiszen biztonsági kockázatot jelenthet, ha az ezekhez kapcsolódó információk a felsőbb rétegekbe szivárognak, esetleg egészen a kliensekig eljutnak. Nem szeretnénk például, ha a szolgáltatások réteg tudná, hogy hogyan történik a jelszavak ellenőrzése, vagy éppen az új egyedi azonosítok legenerálása. Emellett az adattárolás mikéntjére sem vagyunk kíváncsiak a felsőbb szinteken, hiszen a repository réteg alatt lévő tényleges adatbázis megoldás bármikor cserélhető kell legyen.

5.1.3. SERVICE RÉTEG

A service rétegben találhatóak a bonyolult logikai műveleteket tömörítő osztályok. Ezek az egyszerű CRUD metódusok továbbításán túl olyan műveleteket is biztosítanak,

melyekhez esetenként több repositoryra is szükség lehet. Emellett ennek a rétegnek a feladata az is, hogy a WebAPI irányából érkező nyers entitásokat – adattároló, úgynevezett DTO osztályokat, vagyis a különböző kliensoldali kéréseket leíró osztályok példányait – leképezze olyan típusokra, melyeket az adatbázisban tárolni fogunk tudni. Ennek során sok helyen ki tudtam használni, hogy a DTO osztályok, valamint a valós modell entitások között átfedés van: természetes, hogy sok tulajdonságuk megegyezik. Szerencsére rendelkezésre áll egy könyvtár, az AutoMapper, mely képes ezeket a konverziókat magától elvégezni, így nekem csak az olyan különleges tulajdonságok beállításával kellett foglalkoznom, mint például az azonosítók, vagy a navigációs tulajdonságok. Ezzel jelentősen csökkenthető a boilerplate kód mennyisége, hiszen az automapper beállítása pár sort vesz csupán igénybe, és akár 10-20 tulajdonság kézi beállítását is megspórolhatjuk így. Emellett a változásokra is jobban reagálhatunk így, hiszen, ha módosításra kerül az adattároló entitás, illetve a modell osztály is, akkor nem kell kézzel újrakonfigurálnunk a konverziót szerte a kódbázisban, hiszen az AutoMapper segítségével ez automatikusan megtörténik mindenhol.

5.1.4. WEBAPI RÉTEG

Ez a réteg felelős a webes kiszolgálásért, a beérkező kérések kezeléséért. Az itt található osztályok jelentős része kontrollerként funkcionál, vagyis az ASP valamilyen beépített controller ősosztályának leszármazottai. Ezekben az osztályokban használhatunk belső, rejtett segédmetódusokat is, valamint megjelölhetjük azokat a metódusokat, amelyeket végpontnak szánunk, hogy megfelelő paraméterek esetén ezekben csapódjanak le a beérkező kérések.

A kontrollerek metódusainak visszatérési értékei jellemzően valamilyen formájú *ActionResultok*, vagy esetleg konkrét típus, kollekciók lehetnek. Mi magunk szabályozhatjuk, hogy adott esetben milyen HTTP státuszkóddal térjen vissza az alkalmazás, azonban ezt sok esetben felülírhatja az ASP.

HTTP állapotkód	Jelentése
200	„Ok” – sikeres művelet
204	„No Content” – nem keletkezett hiba, sikerese volt a kérés, azonban nem tért vissza semmilyen adat
302	„Found” – átirányítási kérés másik címre
400	„Bad Request” – hibás kérés
401	„Unauthorized” – hibás authentikáció esetén keletkező válasz

404	„Not Found” – a kért erőforrás nem található
405	„Method Not Allowed” – hibás HTTP kérésfajta esetén keletkezhet, például, ha GET kéréssel fordulunk egy POST-ot váró végponthoz

5.2 táblázat: Néhány gyakori HTTP állapotkód és jelentésük

A fentiek alapján látható, hogy hiába térnénk vissza Ok() státusszal (amit az ASP természetesen 200-as HTTP státuszkódra konvertálna), egy olyan metódusban, amely authentikációt igényel, ha a beérkező kérést például nem sikerül azonosítani, akkor automatikusan 401 „Unauthorized” választ fog küldeni a kontroller a válaszban.

Ez a réteg is erősen támaszkodik az annotációk nyújtotta lehetőségekre. A kontrollereink konfigurálását is ezeken keresztül tehetjük meg, valamint az egyes metódusokat is így címkézhetjük fel, hogy milyen URL címen, milyen paraméterrel legyenek elérhetők, mely csoportok férhessenek hozzá a végponthoz, valamint, hogy ezek milyen HTTP szavakra reagáljanak. Példaként álljon itt egy, az authentikáció kontrollerében található metódus kódja:

```
[Route("register")]
[Authorize(Roles = "Admin")]
[HttpPost]
public async Task <ActionResult> InsertUser([FromBody] RegisterData model)
{
    var currentUserName = this.User.FindFirst(ClaimTypes.NameIdentifier).Value;
    model.RegisteringUserName = currentUserName;
    if (await userService.Register(model))
    {
        return Ok();
    }
    else
    {
        return BadRequest();
    }
}
```

5.3 ábra: A WebAPI rétegben található AuthController.cs tartalmának egy részlete

Az 5.3 ábrán látható, hogy ez a metódus a jelenlegi kontroller címe utáni „/register” címen lesz elérhető, valamint POST kérésekre reagál majd csak. Emellett a kontroller ellenőrzi, hogy be van-e jelentkezve a felhasználó, és az „Admin” csoporthoz tartozik-e. Utóbbit a kérés fejlécében továbbított tokenben található claimből olvassa ki a rendszer. Ennek működéséről az 5.2-es, a kliens-oldali alkalmazást tárgyaló fejezetben írok részletesebben.

A fordítóprogram

Az alkalmazás szintén fontos pontja a forráskódokat feldolgozó modul. Mint ahogy a tervezés során is említettem, a Roslyn API egy funkcionálisan szerteágazó interfész, melyet egy saját megközelítéssel próbáltam meg leegyszerűsíteni.

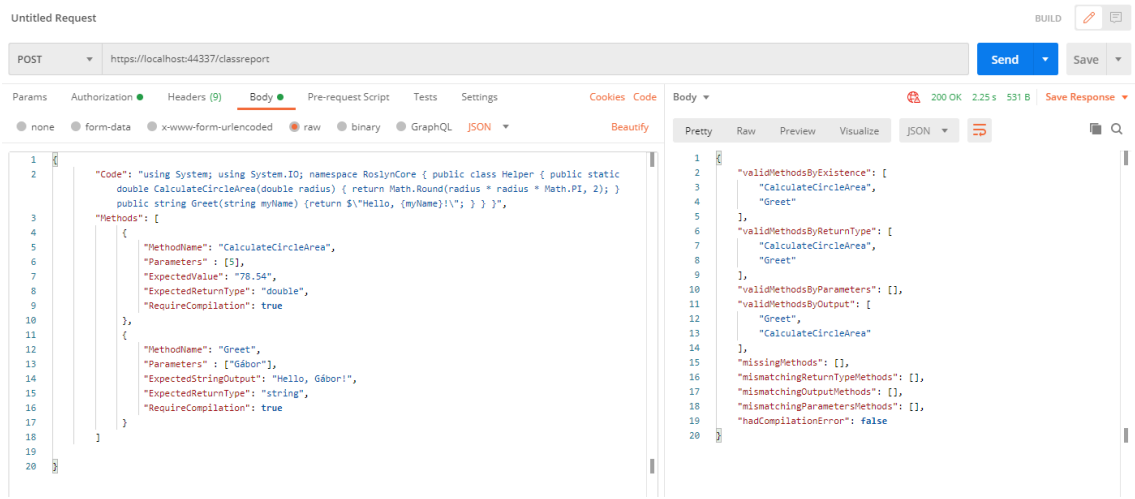
A modul bemenetként egy osztályt vár, de előfordulhat, hogy a feladat alapból nem írja elő az osztály használatát. Ennek hiányában az adott metódust a kliens beágyazza egy egyszerű osztályba, és ezt küldi feldolgozásra a szerver felé. Rengeteg kombináció mutatkozik arra, hogy ez miképpen lehetséges (például leggyakrabban a metódus-szignatúra ellenőrzése, esetleg a kimenetek összevetése). Az ötletet a megvalósításhoz a Gang of Fourból ismert builder tervezési minta adta. Ennek során létrehoztam egy „nyelvtant”, ami a builder objektum alapjául szolgál: ez gyakorlatilag egy interfészgyűjtemény, melynek elemei együtt egy hierarchiát alkotnak, és meghatározzák, hogy melyik metódus milyen sorrendben hívható – tehát hogyan lehet „felépíteni” egy ellenőrzési logikát. Az 5.4 ábrán a nyelvtant alkotó interfészek láthatók:

```
namespace project.Domain.Helpers.ClassReportBuilder
{
    public interface ICanCallBuild
    {
        ClassReport Build();
        ICanRequireCompilation WithMethod(string methodName);
        ICanRequireCompilation WithMethod(string methodName, string expectedReturnType);
        ICanRequireCompilation WithMethod(string methodName, ParamList expectedParameters);
        ICanRequireCompilation WithMethod(string methodName, ParamList expectedParameters, string expectedReturnType);
    }

    public interface ICanRequireCompilation : ICanCallBuild
    {
        ICanCompile AlsoCompile();
    }

    public interface ICanCompile : ICanCallBuild
    {
        ICanCompile WithParameters(object[] parameters);
        ICanCompile SetExpectedStringOutput(string expectedStringOutput);
        ICanCompile SetExpectedValue(object expectedOutput);
    }
}
```

5.4 ábra: Az interfészek egymásra épülése, vagyis a nyelvtan



5.5 ábra: A report-készítő demonstrációja.
Balra: a kérés teste, benne a forráskóddal és a kért validációkkal.
Jobbra: a kapott válasz, benne a validációs szempontokkénti eredményekkel

Az 5.5 ábrán látható választ adó végpontot természetesen csak tesztelés és demonstráció szempontjából hoztam létre. Valójában az elküldött programkódok az adatbázisba kerülnek, majd a tesztkitöltési próbálkozás után minden kérdés kiértékelésre kerül, így a programkód ellenőrzése is megtörténik. Ezután a válaszban látott szempontok alapján pontozza a rendszer a próbálkozást.

5.2. A KLIENSOLDALI ALKALMAZÁS

A kliens-oldalon fejlesztésre kerülő alkalmazás egyik talán könnyebb, ám határozottan érdekes technológia döntése volt a felhasználói felület megalkotásához szükséges könyvtár kiválasztása. Természetesen az olyan ismertebb, szélesebb körben is elterjedt UI megoldásokra is támaszkodhattam volna, mint a *Bootstrap*, azonban ez önmagában főleg egyszerűbb, gyorsan elkészíthető felületek esetén hasznos. Mivel, ahogy az a 4.1-es fejezetben is olvasható, a viszonylag friss Blazort választottam a kliensoldali alkalmazás megvalósításához, így rengeteg új, feltörekvő UI megoldás közül választhattam.

Ezek közül a *MudBlazor* nevű könyvtárra esett a választásom. Ez egy nyílt forráskódú projekt, mely támaszkodik a korábban említett Bootstrapre, és a felhasználói felület dizájnja szempontjából pedig elsősorban a Google-höz köthető „material design” irányvonalát képviseli. A széles eszközpalettával rendelkező könyvtárral szép, letisztult felületek építhetők, emellett a dokumentációja is megfelelően részletes, illusztratív.

5.2.1. AUTHENTIKÁCIÓ, AUTHORIZÁCIÓ A KLIENSBEN

Az authentikáció folyamata egy dedikált oldalon keresztül indul. Itt egy űrlap fogadja a felhasználót, egyszerű validációkkal ellátott mezőkkel. Hibás adatok megadása esetén hibaüzenet jelenik meg egy felugró ablakban, valamint kiürítésre kerül a felhasználónév és jelszó páros is.

A sikeres bejelentkezés után a végpont által visszaadott értékeket a kliens a böngésző localStorage tárolójában eltárolja. Ennek írása csak ezen a ponton történhet meg, hiszen ebbe a tárolóba kulcs-érték párokat tehetünk el - hasonlóan egy nagyon leegyszerűsített NoSQL adatbázishoz - így nem volna szerencsés, ha ellenőrizetlenül módosítana ezen a rekordon az alkalmazás valamely komponense a későbbiekben. A localStorage tárolóhoz való hozzáféréshez egy *ILocalStorageService* implementációt használtam, melyet az IoC container dependency injectionnel biztosít. Ezt nem kell különösebben konfigurálnunk, elegendő a megfelelő csomagot a projekthez adni, majd egy servicet kell hozzáadnunk az alkalmazás konfigurálásáért felelős builder objektumhoz a Main függvényben.

Ahhoz, hogy az alkalmazás ellenőrizni tudja az authentikációt, például az ezt megkövetelő oldalak betöltése esetén, vagy egy-egy kérés elküldése előtt, szüksége van egy *AuthenticationStateProviderre.* Az állapotról szóló információ érkezhet a gyári megoldástól is, azonban én egy saját implementációt készítettem *LocalAuthenticationStateProvider* néven, mivel ebben könnyedén felbonthatom a tokent, elmenthetem a kapott adatokat a korábban említett tárolójába a böngészőnek. Szerencsére nem szükséges minden egyes megjelenített oldalon újra megszereznünk a fent említett helyi authentikációs szolgáltatónak egy példányát, támaszkodhatunk a Blazor - nagyon jól kidolgozott - paraméter-átadási rendszerére ebben az esetben. A View-k, vagyis a nézetek, melyeket megjelenítünk, Razor komponens formátumúak. Ezek az oldalon való navigálás közben egymásba ágyazódnak, így egy nagyon kényelmes mechanika kínálkozik, mégpedig a paraméterek kaszkádolásának lehetősége. Nincs más dolgunk, mint azoknak a komponenseknek a kódblokkjában ellátni a *[CascadingParameter]* attribútummal egy *Task<AuthenticationState>* típusú változót, melyekben szükségünk van authentikációval kapcsolatos információkra. A komponens betöltésekor a felsőbb szintekről tovagyűrűző referenciát „elkapja” a keretrendszer, és betölti ebbe a változóba, amit ezután nyugodtan használhatunk. A *Task<AuthenticationState>* szerkezetből látható, hogy ez a szolgáltató is aszinkron működik. Ennek oka nem sokkal tér el az 5.1-es fejezet elején tárgyaltaktól: a bejelentkezéssel kapcsolatos adatok érkezhetnek akár külső forrásból is, de minden esetben elmondhatjuk, hogy az adatok kiolvasása egy időigényes, a végrehajtást blokkoló feladat. Az aszinkron mechanika korábban említett előnyeit itt hatványozottan kiaknázhatjuk, hiszen nem csak a szálat szabadítjuk fel, de a UI esetén annak reszponzivitást is megőrizzük.

Az ASP beépített, Identity-alapú felhasználó-kezelő rendszere a tokeneket, valamint azok lejárati idejét a szerveroldali alkalmazás segítségével egyaránt nyilvántartja az

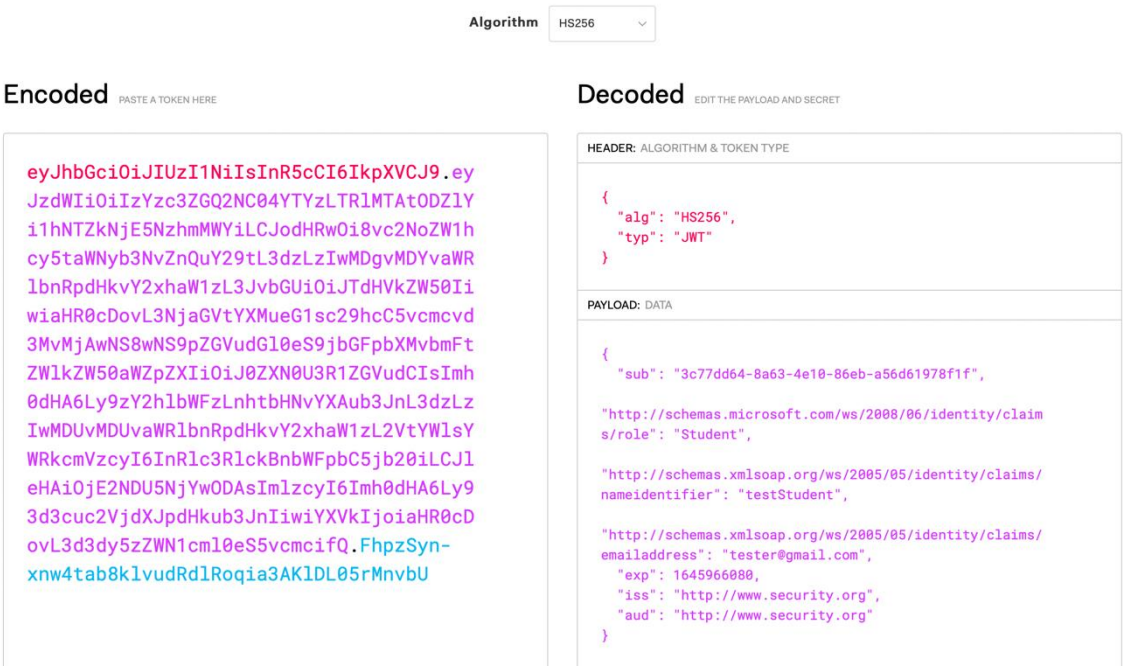
adatbázisban. A JWT formátum használatának nagy előnye, hogy nem kell mindig továbbítani olyan adatokat a böngészőből a végpont felé, mint például a role, vagyis a csoport, melynek tagja a felhasználó, mivel a tokenbe eleve bele van kódolva ez az információ. Ezeken az úgynevezett claimeken keresztül egyszerűen és biztonságosan tudunk szenzitív információkat közölni, pusztán a token továbbításával. Az 5.6 ábrán látható, hogy a kliens milyen adatokat kap meg a bejelentkezés során:


```
{
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
        eyJzdWIiOiIzYzc3ZGQ2NC04YTYzLTRlMTAtODZlYi1hNTZkNjE5NzhmMWYiLCJodHRwOi8vc2NoZW1hcy5taWNyb3NvZnQuY2
        9tL3dzLzIwMDgvMDYvaWRlbnRpdHkvY2xhaW1zL3JvbGUiOiJTdHVkZW50IiwiaHR0cDovL3NjaGVtYXMueG1sc29hcC5vcmcv
        d3MvMjAwNS8wNS9pZGVudGl0eS9jbGFpbXMvbmFtZWlkZW50aWZpZXIiOiJ0ZXN0U3R1ZGVudCIsImh0dHA6Ly9zY2hlbWFzLn
        htbHNvYXAub3JnL3dzLzIwMDUvMDUvaWRlbnRpdHkvY2xhaW1zL2VtYWlsYWRkcmVzcyI6InRlc3RlckBnbWFpbC5jb20iLCJl
        eHAiOjE2NDU5NjYwODAsImlzcyI6Imh0dHA6Ly93d3cuc2VjdXJpdHkub3JnIiwiYXVkIjoiaHR0cDovL3d3dy5zZWN1cml0eS
        5vcmcifQ.FhpzSyn-xnw4tab8klvudRdlRoqia3AKlDL05rMnvbU",
    "expiration": "2022-02-27T12:48:00Z",
    "role": "Student",
    "username": "testStudent"
}
```


5.6 ábra: A bejelentkezésért felelős API végpont által visszaadott adatok sikeres azonosítás után

Látható, hogy néhány adat a tokenen felül is továbbításra kerül. Ennek oka, hogy a kliens így gyorsabban jut ezekhez az információkhoz, és a felhasználói felület kialakítása során rögtön hasznát is vettem ezeknek a mezőknek. A lejárati dátumot azért érdemes továbbítani, eltárolni, mert bár a szerveroldal az adatbázisban tárolt adataira támaszkodva minden kérés feldolgozása során validálja, hogy az adott token nem járt-e még le, a frontenden ebből automatikusan legfeljebb annyit érzékelhetünk, hogy egyik pillanatról a másikra hibára futunk, hiszen két kérés között éppen lejárhat a tokenünk élettartama. Ennek megoldására használtam a korábban már említett saját AuthenticationStateProvider implementációt: az állapot kiadásakor a szolgáltató kiolvassa a lejárati dátumot a böngésző localStorage tárolójából, és amennyiben az lejárt, törli a jelenlegi felhasználóhoz tartozó rekordokat. Ezután bármelyik, bejelentkezést igénylő oldalon az authorizációs attribútumok tiltani fogják a felhasználót, aki átirányításra kerül egy nyitóoldalra, ahol újra bejelentkezhet.

Ami a nyers tokent illeti, ez egy hashelt karakterlánc, mely jelen esetben egy szimmetrikus algoritmussal, HS256-os eljárással készül. Ha egy erre alkalmas program segítségével dekódoljuk a tokent, láthatjuk, hogy milyen claimeket, vagyis kódolt információkat tartalmaz:

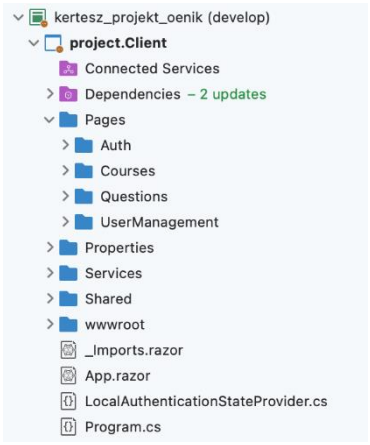


5.7 ábra: Egy, a rendszer által generált JSON Web Token dekódolva, jobbra a claimekkel, melyeket tartalmaz

A fenti, 5.7 ábrán jól látható, hogy a token eleje egy header rész, mely metainformációkat tartalmaz, majd ezt követik a kulcs-érték párok, melyek a tényleges információkat hordozzák. Ezek a kulcsok többféleképpen választhatók, én igyekeztem a sztenderdizált, például jelen esetben a Microsoft és más szervezetek által meghatározott neveket használni, de természetesen teljesen szabadon felvehetünk saját claim típusokat.

5.2.2. A KLIENS ALKALMAZÁS FELÉPÍTÉSE

A kliensoldali alkalmazás felépítésében egészen eltér a szerveroldalitól, hiszen merőben más jellegű és szerepkörű alkalmazásokról beszélünk. Lévén, hogy a Blazor WebAssembly képében még egy korai, formálódó keretrendszerről beszélünk, így az interneten, könyvekben fellelhető segédanyagok tárháza is csak most kezd el bővülni, de szerencsére a Microsoft hivatalos oldalán található dokumentáció kellően részletes, naprakész, így főleg innen tájékozódva könnyen tudtam dolgozni a keretrendszerrel. Az belátható, hogy a projekt strukturálására itt is szükség van, hiszen nem egy pár kódsoros frontend applikációt készítek, hanem viszonylag sok osztállyal, nézettel kellett dolgoznom a munkám során. Éppen ezért a következő, 5.8 ábrán látható projektstruktúrát alakítottam ki:



5.8 ábra: A kliensoldali alkalmazás szerkezete

A mappák – a .NET berkein belüli konvencióknak megfelelően – a névtereket is alakítják, így a struktúra nem csak a fájlokat tekintve, de a kódbázisban is előáll. A project.Pages névtér tartalmazza a weblap különböző oldalaihoz tartozó komponenseket. A Services könyvtárban találhatóak a különböző szolgáltatások: ezek jellemzően hagyományos C# osztályok. Feladatuk, hogy a WebAPI-hoz intézett kéréseket megfelelően továbbítsák, a válaszokat feldolgozzák, az oldalak számára megjeleníthető formában továbbadják azoknak. Ezeknek a serviceknek a példányait a IoC container fogja kezelni, igény esetén biztosítani az oldalak számára, miután ezt beállítottuk az alkalmazás belépési pontján, a Program.cs állományban. Egyéb konfigurációkat is itt eszközölhetünk, bár jelentősen kevesebb beállításra volt szükségem, mint a szerveroldali alkalmazás esetén.

Lévén, hogy az oldalak csupán kódblokkal rendelkeznek, nincs konstruktoruk, így a függőségek esetén a dependency injectiont az *@inject* direktívával kérhetjük a konténertől. Ilyen módon kell hozzáadni a gyári szolgáltatásokat is az oldalhoz, amiket használni szeretnénk. Igen gyakran szükségem volt például egy *NavigationManager* implementációra, mivel ennek segítségével tudtam navigálni az oldalaim között.

Ennek használatát azonban érdemes egy magasabb absztrakciós szintre emelni. Mivel a Single Page Application keretrendszerek esetén – ahogy azt a technológia neve is mutatja - valójában nem navigálunk több fizikai oldal között, csupán egy „nézeten” belül cseréljük a megjelenített információkat, így a navigáció kérdése mindenképpen egy sarokpontot jelent. A komponensek újrahasznosíthatósága révén - melyről a következő bekezdésekben beszélek részletesebben - előfordulhat például, hogy egy-egy oldalra több helyről is el tudunk jutni. Ilyenkor célszerű lenne, ha a művelet végeztével a hívás helyére, vagyis az előző komponensre jutnánk vissza. Ez alapesetben nehezen megoldható, hiszen a rendszer alapértelmezetten nem rendelkezik

Az oldalak megjelenítéséhez az úgynevezett Razor komponenseket használhatjuk, mint fő építőelemeket. Ez egy teljesen új koncepció az ASP égisze alatt, mely csak Blazor appokban elérhető. Az elnevezés talán nem tökéletes, hiszen a korábbi Razor Pages,

valamint MVC applikációkból megszokott Razor oldalak más felépítésűek is voltak, valamint más célt is szolgáltak. A Razor komponensek nem tartalmazhatnak kontroller logikát (hiszen eleve a frontenden használatosak, így erre szükség sincs), viszont mögöttes kódot, egy kódblokk formájában igen. Az elnevezés megtartása valószínűleg onnan ered, hogy ezeken a komponenseken belül is Razor szintaxissal dolgozhatunk, tehát a HTML tagek között a „@” jellel egy utasítást, valamint „@{}” jelöléssel egy egész kódblokkot adhatunk meg. Az oldalak elején direktívákat sorolhatunk fel, a következő táblázat ezek közül mutat be néhány gyakran használatost:

Direktíva	Feladata
@page "/ut/vonal"	Meghatározza, hogy a jelenlegi oldal milyen URL címen jelenik meg. Ezt az út a domainhez hozzáfűzve értendő.
@layout AuthLayout	Biztosítja, hogy az oldal tartalma a megadott Layoutba, vagyis elrendezésbe kerüljön beágyazásra. Saját layoutokat is létrehozhatunk.
@inject NavigationManager navigationManager	Beszúr egy megfelelő típusú komponenst, amely a megadott nevű változón keresztül elérhetővé válik az egész oldalon.
@attribute [Authorize]	Meghatároz egy attribútumot, mely az egész oldalra érvényes

5.3 táblázat: Néhány gyakori Blazor direktíva, és jelentésük

A Razor komponensek hatalmas előnye, hogy újrahasznosíthatók. Elég egyszer létrehoznunk ezeket a modulokat, majd a nevüket egy HTML-szerű tagbe foglalva bármelyik másik komponensbe beilleszthetjük őket. Valójában ezek, és a direktívák is makróként funkcionálnak, fordításkor a preprocesszor dolgozza fel és alakítja át őket a megfelelő utasításokra. Lehetőségünk van paramétereket is átadni a komponensek között, ezt főleg a korábban ismertetett kaszkádoló paraméterek kapcsán találtam hasznosnak.

A tesztek kérdésekkel feltöltésére szolgáló felületen például elhelyeztem egy tesztválasztó komponenst, ennek feladata, hogy megjelenítse a jelenlegi felhasználó tesztjeit, kurzusok szerint rendezve. Ebből a tanár kiválaszthat egyet, azzal a céllal, hogy a kérdésbankból hozzárendeljen a rendelkezésre álló kérdések közül néhányat. Ez a komponens a felület jobb oldalán található, feladata, hogy megjelenítse a felhasználó saját kérdésein túl azokat is, melyeket más tanárok címkékkel láttak el, tehát megosztottak mások számára. A felhasználó ebből is kiválaszt egyet, melyet szeretne az előzőekben jelölt teszthez rendelni. Ekkor, ahhoz, hogy a kérdésbank komponens ezt a műveletet

végre tudja hajtani, szüksége lesz a kiválasztott teszt azonosítójára. Mivel a felületen a két komponens mellérendelésben szerepel, ezért a teszt szelektornak először felfelé kell propagálnia, hogy kiválasztásra került egy teszt: ez egy callback függvényen keresztül történik. Az ős komponensből a kérdésbank számára átadni az információt már szerencsére egyszerűbb, hiszen erre használható a korábban említett kaszkádoló paraméter. Az alábbi, 5.9 ábrán szereplő kódsor a hossza miatt talán kevésbé illusztratív, ám mégis szeretném bemutatni, mivel ezen keresztül jól láthatjuk, hogy hogyan épül fel egy Razor komponens, és több ilyen elem milyen kapcsolatban állhat egymással. Figyeljük meg, hogy a TestSelector komponens OnClick eseményére milyen könnyedén fel tudunk iratkoztatni egy metódust, jelen esetben a ClickHandlert, melyben aztán könnyedén eszközölhetők a szükséges módosítások: a *selectedTest* beállítása, egy címke szövegének aktualizálása, valamint a sikeres kiválasztásról tájékoztató üzenet megjelenítése egy MudBlazor specifikus felugró ablakban, a Snackbarban.

```
@page "/questions"
@attribute [Authorize(Roles = "Teacher")]
@inject ISnackbar Snackbar

<MudGrid Justify="Justify.SpaceEvenly">
    <MudItem xs="12" md="6">
        <MudPaper Class="d-flex align-right justify-center mud-width-full py-8">
            <div class="row">
                <div class="col-12">
                    <MudText Typo="Typo.h2" Align="Align.Center">Teszt kiválasztása</MudText>
                </div>
                <div class="col-12">
                    <TestSelector OnClick="ClickHandler"></TestSelector>
                </div>
            </div>
        </MudPaper>
    </MudItem>
    <MudItem xs="12" md="6" >
        <MudPaper Class="d-flex align-left justify-center mud-width-full py-8">
            <div class="row">
                <div class="col-12">
                <MudText Typo="Typo.h2" Align="Align.Center">Kérdésbank</MudText>
                </div>
                <CascadingValue Value=@selectedTest>
                    <div class="col-12">
                        <QuestionBank></QuestionBank>
                    </div>
                </CascadingValue>
            </div>
            </MudPaper>
    </MudItem>
</MudGrid>


@code {
    private Test selectedTest { get; set; }

    string chipText { get; set; } = "Nincs kiválasztott teszt";

    void ClickHandler(Test selectedTest)
    {
        this.selectedTest = selectedTest;
        Snackbar.Add($"Kiválasztott teszt: {selectedTest.Title}", Severity.Info);
        chipText = "Kiválasztott teszt: " + selectedTest.Title;
    }
}
```

5.9 ábra: A questions.razor fájl tartalma

A dinamikusan változó oldalak is kihívást jelentenek egy modern frontend alkalmazás esetén. Egyes kérdések szerkesztőfelületén például előfordulhat olyan eset, hogy egy sor, opció hozzáadása után egy újabb sornak kell megjelennie, mindezt addig ismételve, amíg a felhasználó be nem fejezi a kérdés szerkesztését, és el nem küldi azt. Ebben az esetben a Blazor biztosította dinamikus komponensekre támaszkodtam. Ezek olyan komponensek, melyeket futásidőben tudunk létrehozni, kirenderelni, így nem kell sem előre a kódba égetni elrejtett komponenseket, sem pedig a láthatóságokukat állítani. Emellett a komponensfajtákat is dinamikusan tudjuk cserélni: célszerű egy *DynamicComponent* típusú kollekciót létrehozni, melynek minden elemét megjelenítjük a HTML oldalon. Amíg üres a lista, természetesen nem jelenik meg semmi, majd, amikor egy komponens hozzáadásra kerül, akkor a rendszer automatikusan érzékeli, hogy változás történt az oldal szerkezetét meghatározó komponensek között, így invalidálja a nézetet, és újrarendereli azt. Ekkor már meg is fog jelenni a legújabb komponensük, melynek típusát akár a felhasználói inputtól is függővé tehetjük, így gyakorlatilag bármilyen típusú komponens kirajzolható ilyen módon. Mivel egy lista elemeihez adjuk hozzá ezeket, így a dinamikus komponens létrehozásakor a korábban már említett @ref attribútum segítségével össze is tudjuk kötni ezeket az entitásokat, tehát azok adattagjait is olvasni tudjuk.

Korábban, a razor komponensek felépítése kapcsán példaként említettem, hogy gyakran van szükség bizonyos szolgáltatásokra, többek között például NavigationManagerre, aminek segítségével az oldalaink közötti navigálást tudjuk vezérelni. Az átirányításokat azonban egy magasabb absztrakciós szinten szerettem volna kezelni, így kiszerveztem az ide kapcsolódó logikát egy PageBase névre hallgató, saját komponensbe, a valós oldalaimat pedig ebből a bázisoldalból származtattam. Ezen mechanika segítségével elértem azt, hogy a navigáció során a felhasználó mindig a PageBase változatai között ugrál, az ős pedig tartalmaz egy referenciát egy másik komponensre, mely a navigációs történetet tárolja. Ez a PageHistoryState osztály, mely singletonként funkcionál, és a frontend alkalmazás teljes élettartama alatt rögzíti és tárolja egy listában a bejárt komponensek útvonalát. Ennek a listának az olvasásával bármikor tudunk visszafelé navigálni, ami azért hasznos funkció, mivel a korábban említett komponens-újrafelhasználhatóság révén gyakran előfordulhat olyan eset, hogy egy-egy komponenst két különböző helyről is meglátogatunk, ilyenkor pedig célszerű arra az oldalra visszatérni a művelet végeztével, ahonnan érkeztünk. Ezt a mechanikát kiegészítettem azzal, hogy bizonyos oldalak esetén - ahol erre szükség van - lementem a komponens állapotát navigáció előtt, például a felhasználó által már kitöltött textboxok szövegét, a kiválasztott checkboxok állapotát, vagy esetleg a már hozzáadott elemeket, amennyiben vannak ilyenek. Mindezeket a korábban már említett, böngésző által biztosított localStorage tárolóban mentettem el, majd betöltéskor elég megvizsgálnom, hogy szerepelnek-e ebben bejegyzések a megfelelő kulcsokkal, és amennyiben igen, rögtön ki is tölthetőek a megfelelő űrlapelemek a mentett értékekkel. Utóbbi ellenőrzést érdemes a *OnInitializedAsync()* metódus felülírásával megtenni, hiszen ez az a metódus a

komponens életciklusában, ami csak az első megjelenítés előtt fut le, viszont ilyenkor a fizikai renderelés előtt, így pont alkalmas arra, hogy egy alkalommal adatokat töltsünk be egy korábbi állapot alapján. Az előző két technika kombinálásával sikerült egy teljesen stateful alkalmazást készítenem, vagyis egy olyat, ami megőrzi az oldalak állapotát, és a navigációt illetően pedig figyelembe veszi a történetet.

5.3. DEPLOYMENT

Minden modern webalkalmazás esetén kulcsfontosságú, hogy milyen módon futtatjuk azt. A web hőskorában kevesebb opció is állt rendelkezésre ehhez, valamint könnyebb is volt az alkalmazás elhelyezését megoldani a világhálón, lévén, hogy ezek az appok sokszor statikus fájlokkal operáltak működésük során, esetleg még az adatbázis kapcsán is. A következőkben az általam fejlesztett alkalmazást illetően mutatom be, hogy milyen lehetőségek mutatkoznak a deploymentre.

A szakdolgozat elkészítése során létrejött projektet elsősorban elméleti jellegű megközelítéssel fejlesztettem, mivel a célom az volt, hogy az adott problémára válaszként megfogalmazott, jól körülhatárolt alkalmazás-specifikációt egy minél korszerűbb, fenntarthatóbb, modernebb applikáció keretein belül valósítsam meg. Ebből kifolyólag, ténylegesen éles környezetbe nem is telepítettem azt, azonban ennek lehetőségeit is vizsgáltam, ennek részleteit egy későbbi bekezdésben tárgyalom. Először azonban érdemes beszélni a lokális környezetről, mivel ez igen fontos egy nagy alkalmazás esetén. Tekintve, hogy az app itt tölti az életciklusa első szakaszát, a fejlesztőknek is alapjaiban véve határozza meg a mindennapi munkáját a helyi környezet.

5.3.1. LOKÁLIS KÖRNYEZET

A helyi környezet a fejlesztés során tartogatott érdekességeket, főleg azért, mert párhuzamosan fejlesztettem az alkalmazást két különböző rendszeren. Ezen a ponton visszautalnék a tervezés fejezetben tárgyalt technológiai választásra, ahol kifejtettem, hogy érdemes az alkalmazás minden pontján a nyílt forráskódú, multiplatform lehetőségeket választani, hiszen így a fejlesztési folyamatot mobilizálni tudjuk. Ezt én is kihasználtam, Macen és Windows alapú rendszeren egyaránt dolgoztam a megvalósítás során.

Windows-os környezetben érezhetően gördülékenyebben működnek a szolgáltatások, ami nem meglepő, hiszen ez tekinthető natív környezetnek a C# alapú keretrendszerek használatakor. Ebben az esetben, a Visual Studio újabb verzióiban már alapértelmezetten telepített LocalDB használható adatbázisként, ennek konfigurálása gyors és egyszerű, mindössze regisztrálnunk kell a dependency injection containerében a DBContext osztályunkat, ezt a Program.cs-ben tehetjük meg. Emellett szükségünk van egy connection stringre, ennek az alapértelmezett formája megtalálható a Microsoft

dokumentációjában, ezt módosítani csak indokolt esetben érdemes. Utóbbit az appsettings.json fájlban kell elhelyeznünk, innen olvassa ki azt a rendszer. Ennek a fájlnak az érdekessége, hogy önmagában egy alapértelmezett konfigurációs állomány, azonban lehetőségünk van *appsettings.verzio.json* formában létrehozni további változatokat. Ekkor, a pontok közé írt rész utal a környezetre, ahol a fájl beállításai érvényesek lesznek. Ezek a különböző változatok tehát mind-mind specifikus beállításokat tartalmaznak, így a megfelelő környezetben könnyedén kiválasztható, hogy a program a konfigurációs állomány melyik verzióját használja, nem kell mindig ezek újraírásával vesződnünk.

MacOS rendszeren például nem elérhető natívan az előzőekben említett MSSQL alapú LocalDB. A probléma áthidalásához virtuális megoldásokra támaszkodhatnunk, vagy a konténerizációt hívhatjuk segítségül. A fejlesztés során az utóbbit választottam, így Dockert használtam, amibe könnyedén telepíthető egy SQL Server 2019-et tartalmazó, előre elkészített képfájl. Ezt maga a Microsoft biztosítja ingyenesen, és macOS mellett Linux platformon is elérhető. Letöltés után a terminálon keresztül néhány beállítást eszközölnünk kell, ebben az esetben - ellentétben a Windows alatti LocalDB implementációval - szükségünk lesz felhasználónév és jelszó párosra is, aminek természetesen a connection stringben is meg kell jelennie. A kezdeti beállítás nehézségeitől eltekintve elmondható, hogy valóban transzparens az alkalmazás működése, hiszen később, a fejlesztés során semmit nem érzékeltem abból, hogy egy másik szolgáltató biztosítja az adatbázist.

5.3.2. DEPLOYMENT LEHETŐSÉGEK

Amennyiben élesíteni szeretnénk az alkalmazásunkat, meg kell vizsgálnunk, hogy milyen szolgáltatókra támaszkodhatunk, hiszen ezek függvényében jelentősen át kell szerveznünk a konfigurációnkat.

Kézenfekvő, azonban a kevésbé rugalmasan skálázódó megoldásnak ígérkezik a helyi szerveren való futtatás. Ennek lépéseit csak nagy vonalakban foglalom össze, lévén, ez a megoldás kevésbé releváns egy nagy alkalmazás esetén. Ebben az esetben szükségünk lesz egy IIS szerverre, illetve a Web Deploy csomagra. Ezeket a Web Platform Installer alkalmazásból érhetjük el a legkönnyebben. Telepítés után az IIS Manager segítségével el kell végeznünk néhány konfigurációs lépést, ezek végeztével az alkalmazásunk készen áll a deploymentre. A probléma, hogy az IIS teljes verziójából futtatott alkalmazás - érthető okokból - nem működik a LocalDB megoldásunkkal. Így szükségünk lesz egy SQL Server Express telepítésre is, amivel adatbázisokat hozhatunk létre saját szerveren. Ezek írásához, olvasásához azonban már jogosultság szükséges, így el kell készítenünk egy „grant” szkriptet, amit lefuttatva az adatbázisban, biztosítjuk, hogy az alkalmazásnak jogosultsága van a hozzáféréshez. Utolsó lépésként, a Visual Studio varázslója segítségével deployolni tudjuk az alkalmazást, azonban ezután még olyan beállítások

eszközölésére is szükségünk lehet, mint például mappa jogosultságok, vagy egy plusz tesztkörnyezet beállítása. Az egész folyamat nehézkes, sok figyelmet igényel, azonban kisebb alkalmazások esetén hasznos lehet, hogy egy számítógépről tudunk hosztolni egy komplett alkalmazást és a hozzá tartozó adatbázist. Tovább bonyolítja a helyzetet, hogy a frontend alkalmazás nem része az ASP alkalmazásnak, így ennek deploymentje még plusz lépéseket foglal magában. Könnyebbséget jelenthet az a lehetőség, mellyel a Blazor alkalmazás is a backend mellett futtatható, úgynevezett hosted deployment módban. A skálázhatóságon valamelyest javítunk, ha egy egyszerűbb elosztott rendszerben gondolkodunk, például a backend, frontend, valamint az adatbázist biztosító alkalmazást három különböző helyről hosztoljuk.

Az előző módszer nehezen skálázódik, hiszen előre adott erőforrásokkal tudunk csak gazdálkodni, melyek beszerzése egyébként is költséges. Érdemes inkább a felhőszolgáltatások nyújtotta lehetőségeket megvizsgálni, mivel ezeket aktuális igényeinknek megfelelően tudjunk alakítani, nem kell a fenntartásukkal, karbantartással vesződnünk, és még a konfiguráció is egyszerűbbé válik. Tekintve, hogy a szakdolgozat egésze a Microsoft égisze alatt futó projektek és technológiák segítségével készült, így kézenfekvő megoldásnak tűnik az Azure használata.

Mind az ASP, mind a frontend alkalmazás gyorsan deployolható az Azure segítségével. Új erőforrásokat kell létrehoznunk, majd ezek paramétereit beállítani igényeink szerint. Ezután a korábbiakban említett, Visual Studioban megtalálható publish varázslóba be tudjuk tölteni az Azureben kapott publish profilt, vagy közvetlen be is tudunk jelentkezni a fiókunkba. Ez a folyamat gyorsan, gördülékenyen megy végbe, azonban ezzel csak magát az alkalmazás magját helyeztük ki, az adatbázist azonban nem. Ehhez az előző publish varázslóba visszalépve kaphatunk segítséget, itt rögtön felajánlja nekünk a rendszer, hogy hozzunk létre egy adatbázist, mivel érzékeli, hogy van ilyen jellegű függősége az alkalmazásnak. Az érzékeny pont ebben az esetben a kapcsolat létrehozásához szükséges felhasználónév és jelszó páros tárolása: ezt nyilvánvalóan nem égethetjük be a programkódba, egy titkos tárolóban kell elhelyeznünk ezt. Az Azure ehhez is biztosít egy szolgáltatást: ez a Key Vault, ami a lokális környezetben található Secret Managerhez hasonlít. Ez utóbbi is arra szolgál, hogy már a fejlesztés közben is biztonságosabban tudjuk tenni a különböző kulcsok, jelszavak tárolását, hiszen azokat nem a programhoz köthető konfigurációs állományokban tárolja, hanem dedikált, strukturált fájlokban – a Secret Manager ehhez konkrétan JSON állományokat használ. A Key Vault valójában pont ezt a mechanikát valósítja meg egy felhőszolgáltatás keretében: miután összekötjük az alkalmazásunkkal, a szükséges kulcs-érték párokat már innen tudja betölteni az app.

6. TESZTELÉS

Az alábbiakban az elkészült alkalmazás tesztelését fogom részletezni. A tesztelés során a célom az volt, hogy a különböző komponenseket egységükben tudjam tesztelni, így megbizonyosodva arról, hogy külön-külön működőképesek. Ezután a program egészét tesztelve, a különálló komponensek egymással történő integrációját vizsgáltam. A három sarokpont, melyeknek fokozott figyelmet szenteltem, a következők voltak:

- A service réteg
- A végpontokat tartalmazó WebAPI réteg
- A frontend alkalmazás

A service réteg tesztelésének kivitelezéséhez elsősorban egységtesztekre hagyatkoztam, a végpontokat célszerűen egy REST kérések küldésére és fogadására alkalmas program segítségével teszteltem, ez volt a Postman, a frontend alkalmazást pedig manuális tesztelés segítségével végeztem.

6.1. A SERVICE RÉTEG UNIT TESZTELÉSE

A service réteg tesztelése során néhány fontosabb komponens tesztelésével foglalkoztam. Ezek az alkalmazás azon részei, melyek a legbonyolultabb üzleti logikát valósítják meg. A két tesztelt szolgáltatás a kérdések kezeléséért felelős, valamint a tesztmenedzser volt. A tesztek kivitelezéséhez az NUnit könyvtárat hívtam segítségül, az adatokat szolgáltató komponensek szimulálásához, vagyis a mockoláshoz, pedig a Moq-ot használtam. Ez utóbbi olyan eszközöket nyújt, amivel ellenőrizhető például, hogy a repository komponensek mely metódusait hányszor, illetve milyen paraméterrel hívta meg a program. Legtöbbször erre, illetve a visszatérési értékek logikai vizsgálatára hagyatkoztam. A minden teszt előtt lefutó előkészítő metódusban létrehoztam a mock objektumokat, ezek beállítása azonban a konkrét tesztek feladata, mivel sok esetben eltérő, a teszthez egyedileg illeszkedő objektumokat kellett visszaadniuk. Emellett itt példányosítom a szolgáltatást is, paraméterként természetesen az előbbi mockok által szimulált objektumokat átadva.

6.1.1. A KÉRDÉSKEZELŐ TESZTELÉSE

Itt a QuestionService implementációm által megvalósított IQuestionService interfész publikus elemeit teszteltem. A két tesztesetben azt vizsgáltam, hogy mi történik sima (tehát hagyományos) kérdésfajták, valamint programozási feladatok hozzáadása esetén. A 6.1 és 6.2 táblázatok a két teszt eredményeit foglalják össze:

Ellenőrzés	Kapott eredmény	Várt eredmény
QuestionDTO osztály valid példányának hozzáadása esetén a beszúrás függvény „igaz” értékkel tér vissza	Igaz	Igaz
A szolgáltatás beszúrás metódusa által létrehozott objektum típusa, amennyiben QuestionDTO példányt kap paraméterként	Question	Question
Beszúrás esetén hívások száma: QuestionRepository létrehozás metódusa, bármilyen Question példánnyal	Egyszer	Egyszer
A létrejött Question példány paraméterei megegyeznek a DTO objektumban specifikáltakkal (maximum pontok, cím, létrehozó felhasználó)	Igaz	Igaz
A kérdés objektum címe hiányzó paraméter hiányában a DTO példány szövegének első három kifejezéséből generálódott	Igaz	Igaz
A kérés részeként átadott címke a címkézésért felelős szolgáltatásnak átadásra került, a címke QuestionID mezője a legutóbb létrehozott kérdés azonosítóját tartalmazza	Igaz	Igaz

6.1 táblázat: A QuestionService szolgáltatás tesztjének eredményei

A második, 6.2 táblázat a programozás kérdések hozzáadásának esetét vizsgálja, hasonló szempontok szerint:

Ellenőrzés	Kapott eredmény	Várt eredmény
ProgQuestionDTO osztály valid példányának hozzáadása esetén a beszúrás függvény visszatérési értéke	Igaz	Igaz
A szolgáltatás beszúrás metódusa által létrehozott objektum típusa, amennyiben ProgQuestionDTO példányt kap paraméterként	Programming Question	Programming Question

Beszúrás esetén hívások száma: ProgrammingQuestionRepository létrehozás metódusa, bármilyen ProgrammingQuestion példánnyal	Egyszer	Egyszer
A létrejött ProgrammingQuestion példány paraméterei megegyeznek a DTO objektumban specifikáltakkal (maximum pontok, cím, létrehozó felhasználó)	Igaz	Igaz
A kérdés objektum címe hiányzó paraméter hiányában a DTO példány szövegének első három kifejezéséből generálódott	Igaz	Igaz
A kérés részeként átadott címke a címkézésért felelős szolgáltatásnak átadásra került, a címke QuestionID mezője a legutóbb létrehozott kérdés azonosítóját tartalmazza	Igaz	Igaz
Az ellenőrizendő metódus az összes paraméterével bekerült a kész ProgrammingQuestion példányba, a JSON-ként kapott kérésből a típusokat sikeresen dekódolja a program, ezek típushelyesen kiolvashatók később (string, char, bool, int, double, bármilyen komplex vagy egyszerű típusból álló tömb vagy lista)	Igaz	Igaz

6.2 táblázat: A ProgrammingQuestionService szolgáltatás tesztjének eredményei

6.1.2. A TESZTMENEDZSER TESZTJEI

A tesztmenedzser modul felelős a következőkért: tesztek kitöltésének indítása és leállítása, befejezett tesztek lezárása, kiértékelése – beleértve a programozási kérdések eredményét is. Emellett ez a szolgáltatás teszi lehetővé, hogy a tesztek kitölthetőségét szabályozzák az oktatók. Ehhez a következő modulok implementációját használja, melyeket természetesen helyettesítenem kellett mockokkal: ITestService, IQuestionService, ICourseService, ITestResult, ICourseRepository. A 6.3 táblázat összefoglalja ennek a modulnak a teszteredményeit:

Ellenőrzés	Kapott eredmény	Várt eredmény
Létező kurzus esetén, az abban található teszt kitöltésre megnyitásának eredménye	Igaz	Igaz
Létező kurzusban található teszt kitöltésének megkezdése és befejezése esetén a courseRepository létrehozás és frissítés metódusok meghívásainak száma	1	1
Kitöltés megkezdésének eredménye olyan teszt esetén, melyet bármilyen IP címmel ki lehet tölteni	Igaz	Igaz
Kitöltés megkezdésének eredménye 192.168.10.14 IP címről, olyan teszt esetén, melyet 1.1.x.x tartományban lévő IP címmel lehet kitölteni	Hamis	Hamis
Kitöltés megkezdésének eredménye 192.168.10.14 IP címről, olyan teszt esetén, melyet 192.168.x.x tartományban lévő IP címmel lehet kitölteni	Igaz	Igaz
Létező kurzus esetén, létező, nyitott teszteredmény (kitöltés) befejezésekor a teszteredmény IsClosed paraméterének állapota	Igaz	Igaz
Létező kurzus esetén, létező, nyitott teszteredmény (kitöltés) befejezésekor a teszteredmény IsCorrectionFinished paraméterének állapota	Igaz	Igaz
Létező kurzus esetén, az abban található teszt megnyitása után a kapcsoló entitásban található IsOpen tulajdonság állapota	Igaz	Igaz
Létező kurzus esetén, az abban található teszt lezárása után a kapcsoló entitásban található IsOpen tulajdonság állapota	Hamis	Hamis
Létező kurzus esetén, az abban található teszt megnyitása és bezárása során a courseRepository frissítés metódus meghívásainak száma	2	2
Nem létező kurzus tesztjének megnyitása	Hamis	Hamis
Létező kurzus nem létező tesztjének megnyitása	Hamis	Hamis

Nem létező kurzus nem létező tesztjének megnyitása	Hamis	Hamis
Létező teszt kérdésére, még nem lezárt teszteredmény esetén, időben érkező válasz feldolgozása	Igaz	Igaz
Létező teszt kérdésére, még nem lezárt teszteredmény esetén, időben érkező válasz bekerül a teszteredménybe, a TestResultRepository frissítés metódusa meghívódik	Igaz	Igaz
Létező teszt kérdésére, még nem lezárt teszteredmény esetén időn kívül érkező válasz feldolgozása	Hamis	Hamis
Létező teszt kérdésére, lezárt teszteredmény esetén időben érkező válasz feldolgozása	Hamis	Hamis
Létező teszt kérdésére, még nem lezárt teszteredménnyel, időn kívül érkező válasz esetén a TestResultRepository frissítés metódus meghívásainak száma	0	0
Létező teszt kérdésére, lezárt teszteredménnyel, időben érkező válasz esetén a TestResultRepository frissítés metódus meghívásainak száma	0	0

6.3 táblázat: A ProgrammingQuestionService szolgáltatás tesztjének eredményei

A fenti tesztesetek forráskódjai megtalálhatóak a QuestionServiceTester.cs, valamint a TestManagerTester.cs osztályban

6.2. A WEBAPI TESZTELÉSE

Az alkalmazás WebAPI-n keresztül történő tesztelését részletezem az alábbiakban. Ezzel nem csak a végpontok által fogadott és visszaadott objektumok szintaktikáját validáltam, de az egész backend alkalmazás működésére is következtetés vonható le, hiszen itt éles adatbázis ellenében végeztem a teszteket. Jellemzően két fajta válasszal találkozhatunk: az egyik esetben érdemi információt nem tartalmaz a válasz, csupán logikailag jelzi, hogy sikeres volt-e a művelet, és ennek megfelelően HTTP 200 (Ok) vagy HTTP 400 (Bad Request) státuszkóddal tér vissza. A másik esetben a válasz valamilyen információt tartalmaz, ilyenkor egy JSON objektum formájában tér vissza a kérés. Utóbbiak esetén, a tesztelés során kapott válaszokat elmentettem JSON kiterjesztésű fájlokba, és ezeket a

függelékben mellékelem. A teszteredményeket kontrollerenként külön táblázatban közlöm.

A 6.4 táblázat tartalmazza az authentikációhoz tartozó tesztek eredményeit:

Ellenőrzés	Kérés fajtája, végpont	Kapott HTTP státuszkód	Várt HTTP státuszkód	Kapott JSON válasz fájl
Bejelentkezés admin felhasználóként	POST: /users/login	200 (Ok)	200 (Ok)	login_response.json
Bejelentkezés hibás felhasználónév – jelszó párossal	POST: /users/login	400 (Bad Request)	400 (Bad Request)	-
Új testTeacher felhasználó hozzáadása	POST: /users/register	200 (Ok)	200 (Ok)	-
Létező felhasználónévvel új regisztráció	POST: /users/register	400 (Bad Request)	400 (Bad Request)	-
Felhasználó törlése admin jogosultságú felhasználóval	DELETE: /users	200 (Ok)	200 (Ok)	-
Felhasználó törlése nem admin jogosultságú felhasználóval	DELETE: /users	403 (Forbidden)	403 (Forbidden)	-
Felhasználók listázása admin felhasználóval	GET: /users	200 (Ok)	200 (Ok)	user_list.json
Felhasználók listázása diák vagy tanár felhasználóval	GET: /users	403 (Forbidden)	403 (Forbidden)	-

6.4 táblázat: Az AuthController.cs tesztjei

A 6.5 táblázatban a kurzusokat kezelő végpont tesztjeinek eredményét foglalom össze:

Ellenőrzés	Kérés fajtája, végpont	Kapott HTTP státuszkód	Várt HTTP státuszkód	Kapott JSON válasz fájl
Új kurzus hozzáadása	POST: /courses	200 (Ok)	200 (Ok)	-
Kurzus törlése	DELETE: /courses	200 (Ok)	200 (Ok)	-
Saját kurzusok lekérése	GET: /courses/OwnCourses	200 (Ok)	200 (Ok)	OwnCourses.json
Felhasználó beiratkoztatása kurzusba, aki nem tagja a kurzusnak	GET: /courses/enrollStudentInCourse/	200 (Ok)	200 (Ok)	-
Felhasználó beiratkoztatása kurzusba, aki már tagja a kurzusnak	GET: /courses/enrollStudentInCourse/	400 (Bad Request)	400 (Bad Request)	-
Teszt hozzáadása kurzushoz	POST: courses/addTestToCourse/	200 (Ok)	200 (Ok)	-
Teszt eltávolítása a kurzusból	GET: courses/removeTestFromCourse/	200 (Ok)	200 (Ok)	-
Összes kurzus lekérdezése	GET: /courses	200 (Ok)	200 (Ok)	courses_list.json

6.5 táblázat: Az CoursesController.cs tesztjei

A következőkben a címkézés kezeléséért felelős végpont tesztjeit mutatom be. Az eredményeket a 6.6 táblázat tartalmazza.

Ellenőrzés	Kérés fajtája, végpont	Kapott HTTP státuszkód	Várt HTTP státuszkód	Kapott JSON válasz fájl
Címke hozzáadása meglévő kérdéshez	POST: /labels	200 (Ok)	200 (Ok)	-
Összes címke lekérdezése	GET: /labels	200 (Ok)	200 (Ok)	labels_list.json

Adott címkével felcímkézett kérdések keresése	GET: /labels/getLabelledQuestions	200 (Ok)	200 (Ok)	label_query.json

6.6 táblázat: A LabelsController.cs tesztjei

A 6.7 táblázat a kérdéseket kezelő végpont teszteredményeit mutatja:

Ellenőrzés	**Kérés fajtája, végpont**	**Kapott HTTP státuszkód**	**Várt HTTP státuszkód**	**Kapott JSON válasz fájl**
Szöveges kérdés hozzáadása	POST: /questions	200 (Ok)	200 (Ok)	-
Programozási feladat hozzáadása	POST: /questions	200 (Ok)	200 (Ok)	-
Összes kérdés lekérdezése	GET: /questions	200 (Ok)	200 (Ok)	questions_list.json
Saját kérdések lekérdezése	GET: /questions/OwnQuestions	200 (Ok)	200 (Ok)	own_questions.json
Összes megosztott kérdés lekérdezése	GET: /questions/SharedQuestions	200 (Ok)	200 (Ok)	shared_questions.json

6.7 táblázat: A QuestionsController.cs tesztjei

A 6.8 táblázatban a tesztekhez tartozó kontroller tesztjeit foglalom össze:

Ellenőrzés	**Kérés fajtája, végpont**	**Kapott HTTP státuszkód**	**Várt HTTP státuszkód**	**Kapott JSON válasz fájl**
Teszt hozzáadása	POST: /tests	200 (Ok)	200 (Ok)	-
Teszt törlése	DELETE: /tests	200 (Ok)	200 (Ok)	-
Teszt frissítése	PATCH: /tests	200 (Ok)	200 (Ok)	-

Teszthez tartozó kérdések lekérdezése	GET: /tests/GetQuestionsOfTest	200 (Ok)	200 (Ok)	test_questions.json
Adott kurzushoz tartozó tesztek listájának lekérdezése	GET: /tests/GetTestsOfCourse	200 (Ok)	200 (Ok)	course_tests.json
Saját kurzusokban található tesztek lekérdezése	GET: /tests/OwnTests	200 (Ok)	200 (Ok)	own_courses.json
Létező kérdés létező teszthez adása	GET: /tests/addQuestionToTest	200 (Ok)	200 (Ok)	-
Létező kérdés nem létező teszthez adása	/tests/addQuestionToTest	400 (Bad Request)	400 (Bad Request)	-
Korábban hozzáadott kérdés eltávolítása tesztből	/tests/removeQuestionFromTest	200 (Ok)	200 (Ok)	-

6.7 táblázat: A TestsController.cs tesztjei

A következő táblázat a teszteredményeket kezelő végpont ellenőrzését összegzi:

Ellenőrzés	**Kérés fajtája, végpont**	**Kapott HTTP státuszkód**	**Várt HTTP státuszkód**	**Kapott JSON válasz fájl**
Adott felhasználó egy tesztre irányuló tesztkitöltéseinek eredménye, bármilyen kurzusban	GET: /results	200 (Ok)	200 (Ok)	user_test_testresult.json
Felhasználó teszteredményeinek lekérése (testStudent felhasználó, 2 kurzusban összesen 3 teszttel)	GET: /results	200 (Ok)	200 (Ok)	user_course_results.json

Egy kurzuson belül, egy teszt összes kitöltésének eredményeinek lekérése	GET: /results	200 (Ok)	200 (Ok)	testresult_in_course.json

6.8 táblázat: A TestsController.cs tesztjei

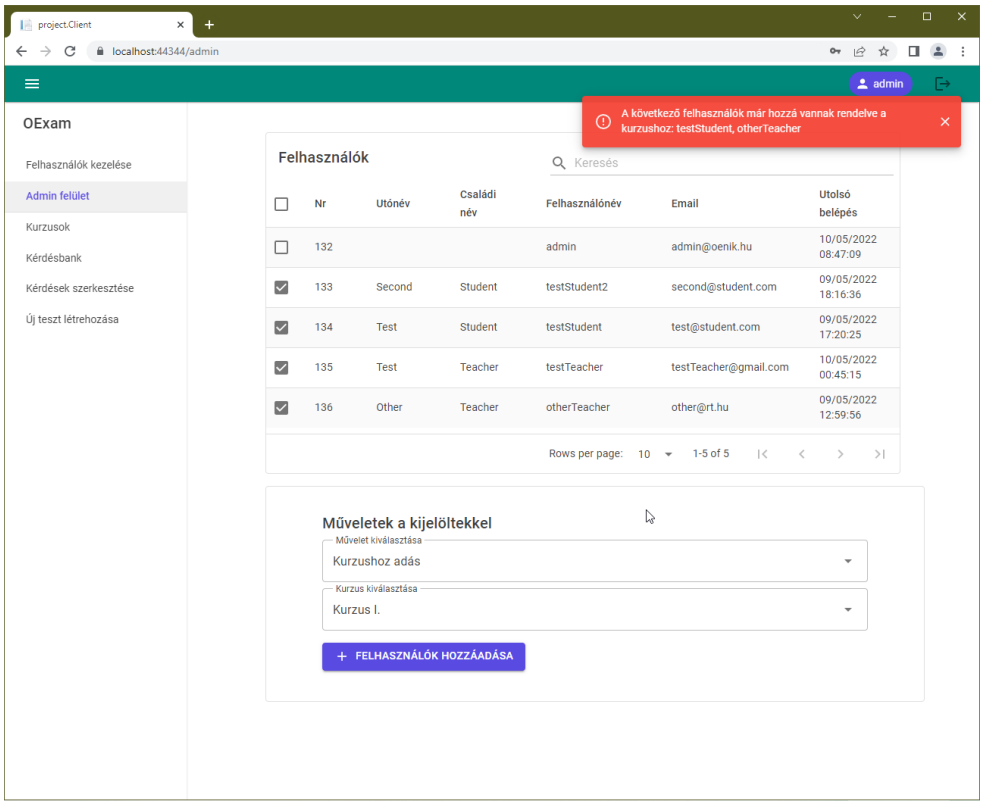
6.3 A FRONTEND ALKALMAZÁS TESZTELÉSE

A frontend alkalmazás teszteléséhez kézi tesztelésre támaszkodtam. Első tesztem során verifikáltam, hogy amennyiben nem vagyunk bejelentkezve, bármilyen URL cím felkeresése esetén valóban átirányításra kerül az ablak a kezdőképernyőre. Valamennyi cím (például: https://localhost:44344/courses vagy „https://localhost:44344/tests/add”) felkeresése esetén az üdvözlőképernyő fogadott. A bejelentkezési űrlapon, majd a bejelentkezés után a következőket ellenőriztem:

- Hibás felhasználónév és jelszó páros esetén az űrlap kiürítésre kerül, hibaüzenet jelenik meg
- Bejelentkezés után az átirányítás megtörténik
- Az authentikált felhasználó szerepkörétől függően jelennek meg a menüben lehetőségek:
    - Felhasználók kezelése, Admin felület csak admin felhasználóval
    - Kérdésbank, Új teszt létrehozása, Kérdések szerkesztése menüpontok csak tanári felhasználóval
    - Diák azonosítás esetén csupán Kurzusok menüpont jelenik meg

Az admin szerepköréhez tartozó két komponenst is teszteltem:

- Az *Admin felület* menüben táblázatosan megjelennek a felhasználók, lehetőség van keresni, illetve csoportosan kurzushoz adni őket. Amennyiben egy vagy több felhasználó már tagja egy kurzusnak, akkor erről felugró ablak figyelmeztet. A 6.1 ábrán látható ennek a tesztnek az eredménye.
- A *Felhasználók kezelése* menüben új felhasználó regisztrálható, a meglévők törölhetők



6.1 ábra: Beiratkoztatási teszt az admin felületen

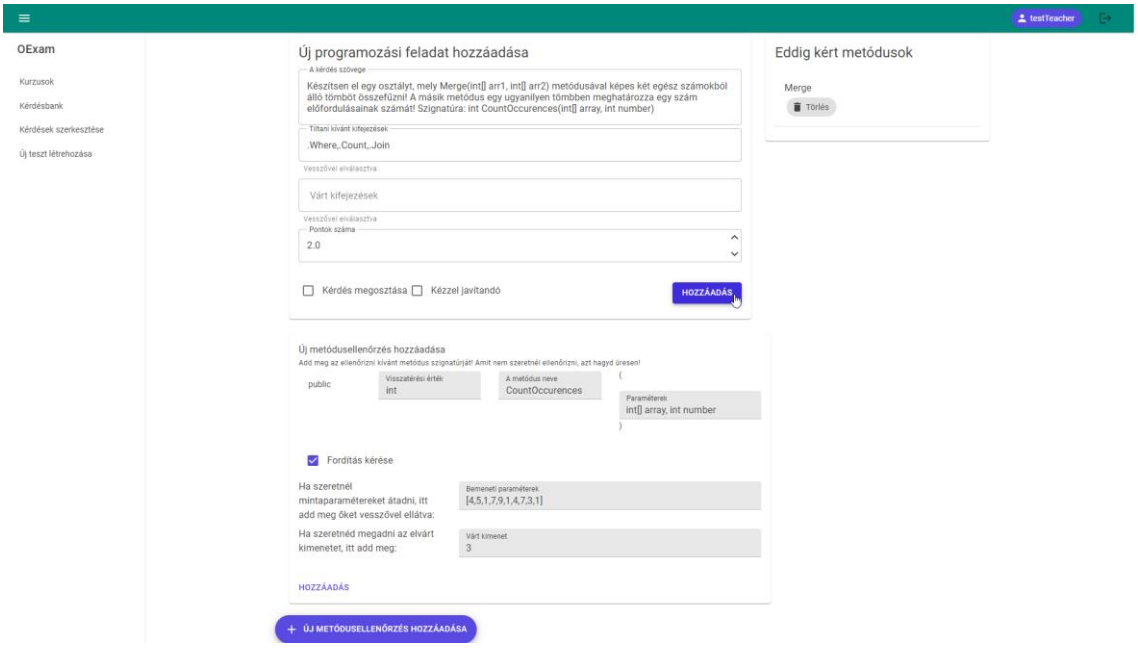
A *Kurzusok* oldalon az alábbiakat vizsgáltam:

- Az oldalon csak azok a kurzusok jelennek meg, melyekbe a felhasználó be van iratkoztatva
- Csak tanári felhasználó esetén jelenik meg a „+” ikon, tehát csak ilyen jogosultság mellett lehet kurzust létrehozni
- A létrehozott kurzusba automatikusan beiratkoztatásra került a felhasználó, aki létrehozta azt

A *Kérdések szerkesztése* menüpont alatt, tanári felhasználóval a következőket ellenőriztem:

- Megjelenik egy komponens, melyben a saját kérdések külön megjeleníthetők a megosztott kérdésektől
- Egy saját kérdés törölhető, azonban a mások által megosztott kérdések esetén nem jelenik meg a törlés ikon
- A megosztott kérdések között a címkék alapján keresni lehet
- Új kérdés adható hozzá, a „+” jel egy új oldalra irányít, ahol kiválasztható, hogy milyen típusú kérdést szeretnénk hozzáadni
- Új kérdés hozzáadásakor, a típus kiválasztása után annak megfelelő szerkesztőfelület nyílik meg

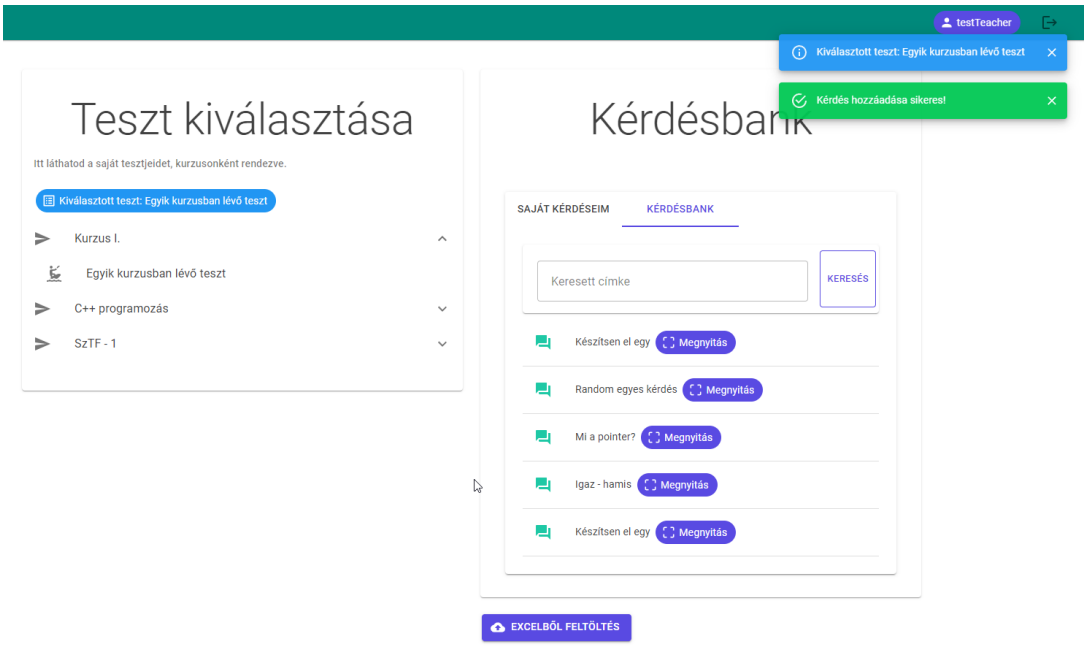
- A programozási kérdések szerkesztő felületén minden ellenőrzés, beállítás helyet kapott, több metódus ellenőrzése is felvihető. Ennek ellenőrzését mutatja a 6.2 ábra.
- A kérdések sikeres hozzáadásáról üzenet tájékoztat, illetve átirányításra kerülünk az előző oldalra, ami jelen esetben a kérdésfajta-választó



6.2 ábra: Új programozási kérdés hozzáadása, 2 metódus ellenőrzésével

A *Kérdésbank* komponens tesztjeit az alábbiakban foglalom össze:

- Az oldalon megjelenik egy tesztválasztó komponens, melyben kurzusonként rendezve jelennek meg a tesztek
- A kérdéseket listázó komponens itt is megtalálható
- Az aktuálisan kiválasztott teszt látható a felületen
- Ha nincs kiválasztva teszt, kérdés hozzáadása esetén üzenet tájékoztat a hibáról
- Az *Excelből feltöltés* gombra kattintva megjelenik egy tallózó, és lehetőség van .xlsx kiterjesztésű fájlból betölteni kérdéseket, melyeket sikeresen hozzáad a rendszer, és megjelennek a kérdésbank saját kérdéseket tartalmazó listájában
- Sikeres hozzáadás esetén üzenet tájékoztat a műveletről, ezt mutatja a 6.3 ábra



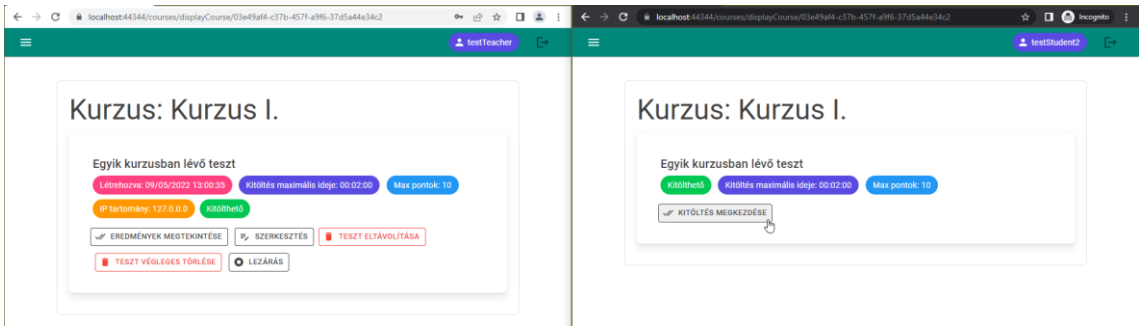
6.3 ábra: Kérdés teszthez adása a kérdésbankból

Az *Új teszt létrehozása* oldal tesztesetei a következők:

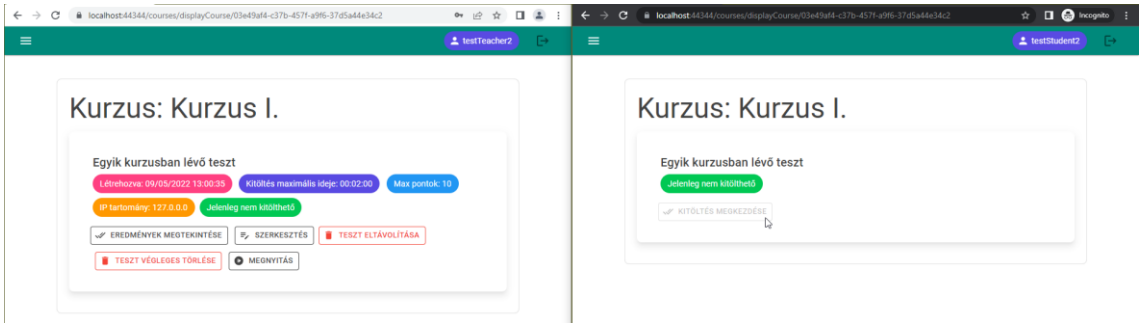
- Az új teszt paraméterei (IP szűrés, megnyitás, cím, kurzus) beállíthatók
- Kérdések már a teszt létrehozása közben hozzáadhatók, ilyenkor a felület átirányításra kerül a kérdésfajta-választó komponsensre. Ennek kapcsán ellenőriztem, hogy a kérdés hozzáadása után valóban nem a választóhoz jutunk vissza, hanem a tesztszerkesztőhöz, ahol a korábbi beállításokat megjegyezte az űrlap
- A hozzáadandó kérdéseket listázza a komponens, ezeket is megjegyzi

A tesztek megjelenítését és kitöltését is vizsgáltam, erre a kurzusoldalon belül van lehetőség:

- Tanár jogosultságú felhasználó esetén technikai információk is megjelennek a teszttel kapcsolatban
- Tanár jogosultságú felhasználó esetén leállítható és megnyitható a teszt kitöltésre
- Tanár jogosultságú felhasználóval törölhető a kurzusból a teszt
- Diák jogosultságú felhasználóval csak megnyitott teszt esetén elérhető a kitöltés megkezdésére szolgáló gomb. Ez a működés látható a 6.4 és 6.5 ábrákon.
- Diák jogosultságú felhasználóval a kitöltés megkezdése után végig lehet lépkedni a kérdéseken, az idő lejárta, vagy az utolsó kérdés után elküldi az oldal a kitöltést, és visszairányít a kurzus oldalára



6.4 ábra: Egy minta kurzusoldal tesztje. Bal oldalon az oktató, jobb oldalon a diák képernyője. Látható, hogy megnyitott állapotban a diáknál elérhető a kitöltés



6.5 ábra: Egy minta kurzusoldal tesztje. Bal oldalon az oktató, jobb oldalon a diák képernyője. Látható, hogy lezárt esetben a diák nem töltheti ki a tesztet

7. ÉRTÉKELÉS, TOVÁBBFEJLESZTÉSI LEHETŐSÉGEK

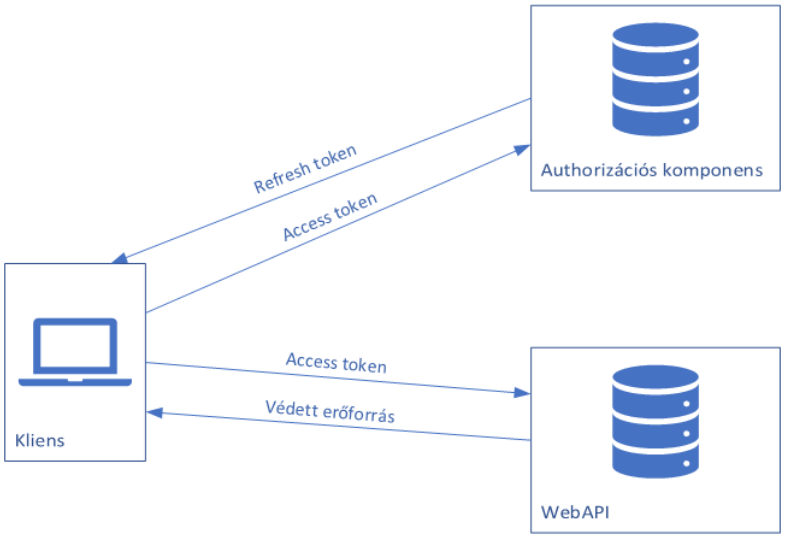
Az elkészült szoftver egy ambiciózus próbálkozás eredménye, melynek megtervezése is rengeteg kutatómunkát igényelt, azonban az implementáció során a felmerülő technikai problémákon és kialakult helyzeteken keresztül rengeteg architekturális, és különböző keretrendszereken átívelő technikai kérdésekben mélyülhettem el. A megvalósított rendszer az ASP beépített felhasználókezelő-rendszerén kívül nem támaszkodik semmilyen előre elkészített megoldásra, az adatbázis felépítése és fenntartása is az általam elkészített logika alapján történik, tehát nem csak szigorúan a programkód-ellenőrző modulra fókuszáltam, hanem igyekeztem egy fenntartható, egyedi megoldásokra épülő, modern alkalmazást felépíteni. Ezzel együtt az elkészült program a kitűzött célokat megvalósítja:

- Létrehozhatóak a segítségével tesztek, melyekbe hagyományos kérdésformák mellett programozási feladatok is hozzáadhatók
- A kérdések megoszthatók, címkézhetők, mások által kereshetők és felhasználhatók
- A program eltérő működést biztosít oktatók és diákok számára, ezekben átfedés a triviális eseteket leszámítva nincs
- A kitöltés ideje beállítható, kézzel megnyitható és lezárható a teszt, valamint IP-cím alapján is lehetőség van szűrni
- A rendszer képes automatikusan pontozni a próbálkozásokat, ezek eredményeit a kurzusokon belül tesztenként megtekinthetik az oktatók
- Elérhetőek a leggyakoribb hagyományos kérdésfajták:
    - igaz - hamis
    - szöveges
    - szám, mint válasz
    - párosítós
    - sorbarendezős
- Mobilon is használható a program, egy átlagos okostelefon kijelzőmérete mellett minden komponens olvasható, kezelhető
- Egyszerű kérdéseket tartalmazó tesztek kérdései létrehozhatók, betölthetők egy Excel munkafüzetből

A következőkben olyan lehetőségeket fogok áttekinteni, melyek megvalósításuk esetén az elkészült programot bizonyos aspektusokból továbbfejlesztenék. Ezen javítások egy részének lehetősége már az implementáció korai szakaszában láthatóvá vált, ezek jellemzően azok a fejlesztések, melyek nagyobb felhasználói bázis, magasabb igénybevétel esetén válnak relevánssá, mások pedig az implementáció során szerzett tapasztalataim alapján fogalmazódtak meg bennem.

7.1. AUTHENTIKÁCIÓ FEJLESZTÉSE

Jelenlegi állapotában a program az ASP saját rendszerével kezeli a felhasználói bejelentkezéseket. Bejelentkezés után - a korábban tárgyaltak alapján - egy token kerül kiadásra, melynek fix lejárati ideje van, ez jelen esetben egy óra. A tokent a kliens eltárolja a böngészőben, majd a kliens-oldali service-k a kérések fejlécében ezt minden egyes alkalommal továbbítják. Ez a mechanika egyszerű, alapvetően biztonságos, hiszen a tokenen kívül nem tárolunk egyéb szenzitív információt a böngészőben, azonban azt megszerezve egy hacker máris hozzáférést kap – legfeljebb egy órára – a felhasználó által kezelt fiókhoz. Ez valamelyest kiküszöbölhető a következő ábrán látható felépítés implementálásával:



7.1 ábra: Refresh token alapuló azonosítás

Ebben az esetben a sima hozzáférési token mellett egy úgynevezett refresh token is kiadásra kerül. Ennek egyetlen célja, hogy biztosítékot adjon arról, hogy korábban sikeres bejelentkezés történt, majd a későbbiekben ennek a segítségével igényelhető új access token. Mivel a refresh token soha nem kerül a WebAPI felé továbbításra, így a legnagyobb biztonság akkor érhető el, ha az authentikációért felelős komponenst és a WebAPI-t tényleges elválasztjuk egymástól. Fontos megjegyezni, hogy az access token élettartama drasztikusan lerövidül ebben az esetben, akár pár percre is korlátozódva. Ha a betolakodó meg is szerzi ezt, a refresh token hiányában néhány percet tölthet csupán el a rendszerben. Utóbbi token hosszabb élettartamú, akár pár órára is kiadható. Ezzel a felépítéssel érhető el továbbá a legbiztonságosabban az, hogy a felhasználónak ne kelljen újra bejelentkezni, hiszen a refresh token nem tartalmaz kódolatlan szenzitív információt, továbbításával azonban mégis friss hozzáférési tokenhez juthatunk.

7.2. EGYÉB FEJLESZTÉSEK

Az elkészült alkalmazás egyes részeit áttekintve felmerültek továbbfejlesztési lehetőségek. Ezek a következők:

- Azure deployment esetén blob storage használata a fordított fájlok tárolásához, képek támogatása.
- Saját kivételosztályok használata: jelenleg hiba keletkezése esetén általános hibaüzeneteket adnak vissza a végpontok, lehetőség lenne saját kivételek kiváltására, melyet egy dedikált végpont tudna kezelni. Ennek alapjait elő is készítettem a programban.
- Programkód komplexitásának ellenőrzése: jelenleg a fordító modul csak viszonylag durván szemcsézett megoldást ad arra, hogy a beérkezett programkód belső szerkezete, működése ellenőrizhető legyen: például lehetőség van „kizárni” kifejezéseket, illetve saját, tetszőleges bemenettel futtatható a kód, melyet a kitöltő nem ismer. Emellett szerencsés lenne, ha bizonyos metrikák mentén meghatározásra kerülnének referencia szintek a komplexitás méréséhez, és a rendszer jelezni tudná, ha túl lassú, esetleg túl hosszú a végrehajtás ideje, vagy éppen túl rövid a programkód.
- Komplex típusok: a fordító modul képes komplex típusokat is kezelni, és a továbbítás sem okozhat különösebb gondot, hiszen ezek jól szerializálhatók, így elegendő lenne a frontend alkalmazást felkészíteni arra, hogy saját típusok is hozzáadhatók legyenek, például egy .cs kiterjesztésű állományból.
- A frontend alkalmazásban, a programozás kérdéseknél hasznos lenne egy olyan beviteli mező, amely valójában egy integrált fejlesztői környezet, ami képes szövegbehúzásra, szintaxis alapján történő kiemelésre. Léteznek ilyen könyvtárak a Blazorhöz, azonban ezek jellemzően nem nyílt forráskódúak. Talán a későbbiekben több, ingyenes alternatíva is megjelenik majd. Emellett, az implementáció fejezetben szereplő 5.5-ös ábrán látható végpont segítségével könnyedén megoldható, hogy a teszt kitöltése közben is visszajelzést kapjon a hallgató a még szerkesztés alatt álló programkódjáról.

8. ÖSSZEFOGLALÓ

A szakdolgozatom írásának megkezdésekor, a tervezés során, célul tűztem ki magamnak, hogy egy olyan modern webalkalmazást készítsek, amely képes az informatikai felsőoktatást támogatni. A 2020 elején megjelent Covid-19 okozta világjárvány rámutatott, hogy igény mutatkozik az online oktatás mellett a digitális számonkérések modernizálására is. Ezen a területen jellemző, főleg a képzések korai szakaszában, hogy a hallgatók általános, egyszerű programozási paradigmákkal, algoritmusokkal foglalkoznak, így egy olyan rendszer ötlete fogalmazódott meg bennem, amelynek segítségével a számítógépes számonkérések keretében előbbiek is számonkérhetők. Emellett célul tűztem ki, hogy kurrens technológiák felhasználásával készítsek egy fenntartható, a mai trendeknek megfelelő, modern alkalmazást, így sokszor nem a könnyebb, hanem a kevésbé járt utat választottam a technológia választás során.

A tervezés előkészítéseként meglévő rendszereket vizsgáltam (Moodle, Redmenta, Socrative), melyek népszerűek, és szerteágazó megoldásokkal operálnak, azonban csak egyszerű számonkérések készíthetőek segítségükkel, általános felhasználásra alkalmasak, így inkább a középfokú oktatásban veszik tényleges hasznukat az oktatók. Ezeknek a rendszereknek az erősségeit és gyengeségeit megvizsgáltam, az így kapott kép alapján alakítottam ki a saját megoldásom specifikációját. Igyekeztem kiemelni minden rendszerből egy-egy olyan egyedi tulajdonságot, melyet érdemesnek tartottam arra, hogy valamilyen formában a saját rendszerem keretei között is implementáljak. Így született meg az elképzelés, hogy egy olyan alkalmazást készítek, melyet oktatók és diákok egyaránt használni tudnak, a felhasználók kurzusokba iratkoztathatók, és a rendszerben található tesztek ugyanezen kurzusokba rendezhetők. Célul tűztem ki továbbá, hogy a tudásanyag újrafelhasználható legyen, így lehetőség van a kérdések megosztására, méghozzá egy címkerendszer segítségével. A hagyományos, általánosan ismert kérdésfajták mellett egyszerű programozási feladatok is kiszabhatók kérdésként. Az alkalmazás jól használható mobilos kijelzőméret mellett is, így nincs szükség külön mobilalkalmazásra a használatához.

9. SUMMARY

In the early days of writing my thesis, during the planning phase, I set it as my objective to make a modern web application, one that can aid the higher education courses specialized in computer science. The outbreak of the Covid-19 pandemic in the beginning of 2020 pointed out that there seems to be a need for modernizing the digital examination techniques, not just the online lecturing. In this field, it is common, especially in the early phases of the courses, that the students learn or work with simple, generally known programming paradigms, algorithms. Thus, I realized that there is a need for a system, one that provides the functionality for the lecturers, to be able to incorporate the previously mentioned knowledge into the digital exams. I have also decided to create the application using currently relevant technologies, so that the end product is a modern, sustainable app. This led to some interesting choices when it came to choosing technology: as per the aforementioned goals, I have mostly chosen the new, coming up technologies, that are mostly harder to work with.

Preparing the application's planning, I studied some already existing systems, such as Moodle, Redmenta and Socrative. These are popular apps, and they utilize a wide variety of functionalities, but this also makes them generic, especially when it comes to creating assignments. This means that they are more suitable for secondary school education. I have analyzed the strengths and weaknesses of these solutions, and based on these, created my own specification. I have tried to look for one unique feature in each of these systems, one that I thought to be worthy of consideration for incorporating into my own work some way. This lead to the specification of a system, that both students and lecturers can use, where the users can be enrolled into courses, and the tests are also part of these courses. I have also decided that the knowledge in the system should be reusable, so I have implemented a labelling mechanism, making the questions shareable among tests. Besides the commonly known question types, programming questions can also be part of these tests. The application is also optimized for most mobile screen sizes, so a separate application is not necessary for these devices.

10. FÜGGELÉK

10.1. FÜGGELÉK 1.

LOGIN_RESPONSE.JSON


```
{
    "token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIyNWY4NjkyOS04OT
g0LTQwNjUtOTliOS0xMzk5OGI3NGU5YjkiLCJodHRwOi8vc2NoZW1hcy5taWNyb3
NvZnQuY29tL3dzLzIwMDgvMDYvaWRlbnRpdHkvY2xhaW1zL3JvbGUiOiJBZG1pbi
IsImh0dHA6Ly9zY2hlbWFzLnhtbHNvYXAub3JnL3dzLzIwMDUvMDUvaWRlbnRpdH
kvY2xhaW1zL25hbWVpZGVudGlmaWVyIjoiYWRtaW4iLCJodHRwOi8vc2NoZW1hcy
54bWxzb2FwLm9yZy93cy8yMDA1LzA1L2lkZW50aXR5L2NsYWltcy9lbWFpbGFkZH
Jlc3MiOiJhZG1pbkBvZW5pay5odSIsImV4cCI6MTY1MjAyNTQyMiwiaXNzIjoiaH
R0cDovL3d3dy5zZWN1cml0eS5vcmciLCJhdWQiOiJodHRwOi8vd3d3LnNlY3VyaX
R5Lm9yZyJ9.AVO26wfx7p87x_q9Dh-k1VGnP854cRVrQuDlP8Yyjkg",
    "expiration": "2022-05-08T15:57:02Z",
    "role": "Admin",
    "username": "admin",
    "id": "25f86929-8984-4065-99b9-13998b74e9b9"
}
```


USER_LIST.JSON (RÉSZLET)


```
[
    […]
    {
        "firstName": "Test",
        "lastName": "Student",
        "lastLogin": "2022-05-08T19:46:48.329109",
        "lastIP": "::1",
        "createdOn": "2022-05-08T19:46:32.060375",
        "modifiedOn": "2022-05-08T19:46:32.059816",
        "createdBy": null,
        "userCourses": null,
        "id": "bc98f638-fcf3-4515-b166-d1b3ab4f75de",
        "userName": "testStudent",
        "normalizedUserName": "TESTSTUDENT",
        "email": "student@test.com",
        "normalizedEmail": "STUDENT@TEST.COM",
        "emailConfirmed": true,
```

```
        "passwordHash": "",
        "securityStamp": "",
        "concurrencyStamp": "",
        "phoneNumber": null,
        "phoneNumberConfirmed": false,
        "twoFactorEnabled": false,
        "lockoutEnd": null,
        "lockoutEnabled": true,
        "accessFailedCount": 0
    },
    […]
]
```


OWNCOURSES.JSON (RÉSZLET)


```
[
    {
       "id":"9531a7c8-b7e1-4c7a-bb9d-bcceba229905",
       "name":"Teszt kurzus - 2.",
       "creationTime":"2022-05-08T19:21:42.762067",
       "userCourses":[
          {
             "id":"118c17bf-4576-4778-a733-6e07b1126a76",
             "userID":"fb3c5745-b42f-48d1-88b9-85dd5aaa65ba",
             "courseID":"9531a7c8-b7e1-4c7a-bb9d-bcceba229905"
          }
       ],
       "courseTests":[]
    },
    {
       "id":"fee780c7-6f10-4550-b77b-aa1ea22e8dd3",
       "name":"Teszt kurzus - 1.",
       "creationTime":"2022-05-08T19:12:14.867375",
       "userCourses":[
          {
             "id":"2b50b168-7f71-4746-8dc3-7302bdbc85e4",
             "userID":"fb3c5745-b42f-48d1-88b9-85dd5aaa65ba",
             "courseID":"fee780c7-6f10-4550-b77b-aa1ea22e8dd3"
          }
       ],
```

```
        "courseTests":[]
    }
 ]
```

COURSES_LIST.JSON (RÉSZLET)

```
[
    […]
    {
        "id": "9531a7c8-b7e1-4c7a-bb9d-bcceba229905",
        "name": "Teszt kurzus - 2.",
        "creationTime": "2022-05-08T19:21:42.762067",
        "userCourses": [
            {
                "id": "118c17bf-4576-4778-a733-6e07b1126a76",
                "userID": "fb3c5745-b42f-48d1-88b9-
85dd5aaa65ba",
                "courseID": "9531a7c8-b7e1-4c7a-bb9d-
bcceba229905"
            }
        ],
        "courseTests": [
            {
                "id": "e9efa0fc-b552-4ada-82bc-da8d83a6f197",
                "testID": "27023470-716c-4ed6-814d-
8d9c63f26292",
                "courseID": "9531a7c8-b7e1-4c7a-bb9d-
bcceba229905",
                "isOpen": true,
                "allowedIPSubnet": "1.2.x.x"
            }
        ]
    },
    […]
]
```

LABELS_LIST.JSON (RÉSZLET)

```
[
    {
        "id": "40fe8fbb-c5f2-49aa-8d09-8c14df6c5ce0",
        "question": {
            "id": "a522f89c-e916-4047-96c4-7a8bed1d5497",
            "title": "Sorrendbe helyezős feladat",
```

```
            "text": "Sorrendbe helyezős feladat",
            "creationTime": "2022-05-08T22:35:21.218421",
            "createdBy": null,
            "isShared": true,
            "correctManually": false,
            "questionType": 6,
            "correctAnswer": "[\"első\",\"második\"]",
            "maxPoints": 0,
            "pictureExtensionType": null,
            "pictureData": "",
            "testQuestions": null
        },
        "labelText": "label1"
    },
    […]
]
```


LABEL_QUERY.JSON


```
[
   {
      "id":"a522f89c-e916-4047-96c4-7a8bed1d5497",
      "title":"Sorrendbe helyezős feladat",
      "text":"Sorrendbe helyezős feladat",
      "creationTime":"2022-05-08T22:35:21.218421",
      "createdBy":null,
      "isShared":true,
      "correctManually":false,
      "questionType":6,
      "correctAnswer":"[\"első\",\"második\"]",
      "maxPoints":0,
      "pictureExtensionType":null,
      "pictureData":"",
      "testQuestions":null
   }
]
```

QUESTIONS_LIST.JSON (RÉSZLET)


```
[
    […]
    {
        "id": "a406cf1c-7d7f-4537-be9c-f3d31c7e5c53",
        "title": "Szöveges teszt kérdés",
        "text": "Szöveges teszt kérdés",
        "creationTime": "2022-05-08T19:25:00.963086",
        "createdBy": {
            […]
            "id": "fb3c5745-b42f-48d1-88b9-85dd5aaa65ba",
            "userName": "testTeacher",
            […]
        },
        "isShared": true,
        "correctManually": false,
        "questionType": 7,
        "correctAnswer": "helyes válasz",
        "maxPoints": 3,
        "pictureExtensionType": null,
        "pictureData": "",
        "testQuestions": []
    },
    […]
]
```


OWN_QUESTIONS.JSON (RÉSZLET)


```
[
    {
       "id":"420b2950-d19d-4372-8299-7bb35d8869e6",
       "title":"Programozási feladat szövege",
       "text":"Programozási feladat szövege",
       "creationTime":"2022-05-08T23:17:37.538494",
       "createdBy":{
          […]
          "id":"fb3c5745-b42f-48d1-88b9-85dd5aaa65ba",
          "userName":"testTeacher",
          […]
       },
       "isShared":true,
```

```
        "correctManually":false,
        "questionType":8,
        "correctAnswer":null,
        "maxPoints":2,
        "pictureExtensionType":null,
        "pictureData":"",
        "testQuestions":[]
    },
    {
        "id":"a522f89c-e916-4047-96c4-7a8bed1d5497",
        "title":"Sorrendbe helyezős feladat",
        "text":"Sorrendbe helyezős feladat",
        "creationTime":"2022-05-08T22:35:21.218421",
        "createdBy":{
            […]
            "id":"fb3c5745-b42f-48d1-88b9-85dd5aaa65ba",
            "userName":"testTeacher",
            […]
        },
        "isShared":true,
        "correctManually":false,
        "questionType":6,
        "correctAnswer":"[\"első\",\"második\"]",
        "maxPoints":0,
        "pictureExtensionType":null,
        "pictureData":"",
        "testQuestions":[]
    },
[…]
]
```

11.IRODALOMJEGYZÉK

[1] Ugur, N.G. (2020). Digitalization in higher education: A qualitative approach. International Journal of Technology in Education and Science (IJTES), 4(1), 18-25.

[2] Berggren, B., Fili, A. & Nordberg, O. (2015). Digital examination in higher education: Experiences from three different perspectives. *International Journal of Education and Development using ICT*, *11*(3), Open Campus, The University of the West Indies, West Indies.

[3] https://docs.moodle.org/dev/Main_Page, utoljára megtekintve: 2020-03-02

[4] https://docs.moodle.org/39/en/Quiz_activity, utoljára megtekintve: 2020-03-02

[5] https://docs.moodle.org/311/en/Question_sharer, utoljára megtekintve: 2020-03-25

[6] 2.2 ábra forrása: https://docs.moodle.org/39/en/Quiz_reports, utoljára megtekintve: 2020-03-26

[7] Személyes beszélgetés Mérő Bálinttal, a Redmenta társalapítójával

[8] https://www.commonsense.org/education/website/socrative, utoljára megtekintve: 2020-04-02

[9] https://help.socrative.com/en/articles/2155335-import-a-quiz, utoljára megtekintve: 2020-04-03

[10] Hussein Toman, Z., Hussein Toman, S., Joundy Hazar, M. (2019). An In-Depth Comparison Of Software Frameworks For Developing Desktop Applications Using Web Technologies. Journal of Southwest Jiaotong University, Vol. 54 No.4.

[11] Muittari, J. (2020). Modern web back-end: What happens in the back end of the web application? Bachelor's Thesis, Oulu University of Applied Sciences.

[12] https://docs.microsoft.com/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-6.0, utoljára megtekintve: 2021-03-15

[13] https://aspnetcore.readthedocs.io/en/stable/intro.html, utoljára megtekintve: 2021-03-15

[14] https://docs.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-6.0, utoljára megtekintve: 2021-04-07

[15] https://webassembly.org/roadmap/, utoljára megtekintve: 2021-04-07

[16] 4.1 ábra forrása: https://docs.microsoft.com/en-us/dotnet/csharp/roslyn-sdk/get-started/syntax-analysis, utoljára megtekintve: 2020-11-19

[17] https://docs.microsoft.com/en-us/dotnet/csharp/roslyn-sdk/get-started/syntax-analysis, utoljára megtekintve: 2020-11-19

[18] https://docs.microsoft.com/en-us/dotnet/csharp/roslyn-sdk/get-started/semantic-analysis, utoljára megtekintve: 2020-11-19

[19] https://docs.microsoft.com/en-us/archive/msdn-magazine/2017/may/net-core-cross-platform-code-generation-with-roslyn-and-net-core, utoljára megtekintve: 2020-11-30

[20] https://medium.com/machine-words/separation-of-concerns-1d735b703a60, utoljára megtekintve: 2021-04-29

[21] Evans, E. J. (2003). Domain-driven Design: Tackling Complexity in the Heart of Software

[22] https://medium.com/@shivendraodean/software-architecture-the-onion-architecture-1b235bec1dec, utoljára megtekintve: 2021-05-01

[23] https://oauth.net/2/jwt/, utoljára megtekintve: 2021-05-01

[24] https://docs.microsoft.com/en-us/ef/core/, utoljára megtekintve: 2021-04-07