

SZAKDOLGOZAT



MISKOLCI EGYETEM

Mesterséges intelligencia tantárgyi webes keretrendszer készítése

Készítette:

Ferencsik Márk

Programtervező informatikus

Témavezető:

Kunné Dr. Tamás Judit

MISKOLC, 2022

MISKOLCI EGYETEM

Gépészmérnöki és Informatikai Kar

Alkalmazott Matematikai Intézeti Tanszék

Szám:

SZAKDOLGOZAT FELADAT

Ferencsik Márk (UJTWLL) programtervező informatikus jelölt részére.

A szakdolgozat tárgyköre: szoftverfejlesztés

A szakdolgozat címe: Mesterséges intelligencia tantárgyi webes keretrendszer készítése

A feladat részletezése:

Gyűjtse össze és a szükséges mértékig ismerje meg a GEIAK130B Mesterséges Intelligencia tárgy keretében eddig hallgatók és oktatók által elkészített oktatási segédleteket, demonstrációs programokat, rendezze azokat a tantárgy tematikájának megfelelően.

Készítse el a rendszertervét egy webes tantárgyi keretrendszernek, amely a következő szolgáltatásokat nyújtja: letölthetővé teszi a tantárgy statikus erőforrásait (záróvizsga kérdések, tematika, előadásanyagok, feladatsorminták, stb.). Letölthetővé, vagy weben futtatható módon tanulmányozhatóvá teszi a meglévő demonstrációs alkalmazásokat. Bővítési felületet nyújt újabb erőforrások beküldéséhez, esetleg az ellenőrzött rendszerbeillesztéshez.

Válassza meg az alkalmas programozási és futtatókörnyezetet a program elkészítéséhez, választását indokolja!

Készítse el az alkalmazás programját és linkelje be, vagy töltsse fel a meglévő erőforrásokat!

Tesztelje a keretrendszer működését és készítsen hozzá egy tömör alkalmazási leírást, mely a keretrendszer részét is képezi!

Témavezető: Kunné Dr. Tamás Judit (egyetemi adjunktus)

A feladat kiadásának ideje: 2021. október 6.

.....
szakfelelős

EREDETISÉGI NYILATKOZAT

Alulírott **Ferencsik Márk**; Neptun-kód: UJTWLL a Miskolci Egyetem Gépészmérnöki és Informatikai Karának végzős Programtervező informatikus szakos hallgatója ezennel büntetőjogi és fegyelmi felelősségem tudatában nyilatkozom és aláírással igazolom, hogy *Mesterséges intelligencia tantárgyi webes keretrendszer készítése* című szakdolgozatom saját, önálló munkám; az abban hivatkozott szakirodalom felhasználása a forráskezelés szabályai szerint történt.

Tudomásul veszem, hogy szakdolgozat esetén plágiumnak számít:

- szó szerinti idézet közlése idézőjel és hivatkozás megjelölése nélkül;
- tartalmi idézet hivatkozás megjelölése nélkül;
- más publikált gondolatainak saját gondolatként való feltüntetése.

Alulírott kijelentem, hogy a plágium fogalmát megismertem, és tudomásul veszem, hogy plágium esetén szakdolgozatom visszautasításra kerül.

Miskolc, év hó nap

.....

Hallgató

1.

szükséges (módosítás külön lapon)

A szakdolgozat feladat módosítása

nem szükséges

.....

dátum

.....

témavezető(k)

2. A feladat kidolgozását ellenőriztem:

témavezető (dátum, aláírás):

konzulens (dátum, aláírás):

.....

.....

.....

.....

.....

.....

3. A szakdolgozat beadható:

.....

dátum

.....

témavezető(k)

4. A szakdolgozat szövegoldalt

..... program protokollt (listát, felhasználói leírást)

..... elektronikus adathordozót (részletezve)

.....

..... egyéb mellékletet (részletezve)

.....

tartalmaz.

.....

dátum

.....

témavezető(k)

5.

bocsátható

A szakdolgozat bírálatra

nem bocsátható

A bíráló neve:

.....

dátum

.....

szakfelelős

6. A szakdolgozat osztályzata

a témavezető javaslata:

a bíráló javaslata:

a szakdolgozat végleges eredménye:

Miskolc,

.....

a Záróvizsga Bizottság Elnöke

Tartalomjegyzék

1. Bevezetés	1
2. Tervezés	2
2.1. Rendszerleírás	2
2.2. Technológia	2
2.3. Követelmény specifikáció	3
3. Az elérhető technológiák összehasonlítása	4
3.1. Webes keretrendszerek	4
3.1.1. Vue.js	4
3.1.2. Angular	6
3.1.3. Node.js	9
3.1.4. Összehasonlítás	11
3.2. Adatbázisrendszerek	11
3.2.1. MySQL	12
3.2.2. Microsoft SQL Server	12
3.2.3. MongoDB	14
3.2.4. Összehasonlítás	16
4. Megvalósítás	18
4.1. Előkészületek	18
4.1.1. Visual Studio Code	18
4.1.2. Node.js	18
4.1.3. MongoDB	19
4.1.4. Egyéb potenciálisan hasznos programok	19
4.2. A program általános jellemzői	21
4.3. A program felépítése	22
4.3.1. A szerver konfigurációja	22
4.3.2. Autentikáció	23
4.3.3. Adminisztrátori műveletek	24
4.3.4. Fájlfeltöltés	25
4.3.5. Címek	26
4.3.6. A hallgatói felület	30
4.3.7. Az adminisztrátori felület	32
5. Összefoglalás	34
Irodalomjegyzék	35

1. fejezet

Bevezetés

A felsőoktatásban részt vevő hallgatók számos tantárgyat tanulmányoznak az itt eltöltött féléveik alatt. Ezeknek a tárgyaknak pedig szükségük van egy, a mai világban már elengedhetetlennek számító internetes felületre, amelyet a hallgatók bárhol, bármikor elérhetnek.

A szakdolgozatom célja az, hogy a GEIAK130B Mesterséges Intelligencia tárgyhoz egy ilyen felületet hozzon létre, webapplikáció formájában. Ezen a felületen elérhetővé válnak a tantárgyhoz kapcsolódó információk és anyagok, például a konzultációs időpontok, a zárthelyi dolgozat időpontjai, vagy az előadásfóliák.

Ügyelve arra, hogy csak illetékesek (egyetemi hallgatók) férjenek hozzá ehhez az oldalhoz, egy bejelentkezési felületre lesz szükségünk. Erről az oldalról csak akkor léphetnek tovább a felhasználók, ha rendelkeznek fiókkal. A regisztrált fiókok adatait pedig egy online adatbázisban tároljuk.

Tekintettel kell lennünk továbbá arra is, hogy szükségünk lesz egy olyan személyre is, aki felügyeli és karbantartja az alkalmazást, valamint a fiókokat. Ezt a feladatkört látja el majd a későbbiekben az adminisztrátor, akinek fő feladata a felhasználók regisztrálása, illetve adatbázisból való eltávolítása lesz.

2. fejezet

Tervezés

2.1. Rendszerleírás

A szakdolgozat feladat egy olyan rendszer létrehozását igényli, melybe a Miskolci Egyetem hallgatói – előzetes adminisztrátori regisztráció után – beléphetnek, így hozzáférve a Mesterséges Intelligencia című tantárgyhoz kapcsolódó anyagokhoz, példának okáért:

- az ütemtervhez,
- az előadásanyagokhoz,
- a zárthelyi dolgozat eredményeihez, vagy akár
- a záróvizsga tételeihez.

Elérhetőek lennének a tárgy előadásain, valamint gyakorlatain bemutatott programok, például a BrainMaker vagy a betanítható sakkrobot. Továbbá, biztosítani kell egy feltöltésre alkalmas felületet, amennyiben a felhasználók saját tartalmat szeretnének rendelkezésre bocsátani. Természetesen ez nem jelenne meg azonnal a weboldalon, valószínűleg illetékes (például adminisztrátori) jóváhagyás után kerülne fel ténylegesen a rendszerbe. Mindezek mellett az adminisztrátor(ok) számára is biztosított lenne egy felület, ahol

- hozzáadhatnak,
- módosíthatnak, vagy akár
- törölhetnek

felhasználói fiókokat.

2.2. Technológia

Ezen feladat egészen szerteágazó tudást igényel, továbbiakban ezeket fejteném ki. Tekintettel arra, hogy lényegében egy weboldal az alapja az egésznek, így elengedhetetlen némi HTML ismeret, melyet – a középiskolai tanulmányaimon kívül – a már korábban említésre került Webtechnológiák I című tárgy keretein belül volt szerencsém elsajátítani. Ebben segítségemre lesz a tárgy előadója (Agárdi Anita), valamint az általa

létrehozott GitHub repository [2]. Itt rendelkezésemre állnak a gyakorlati feladatok, valamint olyan példafeladatok, melyek segítségével könnyebben megértheti bárki a nyelv sajátosságait. Emellett, amennyiben ez mégsem lenne kielégítő, úgy a W3Schools oldalán további segítség található [6]. Ezen weboldalon számtalan programozási nyelv számára - így természetesen a HTML számára is – elérhetőek ismeretterjesztő szövegek, demonstrációs programok.

Ha már webprogramozás a téma, nem maradhat említés nélkül a dinamikus weboldal szíve – a JavaScript. A korábban már említett HTML nyelv mellett a JavaScript-tel is foglalkoztunk a Webtechnológiák I. tantárgy keretein belül. Továbbá, szakmai gyakorlatom során is volt részem nem kevés JavaScript programozásban.

Egy weboldal harmadik, egyben utolsó összetevője a stílus, a kinézet. Ezt CSS stíluslapokkal tervezem megoldani. Ebből a nyelvből mind középiskolában, mind a már oly sokszor említésre került Webtechnológiák I. tárgy tanulmányozása során mélyíthettem tudásomat.

Viszont itt még nem állhatunk meg. Ahhoz, hogy ebből egy szerteágazó webalkalmazás legyen, elengedhetetlen valamilyen backend keretrendszer használata. A következő alfejezetben felsorakoztatunk ezek közül néhányat.

Adattárolási módszerekre is szükségünk lesz. Tekintettel a potenciális felhasználók körére, valamint a biztonsági elvárásokra, valamilyen online adatbázisra is szükségünk lesz. Ezek lehetőségeire szintén hozunk pár példát a későbbiekben.

2.3. Követelmény specifikáció

Az igényes munka, a biztonság és a felhasználói élmény kívánalmai miatt követelményeket állítottunk fel magunknak a megvalósítani kívánt rendszerrel kapcsolatban.

Először is, a bejelentkezési rendszernek megfelelően biztonságosnak kell lennie. Ezt online adatbázisrendszerrel, titkosított jelszavakkal, valamint érvényesítési módszerekkel terveztük elérni.

A felhasználói kezelhetőség javítása érdekében pedig menürendszert, valamint stíluslapokat vezetünk be, ezáltal fejlesztve a rendszer megjelenését.

Természetesen ezeket az elvárásokat a felhasználói felület mellett az adminisztrációs felületre is alkalmazzuk.

3. fejezet

Az elérhető technológiák összehasonlítása

Alapvető technológiaként tekintünk a HTML, CSS, valamint a JavaScript nyelvekre, mivel ezek szolgálnak a legtöbb weboldal alapjául, ezek a legelterjedtebbek. A HTML a weboldal tartalmáért, a CSS a weboldal esztétikájáért, a JavaScript pedig az oldal dinamikus mivoltáért fog felelni.

A továbbiakban viszont szélesebb körű rendszerek álltak rendelkezésünkre. Először is tekintsük át néhány releváns, számunkra is elérhető keretrendszert.

3.1. Webes keretrendszerek

3.1.1. Vue.js

A Vue.js egy nyílt forráskódú frontend JavaScript keretrendszer, melyet legtöbbször felhasználói felületek, valamint single-page (egyetlen weboldalt használó, azt dinamikusan változtató) applikációk fejlesztésekor alkalmaznak.

Nagy előnye, hogy magasabb szintű funkcionalitást kölcsönöz a HTML applikációknak, melyet az ún. „direktíváival” ér el. Ezeknek egy része beépített, viszont a felhasználók saját maguk is elkészíthetik egyéni direktíváikat [7]. Emellett „virtuális DOM-ot” használ. Ez azt jelenti, hogy csak akkor módosul a valódi DOM, ha az applikáció változói frissülnek, így pedig a feldolgozási sebesség megnő például a közismert, és gyakran használt JQueryhez képest.

Erről a rendszerről egy cikk számol be bővebben [4]:

Mi az a Vue.js?

Manapság minden egyes weboldalnak, webáruháznak szüksége van a megfelelő böngészőoldali élmény, valamint funkcionalitás megteremtésére. Ez alapvető fontosságú: ugyanis ezek nélkül az adott online megjelenés nem fogja azokat az eredményeket hozni, amiket szeretnénk, és elvárnánk. A Vue.js ebben tud nekünk segíteni.

Aki dolgozott már JavaScripttel, az tisztában van vele, hogy egy viszonylag könnyen használható scriptnyelv, viszont hosszú távon natív JavaScriptet használni-na, ez sokak szerint nem igazán jó ötlet.

Gondolhatunk itt számos különféle funkcionalításra: egy felugró ablak megnyitása, bizonyos elemek elrejtése adott feltételek mellett: mindez szimpla JavaScripttel megoldva

több időbe telhet. Akkor pedig, ha komplexebb applikációról lenne szó, egészen biztosan rosszul járunk.

Ennek kapcsán érdemes lehet valamilyen jobb megoldás után néznünk.

Miért van szükség keretrendszerekre, könyvtárakra?

Keretrendszerekre (framework), valamint könyvtárakra (library) általában azért lehet szükség, mert ezeknek a segítségével jóval gyorsabban dolgozhatunk, valósíthatjuk meg az adott funkcionalitást.

Ezen felül, gondoljunk csak a böngésző kompatibilitásra: ez a probléma nem csupán CSS esetén merülhet fel. A megírt kódunk nem egyformán fog működni minden böngésző, illetve azok verziói alatt.

Mindezek eredményeként a mai modern webfejlesztésben gyakorlatilag mindenki keretrendszert, és könyvtárat is használ.

Mi az a Vue.js?

A Vue.js egy JavaScript könyvtár. Sok esetben keretrendszernek szokták hívni, pedig valójában nem az: az Angular, React, Vue trióból egyedül az Angular minősül frameworknek.

2014-ben jelent meg először, a Google egyik volt fejlesztője készítette eredetileg. Azóta hihetetlenül sok újításon ment keresztül, és a használati tendenciája egyre inkább felfelé ívelőnek látszik. Miért?

Miért a Vue.js?

A Vue.js egy nagyon könnyen megtanulható, és használható JavaScript könyvtár. Alacsony Learning Curve-vel rendelkezik, én annak idején az Udemy-n egy 20 órás tananyagot végig csináltam a témához kapcsolódóan, és egy kicsi gyakorlás után már teljesen tudtam is használni azokra a feladatokra, amikre szerettem volna.

A Vue.js-t annak ellenére, hogy könnyebben elsajátítható, rengeteg feladatra könnyedén fel lehet használni. Kifejezetten szeretik startupok is használni, valamint kiváló párosítást alkot Laravel keretrendszerrel.

Abban az esetben, ha még soha nem használtunk JavaScript keretrendszert, kifejezetten jó opció lehet, ugyanis a sikerélményünk garantált lesz.

Milyen feature-ök vannak a Vue.js-ben?

A Vue.js a háttérben egy virtuális DOM-ot használ. Amennyiben az applikációnk változói frissülnek, az igazi DOM is módosulni fog, ez egy sokkal gyorsabb performanszt, teljesítményt nyújt, mint például a jQuery.

Ez a könyvtár egy MVC-hez hasonló architektúrát követ. Az adataink, a különféle metódusok, valamint az oldalunk HTML része könnyedén szétválasztható a segítségével, ebből adódóan egy jól áttekinthető, átlátható kódot tudhatunk a magunkénak. Az adatok „egy helyre gyűjtésében” lehet segítségünkre a Vuex, ami teljesen engedi különválasztani az applikációnktól azokat, ezáltal a különféle állapotokat meghatározó adatok egy helyre kerülnek.

A Vue.js komponensekből épül fel. Ennek kapcsán könnyedén hozhatunk létre „újra-

hasznosítható” elemeket az alkalmazásunkon belül: ha például egy képgalériát szeretnénk létrehozni, akkor az egyetlen képet megtestesítő elem lehet egy ilyen komponens. Vagy akár egy űrlap, melyet a weboldalunkon máshol is használni szeretnénk.

A Vue segítségével képesek lehetünk akár többoldalas appokat is építeni, ebből adódóan az úgynevezett routing is jól meg van benne oldva.

Az Vue.js mellé rengeteg kiváló paketet, és third party plugint készítettek. Ezek közé tartozik például a már említett Vuex, vagy a Vuelidate, aminek a segítségével űrlap-érvényesítést valósíthatunk meg könnyedén.

3.1.2. Angular

A címben említett Angular egy elterjedt TypeScript alapú keretrendszer. Érdemes megjegyezni, hogy az ebben a dolgozatban felsorolt három keretrendszer közül ez az egyetlen tényleges keretrendszer, a Vue.js, valamint a Node.js pusztán JavaScript-könyvtárak.

Ennek a rendszernek a jellegzetessége, hogy HTML-eket egészít ki attribútumokkal, ebből adódik, hogy legtöbbször single-page alkalmazások fejlesztésekor szokták alkalmazni, melyekhez számtalan bemutató videó, valamint demonstrációs anyag létezik.

Továbbá, a három felsorolt lehetőség közül ezen keretrendszer tanulmányozásával foglalkozunk kizárólagosan Webtechnológiák II. tantárgy berkein belül.

Egy rövid (angol nyelvről fordított) szöveg mutatja be részleteiben a rendszert [1]:

Bevezetés az Angular koncepcióiba

Az Angular egy felület és egy keretrendszer, mely single-page kliens applikációk fejlesztését biztosítja HTML, valamint TypeScript segítségével. Megvalósítja az alapvető és az opcionális funkcionalitást TypeScript függvénykönyvtárak készleteként, melyek beilleszthetők az applikációkba.

Egy Angular applikáció architektúrája bizonyos alapvető koncepcióra támaszkodik. Az Angular keretrendszer építőkövei olyan Angular komponensek, melyek úgynevezett NgModulokba szervezettek. Az NgModulok összegyűjtik az összefüggő kódot funkcionális készletekbe; egy Angular applikáció ilyen NgModulok által van definiálva. Egy applikáció mindig rendelkezik legalább egy modullal, mely lehetővé teszi az úgynevezett "bootstrapping"-et, és jellegzetesen számos más modullal rendelkezik.

*A komponensek "view"-okat (későbbiekben: **nézeteket**) definiálnak. Definíció szerint ezek olyan képernyőelem-készletek, melyek közül az Angular választhat és módosíthat, a programlogika, valamint az adat alapján. A komponensek szolgáltatásokat használnak, amelyek olyan meghatározott funkcionalitást biztosítanak, amik nem kapcsolódnak közvetlenül a nézetekhez. A szolgáltatások biztosítói beilleszthetők a komponensekbe függőségek formájában, ezzel modulárisná, újrahasználatóvá, valamint hatékonyá téve a programunkat. A modulok, a komponensek, illetve a szolgáltatások olyan osztályok, melyek "dekorátorokat" használnak. Ezek jelzik a típusukat, és metaadatokat biztosítanak, melyek meghatározzák az Angular számára, hogyan használhatják őket.*

A komponensek osztályaihoz tartozó metaadatokhoz társul egy sablon, mely meghatározza a nézetet. A sablonban átlagos HTML elemekhez kapcsolódnak Angular direktívák és összeköttetésjelzők, melyek lehetővé teszik az Angular számára, hogy módosítsák a HTML elemeket a megjelenítés legenerálása előtt.

A szolgáltató osztályhoz tartozó metaadatok elérhetővé teszik az Angular számára szükséges információkat, így lehetővé téve a függőség beillesztést (DI, Dependency In-

jection) a komponensek részére.

Egy applikáció komponensei jellemzően definiálnak hierarchikusan rendezett nézeteket. Az Angular elérhetővé teszi az átírányító (Router) szolgáltatás számára, hogy segítséget nyújtson a fejlesztőnek a nézetek közötti navigációban. Az átírányító biztosítja a kifinomult, böngészőn belüli navigációs lehetőségeket.

[...]

Modulok

Az Angular NgModuljai különböznek, illetve kiegészítik a JavaScript (ES2015) modulokat. Egy NgModul meghatároz egy fordítási kontextust a komponensek készlete számára, melyek az applikáció tartományához, munkafolyamatához, vagy a hozzá kapcsolódó képességekészlethez kötődnek. Egy NgModul a komponensekhez társíthat hozzá kapcsolódó kódot (például szolgáltatást), hogy funkcionális egységet alkossanak.

Minden Angular applikáció rendelkezik egy gyökérmodullal, melynek megegyezés szerint AppModule a neve. Ez biztosítja a bootstrap mechanizmusokat, melyek indítját az alkalmazást. Az applikáció általában számos funkcionális modult tartalmaz.

Az NgModul - a JavaScript modulokhoz hasonlóan - képesek funkcionális beilleszténi még NgModulokból, valamint engedélyezni a saját funkcionálisuk exportálását, melyet így más NgModulok is használatba vehetnek. Például, hogy valaki használhassa az átírányító szolgáltatást az alkalmazásában, be kell importálnia a Router NgModult.

A kód különböző funkcionális modulokba való átszervezésével könnyebben kezelhetjük a komplex applikációk fejlesztését, akár a desing és az újrahasználhatóság szempontjából is. Továbbá, e technika az úgynevezett "lazy loading" ("lusta töltés") előnyét nyújtja. Ez lehetővé teszi, hogy igény szerint tölthetünk be modulokat, ezzel minimalizálva az indításnál betöltendő kód mennyiségét.

Komponensek

Minden Angular alkalmazás rendelkezik legalább egy komponenssel (a gyökér komponenssel), mely kpcsolatba hozza a komponens hierarchiát az oldal dokumentum objektum modelljével (DOM). Minden komponens egy osztályt definiál, mely tartalmazza az applikáció adatait és logikáját, és kapcsolatban áll a HTML sablonnal, amely definiálja a célkörnyezetben megjelenítendő nézetet.

A @Component() dekorátor azonosítja az alatta lévő osztályt komponensként, és rendelkezésre bocsátja a sablont, és a hozzá kapcsolódó komponens-specifikus metaadatokat.

A dekorátorok olyan funkciók, melyek JavaScript osztályokat módosítanak. Az Angular számos dekorátort definiál, melyek az osztályokhoz bizonyos metaadatokat rendelnek, így a rendszer tudja, mit jelentenek, illetve hogyan működnek az adott osztályok.

[...]

Sablonok, direktívák, és adatösszeköttetés

A sablonok HTML elemeket kombinálnak Angular jelölőkkel, melyek módosíthatják a HTML objektumokat a megjelenítésük előtt. A sablon direktívák programlogikát biztosítanak, az összeköttetésjelzők pedig összekapcsolják az alkalmazás-adatokat a DOM-mal. Kétféle adatösszeköttetést különböztetünk meg:

Az **eseményösszeköttetés** segítségével az applikáció képes "válaszolni" a felhasználói inputra a célkörnyezetben az alkalmazásadatok frissítésével.

A **tulajdonságösszeköttetés** lehetővé teszi, hogy számított értékeket illeszthessünk be az alkalmazásadatból a HTML elemekbe.

Mielőtt egy nézet megjelenítésre kerülne, az Angular kiértékeli a direktívákat, illetve feloldja a sablonban található összeköttetési szintaxisokat, így módosítva a HTML elemeket és a DOM-ot a program adatai, valamint logikája alapján. Az Angular támogatja a kétirányú adatösszeköttetést. Ez azt jelenti, hogy a DOM-ban bekövetkezett változások, mint például a felhasználói választások a programadatban is bekövetkeznek.

A sablonok használhatnak úgynevezett "pipe"-okat, melyek javítják a felhasználói élményt azáltal, hogy átalakítják a megjelenő értékeket. Például, pipe-okat használunk akkor, ha a felhasználó elhelyezkedése alapján szeretnénk megjeleníteni dátumot vagy valutát. Az Angular biztosít előre definiált pipe-okat gyakori átalakításokra, továbbá a fejlesztő is írhat saját pipe-okat.

[..]

Szolgáltatások és függőségbeillesztések

Az olyan adathoz vagy logikához, mely nem kapcsolódik az adott nézethez, és szeretnénk megosztani más komponensekkel, szolgáltatásoztlányokat hozunk létre. Ezek definícióját az @Injectable() dekorátor előzi meg. Ezen dekorátor biztosítja a metaadatokat, melyek lehetővé teszik, hogy más szolgáltatások beilleszthetők legyenek függőségekként az osztályunkba.

A függőségbeillesztés (DI) segítségével a komponens osztályok megtarthatják tömör és hatékony mivoltukat. Ezek nem kérnek le adatot a szerverről, nem érvényesítenek felhasználói adatbevitelt, és nem naplóznak közvetlenül a konzolra; ezek csak elküldik az adott feladatokat a felelős szolgáltatásoknak.

[...]

Átírányítás

Az Angular átírányító NgModul olyan szolgáltatást biztosít, mely lehetővé teszi az alkalmazáson belüli különböző applikáció állapotok, valamint nézet hierarchiák közötti navigációs útvonal definíálását. Ez a modell a böngésző-navigációs konvenciók által lehet ismerős:

- Ha a felhasználó beír egy URL-t a címsorba, a böngésző a kapcsolódó oldalra navigál.
- Az oldalon található linkre kattintva a böngésző egy új oldalra navigál.
- A böngésző "Vissza", valamint "Tovább" gombjaira kattintva a böngésző előzménye alapján az előző, illetve a következő (már meglátogatott) oldalra navigál.

Az átírányítás feltérképezi az URL-szerű útvonalakat. Viszont ezeket nem oldalak, hanem nézetek formájában teszi. Amikor egy felhasználó végrehajt valamit az oldalon (például rákattint egy linkre, mely a böngészőben egy új oldalt töltene be), az átírányító megszakítja a böngésző folyamatát, és megjeleníti, vagy eltünteti a nézetek hierarchiáját.

Ha az átirányító meghatározza, hogy a jelenlegi applikáció állapot egy bizonyos funkcionalitást igényel, és a modul, mely ezt definiálja, még nem töltődött be, úgy az átirányító igény szerint "lusta töltést" alkalmazhat a modulon.

Az átirányító értelmezi a link URL címét az applikáció nézeteinek navigációs szabálykészlete, valamint adatállapota alapján. Navigálhatunk új nézetekre amikor a felhasználó egy gombra kattint, vagy kiválaszt egy opciót a legördülő listából, illetve, ha bármi más forrásból "ingert" kapunk. Az átirányító naplózza a tevékenységeket a böngésző előzményeiben, így a "Vissza", valamint a "Tovább" gombok is rendeltetésszerűen működnek.

Ahhoz, hogy navigációs szabályokat létrehozassunk, a navigációs útvonalakat a komponensekkel kell társítani. Egy útvonal URL-szerű szintaxist használ, mely egyesíti a programadatot, pontosan úgy, ahogyan a sablonszintaxis egyesíti a nézeteket a programadattal. Ezután alkalmazhatjuk a programlogikát, hogy kiválasszuk, melyik nézetet jelenítsük meg, vagy rejtjük el, válaszként a felhasználói inputra, a saját hozzáférési szabályaink alapján.

3.1.3. Node.js

A Node.js egy JavaScript függvénykönyvtár, melyet kifejezetten webszerverek fejlesztésére hoztak létre.

Tulajdonságai közé tartozik az a tény, hogy skálázható, illetve aszinkron programozásra is képes, így nagyobb volumenű, jelentős áteresztőképességet igénylő projektekhez kiválóan alkalmazható.

A rendszer ezen felül backend és frontend projekteknek is remek alapot ad. Hogy pontosan milyen esetekben érdemes használni, azt megtudhatjuk az alábbi értekezésből [5]:

Node.js: miért, mikor, és mikor ne?

Az utóbbi pár évben a Node.js végigment azon az úton, amelyen a legtöbb új, felkapott technológia végigmegy. Amikor megjelent, hatalmas hype övezte, mindenki megváltónak kiáltotta ki, és a boldog-boldogtalan elkezdett gyakorlatilag mindent Node.js-ben írni. Ezt követte a keserű ébredés, amikor kiderült, hogy ez sem lesz az "egy és igaz" platform, amiben minden megvalósítható (ezt persze sosem állította senki), és sokan elfordultak a projekttől. Manapság leginkább a konszolidáció időszaka zajlik, és a megváltó státusz helyett inkább egy újabb lehetséges eszközként tekintünk a reportoárunkban a Node.js-re. Ebben a posztban azt vizsgáljuk meg, milyen esetekben lehet jó döntés a Node.js-t választani egy adott feladatra, és mely esetekben kifejezetten ellenjavallt.

Először nézzük meg, mik azok a tipikus alkalmazás kategóriák, amelyeknél érdemes megfontolnunk a Node.js platformot.

Realtime többfelhasználós alkalmazások

Ebben a kategóriában népszerű példaként szokták hozni a chatszobát, mint olyan alkalmazást, melyet mintha csak a Node.js-re öntöttek volna: nagyforgalmú, adatintenzív (de kis számításigényű) alkalmazás, amely elosztott platformon fut. Ezek az appok (és a példánk is) nagyon gyakran eseményvezéreltek, nagyszámú konkurens kapcsolatot kell kezelniük, és nem mellékesen mindezek mellett elfogadható felhasználói élmény is kell

nyújtaniuk.

API egy objektumadatbázis felett

Bár a realtime alkalmazások jelentik a Node.js savát-borsát, viszonylag adja magát olyan esetekben is, amikor egy API-t kell fejlesztenünk egy objektum alapú adatbázishoz (pl. MongoDB). Mivel az adatainkat már eleve JSON-ban tároljuk, a JavaScriptben történő kezelésük igen kényelmessé válik, illetve a szokásos OO-relációs eltérést sem kell lekezelnünk.

Ha például egy MongoDB adatbázis fölé Railsben, PHP-ban vagy más hasonló környezetben építenénk API-t, az adatok olvasásához és írásához számos extra lépést kellene beépítenünk: a JSON-t konvertálni kellene a saját modellünkre, azokat pedig visszaalakítani JSON-ra, hogy HTTP válaszként küldhessük őket. Ezeket Node.js esetén mind megspóroljuk, és nagyon hatékony tud lenni, ha az adatunk az egész stacken ugyanazon formátumban van.

Üzenetsorok kialakítása

Ha az alkalmazásunk nagyszámú kérést fogad, az adatbázis könnyen szűk keresztmetszetté válhat. Mint azt jól tudjuk, a Node.js nagyon jó a sok konkurrens kérés kezelésében, azonban mivel a DB hozzáférés egy blokkoló művelet, könnyen bajba kerülhetünk a teljesítményt illetően. A megoldás erre az, hogy a kliensnek már az adatbázis művelet elvégzése előtt választ adunk, mintha az sikeresen megvalósult volna, a valódi kérést pedig betesszük valamilyen sorba (erre használhatunk külső toolt, pl. ZeroMQ).

Ezzel a megközelítéssel a rendszerünk nagy terhelés alatt is válaszképes marad. Ez főleg olyan esetekben jó, ahol a változtatásoknak nem kell azonnal megjelenniük az adatbázisban, és az “eventual consistency” is elég (pl. likeok száma a Facebookon, felhasználói statisztikák, stb.).

Egy ilyen queue természetesen más környezetben is megvalósítható, azonban legtöbbször nem ugyanazon a hardveren, vagy sokkal alacsonyabb válaszidővel.

A következőkben nézzünk meg néhány esetet, amelyen a Node.js **kifejezetten elenjavallt** lehet.

Szerveroldali alkalmazás, mögötte relációs adatbázissal

Ez egy máig nagyon elterjedt és klasszikus felállás, és egy olyan, amelyre a Node.js kifejezetten rossz választás; egy Node.js/Express.js kombináció alkalmazása egyértelműen hátrányosabb mondjuk egy Railshez vagy Django-hoz képest.

Sajnos a Node.js ökoszisztémában az RDBMS eszközök még mindig elég fejletlennek mondhatók, az egyetlen említésre méltó ORM a Sequelize, de tudásban és használhatóságban máig elmarad a riválisoktól és relációs adatbázisok használatakor (főleg ha pl. Rails háttérrel rendelkezünk) sokszor úgy érezhetjük, újra és újra “fel kell találnunk a kerekét”.

Magas számításigényű szerveroldali alkalmazások

A Node.js által megoldott legfőbb probléma a blokkoló I/O általi teljesítménycsök-

kenés, amit jól meg is old. Amennyiben azonban számításigényes feladatokat végzünk benne, viszonylag gyorsan semmissé tehetjük a nagy áteresztőképesség nyújtotta előnyöket, hiszen bármely bejövő kérést blokkolni fog a jelenleg zajló számítás, hiszen ne feledjük: a Node.js egyszálú, és mindig egyetlen CPU magot használ.

Ha mégis szükség van ilyesmire az alkalmazásunkban, jobb azt afféle külső erőforrásként megvalósítani, és a korábban említett üzenetsoros megoldással kezelni.

3.1.4. Összehasonlítás

3.1. táblázat. A felvázolt keretrendszerek összehasonlítása (2/1)

Rendszer	Íródott	Cross-platform
Vue.js	JavaScript nyelven	Támogatja
Angular	JavaScript nyelven	Támogatja
Node.js	C/C++/JavaScript nyelveken	Támogatja

3.2. táblázat. A felvázolt keretrendszerek összehasonlítása (2/2)

Rendszer	DOM	Erősség
Vue.js	Használ (virtuálisit)	Single-page alkalmazások, felhasználói felületek
Angular	Használ	Single-page alkalmazások
Node.js	Nem használ	Skálázható webszerverek

3.2. Adatbázisrendszerek

Adatbázisrendszereken belül két nagy csoportot tudunk megkülönböztetni:

- SQL, és
- NoSQL adatbázisok.

Az SQL-alapú rendszerek táblázatok formájában tárolják az adatot, melyekben mezők, valamint rekordok találhatóak. Például, egy iskolai adatbázisban a tanulók nyilvántartása egy táblában történik, melyben változatos mezőket vesznek fel (tanuló neve, születési ideje stb.). Egy tanuló adatait tartalmazó sort rekordnak nevezzük. Fontos még megemlíteni, hogy elsődleges kulcsot erősen ajánlott használni az ilyen, és hasonló jellegű adatbázisokban. Elsődleges kulcsnak nevezzük azt a mezőt, mely a tábla rekordjait egyértelműen meghatározza. Ebben az esetben az előző példában elsődleges kulcs szerepét veheti fel a tanulók diákigazolvány száma.

Ezzel szemben a NoSQL (a nevéből is adódóan) nem SQL-alapú, vagyis nem táblákban tárolják az adatokat. Legtöbbször dokumentumok formájában tárolnak adatot. Ezen belül is léteznek különböző megoldások a rendszerezés tekintetében, így például kollekciók, címkék, vagy akár metainformációk alapján is rendezhetjük az adatbázisunkat. Általánosságban elmondható, hogy JSON-szerű formátumban tárolnak. Ebből adódik, hogy a lekérdezések változatossága leszűkül, cserébe viszont nagyobb teljesítményre, valamint skálázhatóságra lesz képes.

3.2.1. MySQL

A MySQL egy (a nevéből is adódóan) SQL-alapú relációs adatbázis-kezelő szerver (RDBMS, Adatbáziskezelés I. tárgyból foglalkoztunk ezzel). Ez az egyik legelterjedtebb adatbázis-kezelő, mivel ingyenesen elérhető bárki számára szabad szoftverlicen-szen keresztül, valamint számos platformon és programozási nyelvvel kompatibilis a használata. Elsődlegesen CLI-n (Command Line Interface-en, parancssoros kezelői fe-lületen) keresztül használható, viszont létezik hozzá GUI (Graphical User Interface, grafikus felhasználói felület), melyet elérhetünk a MySQL hivatalos weboldaláról.

Alább olvasható egy rövid leírás a MySQL általános jellemzőiről [8]:

MySQL

A MySQL egy többfelhasználós, többszálú, SQL-alapú relációs adatbázis-kezelő szer-ver.

A szoftver eredeti fejlesztője a svéd MySQL AB cég, amely kettős licencceléssel tet-te elérhetővé a MySQL-t; választható módon vagy a GPL szabad szoftver licenc, vagy egy zárt (tulajdonosi) licenc érvényes a felhasználásra. 2008 januárjában a Sun felvá-sárolta 800 millió dollárért a céget. 2010. január 27-én a Sun felvásárolta az Oracle Corporation, így a MySQL is az Oracle tulajdonába került.

A MySQL az egyik legelterjedtebb adatbázis-kezelő, aminek egyik oka lehet, hogy a teljesen nyílt forráskódú LAMP (Linux–Apache–MySQL–PHP) összeállítás részeként költséghatékony és egyszerűen beállítható megoldást ad dinamikus webhelyek szolgálta-tására.

Elérhetősége programnyelvekből

Egyedi illesztőfelületekkel az adatbázis-kezelő elérhető C, C++ , C#, Delphi, Eif-fel, Smalltalk, Java, Lisp, Perl, PHP, Python, Ruby és Tcl programozási nyelvvel. Egy MyODBC nevű ODBC interfész további, ODBC-t kezelő nyelvek számára is hozzáfér-hetővé teszi az adatbázis-kezelőt. A MySQL számára az ANSI C a natív nyelv.

Adminisztrációja

A MySQL adatbázisok adminisztrációjára a mellékelt parancssori eszközöket (mysql és mysqladmin) használhatjuk. A MySQL honlapjáról grafikus felületű adminisztráló eszközök is letölthetők: MySQL Administrator és MySQL Query Browser.

Széles körben elterjedt és népszerű adminisztrációs eszköz a PHP nyelven írt, nyi-tott forráskódú phpMyAdmin. A phpMyBackupPro (amelyet szintén PHP-ban írtak) adatbázisok (akár időzített, ismétlődő) mentésére szolgál eszközkül.

3.2.2. Microsoft SQL Server

A Microsoft SQL Server a világszerte híres Microsoft cég által fejlesztett relációs adatbázis-kezelő rendszer. Természetesen elsősorban Microsoft alapú rendszerekre ter-vezték, így a lefedettsége kevésbé szerteágazó a MySQL-hez képest. Viszont rengeteg szolgáltatással kompenzálja ezt a hátrulütőjét. A cég ismeretes arról, hogy különböző (például „Standard”, vagy „Enterprise”) kiadások formájában bocsátja piacra terméke-it, hogy a felhasználók minél szélesebb körét tudja kiszolgálni, és ez az említett SQL

Server esetében sincs másképp. A Microsoft hivatalos weboldalán tájékozódhatunk az elérhető kiadásokról, valamint tulajdonságaikról.

Itt olvasható egy kis bemutatás az adatázisról [3]:

Mi az SQL Server?

Ebben az oktatóanyagban megismerjük, hogy milyen szoftver ez, hogyan tudjuk az SQL-t alkalmazni a rendszerünkben annak kihasználására, és hogyan lehet az Ön számára működni annak érdekében, hogy megkönnyítse a feladatát. Bár röviden áttekintjük az SQL szerveret, hadd mondjam el, hogy az SQL adatbázis-kezelő rendszer. Most az volt az oka, hogy rövid ötletet kellett adnom neked az SQL szerverről: az, hogy mielőtt továbblépnénk, tudnod kell, mi az adatbázis.

Tehát azoknak, akik nem tudják, az adatbázis úgy definiálható, mint a számítógépen elektronikusan tárolt, bőséges adatgyűjtemény. Olyan platformot biztosít nekünk, amelyet általában ugyanazon entitás adatainak tárolására használnak, ahol a követelményeinknek megfelelően frissíthetjük és kezelhetjük az adatokat. Most már tudja, hogy mi az adatbázis, és itt az ideje, hogy felfedezzük az SQL szerveret!

Meghatározás

Ez egy relációs adatbázis-kezelő rendszer, amelyet a Microsoft fejlesztett ki 1988-ban. Valójában egy háttér alkalmazás, amely lehetővé teszi az adatok tárolását és feldolgozását. Egyszerűen fogalmazva, platformot kínál nekünk, ahol frissíthetjük, módosíthatjuk és kezelhetjük az adatokat. Relacionális adatbázis-kezelő rendszernek nevezzük annak jellege miatt, hogy az adatokat táblákban tárolja, ahol a táblák ugyanazon entitás adatait tárolják.

Hogyan könnyíti meg az SQL Server a munkát?

Nagyon hasznos, ha az adatokat a háttérrendszerben tároljuk annak feldolgozása érdekében. Egy nagyon egyszerű felülettel rendelkezik, amely segít a háttérkép-fejlesztőnek, hogy sokkal inkább az adatok gondozására összpontosítson, ahelyett, hogy félne az adatok működésétől. Mivel a Microsoft terméke, a .Net keret könnyen integrálható vele, mivel ugyanabba a szervezetbe tartozik. Az adatok tárolásának más eszközeivel összehasonlítva, például az Excel, a szöveg és így tovább, az adatbázist mindig előnyben részesítik, mivel nagyszerű adatfeldolgozási képességgel rendelkezik, magas szintű biztonságot nyújt, és főleg nagy adattárolási képessége miatt.

[...]

Az SQL Server előnyei

Számos pluszpont van az SQL szerver használatához az egyéb adattárolási eszközökkel szemben. Néhányat az alábbiakban felsoroljuk.

- **Adatfeldolgozás** - Az SQL szerver lehetővé teszi az adattárolások feldolgozását a kívánt kimenet előállításának érdekében. Az adatok kiszámíthatóak voltak az SQL szerver segítségével.
- **Nagy tárhely** - nagyszámú adat tárolható az SQL szerverben. Magas tárolóka-

pacitása miatt ez a legjobb a szervezetben történő adattárolás során.

- **Integráció a kezelőfelülettel** - Az SQL szervert integrálhatják a kezelőfelület alkalmazásba is, hogy a dinamikus adatcsere mechanizmusát biztosítsák. Nagyon gyakran használják a webes alkalmazásokkal való integrációban.
- **Könnyen csatlakoztatható a .Net-hez** - Mivel az SQL szerver és a .Net keretrendszer egyaránt a Microsofthoz tartozik, mindkettő nagyon könnyen csatlakoztatható. Az SQL szerver nagyon jól működik vagy simán működik, ha összekapcsolják a .Net webhelyen kifejlesztett alkalmazással.

Ki a megfelelő közönség az SQL Server technológiák megtanulásához?

Azok az emberek, akik érdeklődnek a háttérfejlesztés karrierjének növeléséről, nagyszűrű közönség lehetnek az SQL szerver technológiába való mély merüléshez. Ez a technológia fantasztikus növekedési lehetőséget is kínál, amelyet várhatóan tovább fog tartani az e-kereskedelem és a közösségi média exponenciális növekedése mellett.

Következtetés

Nagyon rövid és éles szavakkal, az SQL szerver az az eszköz, amelyet egy relációs adatbázis-kezelő rendszer mechanizmusának megvalósításához használnak. Ez lehetővé teszi a fejlesztőknek, hogy dolgozzanak az adatokkal annak érdekében, hogy jó élményt nyújthassanak a felhasználó számára. A szervezetekben az SQL szerver az adatfeldolgozás legelőnyösebb eszköze, mivel nagy mennyiségű adat kezelésére képes.

3.2.3. MongoDB

A MongoDB kitűnik az előbbi két adatbázis-kezelő közül azzal, hogy az ún. NoSQL adatbázisszerverek közé tartozik. Csoportjának legelterjedtebb adatbáziskezelője. Dokumentumorientált, az adatokat JSON-szerű formátumban tárolja. A Microsoft SQL Server-hez hasonlóan itt is különböző kiadások érhetőek el a potenciális felhasználók számára, attól függően, hogy milyen, valamint mekkora méretű projektet terveznek megvalósítani.

Alább található egy rövid bemutató a rendszerről [9]:

NoSQL adatbázisok

A NoSQL adatbázisok a relációs sémától eltérően működő adatbázisok összefoglaló neve. A név valamennyire megtévesztő, mert a fogalomnak kevés köze van az SQL nyelvhez - ehelyett a releváns különbség az adatrepresentációban és a sémában van. De mégis miért van szükségünk új fajta adatbázisokra, amikor a relációs adatbázisok régóta jól használhatóak? Egy kis méretű adatbázis egyszerű sémával könnyen leírható relációs modellben, még kényelmes is. De az alkalmazásaink fejlődnek, egyre több funkciót kell ellátniuk, ezzel együtt komplexebbé válik a séma is, illetve egyre több adatot kell eltárolni és nő az adatbázis. Ez egy bizonyos határ felett komplikálttá válik.

A relációs adatbázisok hátránya, hogy a folyamatos változások, séma változtatást igényelnek. Ahhoz, hogy ezt karban tudjuk tartani folyamatosan migrálni kell az adatokat és ez nem egyszerű feladat. Továbbá teljesítmény problémákkal, azaz inkább konzisztencia-

és skálázási problémákkal járhat, ha relációs adatbázist használunk - ezzel azonban nem foglalkozunk mélyebben.

Ezekre a problémákra a NoSQL adatbázisok nyújtanak megoldást. Ebben a világban elhagyjuk a szigorú sémákat, helyette egy flexibilis sémát fogunk alkalmazni. Azaz nem lesznek erős elvárásaink az adatbázisban tárolt adatokkal szemben.

A MongoDB alap koncepciói

A MongoDB egy kliens-szerver architektúrájú nem-relációs adatbázis. A kép jobb oldalán látható a mongod, azaz Mongo démon, vagyis az a processz, ami az adatbázis elérését biztosítja. A másik oldal a mi alkalmazásunk, ahonnan a kliens kapcsolódik a szerverhez egy hálózati kapcsolaton keresztül. Ez a hálózati kapcsolat az ún. wire protocol-on keresztül történik, ez a MongoDB saját protokollja. Ebben a protokollban JSON formájú adat kommunikáció zajlik binárisan (azaz BSON).

A MongoDB architektúrája

Logikai felépítés

Egy MongoDB-alapú adatbázis rendszer legfelső rétege az ún. klaszter, ebbe szervezzük a szervereket. Mi klaszterekkel ebben a tárgyban nem foglalkozunk, azok a skálázás eszközei. A második szint a szerver szintje (a mongod processz), ami alatt az adatbázis foglal helyet. Egy szerver/klaszter több adatbázist tárolhat. Az adatbázisok pedig gyűjteményekből (collection) épülnek fel. Ha a relációs adatbázisokkal meg akarjuk feleltetni, akkor a gyűjtemények a táblák megfelelői, ezen belül a sorok/rekordok pedig a gyűjteményben tárolt dokumentumok lesznek.

Nézzük ezeket pontosabban.

Dokumentum

A dokumentum a MongoDB alap tárolási egysége. Egy dokumentum egy JSON (jellegű) fájl, tehát kulcs-érték párokat tartalmaz. Maga a MongoDB BSON-ként, bináris reprezentációként tárolja mindezt.

```
{
  name: "sue",
  age: 26,
  status: "A",
  groups: [ "news", "sports"]
}
```

Kulcsoknak többnyire bármilyen szabad szöveget választhatunk, de a neveknek egyedinek kell lenniük és nem kezdődhetnek a \$ karakterrel. A nevek case sensitive-ek. Az érték lehet szöveg, szám, dátum, bináris, beágyazott elem, null, vagy akár a fenti példában a groups kulcsnál láthatóan tömb is - relációs adatbázisban ezt így nem lehet reprezentálni.

Az objektum orientált világban egy dokumentum felel meg egy objektumnak. Fontos megkötés, hogy a dokumentumok maximális mérete 16MB lehet, és ez nem konfigurál-

ható érték.

Gyűjtemény

Relációs tábla analógiája a gyűjtemény, de nincs sémája, így ezeket létrehozni, definiálni se kell, első használatkor a rendszer automatikusan létrehozza őket. Úgy fogalmazhatjuk meg, hogy a gyűjtemény a "hasonló" dokumentumok gyűjtőhelye. Bár nincs séma, indexeket ennek ellenére definiálhatunk a gyűjteményeken, amely a gyors keresést fogják segíteni. Séma hiányában nincs tartományi integritási kritérium, tehát például a helyes adattípusok és tartományi kritériumok biztosításában az adatbázis nem nyújt segítséget.

Adatbázis

Az adatbázis ugyanazt a célt szolgálja, mint relációs adatbázisban. Ez fogja össze az alkalmazás adatait logikailag. Illetve hozzáférési jogosultságokat adatbázis szinten tudunk adni. Az adatbázisok neve case sensitive és konvenció szerint tipikusan csupa kisbetű.

Kulcs

Minden dokumentum egyértelmű azonosítója az `_id` mező, más kulcsot nem tudunk definiálni. Ezt a mezőt beszúrásakor nem szükséges explicit megadni (de lehet), tipikusan a kliens driver vagy a szerver generálja (alapértelmezésben egy 12 bájtos ObjectId-t készít).

Az `_id` mezőtől függetlenül egyediséget indexek segítségével tudunk garantálni. Amennyiben szükséges, definiálhatunk tehát más, kulcs-szerű mezőket is. Az így definiált egyedi mezők lehetnek összetettek is (tehát lehet több mező együttes egyediségét előírni).

Külső kulcs hivatkozások MongoDB-ben nincsenek. Tudunk hivatkozni más dokumentumokra azok kulcsainak bemásolásával, azonban ezekre a rendszer nem vállal garanciát (pl. a hivatkozott dokumentum törölhető).

3.2.4. Összehasonlítás

3.3. táblázat. A felvázolt adatbáziskezelők összehasonlítása (2/1)

Adatbázis	Rendszertípus	Kezelőfelület
MySQL	SQL	Elsődlegesen parancssoros
Microsoft SQL	SQL	Elsődlegesen grafikus
MongoDB	NoSQL	Elsődlegesen parancssoros

3.4. táblázat. A felvázolt adatbáziskezelők összehasonlítása (2/2)

Adatbázis	Alkalmas	Séma
MySQL	Struktúrált adatokhoz	Fix
Microsoft SQL	Struktúrált adatokhoz	Fix
MongoDB	Szemistruktúrált adatokhoz	Dinamikus, opcionális

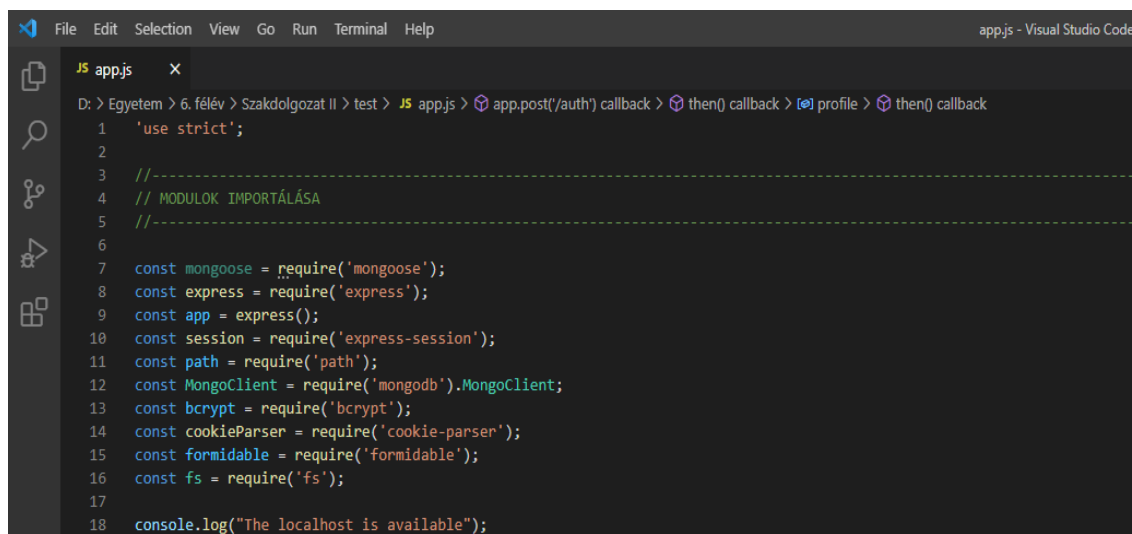
4. fejezet

Megvalósítás

4.1. Előkészületek

4.1.1. Visual Studio Code

Első lépésben szükségünk van egy olyan fejlesztőkörnyezetre, mely rendelkezik terminállal. Választásom a *Visual Studio Code* nevű környezetre esett. Egyszerű, segítőkész, könnyen átlátható fejlesztőkörnyezetről beszélünk, mely számos programozási nyelvet támogat, illetve segítséget nyújt használatukkor (4.1).



4.1. ábra. A Visual Studio Code felülete

4.1.2. Node.js

Az előző fejezetben összehasonlításra került néhány potenciális rendszer, jómagam pedig a *Node.js* mellett döntöttem, abból az egyszerű okból kifolyólag, hogy ezt ajánlotta a témavezetőm, valamint ez volt a legszimpatikusabb számomra. Következő lépésben üzembe kell helyoznunk az említett keretrendszert, melyben a programot írjuk. A rendszert a hivatalos weboldaláról érhetjük el. Letöltés és telepítés után rendelkezésünkre is áll a szoftver. Első lépésben konfigurálnunk kell a program "szívét", a *package.json* fájlt. Ez tartalmazza a fontos metaadatokat a projektünkhöz (például a program belépési pontját), ezek közül is kiemelve a "függőségeket" (dependency-eket). A függőségek

azt jelölik, hogy milyen modulokra (és azok mely verzióira) van szükség a program teljesértékű futtatásához. A *package.json* konfigurálásához az alábbi parancsot kell begépnünk:

```
$ npm init
```

A folyamaton belül, pár további változó megadása után már ténylegesen készen áll a projektünk.

Előreláthatólag szükségünk lesz modulokra, így (a megfelelő mappába való elnavigálás után) a következő parancsot kell begépnünk a terminálba:

```
$ npm install <modulnev>
```

Telepítés után a függőség bekerül a *package.json* fájlba, így a rendszer tudni fogja, hogy a megadott modult telepíteni kell használat előtt (abba az esetben, ha például más mappából futtatjuk a projektet).

Ha minden készen áll a használatra, akkor a lentebb látható paranccsal futtathatjuk a programunkat:

```
$ node <fajl.kiterjesztes>
```

Ezzel készen is vagyunk a Node.js telepítésével, viszont még szükségünk lesz további előkészületekre.

4.1.3. MongoDB

Ahogy webes keretrendszert, úgy adatbázist is választani kell. A választásom pedig a *MongoDB*-re esett, mégpedig azért, mert eddig csak SQL alapú rendszerekkel volt dolgom (Oracle SQL, MySQL), és ki szerettem volna próbálni valami újat, továbbá azért is, mert Webtechnológiák II tárgyból még hasznát vehetem ennek az adatbázis-rendszernek.

Ahogy a Node.js-t, úgy a MongoDB-t is a hivatalos weboldaláról érhetjük el. Ebben az esetben is telepítésre, valamint installációra lesz szükség. Telepítés után pedig csatlakoznunk kell az adott adatbázishoz. Ezt elősegítendő, a program rendelkezik egy saját parancssoros felülettel. Itt az alábbi parancsot kell megadnunk:

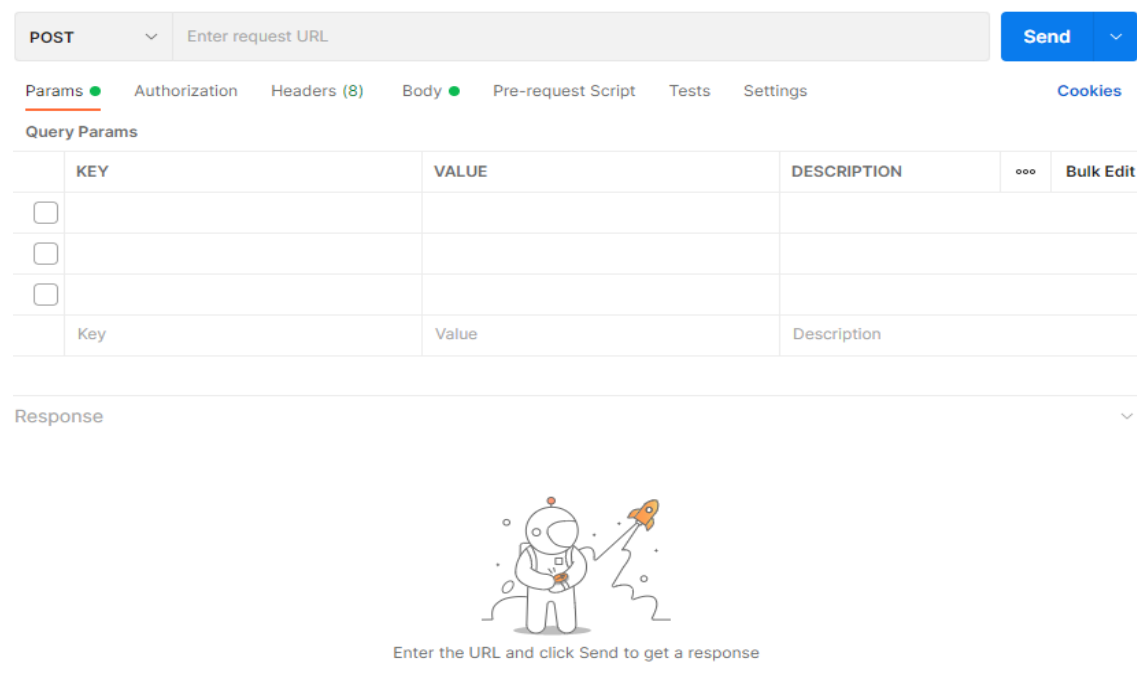
```
$ mongod
```

Ennek hatására a konfigurációs fájl (*mongo.cfg*) alapján csatlakozik az adatbázishoz, így a Node.js-ből is hozzáférhetünk a csatlakoztatott adatbázishoz, és annak tartalmához (például a kollekciókhoz).

4.1.4. Egyéb potenciálisan hasznos programok

Postman

Említést érdemel egy további alkalmazás, mely nagy segítséget nyújt az applikációnk tesztelésében. Ez pedig nem más, mint a *Postman*. Ez az asztali alkalmazás képes - többek között - felhasználói adatbevitelt szimulálni, így tesztelhetjük a webapplikációnkat különböző kérések elküldésével (4.2).



4.2. ábra. A Postman grafikus felülete

Például bejelentkezési folyamatot így futtathatunk:

1. Be kell írunk az URL-t, melyre tesztelni szeretnénk, valamint a kérés típusát (**get** és **post** típusokat különböztetünk itt meg, mivel ez a két leggyakrabban alkalmazott módszer).
2. Meg kell adnunk a küldendő adatokat. Ezek elég változatosak lehetnek, leggyakrabban **header** és **body** paramétereket és értékeiket adunk át.
3. Végül pedig a "**Küldés**" gombbal elküldjük a kérést a megadott szerverre.

Küldés után a válaszlakban találhatjuk a kérés eredményeit. Itt láthatjuk például a HTTP válasz státuszkódját, a futás idejét miliszekundumokban, viszont a legfontosabb a válaszkód, melyet akár többféle nyelven tudjuk megjeleníteni. E kód segítségével tudjuk ellenőrizni azt, hogy valóban helyesen működik-e a programunk.

MongoDB Atlas

Említésre került, hogy a MongoDB applikációja alapvetően parancssoros felületű. Ennek értelmében a csatlakoztatott adatbázist és annak tartalmát (például a kollekcióit) szintén parancssoros felületen keresztül érhetjük el. Amennyiben viszont igényünk van rá, megtehetjük mindezt grafikus felületen is. Ezt a *MongoDB Atlas* nevű webalkalmazás teszi lehetővé (4.3. ábra).



4.3. ábra. A MongoDB Atlas felületén egy kollekció dokumentumai

4.2. A program általános jellemzői

A weboldalak mind HTML nyelvben íródnak, viszont *.ejs* kiterjesztésűek. Ennek az az oka, hogy egy bizonyos *EJS view engine*-t alkalmazunk. Ennek okát a későbbiekben kifejtjük. A weboldalak stílusát CSS stíluslapokkal írjuk meg. Külön témát alkotunk a bejelentkezési oldalra (kék), a felhasználói-, adminisztrátori felületekre, illetve a sikeres műveletet értesítő oldalakra (zöld), valamint a hibajelző oldalakra (piros).

A továbbiakban fontos megjegyezni, hogy az átirányításokat *Express* rendszerrel hajtjuk végre. Szándékosan nem modulként hivatkozunk rá, hiszen ez egy, a Node.js számára fejlesztett backend keretrendszer, mely nagyban megkönnyíti a webapplikációk fejlesztésének menetét. Telepíteni viszont modulként kell:

```
$ npm install express
```

Ahogy már említettük, az Express végzi az átirányításokat (is). Például:

```
app.get('/invalid', function(req, res) {  
  res.render("invalid");  
});
```

Ebben a példában a */invalid* nevű oldalra navigálás esetén az *invalid* nevű fájlt tölti be a rendszer. Itt viszont vissza kell térnünk a *.ejs* kiterjesztésre. Ugyanis az Express HTML oldalakat nem tud betölteni. Emiatt kell használnunk az EJS-t, amely viszont megköveteli, hogy *.html* helyett *.ejs* legyen az oldalak kiterjesztése. Ez lényegében semmilyen változtatással nem jár, szintaktikájuk (ebben az esetben teljesen) megegyezik.

Szóba került a munkamenet-menedzselés is. Ezt az Express egyik alrendszerével, az *Express-session*-nel, valamint a *Cookie-parser* nevű modullal oldjuk meg. Míg az előbbi magát a munkamenetet kezeli, az utóbbi segítségével tudjuk módosítani a sütitket (példának okáért a süti elévülési idejét - ez határozza meg a munkamenet hosszát, mely esetünkben 30 perc).

A projekten belüli navigációt, illetve útvonalkezelést a *Path* nevű modullal végezzük. E modul képes többek között fájlok kiterjesztését, kiterjesztés nélküli nevét ("basename") lekérdezni, útvonalak abszolút voltát, vagy akár két útvonal közti relatív útvonalat meghatározni. Nekünk a `resolve()` függvény lesz segítségünkre, mellyel az argumentumként megadott útvonalat és más mappákat lehet abszolút útvonalként beállítani. Emellett a `join()` metódus is hasznunkra lesz, mivel ezzel útvonalakat lehet összevonni. Továbbá egy bizonyos `__dirname` változót is biztosít számunkra a modul, mely a futtatás helyét tárolja (mindig azt az útvonalat veszi fel értékként, amelyből futtattuk a programot).

Igényt tartunk még a *Formidable* modulra is. Ez lehetővé teszi számunkra, hogy bejövő (felhasználó által elküldött, így rendelkezésünkre bocsátott), főleg fájlfeltöltési űrlapokat kezeljünk. Leglényegesebb funkciója a `parse()` függvény. Ezzel tudjuk a kapott űrlapot értelmezni, mind a kiválasztott fájl(ok), mind a hozzá kapcsolódó mezők szempontjából.

Végül pedig beimportáljuk az *fs* nevű modult. Ezen modullal fájlműveleteket végezhetünk, például:

- Fájl létrehozása
- Fájl létezésének ellenőrzése
- Fájl módosítása
- Fájl törlése
- Fájl átnevezése

Számunkra a fájlfeltöltés folyamatában lesz hasznos. Erre a feltöltés részletezésénél térünk vissza.

4.3. A program felépítése

4.3.1. A szerver konfigurációja

Az applikációnk forráskódja elején a modulok importálása található. Ezután a szerver konfigurációja következik.

Első körben definiáljuk a munkamenethez szükséges paramétereket az Express-session segítségével:

- **secret:** A munkamenet egyedi azonosító kulcsa. Ez általában egy bonyolult, véletlenszerűen generált string, melyet nem szabad nyilvánosságra hozni.
- **saveUninitialized:** Egy boolean (logikai) típusú változó, mely jelzi, hogy az *uninitialized* (létrehozott, de értékeket nem hordozó) munkamenetek mentésre kerüljenek-e, vagy sem.
- **resave:** Szintén egy logikai értéket hordoz. Ez határozza meg, hogy mi történjen akkor, ha egy kérés során nem módosul a munkamenet. **True** érték esetén mentésre kerül, **false** esetén viszont nem. Versengéses kérések esetén (például amikor a felhasználó több párhuzamos kérést küld a szerver felé) a *false* a jobb választás.

-
- **cookie:** Itt tudjuk módosítani a süti változóit. Mi csak egyetlen változót állítunk be, a `magAge`-et. Ez határozza meg a süti (ezzel együtt a munkamenet) lejáratát. A mi esetünkben ez az érték fél óra.

A következő lépésben az Express működését szabályozzuk.

Az Express `json()` metódusa egy úgynevezett "middleware" metódus, melynek segítségével JSON-ként tudjuk értelmezni a bejövő kéréseket. Ennek a funkciónak köszönhető többek között az is, hogy a `req` változót és annak paramétereit használni tudjuk.

A következő metódus, melyet igénybe veszünk, a `urlencoded()`. Ez szintén egy middleware funkció, mely meghatározza, hogy csak UTF-8 kódolású oldalakat értelmezzünk.

Ezután a `static()` függvényt hívjuk meg. Ezzel határozzuk meg, hogy melyik mappából töltsük be a futáshoz szükséges eszközöket (ebben az esetben, az EJS-eket, a stílusokat, valamint a letölthető tartalmat).

Következik az átirányítást kezelő motor konfigurációja. Ezt az Express `set()` metódusával tudjuk beállítani. A konfiguráció két fontos lépést foglal magában:

- Beállítjuk a `view engine` paramétert. Mi az EJS-t használjuk, így az értékét erre állítjuk.
- Majd pedig meghatározzuk a `view` változót, ennek értékét az EJS fájlok főmappájában határozzuk meg

Utolsó változtatásként pedig meghívjuk a Cookie-parser modult.

4.3.2. Autentikáció

A program következő részében az autentikációval foglalkozunk. Ez egy `post` kérés, mely azt jelenti, hogy adatot küldünk a szervernek, mivel új adatot szeretnénk feltölteni, vagy meglévőt szeretnénk módosítani. Itt a bevitt adat hosszára, illetve típusára vonatkozó megkötések nincsenek. Az URL-ben nem jelenik meg a küldött adat, ezáltal a böngésző előzményei közt sem tárolódik. Az előző tények miatt érzékeny (fontos/titkos) adatokat érdemesebb `post` metódussal küldeni. Ennek értelmében tehát az autentikációt is `post` metódussal kezeljük.

A `login.ejs` fájlban létrehozuk a bejelentkezési űrlapot, melynek `action` paraméterét `post`-ra állítjuk, így jelezve azt, hogy az űrlap `post` metódussal kerül küldésre. Az űrlap két `<input>` mezőből áll, egy a felhasználónak, egy a jelszónak, előbbi `text`, utóbbi `password` típusú. A `text` mezőben minden karakter megjelenik, míg a jelszó mezőben csillagok jelennek meg, a titkosítást elősegítendő. Az űrlaphoz tartozik még a *Bejelentkezés* gomb is, mellyel az űrlap tartalma elküldhető.

Az elküldött űrlapot az applikációnkban kezeljük. Először is bekérjük a felhasználó által bevitt adatokat, a nevet és a jelszót. Kiíratjuk a konzolra a felhasználói adatokat, majd csatlakozunk az adatbázishoz, ahol első körben rákeresünk a felhasználónévre. Abban az esetben, ha nem létezik, hibaüzenetet küldünk. Ha mégis létezik, a `bcrypt` modul segítségével összehasonlítjuk a begépelt jelszót az adatbázisban szereplő, kódolt jelszóval. Magától értetődő, hogy abban az esetben, ha nem egyeznek, szintén hibaüzenetet küldünk a felhasználónak. Egyezés esetén viszont eltároljuk a munkamenet adatait (például a felhasználónevet), a Cookie-parser modul legenerálja a felhasználóhoz tartozó süti-adatokat (így a nagy jelentőséggel bíró lejárat dátumot is), végül

pedig átirányít a megfelelő oldalra (a "nem-admin" felhasználókat a főoldalra, az adminisztrátort pedig a felhasználó hozzáadása oldalra).

4.3.3. Adminisztrátori műveletek

Következőleg az adminisztrátori műveletek kezelését határozzuk meg. Fontos megjegyezni, hogy mivel mindhárom művelet adatot oszt meg a szerverrel, ezért ezek mindegyike `post` metódusként kezelendő.

Felhasználó hozzáadása

Első lépésként az `adminadd.ejs` fájlban létrehozunk a megfelelő űrlapot, az `action` paraméterét pedig `post`-ra állítjuk. Az űrlap tartalma:

- **Felhasználónév:** Egy text típusú `<input>` mező, ide kerül a hozzáadandó felhasználó neve.
- **Jelszó:** Password típusú `<input>` mező, ez a leendő felhasználó jelszavát tartalmazza.
- **Jelszó újra:** Password típusú `<input>` mező, a jelszó helyes bevitele miatt van szükség erre.
- **Hozzáadás:** Egy submit típusú input-gomb, a kitöltött űrlapot ennek segítségével tudjuk elküldeni a szervernek.

Elküldés után az applikációban kezelhetjük a felhasználói fiók hozzáadásának folyamatát.

Itt először különböző változókból eltávolítjuk a felhasználó által bevitt adatokat: a felhasználónevet, a jelszót, valamint az újbóli jelszóbevittet. Ezután ellenőrzést hajtunk végre a jelszó mezőkön. Ha nem egyeznek, az adminisztrátor *"A jelszavak nem egyeznek!"* hibaüzenetet kapja meg, majd vissza kell lépnie. Egyezés esetén csatlakozunk az adatbázishoz, és egy `findOne()` lekérdezéssel keresztül megvizsgáljuk, hogy létezik-e az adott névvel felhasználói fiók. Ezzel lezárjuk az első lekérdezést. Ha létezik, szintén hibaüzenet kerül megjelenítésre (*"Már létezik ilyen felhasználói fiók!"*). Ha nem létezik, elindítjuk a második lekérdezést, mely viszont már `insertOne()` típusú, ugyanis ezzel tudjuk beilleszteni a dokumentumba a felhasználói fiókot. Ezután a sikeres fiókhozzáadásról üzenetet kapunk, majd visszalépünk az adminisztrációs felületre.

Felhasználó módosítása

Az előző metódushoz hasonlóan először az adatbeviteli felületet hozzuk létre (`adminmodify.ejs`). Ugyanazokat a bevitt mezőket tartalmazza, mint a fiók hozzáadása, viszont az itt található mezők azonosítója más, a megkülönböztethetőség végett (Például a hozzáadás folyamatában a felhasználónevet tartalmazó mező azonosítója `addUName`, míg a módosítás űrlapjában található felhasználónév-mező azonosítója `modifyUName`).

Adatfeltöltés, és elküldés után az alkalmazásban folytatjuk a kezelést.

Változókból eltávolítjuk a felhasználó által beírt adatokat. Az első lépés itt is a jelszavak egyezőségének ellenőrzése. Ha nem egyeznek, hibaüzenetet küldünk. Ha egyeznek,

inicializáljuk az első csatlakozást a szerverhez, aholis a felhasználónév meglétét ellenőrizzük. Ha nem létezik, hibaüzenetet küldünk, mivel csak meglévő felhasználói fiókot tudunk módosítani. Amennyiben létezik, egy `updateOne()` típusú lekérdezéssel módosítjuk a felhasználói fiók adatait. Visszajelzést küldünk a sikeres módosításról, majd visszalépünk az oldalra.

Felhasználó eltávolítása

Itt már csak egy felhasználónévre (és természetesen a *Törlés* gombra) lesz szükségünk, hiszen a név alapján egyértelműen meghatározható, melyik fiókot szeretnénk eltávolítani a dokumentumból. Így az *adminremove.ejs*-ben található űrlap csak a felhasználónévhez szükséges `<input>` mezőt, valamint az elküldést intéző gombot tartalmazza.

A következő lépés már ismert. Felhasználói adatbevitel, majd pedig elküldés. Továbbiakban az applikáció foglalkozik a metódussal.

A törlendő felhasználói fiók nevének változóban történő eltárolása után csatlakozunk az adatbázishoz, majd `findOne()` típusú lekérdezéssel rákeresünk a felhasználónévre. Ha a keresés nem járt sikerrel, hibaüzenet jelenik meg. Találat esetén egy új csatlakozást kezdeményezünk, ahol már `deleteOne()` típusú lekérdezés segítségével töröljük az adott névhez tartozó felhasználói fiókot. Sikeres adattörlés esetén visszajelzést küldünk az adminisztrátornak, majd visszalépünk.

4.3.4. Fájlfeltöltés

A következőekben a fájlok feltöltésének kezelését részletezzük. Az eddigiekhez hasonlóan - tekintettel arra, hogy ez a művelet is adatot oszt meg a szerverrel - ez a metódus is `post` típusú.

Fájl feltöltése

A folyamat a */fileupload* oldalon zajlik. Első körben a *formidable* modul segítségével letároljuk a beérkező űrlapot, mely tartalmazza a feltöltendő fájlt. Ezután a modul `parse()` metódusát vesszük igénybe, ezzel értelmezzük és dolgozzuk fel az űrlap tartalmát. Alapvető hibaellenőrzést végzünk: Amennyiben a fájl neve megegyezik az üres string-gel (nincs fájlnev, azaz nincs kiválasztott fájl), úgy hibaüzenetet küldünk a felhasználónak (*Nincs kiválasztott fájl!*). Ellenkező esetben először változóban eltároljuk:

- a fájl(ok) feltöltésének mappáját (*./views/files/uploads/*), a leendő fájlnévvel és kiterjesztésével együtt (a feltöltési útvonalat a *path* modul segítségével hozzuk létre),
- az eredeti fájl elérési útvonalát, és
- az eredeti fájl tartalmát az *fs* fájlkezelő modul `readFileSync()` függvénye segítségével.

Következő lépésben szintén az *fs* modul lesz hasznunkra, most a `existsSync()` metódust hívjuk meg, ennek segítségével a fájlok feltöltésének mappáját vizsgáljuk, azt keresve, hogy van-e a mappában azonos nevű fájl. Ha létezik ilyen, megakadályozzuk a feltöltést, és hibaüzenetet küldünk (*Már létezik ilyen nevű fájl!*), így kiszűrve az azonos

nevű fájlok feltöltésének problémáját. Viszont ha nem létezik, szabad az út a feltöltéshez. Még egyszer az *fs* modult hívjuk segítségül, majd a `writeFile()` funkcióját meghívva a változóban korábban már eltárolt feltöltési útvonalra kiírunk egy új fájlt, és feltöltjük az eredeti fájl `readFileSync()` által feldolgozott tartalmával.

Program feltöltése

Ez a metódusa */programmeupload* oldalon történik, viszont a feltöltés kezelése szinte azonos a fájlfeltöltéssel. Az űrlap tartalmának változóban történő letárolása után a kiválasztott fájl ellenőrzését vizsgáljuk. Ha van kiválasztott fájl az űrlapban, letároljuk a fájlfeltöltéshez hasonlóan 3 változóban a feltöltés helyét (*./views/programmes/uploads/*), fájlnevvvel és kiterjesztéssel együtt, az eredeti elérési útvonalat, valamint az eredeti fájl tartalmát. Keresünk a mappában azonos nevű fájlt. Ha egyedi a fájlnev, akkor az eredeti fájl tartalmát kimásoljuk az általunk létrehozott, új fájlba, majd a sikeres feltöltő művelet után visszajelzést küldünk.

4.3.5. Címek

Az alkalmazás eddigi részében kizárólag `post` metódust használtunk az *Express* modulal, ezután viszont csak `get`-re lesz szükségünk, mivel ezekben a kérésekben a szerverről kérünk le adatot. További jellemzői közé tartozik még, hogy a `post`-tal ellentétben ezek az oldalak könyvjelzőként menthetők, valamint újratöltésükkor nem ütközünk problémába (`post` kérés újratöltése esetén az adatok újra elküldésre kerülnek, például **netbankon belüli tranzakciós kérés oldalának újratöltése esetén a tranzakció megismétlődik**, ezt kiküszöbölendő a böngésző értesítést küld ilyen esetben). Továbbá a küldés paraméterei megjelennek a böngésző címében, illetve mentésre kerülnek a böngésző előzményei közt. Ebből kifolyólag érzékeny adat továbbításakor nem érdemes ezt a metódust használni!

A címeket funkciójuk alapján különböző csoportokba osztjuk:

- bejelentkezési oldal
- felhasználói felület oldalai
- adminisztrációs felület oldalai
- sikerességet visszajelző oldalak
- hibát jelző oldalak

Bejelentkezési oldal

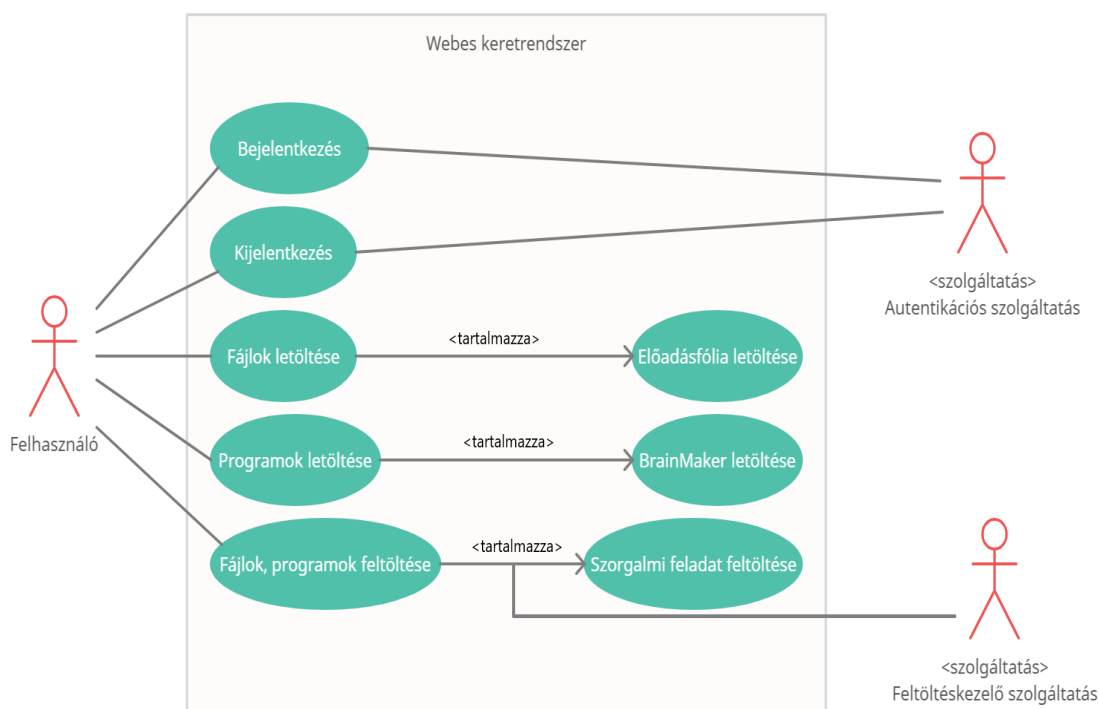
Ide egyetlen oldal tartozik, a bejelentkezési oldal, mely egyben az URL gyökerét is jelzi (vagyis, domain név betöltése esetén ez jelenik meg). Elérési útvonala `"/`. Ezen az oldalon tudnak a felhasználók (és az adminisztrátor) bejelentkezni a megfelelő oldalra (a felhasználók a *home* nevű főoldalra, az adminisztrátor pedig az */adminadd* oldalra), valamint a munkamenet lejárataát követően is erre az oldalra navigálhatnak.

Felhasználói felület oldalai

Ide soroljuk azokat az oldalakat, melyek között a (bejelentkezett) felhasználók navigálni tudnak. Amint azt már korábbi fejezetekben részleteztük, a felhasználók képesek lesznek:

- böngészni fájlok között
- letölteni fájlokat, programokat, illetve
- feltölteni saját tartalmat.

Ezen lehetőségeket szemlélteti az alábbi use-case diagram (4.4. ábra):



4.4. ábra. A felhasználói felület

A navigációt a weboldal felső részében található zöld menü segítségével végezhetik a felhasználók. A menü felett jelenik meg annak az oldalnak a neve, melyen a felhasználó éppen tartózkodik. A *mainStyle.css* nevű CSS stíluslap adja ezen oldalak kinézetét, mely szerint a kurzor menüpontokra való húzása (**hover**) esetén a kijelölt menüpont szürke kiemelő színt kap, így megkülönböztetve a menüpontokat. A *Fájlok*, valamint a *Programok* menüpontok *hover* eseménye esetén legördülnek, és két almenüpont válik előrhetővé számunkra: a *Fájlok* menüpontból a *Tananyagok* és a *Feltöltés*, a *Programok* menüpontból pedig a *Demonstrációk* és a *Feltöltés* almenüpontok jelennek meg. Az almenüpontok háttérszíne fekete, kijelölés esetén viszont ugyanazt a szürkés árnyalatot kapják, mint a főmenüpontok. Kattintás hatására pedig a tényleges navigáció zajlik le. Attól függően, hogy melyik menüpontra kattintottunk, a navigáció változatos lehet:

- **Főoldal:** Navigáció a *" /home "* oldalra.
- **Fájlok/Tananyagok:** Navigáció a *" /files "* oldalra.

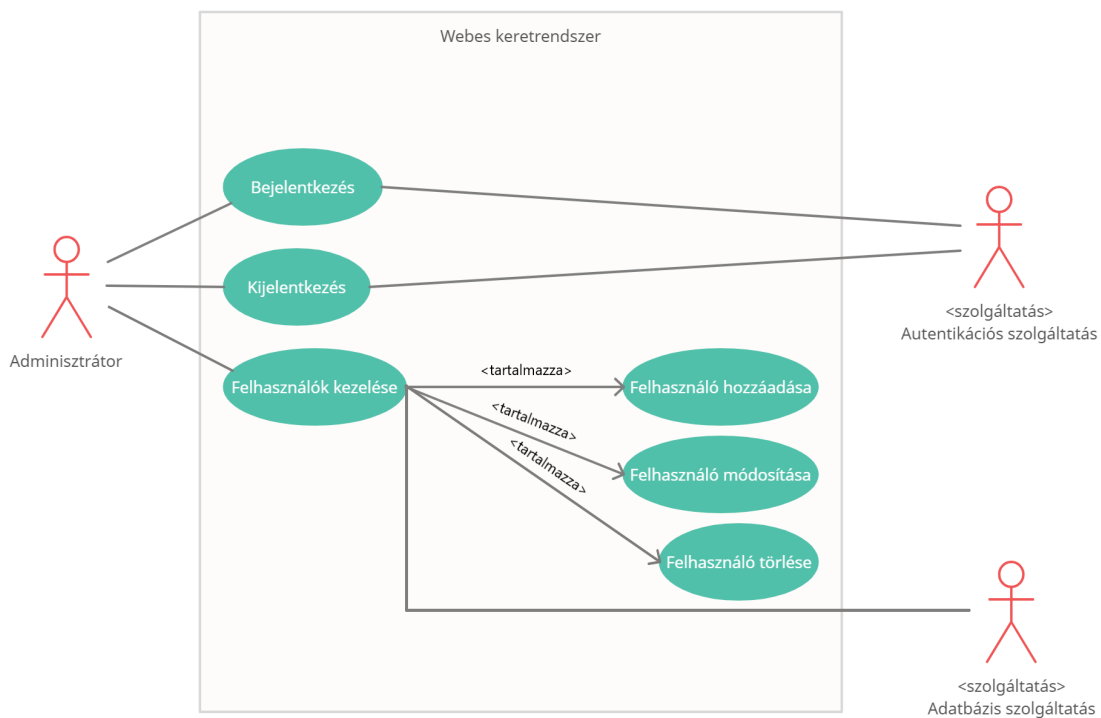
- **Fájlok/Feltöltés:** Navigáció a `" /fileupload"` oldalra.
- **Programok/Demonstrációk:** Navigáció a `" /programmes"` oldalra.
- **Programok/Feltöltés:** Navigáció a `" /programmeupload"` oldalra.

Adminisztrációs felület oldalai

Ide azokat az oldalakat soroljuk, melyek között az adminisztrátor böngészni tud. Ezen oldalakon az adminisztrátornak lehetősége van:

- felhasználót hozzáadni,
- felhasználó meglévő adatait módosítani, valamint
- felhasználói fiókot törölni

A megfelelő felhasználóba való bejelentkezés után a program automatikusan az adminisztrátori felületbe továbbít minket. Az itt rendelkezésünkre álló lehetőségeket szemlélteti az alább található use-case diagram (4.5. ábra):



4.5. ábra. Az adminisztrátori felület

A felület stílusa megegyezik a felhasználói felület stílusával, mivel ezen oldalak is a `mainStyle.css` stíluslapot használják. A főmenüpontok zöldek, a jelenlegi oldalt jelző cím szürke, menüpont kijeöléskor pedig szürke háttérrel kapnak. Itt nincsenek almenüpontok, csak három főmenüpont:

- **Felhasználó hozzáadása:** Navigáció az `" /adminadd"` oldalra.
- **Felhasználó módosítása:** Navigáció az `" /adminmodify"` oldalra.

-
- **Felhasználó törlése:** Navigáció az `"/adminremove"` oldalra.

Sikerességet visszajelző oldalak

Ide tartoznak azok az oldalak, melyek a sikeres műveletek (például adatbázis sikeres frissítése) után kerülnek megjelenítésre. Ezek mindegyike a *successStyle.css* nevű stíluslapot veszi igénybe. Ebbe a kategóriába soroljuk:

- **Adatbázis sikeres frissítése:** Navigáció a `"/dbsuccess"` oldalra. Ide az adminisztrátor oldalán található *Felhasználó hozzáadása*, */Felhasználó módosítása*, valamint a */Felhasználó törlése* műveletek sikeres elvégzése után jutunk el.
- **Feltöltés sikeres:** Navigáció az `"/uploadsuccess"` oldalra. A */fileupload*, illetve a */programmeupload* oldalakon elvégezhető feltöltés (végbement, sikeres) művelete után jelenik meg a felhasználók számára ez az oldal.

Hibajelző oldalak

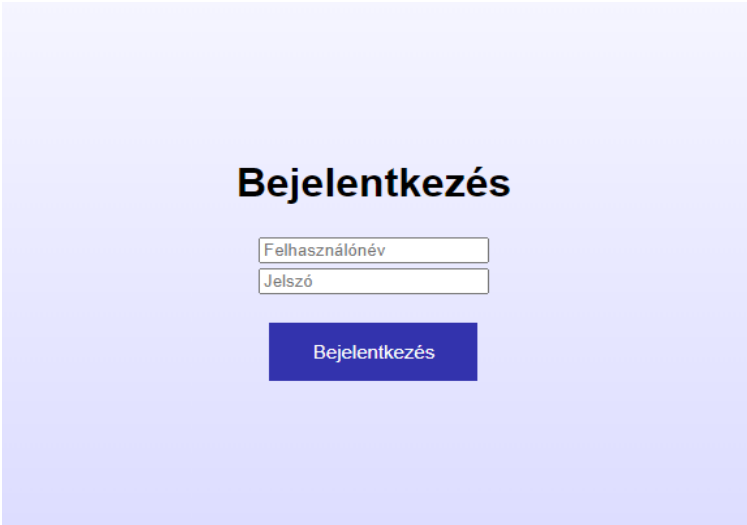
Ebbe a csoportba tartoznak azok az oldalak, melyek a hibás műveleteket jelzik vissza, vagy elkerülendő problémák miatt van szükség rájuk. Ezen oldalak megjelenése a */errorStyle.css* stíluslap alapján történik. Ide tartoznak:

- **Helytelen jelszó:** Navigáció a `"/badauth"` oldalra. Ezt akkor jelenítjük meg, ha a felhasználó, vagy az adminisztrátor nem a fióknévhez kapcsolt jelszót gépele be a jelszó mezőbe.
- **Már létező fájl:** Navigáció a `"/fileexists"` oldalra. Ezzel a hibaüzenettel találkozunk abban az esetben, *"Fájlok/Feltöltés"*, vagy a *"Programok/Feltöltés"* menüpont alatt olyan állományt szeretnénk feltölteni, melynek neve már megtalálható a megfelelő *"files/uploads"*, vagy *"programmes/uploads"* mappában (tehát már feltöltésre került azonos nevű fájl).
- **Helytelen vagy lejárt munkamenet:** Navigáció a `"/invalid"` oldalra. Akkor szembesülünk ezzel az üzenettel, ha a bejelentkezéstől számított 30 perces munkamenet lejár, és az idő lejártá után próbálunk meg műveleteket végezni az oldalon (például másik oldal betöltése a menün keresztül). Továbbá, amennyiben a címsorból próbáljuk meg elérni az oldalakat, szintén ezzel a hibával találkozunk. Érdekes lehet megjegyezni, hogy míg a többi hibajelző oldalon *"Vissza"* gomb található, mely a böngésző előzménye alapján vezet vissza az előző meglátogatott oldalra a `history.back()` paranccsal, addig ezen az oldalon egy *"Bejelentkezés"* gomb található, mely visszairányítja a böngészőt a `window.location` paraméter változtatásával a bejelentkezési felületre.
- **Nincs kiválasztott fájl:** Navigáció a `"/nofile"` oldalra. Amikor a felhasználó úgy küldi el a *"Fájlok/Feltöltés"*, vagy a *"Programok/Feltöltés"* oldalon található űrlapot, hogy nem választott ki fájlt, akkor ez a hibaüzenet jelenik meg számunkra.
- **Jogosulatlan felhasználó:** Navigáció a `"/notadmin"` oldalra. Erre akkor van szükség, ha egy nem-adminisztrátor szeretne hozzáférni (címsor segítségével) az adminisztrátori felülethez.

-
- **Nincs ilyen felhasználó:** Navigáció a `" /notfound"` oldalra. A bejelentkezési oldalról kaphatunk ilyen hibajelzést. Az autentikáció során a felhasználó által begépett név alapján keresünk felhasználói fiókot az adatbázisban. Ha nem találunk egyet sem (nincs ilyen névvel felhasználói fiók), akkor erre az oldalra irányít minket az applikáció.
 - **Nem egyező jelszavak:** Navigáció a `" /notmatching"` oldalra. Ezt a hibaüzenetet két oldalról is kaphatjuk: az egyik a *Felhasználó hozzáadása*, a másik pedig a *Felhasználó módosítása* nevű, adminisztrátori felületen fellelhető oldal. Értelemszerűen mindkét oldalon ugyanaz a hiba váltja ki ennek a hibaüzenetnek a megjelenítését: Ha az adminisztrátor nem ugyanazt a jelszót gépeli be a két *Jelszó* mezőbe.
 - **Már létező felhasználó:** Navigáció a `" /userexists"` oldalra. Ez az adminisztrátor számára megjelenő üzenet. Akkor következik be ez a hiba, ha olyan felhasználói fiókot próbál hozzáadni az adatbázishoz, melynek neve már szerepel az adatbázisban. Könnyű belátni, hogy két azonos nevű felhasználó nem szerepelhet a fiókok nyilvántartásában.

4.3.6. A hallgatói felület

A hallgatói felület eléréséhez először a login oldalon való bejelentkezés szükséges. Így tehát a felület elérése előtt elsőként a bejelentkezési felülettel találkozunk a hallgatók/felhasználók (4.6. ábra).



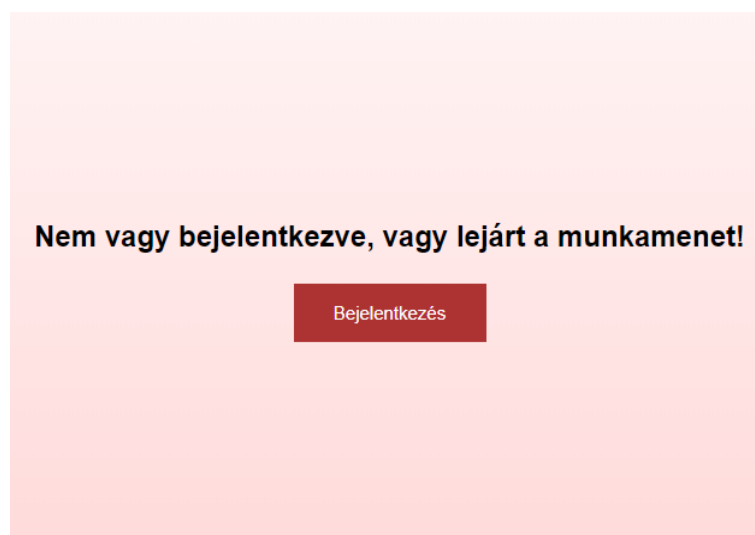
4.6. ábra. A bejelentkezési felület

Sikertelen bejelentkezés esetén hibaüzenettel szembesülünk. Ezt okozhatja az, hogy a felhasználónév nem található az adatbázisban, vagy az, hogy a névhez tartozó jelszó nem megfelelő. Sikeres bejelentkezési folyamat esetén viszont a hallgatói főoldalon találjuk magunkat (4.7. ábra).



4.7. ábra. A főoldal menüje

Fontos megjegyezni, hogy a munkafolyamat a bejelentkezés pillanatától számított 30 perc múlva kiléptet, és újabb bejelentkezési folyamat után enged vissza (4.8. ábra). Továbbá, a *Kijelentkezés* gombra kattintva - azon kívül, hogy átirányít a bejelentkezési felületre - véget ér a munkafolyamat.



4.8. ábra. A munkamenet lejáratát jelző hibaüzenet

A főoldalon a legördülő menü lesz a felhasználók segítségére, ugyanis ezen keresztül történik a navigáció, melyet a korábbi fejezetekben részleteztünk. Maga a menü egyszerű, gombokból (`<button>`-okból) áll, melyeket stíluslapok segítségével kényelmesebbé, esztétikusabbá alakítunk.

Annak érdekében, hogy egyéb fontos információt meg tudjunk jeleníteni, létrehoztunk egy "sidebar"-t. Ez nem más, mint a weboldalak bal oldalán elhelyezkedő oszlop,

melyet adattárolásra rendszeresítettünk. Ennek szélessége mindig az adott képernyő méretéhez igazodik, pontosan a képernyő-szélesség 12%-át foglalja el. Ennek megvalósítása egy egyszerű táblázattal történt. A menü alatt egy "láthatatlan" táblázatot hoztunk létre (egy olyan táblázatot, melynek 0 képpontnyi vastag szegélye van, így ugyan van szegélye, de nem látható). Ennek a táblázatnak két cellája van. A bal oldali cella szolgáltatja a már említett sidebar-t. A jobb oldali cella tölti ki a képernyő-szélesség maradékát, melyben a menüpontok szerinti tartalmat jelenítjük meg.

Mindezen felül a képernyő jobb felső sarkában található a *Kijelentkezés* gomb, melynek segítségével visszatérhetünk a bejelentkezési oldalra. Értelemszerűen a kijelentkezés folyamata után a munkamenetünk véget ér, így ha vissza szeretnénk térni a felhasználói felületre, újabb bejelentkezésre lesz szükségünk.

4.3.7. Az adminisztrátori felület

A login oldalon a megfelelő (adminisztrátori) felhasználóba való bejelentkezés után a program automatikusan az adminisztrátori felületbe továbbít minket (4.9. ábra).



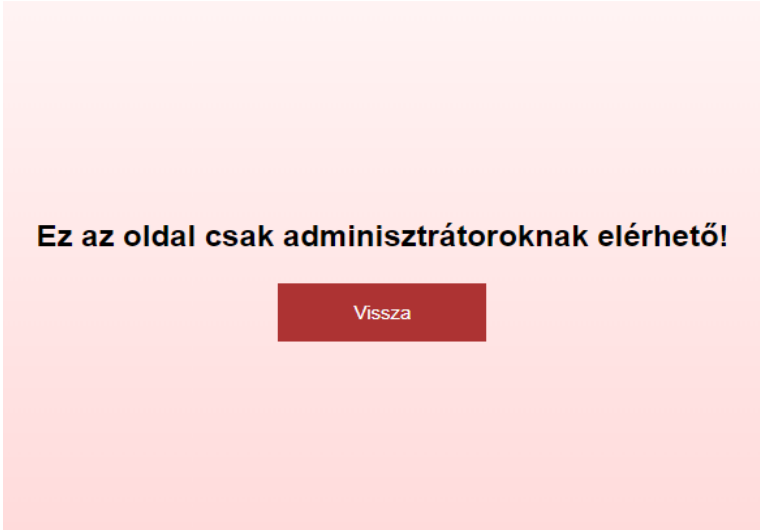
The screenshot displays the administrator interface. At the top, a green banner contains the text "Üdv a szerveren, adminisztrátor!". Below this, a green navigation bar features three buttons: "Felhasználó hozzáadás", "Felhasználó módosítás", and "Felhasználó törlés". The main content area is light green and contains a login form with the following elements: a label "Felhasználónév:" followed by a text input field, a label "Jelszó:" followed by a text input field, a label "Jelszó újra:" followed by a text input field, and a green "Hozzáadás" button at the bottom.

4.9. ábra. Az adminisztrátori felület

Amennyiben illetéktelenül próbálkozunk hozzáférni ehhez a felülethez (tehát, ha hallgatói fiókba jelentkezők be, vagy ha egyáltalán nem jelentkezők be), úgy hibaüzenet fogad minket (4.10)

Ellenkező esetben (adminisztrátori fiókba való bejelentkezéskor) a rendszer átirányít a "*Felhasználó hozzáadása*" (*/adminadd*) oldalra. Ez a felület hasonló a hallgatói felülethez, hiszen ugyanazt a stíluslapot (*mainStyle.css*) használja. Itt is megtalálható egy menü, azonban itt nem ugyanazok a menüpontok találhatók:

- **Felhasználó hozzáadása:** Ezen az oldalon lehet új felhasználói fiókot hozzáadni az adatbázishoz.
- **Felhasználó módosítása:** Itt lehet egy felhasználó már meglévő fiókjának jelszavát módosítani.
- **Felhasználó törlése:** Itt pedig egy meglévő felhasználói fiókot lehet a neve alapján törölni az adatbázisból.



Ez az oldal csak adminisztrátoroknak elérhető!

Vissza

4.10. ábra. Illetéktelen felhasználó esetén megjelenő hibaüzenet

Az előző alfejezetben ismertetett *sidebar*, valamint a *Kijelentkezés* gomb itt is megtalálható.

Érdemes megjegyezni azt a tényt, hogy munkamenet szempontjából az adminisztrátort is felhasználóként kezeljük! Ez azt jelenti, hogy a bejelentkezéstől számított 30 perc után az adminisztrátort is kijelentkezteti a program, és újra be kell jelentkeznie.

5. fejezet

Összefoglalás

A szakdolgozatom létrehozása során betekintést nyerhettem a manapság aktuális, web-fejlesztéshez használatos eszközökhöz, így a Node.js, valamint a MongoDB rendszerekkel való munkavégzés által számottevő tudásra tettem szert. Ezeken belül megtanulhattam, hogy hogyan épül fel egy autentikációs folyamat, mely magában foglalja az adatbázissal való kommunikációt is. Emellett elsajátíthattam egy reszponzív webalkalmazás fejlesztését, a menürendszerrel a fájlfeltöltés-kezeléseken át egészen a munkamenet-kezelésig. Alkalmam nyílt a szoftvertesztelés apróbb, ám annál hatásosabb és segítőkészebb folyamataiba is.

A MongoDB keretein belül betekintheztem egy NoSQL adatbázisrendszer működésének folyamatába. Megtanultam e rendszernek adattárolási metódusait, műveleteit. Ezen felül képes lettem titkosított adatokkal dolgozni, mely például a felhasználói fiókok implementációjánál különösen hasznos (sőt, elvárható).

A szakdolgozatom készítése során segítségemre volt a Webtechnológiák I tantárgy során elsajátított tudás, illetve a szakmai gyakorlatom során, JavaScript nyelven végzett munkatapasztalat is.

Annak ellenére, hogy számos területen mélyíthettem tudásomat a szakdolgozat készítése során, természetesen van lehetőségem fejlődni ezen a területen. Például a visszajelzések és hibaüzenetek rendszerét meg lehetett volna oldani úgy, hogy ne kelljen új weboldalt megjeleníteni. Továbbá, a teljes rendszer single-page megoldásként is elkészíthető lett volna, viszont ehhez érdekesebb lett volna egy alkalmasabb keretrendszert (például Angular-t) használni.

Összességében, sikerült egy - nem tökéletes, de - kielégítő webalkalmazást létrehozni.

Ez viszont nem sikerülhetett volna a témavezetőm, *Kunné Dr. Tamás Judit* segítségével nélkül. Utólag is köszönettel tartozok neki, hálás vagyok azért, hogy felelősségteljesen végezte a szakdolgozatom témavezetői feladatát.

Irodalomjegyzék

- [1] Angular.io. Introduction to angular concepts. <https://angular.io/guide/architecture>, 2022.
- [2] Agárdi Anita. Webtechnológiák i. <https://github.com/anitaagardi/web-technologies>, 2021.
- [3] Education-wiki. Mi az sql server? kulcskonceptiók az sql server előnyeivel. <https://hu.education-wiki.com/7023109-what-is-sql-server>, 2022.
- [4] Gremmédia. Mi az a vue.js? <https://ak-akademia.hu/mi-az-a-vue-js/>, 2022.
- [5] ITHub. Node.js: miért, mikor, és mikor ne? https://ithub.hu/blog/post/Nodejs_miert_mikor_es_mikor_ne, 2017.
- [6] W3Schools. W3schools. <https://www.w3schools.com/>, 2021.
- [7] W3Schools. What is vue.js? https://www.w3schools.com/whatis/whatis_vue.asp, 2021.
- [8] Wikipédia. Mysql. <https://hu.wikipedia.org/wiki/MySQL>, 2022.
- [9] Dudás Ákos. MongoDB alapok, műveletek, és a mongodb .net driver. <https://bmeviauac01.github.io/adatvezzerelt/jegyzet/mongodb/>, 2022.

CD Használati útmutató

Ennek a címe lehet például *A mellékelt CD tartalma* vagy *Adathordozó használati útmutató* is.

Ez jellemzően csak egy fél-egy oldalas leírás. Arra szolgál, hogy ha valaki kézhez kapja a szakdolgozathoz tartozó CD-t, akkor tudja, hogy mi hol van rajta. Jellemzően elég csak felsorolni, hogy milyen jegyzékek vannak, és azokban mi található. Az elkészített programok telepítéséhez, futtatásához tartozó instrukciók kerülhetnek ide.

A CD lemezre mindenképpen rá kell tenni

- a dolgozatot egy `dolgozat.pdf` fájl formájában,
- a LaTeX forráskódját a dolgozatnak,
- az elkészített programot, fontosabb futási eredményeket (például ha kép a kimenet),
- egy útmutatót a CD használatához (ami lehet ez a fejezet külön PDF-be vagy Markdown fájlként kimentve).