Szakdolgozat

Miskolci Egyetem

Webalkalmazás használata az egyetemi ügyintézésben

Készítette:

Kolozsvári Patrik

Programtervező informatikus

Témavezető:

Dr. Hriczó Krisztián

Miskolc, 2024

Miskolci Egyetem
Gépészmérnöki és Informatikai Kar
Alkalmazott Matematikai Intézeti Tanszék **Szám:**

Szakdolgozat Feladat

Kolozsvári Patrik (WYQ5JK) programtervező informatikus jelölt részére.

A szakdolgozat tárgyköre: full stack webfejlesztés

A szakdolgozat címe: Webalkalmazás használata az egyetemi ügyintézésben

A feladat részletezése:

Egy full stack webalkalmazás tervezése és megvalósítása. Frontend részen egy űrlap kitöltését és a szervernek való elküldését kell megvalósítani, backend részen pedig az űrlap adataival egy új PDF létrehozása és kezelése a cél.

Témavezető: Dr. Hriczó Krisztián (egyetemi docens)

A feladat kiadásának ideje: 2023. szeptember 20.

.....
szakfelelős

Eredetiségi Nyilatkozat

Alulírott **Kolozsvári Patrik**; Neptun-kód: `WYQ5JK` a Miskolci Egyetem Gépészmérnöki és Informatikai Karának végzős Programtervező informatikus szakos hallgatója ezennel büntetőjogi és fegyelmi felelősségem tudatában nyilatkozom és aláírásommal igazolom, hogy *Webalkalmazás használata az egyetemi ügyintézésben* című szakdolgozatom saját, önálló munkám; az abban hivatkozott szakirodalom felhasználása a forráskezelés szabályai szerint történt.

Tudomásul veszem, hogy szakdolgozat esetén plágiumnak számít:

- szószerinti 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. A szakdolgozat feladat módosítása

szükséges (módosítás külön lapon)

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. A szakdolgozat bírálatra

bocsátható

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. fejezet

Bevezetés

Szakdolgozatomban egy webalkalmazás tervezését és implementációját mutatom be, ami egy modern megoldást kínál egy hagyományos problémára az ügyintézések területén. Az online alkalmazás működési elve: a weboldal megnyitását követően egy űrlap kitöltőt látunk, amely egy féléves kredit összesítő nyomtatvány kitöltését teszi lehetővé [7]. A kitölendő mezők között van, amely legördülő mezőből választható és van amelynél saját adatbevitel szükséges, továbbá lehetőséget biztosít adott teljesítése igazolásainak feltöltésére. A kitöltés befejezését követően létrejön egy új .pdf fájl, ami tartalmazza azon adatokat, amelyekkel a felhasználó kitöltötte az űrlapot, valamint a feltöltött .pdf formátumú fájlokat, amelyek az elvégzett tevékenység igazolására szolgálnak [8].

Napjainkban az emberek többsége már digitálisan intézi az ügyeit, a hagyományos, papíralapú ügyintézés idejétmúlt, elavult, mivel jóval körülményesebb, több időt vesz igénybe, azaz hosszadalmasabb az elektronikus ügyintézéshez képest. Ezért is ért meglepetésként, amikor megtudtam, hogy a Sályi István Gépészeti Tudományok Doktori Iskolában a kredit összesítőket minden szemeszter végén papíron készítik el a hallgatók és nyújthaták azt be a szükséges jóváhagyó aláírások tán szkennelt formában.

Témavezetőmmel történő megbeszélés után arra a következtetésre jutottam, hogy a legkézenfekvőbb megoldás ezen feldat digitalizálására egy webalkalmazás létrehozása lenne.

Léteznek már ezen az elgondoláson alapuló, kisebb-nagyobb hasonlósággal rendelkező webalkalmazások. Véleményem szerint ezek közül a leginkább említésre méltó a Nemzeti Adó- és Vámhivatal által fejlesztett Adóbevallás kitöltő online felülete. Egyszerűsége és hatékonysága kiemelkedő, ráadásul az adatvédelem és biztonság is kellő figyelmet kapott [13].

Az alkalmazás hallgatók számára a Doktori Iskola weboldalán lesz elérhető és a felhasználók részéről nem igényel semmilyen extra ismeretet, mint például különböző programok/alkalmazások telepítését. Továbbá így megoldható, hogy csak korlátozott időben lehessen használni az alkalmazást, jelen esetben a kredit összesítők benyújtási idejében, valamint egyszerűen bővíthetőek és aktualizálhatóak a jelenlegi funkciók a

változó szabályokzások/elvárások mellett.

Az ügyintézők, illetve az ügyintézést végzők számára hatalmas segítséget jelent ez a webalkalmazás, mivel az adatok gyorsan és pontosan kerülnek feldolgozásra, ezzel minimalizálva az ügyintézés idejét, és az emberi hibákat. Arról nem is beszélve, hogy a jelenlegi megoldással rengeteg papírt és ezzel együtt nyomtatófestéket használnak a hallgatók a kredit összesítők beadása miatt. Ez elkerülhető lenne az általam írt program használatának bevezetésével.

Mindenképpen olyan programot szerettem volna elkészíteni szakdolgozatként, amit a jövőben tudnak és fognak is használni, mások hasznára válik. Továbbá olyan technológiák, nyelvek és eszközök segítségével elérni ezt, amikkel még nincs tapasztalatom, ezáltal is tudásomat bővítve. A programozás kivitelezéséhez szükségem volt további ismeretek megszerzésére, így előzetes tanulást és kutatást igényelt a szakdolgozat megírása. Így témavezetőm ajánlása után tökéletes választásnak tűnt ennek a webalkalmazásnak a létrehozása, ugyanis mindkét általam fontosnak tartott kritériumomnak eleget tesz.

A szakdolgozat következő fejezeteiben részletesen ismertetem, a fejlesztés menetét annak nehézségeivel együtt. Megadom a fejlesztés során adódó megoldandó feladatokat és az általam létrehozott megoldásokat. Továbbá röviden ismertetem a frissítések és bővítések lehetőségeit a jövőbeni szélesebb körű felhasználás támogatására.

2. fejezet

Koncepció

2.1. A fejezet célja

Ebben a fejezetben bemutatásra kerülnek az alkalmazás tervezésének alapvető elemei, kitérve a kredit összesítőre [7] és annak benyújtására [6], valamint a program technológiai megvalósításra.

2.2. Az iskola kreditrendszere

A Sályi István Gépészeti Tudományok Doktori Iskola képzési rendszere úgy lett kialakítva, hogy nagy súlyt fektessen a természettudományi és szakmai megalapozásra, széles témacsoportokat kínáljon, valamint biztosítsa a mélyebb kutatáshoz szükséges ismereteket. A doktori iskola számos tevékenységet jutalmaz kreditekkel, többek között a tudományos szemináriumokon való részvételt, kutatási projektekben való közreműködést, önálló kutatómunka eredményeinek publikálását, második nyelvvizsga letételét és a doktori értekezés elkészítéséhez szükséges publikációs követelmények teljesítését [8].

2.2.1. Kredit összesítő felépítése

A kredit összesítő egy olyan dokumentum, amit a doktoranduszoknak minden félév után be kell nyújtaniuk. A szemeszter során megszerzett kreditek tulajdonképpen ennek a lapnak a segítségével kerülnek könyvelésre. Az összesítőhöz mellékelni kell valamennyi kutatást, beszámolót, publikációt, így ezek összeadódva egy nagy kupac papírt tesznek ki.

2024 February

CREDIT SUMMARY

Doctoral School: István Sályi Doctoral School of Mechanical Engineering Sciences

Name: …… MTMT No.: …….
Topic field: ……..
Topic group: ……

Academic year: …… Year of Ph.D. Study: …….. Number of Semester: ……

Scope of activities	Obtained credits
Exam of obligatory and elective subjects (with Course code from NEPTUN)	
Exam of optional subjects (with Course code from NEPTUN)	
Research seminar	
Institutional/departmental research	
Educational activity (with Course code from NEPTUN)	
Research report (at the end of 2., 4., 5., 6., 7. semester with the signatures of the topic field/group leader and the supervisor(s))	
Complex report	
Second language exam[1]	
List of publications[2] and professional presentations[3]	
Total credits	

Miskolc, 2024. ……..

Topic field/group leader's signature: ……
Topic field/group leader's name: ……..

Supervisor's signature: ……..
Supervisor's (& co-supervisor's) name: ……

1. Credit can be obtained only once in the semester when the exam was completed.
2. Publications have to be uploaded in MTMT and approved.
3. Publications and presentations must be given in details (see example Publications.pdf)

2.1. ábra: Kredit összesítő lap [7]

2.2.2. Kredit összesítő benyújtása és szabályai

A formanyomtatványt megfelelően kitöltve, a témavezetők által aláírva a szükséges mellékletekkel kell benyújtani a témacsoport vezetőknek. A hallgatók különböző mintákat, sablonokat találnak az iskola honlapján, amiket például az oktatási tevékenység vagy intézeti kutatási tevékenység elvégzéséről szóló igazoláshoz vagyszükséges használni. A benyújtási határidőt érdemes észben tartani, sajnos gyakran csúsznak ki ebből a tanulók. [6]

2.3. PDF kezelő eszköz

2.3.1. Követelmény specifikáció

Az alkalmazás célja, hogy lehetővé tegye a hallgatók számára a kredit összesítő formanyomtatványának digitális kitöltését, ezzel leegyszerűsítve a dolgukat, és minimalizálva a felmerülő hibázási lehetőségeket. Az előbbi ábrán láthatóak szerint kell mezőcímkéket megjeleníteni, valamint feltöltési lehetőségeket elhelyezni a webalkalmazásban, egy hasonló, formanyomtatványhoz illő megjelenéssel.

Ha úgy nézzük, felbonthatjuk két részre az alkalmazást:

- Űrlapkitöltő
- PDF összefűző

2.3.2. PDF összefűzés

Az alkalmazás koncepciójának bemutatása mellett konkrét példán keresztül is megközelíteném a hasonló alkalmazások működését és funkcionalitását.

Űrlapkitöltéssel számtalan helyen találkozhatunk az interneten, például regisztrációnál, kapcsolatfelvételnél vagy akár online vásárlásnál. Az alkalmazás másik részéhez, a PDF összefűzőhöz már ritkábban nyúlnak az átlagos felhasználók, ugyanakkor erre is találunk eszközöket a világhálón.

Személy szerint ezelőtt már többször is használtam az *iLovePDF* csapatának a PDF kezelő eszközeit, ezen belül is azt, amelyik összefűz több PDF-et (PDF Merger) egyetlen PDF fájllá [10]. Ez egy végtelenül egyszerű alkalmazás, lehetőséget ad .pdf fájlok feltöltésére, majd ezekből – a sorrend beállítása után – készít egy új .pdf fájlt, amit aztán le lehet tölteni. Általánosságban ezen az elven működnek a PDF összefűzők [12].

Ha konkrétan ezt a példát nézzük, akkor a következőek szólnak a használata ellen:

- Adatvédelem – nem mindegy, hogy milyen oldalra bízzuk az adatainkat, fájljainkat.

- Költségek – csupán a *Free* csomag ingyenes, a *Premium* és *Business* előfizetési tervek választása esetén már havi vagy éves díjat kell fizetni.
- Praktikusság – az én megoldásommal egy helyen el lehet készíteni a teljes kredit összesítőt, míg ellenkező esetben ez több bonyodalommal jár.

2.4. Webalkalmazás

2.4.1. Webalkalmazás Felépítése és Technológiák

A webalkalmazás (web application) egy olyan szoftveralkalmazás, amelyet a felhasználók egy webböngészőn keresztül érnek el és használnak. A webalkalmazások különböznek a hagyományos, asztali alkalmazásoktól abban, hogy nem igényelnek külön telepítést a felhasználó számítógépén. Ehelyett a webalkalmazásokat a felhasználók böngészője futtatja, ami lehetővé teszi számukra, hogy szinte bármilyen eszközről hozzáférjenek, például számítógépek és okostelefonok használatával is.

A webalkalmazások általában szerveroldali *(backend)* és kliensoldali *(frontend)* komponensekből állnak. A frontend és backend közötti kommunikáció általában HTTP protokollon keresztül történik, ahol a frontend kliens oldali kéréseket (HTTP kéréseket) küld a backend szerver felé, amely feldolgozza ezeket a kéréseket, és válaszol a megfelelő adatokkal vagy eredményekkel. A frontend és backend együttesen alkotja a teljes alkalmazást, amelyet a felhasználók böngészőjükben futtatnak [1][14].

Frontend

A kliensoldalon a böngészőben futó alkalmazás a felhasználói felületet biztosítja, vagyis a felhasználó ezzel találkozik, ezt látja és használja közvetlenül. Ez magában foglalja azokat az elemeket és funkciókat, amelyeket a felhasználó böngészőjében lát, például az elrendezést, a gombokat, a menüket, az űrlapokat stb. A frontend tervezése és fejlesztése a felhasználói élményre (*UX*) és az interfész (*UI*) megtervezésére összpontosít. A frontend fejlesztése HTML (*HyperText Markup Language*), CSS (*Cascading Style Sheets*) és JavaScript nyelvek segítségével történik általában.

- HTML: A weboldal strukturálásához és tartalmának meghatározásához.
- CSS: A weboldal stílusának, elrendezésének és megjelenésének meghatározásához.
- JavaScript: Interaktív funkciók hozzáadásához, például űrlap validálás, animációk, dinamikus tartalmak.

Backend

A szerveroldalon a szerver a kérelmeket dolgozza fel, adatbázisokhoz fér hozzá és visszaküldi a válaszokat. A backend kezeli az adatbázis lekérdezéseket, és válaszol az ügyfél (frontend) kéréseire. A backend fejlesztése általában olyan programozási nyelvekkel történik, mint például *JavaScript (Node.js), Ruby, PHP, Java, Python,* stb. Emellett a backend fejlesztéséhez keretrendszerek (*frameworks*) is használatosak, amelyek megkönnyítik az alkalmazás logikájának és adatkezelésének megvalósítását.

2.5. Felhasznált Technológiák

Git

Egy nyílt forráskódú, elosztott verziókezelő szoftver, szoftverforráskód-kezelő rendszer, amely a sebességre helyezi a hangsúlyt. A Gitet eredetileg Linus Torvalds fejlesztette ki a Linux kernel fejlesztéséhez. Minden Git munkamásolat egy teljes értékű repository teljes verziótörténettel és teljes revízió követési lehetőséggel, amely nem függ a hálózat elérésétől vagy központi szervertől [3].

GitHub

A GitHub egy webes alapú Git verziókezelő rendszer, amely lehetővé teszi a felhasználók számára, hogy tárolják, kezeljék és megosszák a számítógépes fájlaikat és projekteiket. A GitHubon keresztül a felhasználók nyomon követhetik a változtatásokat, kommentálhatnak és visszaállíthatják korábbi változataikat. Emellett lehetőség van mások projekteinek a forkolására, azaz saját verzió készítésére belőlük, majd a módosított változat visszatöltésére a közösség számára. A GitHubot az open-source közösség mellett egyre több vállalat és szervezet használja, hogy nyilvános és zárt forráskódú projektek kezelését és együttműködését könnyítsék meg [11].

Visual Studio Code

Egy ingyenes, nyílt forráskódú kódszerkesztő, melyet a Microsoft fejleszt Windows, Linux és macOS operációs rendszerekhez. Támogatja a hibakeresőket, valamint beépített Git támogatással rendelkezik, továbbá képes az intelligens kódkiegészítésre az IntelliSense segítségével. A VSCode-ban a felhasználók megváltoztathatják a kinézetet (témát), megváltoztathatják a szerkesztő gyorsbillentyű-kiosztását, az alapértelmezett beállításokat és még sok egyebet. Támogatja a kiegészítőket, melyek segítségével további funkciók, testreszabási lehetőségek érhetőek el [4].

JavaScript

A JavaScript egy dinamikus, magas szintű programozási nyelv, melyet leggyakrabban webes alkalmazásokhoz használnak. Segítségével interaktív felhasználói felületeket készíthetünk, kezelhetünk adatokat és kommunikálhatunk a böngészővel. JavaScript alkalmazható frontend és backend fejlesztéshez egyaránt alkalmas, és népszerűségét rugalmasságának és könnyen tanulhatóságának köszönheti. Ebből fejlődött ki a TypeScript, ami a JavaScript típusos változatának tekinthető [5].

CSS

Stílusleíró nyelv, mely a HTML vagy XHTML típusú strukturált dokumentumok megjelenését írja le. A CSS-t a weblapok szerkesztői és olvasói egyaránt használhatják, hogy átállítsák vele a lapok színét, betűtípusait, elrendezését, és más megjelenéshez kapcsolódó elemeit. A tervezése során a legfontosabb szempont az volt, hogy elkülönítsék a dokumentumok struktúráját (melyet HTML vagy egy hasonló leíró nyelvben lehet megadni) a dokumentum megjelenésétől (melyet CSS-sel lehet megadni). Az ilyen elkülönítésnek több haszna is van, egyrészt növeli a weblapok használhatóságát, rugalmasságát és a megjelenés kezelhetőségét, másrészt csökkenti a dokumentum tartalmi struktúrájának komplexitását. A CSS ugyancsak alkalmas arra, hogy a dokumentum stílusát a megjelenítési módszer függvényében adja meg [16].

React

A React egy deklaratív, effektív, és rugalmas JavaScript könyvtár, felhasználói felületek készítéséhez. Lehetővé teszi komplex felhasználói felületek összeállítását izolált kódrészletekből, amiket "komponenseknek" hívunk. A komponensek segítségével mondjuk meg a Reactnek, hogy mit szeretnénk látni a képernyőn. Ha az adatunk megváltozik, a React hatékonyan frissíti és újrarendereli a komponensünket. A legtöbb React fejlesztő egy speciális szintaxist használ, ezt "JSX"-nek hívják, ami könnyebbé teszi ezen struktúrák írását. A JSX rendelkezik a JavaScript minden erejével. A JSX-be bármilyen JavaScript kifejezést tehetsz, kapcsos zárójelek közé. Minden React elem egy JavaScript objektum, amit változókban lehet tárolni, vagy körbeküldeni a programban [17].

Node.js

A Node.js egy szoftverrendszer, melyet skálázható internetes alkalmazások, mégpedig webszerverek készítésére hoztak létre. A programok JavaScript-ben írhatók, eseményalapú, aszinkron Input/Outputtal a túlterhelés minimalizálására és a skálázhatóság maximalizálására. Remekül használható online alkalmazások és dinamikus weboldalak fejlesztésére. Azonnali grafikai változtatásokhoz is kiváló terepet nyújt [15].

3. fejezet

Tervezés

A tervezési fázis egy nagyon fontos része a fejlesztésnek, felelőtlenség lenne előzetes átgondolás nélkül nekikezdeni a programozásnak. Még a kódolás megkezdése előtt célszerű fejben összerakni az alkalmazást, annak logikáját, esetleg mindezt "papírra vetni", vagy legalább digitális formában, folyamatábrán megtervezni.

A frontend és backend tervezés közötti hatékony együttműködés létfontosságú annak érdekében, hogy az alkalmazás megfelelően működjön és jó felhasználói élményt nyújtson. A frontend tervezése meghatározza, hogy hogyan jelennek meg és működnek az alkalmazás funkciói a felhasználó számára. Míg a backend tervezése meghatározza, hogy az adatok hogyan fognak kezelésre, tárolásra kerülni, valamint hogyan lesznek elérhetők a frontend részére.

3.1. Frontend

Frontend terén viszonylag egyszerű feladatom volt, hiszen a kredit összesítő minta adott, az azon szereplő elemeket szükséges megjelenítenem.

3.1.1. Megoldandó feladatok

- Űrlap megjelenítése
- Űrlap kitöltésének ellenőrzése
- Űrlap elküldése a szervernek

Űrlap megjelenítése

A űrlap legfontosabb része természetesen a mezők és feltöltési lehetőségek megléte. Emellett célszerű egy átlátható, egyszerű design kitalálása, megvalósítása. Mivel elegendő, hogy a webalkalmazás egyoldalas legyen, ezért nincs szükség például navigációs sáv vagy akármilyen más egyéb – például *Vissza* – gomb elhelyezésére az oldalon.

A már meglévő kredit összesítő dokumentumnak köszönhetően még az implementálás előtt tudom, hogy pontosan milyen mezőket kell majd létrehoznom.

A mezőkön kívül csak egy *Űrlap elküldése*, más szóval *Submit* gombot kell biztosítani a felhasználóknak.

Háromféle mező típust alkalmaztam a fejlesztés során:

1. **Egyszerű mező** – szöveg vagy szám bekérésére alkalmas, a felhasználó szabadon adja meg az értékét. Lehetséges annak a korlátozása, hogy milyen formában írhatja meg a felhasználó a szöveget vagy számot az űrlapon.

2. **Mező több értékkel** – hasonló az egyszerű mezőhöz, annyi különbséggel, hogy itt egy mezőhöz több érték fog tartozni, ezek egy tömbben lesznek tárolva.

3. **Fájl kiválasztó mező** – ez a mező típus lehetővé teszi a felhasználó számára, hogy egy vagy több fájlt kiválasszon a számítógépéről a feltöltéshez.

Egyszerű mezők:

- Name – A doktorandusz neve
- MTMT Number – MTMT azonosító szám
- Topic Field – Tématerület
- Topic Group – Témacsoport
- Academic Year – Érintett szemeszter
- Year of PhD Study – Képzésben eltöltött évek száma
- Number of Semester – Képzésben eltöltött félévek száma
- Educational Activity – Oktatásban való részvétel
- Second Language Exam – Második nyelvvizsga megszerzése

Mezők több értékkel:

- Exam of Obligatory and Elective Subjects – Kötelező tárgyak teljesítése
- Exam of Optional Subjects – Szabadon választható tárgyak teljesítése

Fájl kiválasztó mezők:

- Research Seminar – Kutató szemináriumon való szereplés
- Institutional/Departmental Research – Tanszék/Intézet kutatási projektjeiben való közreműködés

- Research Report – Kutatási beszámoló
- Complex Report – Komplex vizsgához tartozó adatlap
- List of Publications and Professional Presentations – Kutató munka eredményeinek publikálása

Űrlap kitöltésének ellenőrzése

Meg kell határoznunk, hogy melyek azok a mezők, amelyeknek kitöltése kötelező, és melyek azok, amelyek csak opcionálisak. Első körben szűrnünk kell, hogy a kötelező mezők valóban ki vannak-e töltve. Azután megnézhetjük, hogy a különböző formai követelményeknek is megfelelnek-e a mezőkbe írt adatok, értékek. A fájl kiválasztó mezőknél kikötést tehetünk, hogy ténylegesen csak .pdf kiterjesztésű fájlokat fogadjon el az űrlap.

Űrlap elküldése a szervernek

Ha ellenőrzés után sincsenek hibák, akkor továbbítani kell az űrlap tartalmát a szervernek. Ezt ugyebár a *Submit* gombbal tehetjük meg.

Ha a szerver visszaküldte a kliensnek a PDF-et, akkor lehetővé kell tenni ennek a letöltését.

3.1.2. React

Azért esett a választásom a *Reactra*, mert más népszerű keretrendszerekkel szemben a React sokkal tisztább, jobban olvasható, átfogóbb kódot tesz lehetővé. Nem véletlenül ez jelen pillanatban a legnépszerűbb könyvtár, keretrendszer, amit frontend fejlesztéshez használnak.

3.2. Backend

A szerveroldalon fog lezajlani a PDF létrehozása, ehhez biztosítanunk kell az űrlap adatainak feldolgozását, tárolását. Ezen kívül még megoldást kell találnunk a PDF-ek kezelésére.

3.2.1. Megoldandó feladatok

- Űrlap fogadása után az adatok kezelése
- Fájlkezelés
- PDF generálás

- PDF-ek egyesítése
- PDF visszaküldése a kliensnek
- Hibakezelés

Űrlap fogadása után az adatok kezelése

A szerveren fogadjuk az űrlapból érkező adatokat, fájlokat, amiket a felhasználó a böngészőben ki- és feltöltött, aztán elküldött. Ezt követően validáljuk, és helyes formátumban kezeljük, vagyis ellenőrizzük a kapott adatokat, például megnézzük, hogy kaptunk-e üres értékkel ellátott mezőt, vagy valamilyen .pdf-től eltérő kiterjesztésű fájlt. Ha az adatok sikeresen átmentek a validáláson, akkor ezeket a szerveroldalon további feldolgozásnak vetjük alá, ilyen például a szükséges formátumúra, típusúra való konvertálás.

Fájlkezelés

Fájlkezelés során fontos ügyelnem a szerver leterhelésének elkerülésére. Ezért a feltöltött fájlokat átmenetileg tároljuk a szerveren, csak addig, amíg el nem készül belőlük a PDF dokumentum. Ezt követően a feltöltött fájlok automatikusan törlésre kerülnek, így minimalizálva a felesleges tárolást és a terhelést a szerveren. Emellett célszerű beépíteni olyan funkciókat, amelyek limitálják a fájlok méretét, így tovább csökkentve a terhelést a szerveren és biztosítva a hatékony fájlkezelést.

PDF generálás

PDF generálás során létrehozunk egy új PDF dokumentumot, amelyet aztán elküldünk a kliens felé. A PDF dokumentum első oldalát a beküldött űrlap szöveges tartalmával töltjük fel, és további elemeket adunk hozzá, amelyek javíthatják a megjelenést és növelhetik a dokumentum esztétikai értékét.

PDF-ek egyesítése

A PDF-ek egyesítése során össze kell fűznünk az űrlapban elküldött fájlokat vagy azok tartalmát az előzőleg generált PDF dokumentummal. Ez lehetővé teszi, hogy az űrlap kitöltésének eredményeit és a kapcsolódó fájlokat egyetlen összefüggő PDF-dokumentumban jelenítsük meg a felhasználónak. Fontos figyelembe venni az egyesített PDF fájlok sorrendjét és elrendezését, hogy a dokumentum logikus és áttekinthető legyen.

PDF visszaküldése a kliensnek

Miután összeállítottuk és létrehoztuk a kész PDF dokumentumot az űrlap tartalmával és az elküldött fájlok összefűzésével, visszaküldjük ezt a PDF-et a kliens felé. A kliens számára biztosítunk lehetőséget a PDF letöltésére, így a felhasználó könnyedén hozzáférhet és megtekintheti az elkészült dokumentumot.

Hibakezelés

Felmerülhető hibák között szerepel például az, ha nem tudjuk legenerálni a PDF-et, vagy a fájlkezelésnél csúszik valami félre. Ha valamilyen okból kifolyólag mégis rossz adatot vagy fájlot kapnánk a klienstől, akkor ezeket is megfelelően kell tudni kezelni.

3.2.2. Node.js

A megoldandó feladatok feltérképezése után, ki kell választani egy szerveroldali keretrendszert, amely segítségével ezeket meg lehet oldani. Ahogy a frontendnél, itt is elég sok lehetőség közül tudunk választani.

Hasonlóan a frontendhez, backendnél is a legnépszerűbb keretrendszerre, a *Node.js*-re esett a választásom, mivel nagyon jó a sok konkurrens kérés kezelésében, nagy terhelés alatt is válaszképes marad, valamint hatékony a fájlkezelése. Képes kezelni HTTP kéréseket, hozzáférni adatbázisokhoz, lehetővé teszi az valós idejű kommunikációt, és más szerveroldali feladatokat is elvégezhet.

4. fejezet

Megvalósítás

A Tervezés fejezetben leírtak alapján egy kész, használható webalkalmazás létrehozása a cél. Mielőtt elkezdenénk az implementálást, le kell tölteni és telepíteni kell a Node.js-t a hivatalos weboldalról.

4.1. Frontend

Elsőnek a frontenddel foglalkoztam, Visual Studio Code-ban elkészítettem a React App-ot az alábbi paranccsal:

```
npx create-react-app form-app
```

A parancs a Create React App nevű eszköz használatát jelenti, amelyet a React alkalmazások gyors létrehozásához és konfigurálásához fejlesztettek ki. A parancs lényege, hogy egy új React alkalmazást hozzon létre az általad megadott form-app nevű könyvtárban.

Többek között létrejött a *App.js* fájl, frontend részen lényegében csak erre lesz szükségem. Ezt követően elkezdtem összerakni az űrlapot.

React Hook Form

Az űrlap elkészítéséhez a *React Hook Form* könyvtárat használtam, ezzel lehetővé téve a mezők hatékony kezelését és folyamatos frissítését [2] [9].

Főbb funkciói:

- handleSubmit
- register
- formState: errors

A *handleSubmit* az űrlap beküldéséért felelős, a *register*, mint ahogy a neve is sugallja, a mezők regisztrálását intézi, végül pedig a *formState* a hibákat kezeli.

4.1.1. Egyszerű mezők

Az egyszerű mezőket létrehoztam: Name, MTMT Number, Topic Field, Topic Group, Academic Year, Year of PhD Study, Number of Semester, Educational Activity, Second Language Exam. Továbbá ide sorolnám azokat a mezőket, amik a megszerzett kreditekért felelnek. Kezdésnek mindet sima *input* mezőként állítottam be.

Később a következő módosításokat végeztem el ezeken:

Topic Field

Csak 3 képzési program van és ezek adottak, ezért *select* mezőre cseréltem a következő opciókkal:

- Basic engineering sciences
- Design of machines and structures
- Engineering material science, production systems and processes

Topic Group

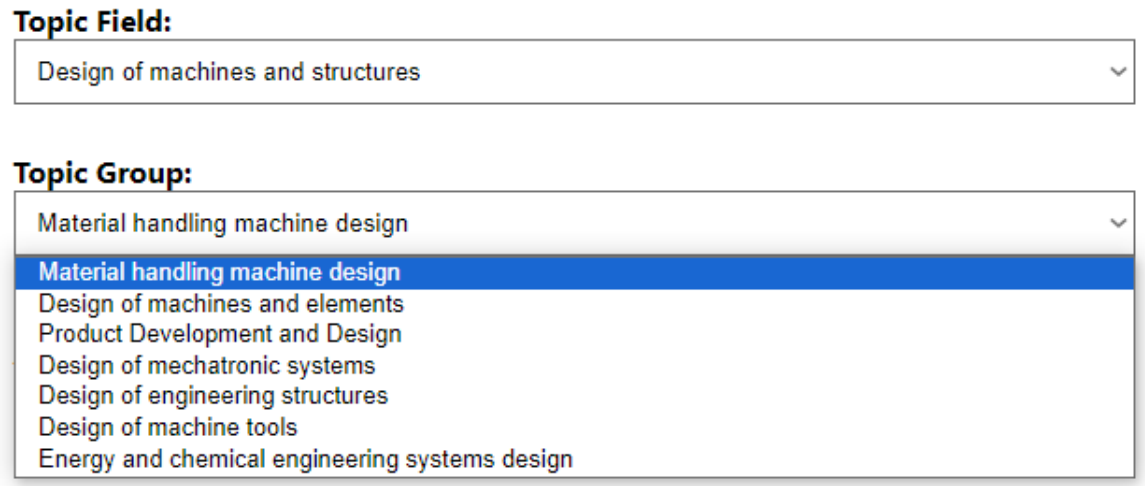
Az egyes képzési programokon belül további témacsoportok találhatóak. Ezt az alábbi módon kell elképzelni:

Education and Research Programmes

Basic engineering sciences (Prof. I. Páczelt)	***Mechanics of solids*** (Prof. Gy. Szeidl)
	Transport processes and machines (Prof. L. Baranyi)
Design of machines and structures (Prof. G. Bognár)	***Material handling machine design*** (Prof. B. Illés)
	Design of machines and elements (Prof. G. Bognár)
	Product Development and Design (Prof. G. Bognár)
	Design of mechatronic systems (Dr. T. Szabó)
	Design of engineering structures (Prof. K. Jármai)
	Design of machine tools (Dr. Gy. Hegedűs)
	Energy and chemical engineering systems design (Prof. Z. Siménfalvi)
Engineering material science, production systems and processes *(Prof. J. Kundrák)*	***Materials engineering and mechanical technology*** (Prof. J. Lukács)
	Manufacturing systems and processes (Dr. Zs. Maros)
	Assembly systems (Prof. Gy. Kovács)
	Structural integrity (Prof. J. Lukács)

4.1. ábra: Az iskola tématerületei

Így célszerű ezt is *select* mezőre váltani, és a Topic Fieldben kiválasztott érték függvényében megadni az opciókat. Azaz, ha például a Topic Fieldben a *Basic engineering sciences* van kiválasztva, akkor a Topic Group értéke csak *Mechanics of solids* vagy *Transport processes and machines* lehet. Ehhez hasonlóan a másik két képzési program szerint is elvégeztem a logikai szűrést és azok szerinti opciókkal láttam el ezt a mezőt.



4.2. ábra: Példa

Academic Year

Ennél a mezőnél jelöli meg a hallgató, hogy melyik félévre vonatkozóan készíti el a kredit összesítőjét. Ezt is *select*-re állítottam a lehetséges értékkel: *2023/2024 II.* és *2024/2025 I.*, ezzel viszont annyi a baj, hogy manuális átírást igényel minden félévben.

Year of PhD

Itt a képzésben eltöltött évek számát kell bejelölnie a hallgatónak, ami maximum 4 lehet, így itt is *select* mezőre váltottam, 1-2-3-4 opciókkal.

Number of Semester

Ennél a mezőnél a képzésben eltöltött félévek számát kell megjelölni. Olyan szűrést végezhetünk, hogy csak az adott évhez tartozó szemeszterek jelenjenek meg opcióként. Például ha a Year of PhD mezőben a 2-es érték van kiválasztva, akkor csak a 3-as és 4-es opciók legyenek érvényben.

Ezt az alábbi módon sikerült kivitelezni:

```
<select {...register('Number of Semester')}>
    {[...Array(2).keys()].map((semester) => (
        <option key={(phdStudyYear - 1) * 2 + semester + 1}
        value={(phdStudyYear - 1) * 2 + semester + 1}>
            {(phdStudyYear - 1) * 2 + semester + 1}
        </option>
    ))}
</select>
```

Second Language Exam

A Doktori Iskola Működési Szabályzatában rögzített, második nyelvvizsga sikeres letételéért az alábbi kreditpontok adhatók:

- az IOK (Idegennyelvi Oktatási Központ) által lefolytatott nyelvvizsga: 15 kredit
- középfokú állami nyelvvizsga: 20 kredit
- felsőfokú állami nyelvvizsga 25 kredit

Így ez a mező is *select* lett, az előbb felsorolt 3, valamint egy *None* opcióval. Ehhez a mezőhöz tartozik egy *Obtained Credits* mező, ami az előbbi opciók alapján kapja az értékét – és automatikusan frissül – azaz lehet 0, 15, 20 és 25.

Utóbbira egy viszonylag furcsa megoldással álltam elő. Adtam egy select mezőnek egy "disabled" osztályt, az *App.cs* fájlban pedig az alábbi két tulajdonságot állítottam be:

```
.disabled {
    pointer-events: none;
    cursor: not-allowed;
}
```

Így a felhasználók nem tudják módosítani az értéket, azonban ugyanúgy mezőként érzékeli az űrlap, ezáltal a többi adattal együtt kapja majd meg a szerver.

4.1.2. Mezők több értékkel

Ezeknél a mezőknél az az elgondolás, hogy a felhasználó tudja szabályozni a mezők mennyiségét, plusz mezőket tud hozzáadni az űrlaphoz, vagy akár törölni is tudja ezeket.

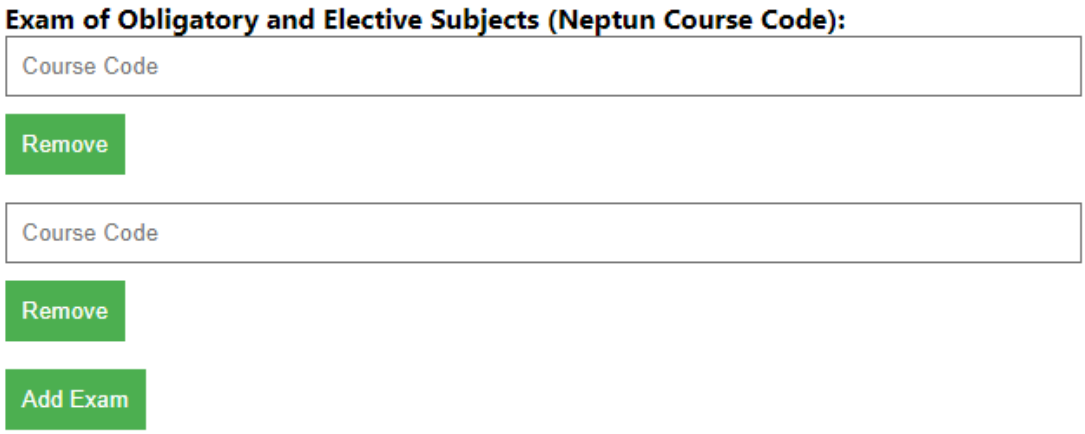
Két mező kerül így felépítésre, az egyik az *Exam of Obligatory and Elective Subjects*, a másik pedig az *Exam of Optional Subjects*.

A mezők mennyiségének szabályozását két gombbal értem el:

- Add Exam
- Remove

Létrehoztam egy új függvényt, ami kezeli az Add gomb megnyomását, ilyenkor új mező jelenik meg:

```
  const addObligatoryExams = () => {
  setObligatoryExams([...obligatoryExams,
  { courseCode: '' }]);
};
```



4.3. ábra: Add és Remove gombok

Az előzőhöz hasonlóan a Remove gombhoz is kellett készítenem egy függvényt, ennek segítségével az űrlapból eltűnik az adott mező:

```
const removeObligatoryExams = (index) => {
  const updatedExams = [...obligatoryExams];
  updatedExams.splice(index, 1);
  setObligatoryExams(updatedExams);
};
```

A formban pedig így kerülnek meghívásra:

```
  <button type="button" onClick={() =>
  removeObligatoryExams(index)}>
      Remove
  </button>
```

```
  <button type="button" onClick=
  {addObligatoryExams}>
      Add Exam
  </button>
```

Több érték kezelése

Több érték kezelése az alábbi módon történik:

```
  {obligatoryExams.map((exam, index) => (
          <div key={index} className="input-group">
            <input
              {...register(`obligatoryExams[${index}]
```

```
                .obligatoryCourseCode`)}
                placeholder="Course Code"
                value={exam.courseCode}
                onChange={(e) => handleExamCodeChange(index,
                e.target.value)}
              />
```

A handleExamCodeChange függvényt azért hoztam létre, hogy kezelje azt, amikor a felhasználó módosítja a kötelező vagy opcionális tárgyakhoz tartozó kurzuskódokat az űrlapon:

```
const handleExamCodeChange = (index, value) => {
    const updatedExams = [...obligatoryExams];
    updatedExams[index].courseCode = value;
    setObligatoryExams(updatedExams);
};
```

A függvény tehát dinamikusan frissíti az űrlapon megjelenő kurzuskódokat a megfelelő tárgyhoz, amikor a felhasználó az input mezőkben módosítást végez. Az *index* paraméter segítségével azonosítja, hogy melyik kurzusnak a kurzuskódját kell módosítani, és a *value* pedig az új beírt értéket jelenti. Így lényegében egy tömbbe kerülnek elmentésre a mezők értékei.

4.1.3. Fájl kiválasztó mezők

Az alkalmazás egyik fő részét a PDF fájlok kezelése teszi. Ehhez biztosítani kell ezeknek a feltöltését.

Így a következő *"file"* típusú mezőket hoztam létre: *Research Seminar, Institutional/Departmental Research, Research Report, Complex Report, List of Publications and Professional Presentations.*

Ezt követően elhelyeztem egy bepipálható négyzetet, amellyel azt jelöljük, hogy szeret-
nénk-e feltölteni fájlt. Ennek azért láttam szükségét, mert nem kötelező minden fájlt feltölteni, és így alapesetben nem követeli meg az űrlap, hogy olyan fájlt töltsön fel a felhasználó, amit nem szeretne. Így kompaktabb, átláthatóbb az űrlap.

A fájl kiválasztó mező és a hozzá tartozó *Obtained Credits* mező csak akkor jelenik meg, ha a négyzet be van pipálva, akkor viszont kötelező a feltöltés és a kreditszám megadása.

4.1.4. onSubmit

Az onSubmit függvényben juttatjuk el az űrlap tartalmát a szervernek. Mindenek előtt kiírjuk ezeket az adatokat a konzolra is.

FormData

A *FormData* egy beépített JavaScript objektum, amely lehetővé teszi, hogy könnyen kezeljük és küldjük az űrlap adatokat HTTP kérésekben, például *fetch* használatával. A FormData objektum lehetővé teszi, hogy strukturált adatokat küldjünk az űrlapból, amely tartalmazhat különböző típusú adatokat, például szöveges mezőket vagy akár fájlokat is.

Én ennek az objektumnak a segítségével kezeltem az űrlap tartalmát. Létrehoztam egy üres FormData típusú változót, aztán egy for ciklus segítségével feltöltöttem azt. Három részre kellett bontanom a feltöltést:

- Fájl
- Több értékű mező
- Egyszerű mező

Különböző *if* szerkezetekkel néztem meg, hogy melyik csoportba fog tartozni az adott mező, majd eszerint feltöltöttem a FormData objektumot a csoportjának megfelelő módon:

```
//fajl
for (let i = 0; i < data[key].length; i++) {
    formData.append(key, data[key][i]);
}

//tobb erteku
data[key].forEach((item, index) => {
    Object.entries(item).forEach(([subKey, subValue]) => {
        formData.append(`${key}[${index}].${subKey}`, subValue);
    });
});

//egyszeru
if (data[key] !== undefined && data[key] !== '') {
    formData.append(key, data[key]);
}
```

4.1.5. Axios

Az Axios egy JavaScript könyvtár, amely lehetővé teszi HTTP kérések kényelmes kezelését böngészőkben és Node.js környezetben is. Az Axios széles körben használható az adatkérések kezelésére azáltal, hogy egyszerű és intuitív módon lehetőséget biztosít aszinkron HTTP kérések küldésére és válaszok fogadására.

Az Axios könyvtár segítségével egy *POST* kérést küldünk a szervernek, aminek a tartalma a formData lesz – a kérés tartalmának típusát meg kell adni, ami nem mást, mint *multipart/form-data*. A választ *arraybuffer* típusként várjuk vissza, ez a típus lehetővé teszi, hogy a válasz egy bufferként vagy byte tömbként legyen kezelve.

Fontos megjegyezni, hogy ez a kérés aszinkron módon fut, és az *await* kifejezéssel várjuk meg a választ a szervertől, mielőtt a kód tovább folytatódna.

4.1.6. PDF letöltése

A szerver válaszát feldolgozzuk, a válaszadatokból létrehozunk egy *Blob* objektumot.

A Blob egy bináris adatobjektum, amelyet lehet kezelni, például fájlként lementeni vagy megjeleníteni. Aztán létrehozunk egy URL-t a Blob objektumhoz, amelyet a böngésző meg tud jeleníteni.

Ahhoz, hogy le tudjuk tölteni a PDF-et kicsit játszani kell a HTML dokumentummal. Először is hozzáadunk egy "<a>" elemet, majd ennek az elemnek beállítjuk a "href" attribútumát az előbb készített URL-re.

Beállítjuk a letöltendő fájl nevét, itt csak egy ideieglenes "combined_pdf.pdf"-et adtam meg, a felhasználóknak saját maguknak kell a kért módon átírni a letöltést követően. Az előbb létrehozott "<a>" elemet a dokumentum törzsébe, hogy látható legyen.

Ezt követően a *link.click();* segítségével megnyitjuk vagy letöltjük a .pdf fájlt – a böngésző beállításaitól függően. Végül töröljük az "<a>" elemet és felszabadítjuk az URL-t.

4.1.7. App.cs

Az App.css fájl tartalmazza a stílusdefiníciókat a frontend alkalmazáshoz, amit az App.js program használ. Az alábbiakban röviden összefoglalom a fájlban található osztályok és tulajdonságaik jelentését:

- **.app-container:** A teljes alkalmazás szélességét és elhelyezkedését határozza meg. A *max-width: 600px;* beállítás korlátozza a konténer maximális szélességét 600 pixelre, a tartalom középre fog igazodni (*margin: auto;*). A *padding: 20px;* beállítás pedig belső tér hozzáadása a containerhez.

- **.form:** A *display: flex;* segítségével az űrlap elemei egy oszlopba rendeződik *(flex-direction: column;)*.
- **.form-group:** Egy űrlapelem csoport stílusa. A *margin-bottom: 15px;* beállítás biztosítja a függőleges térközt az egyes csoportok között. A *font-weight: bold;* a szöveg vastagságát állítja be félkövérre.
- **.label:** Az űrlapelemekhez tartozó címkék stílusa. A *margin-bottom: 5px;* beállítás biztosítja a kis térközt a címke és az alatta lévő beviteli mező között.
- **input, select:** Az űrlapelemek (input mezők és legördülő listák) stílusa. A *width: 100%;* beállítás teljes szélességűvé teszi az elemeket. A *padding: 8px;* a belső tér beállítása az input mezőkben. A *margin-bottom: 10px;* pedig a függőleges térköz az elemek között.
- **button:** A gomb stílusa az űrlapban. A *padding* beállítás biztosítja a gomb belső térét. A *background-color* a háttérszín beállítása. A *color* a szövegszín beállításáért felel. A *border*-rel eltávolítom a keretet a gomb körül. A *cursor: pointer;* az egérmutatót mutatja gomb fölé helyezéskor.
- **button:hover:** A gomb stílusa, amikor az egérmutató a gomb fölé kerül. Itt a háttérszín sötétebb zöldre vált.
- **.error-message:** Hibaüzenet stílusa, ilyenkor a szöveg piros színnel jelenik meg.
- **.disabled:** Erre az osztályra már kitértem korábban, a Second Language Examnél a megszerzett kreditek mezője kapta egyedül ezt az osztályt. A gombok nem reagálnak kattintásra, és az egérmutató sem változik, amikor föléjük mozgatjuk.

A .css fájl nagyon alap, kezdetleges stílusdefiníciókat tartalmaz. Sokat lehetne még rajta fejleszteni és finomítani a jövőben, valószínűleg erre sort is fogok keríteni.

4.2. Backend

A szerver fog felelni a PDF létrehozásáért, ehhez biztosítanunk kell az űrlap adatainak megfelelő feldolgozását.

4.2.1. Technológiák

Az alkalmazás Backend részét Node.js környezetben írtam, Express.js keretrendszer segítségével. Létrehoztam egy .js fájlt *index.js* névvel, és importáltam a szükséges könyvtárakat: *express, cors, multer, fs, path, pdf-lib*.

Aztán inicializáltam egy Express alkalmazást, és beállítottam *cors* és *express.json() middleware*-ekkel.

Middleware

Az Express.js keretrendszerben a middleware egy olyan funkció, amely meghatározza, hogy az alkalmazás hogyan reagáljon a beérkező HTTP kérésekre. Ezek a middleware függvények a req (kérés) és res (válasz) objektumokat manipulálják, majd átadják őket a következő middleware-nek, vagy műveletet hajtanak végre (pl. válaszgenerálás, adatfeldolgozás stb.).

CORS

Az Express.js alkalmazások gyakran szükségessé teszik a CORS engedélyezését, mivel alapértelmezés szerint a modern webböngészők korlátozzák a cross-origin kéréseket biztonsági okokból. Ha a frontend alkalmazásunk más domainről (pl. localhost:3000) érkező kérést küld a backend Express szerverünkre (pl. localhost:3001), akkor a böngésző alapértelmezés szerint blokkolja ezt a kérést. A CORS middleware beállítások segítségével az Express alkalmazásunk engedélyezheti az ilyen kéréseket, illetve pontosan megadhatjuk, hogy mely domainről érkező kérések fogadhatók el, és milyen HTTP metódusokat, fejléceket engedélyezünk.

express.json()

Az express.json() egy beépített middleware az Express.js keretrendszerben, amelyet arra használnak, hogy a *req.body* JSON formátumban parse-olja. A middleware segítségével az Express automatikusan értelmezi a beérkező JSON adatokat, így a további kód könnyen hozzáférhet ezekhez az adatokhoz.

Multer

A multer egy Node.js modul, amelyet fájl feltöltési funkciók kezelésére használnak Express alkalmazásokban. A multer lehetővé teszi, hogy könnyedén kezeljük a HTTP kérések mellékleteit (például fájlok feltöltését) az Express alkalmazásban.

4.2.2. Fájlkezelés

A multer segítségével lehetőség van arra, hogy a feltöltött fájlok csak átmenetileg legyenek tárolva a szerver memóriájában. Inicializáltam a multert, létrehoztam egy konstanst, amiben a mentett fájlok elérési útját határoztam meg.

A *createUploadsFolder()* aszinkron függvénnyel létrehozom az előbb meghatározott elérési úton a mappát – amennyiben még nem létezik.

A *saveFile()* függvény segítségével kerülnek ideiglenes elmentésre a fájlok a szerveren a *savedFiles* objektumba, és a *removeFilesInFolder()* függvénnyel törlöm őket, ha az elkészült PDF el lett küldve a kliensnek.

4.2.3. PDF generálás

Egy nagy függvényben, a *combinePDFs()*-ben történik a generálás és az egyesítés is.

Adatok előkészítése

Ajánlott backend oldalon is szűréseket végezni, ha űrlapok feldolgozásáról van szó, így én is ezt tettem. Létrehoztam egy *filteredData* objektumot, amiben az űrlapból érkező adatokat helyeztem el megszűrve. A szűrés itt csak annyit jelent, hogy nem fájl típusú mezőből származik – szóval ténylegesen csak a szöveges adatokat helyezzük el ebben az objektumban – továbbá az üres vagy *undefined* értékekkel rendelkező mezőket figyelmen kívül hagyjuk.

Definiáltam egy függvényt, az *objectToFormData()*-t, aminek segítségével a filteredData objektumot átkonvertálhatom FormData típusú objektummá, hogy egyszerű legyen vele dolgozni.

```
    function objectToFormData(obj) {
  const formData = new FormData();
  for (const [key, value] of Object.entries(obj)) {
    formData.append(key, value);
  }
  return formData;
}
```

Ahhoz, hogy praktikusan tudjam kezelni a több értékű mezőkből származó értékeket, át kellett alakítanom a tömbös struktúrát egy olyanná, ahol egy kulcshoz csak egy érték tartozik. Így a tömb elemeinek értékét egy darab stringgé alakítottam a következő módon:

```
    const obligatoryExamsValues = [];
      for (const [key, value] of formData.entries()) {
        if (key.endsWith('obligatoryCourseCode')) {
          obligatoryExamsValues.push(value);
        }
      }
      const obligatoryExamsString = obligatoryExamsValues.join(', ');
      formData.set('obligatoryExams', obligatoryExamsString);
```

```
      const optionalExamsValues = [];
      for (const [key, value] of formData.entries()) {
        if (key.endsWith('optionalCourseCode')) {
```

```
        optionalExamsValues.push(value);
      }
    }
    const optionalExamsString = optionalExamsValues.join(', ');
    formData.set('optionalExams', optionalExamsString);
```

A kapott stringeket a hozzáadtam a formData-hoz, hogy a PDF generálásakor minden szükséges adat egy helyen legyen.

Létrehoztam egy új változót, a *creditSum*ot, amivel a különböző módon elért kreditek összegét szemléltetem majd a PDF-en. Ezt is hozzáadtam a formData-hoz kulcs-érték párosként.

pdf-lib

A különböző PDF-ek kezelését a *pdf-lib* könyvtárral valósítottam meg. A pdf-lib egy JavaScript könyvtár, amely lehetővé teszi PDF fájlok kezelését és generálását a böngészőben vagy Node.js környezetben. Ennek a könyvtárnak segítségével lehetőség van PDF fájlok létrehozására, szerkesztésére, összevonására, felbontására és más műveletekre. Kiváló választás lehet olyan alkalmazások számára, amelyeknek dinamikusan kell generálniuk PDF dokumentumokat a szerveroldalon.

Az alábbi függvénnyel hoztam létre a PDF-et:

```
const combinedPdfDoc = await PDFDocument.create();
```

Első oldal összeállítása

Az előbb létrehozott PDF jelenleg üres. Deklarálok egy *firstPage* konstanst a *const firstPage = combinedPdfDoc.addPage();* függvénnyel, ez lesz az első oldala a PDF-nek.

Az első oldal tartalmazza a szöveges adatokat, és kikötés volt, hogy ez el is férjen egy oldalon. Így viszonylag össze kellett sűrítenem a különböző elemeket, egy átlátható kialakítás megtartása mellett. Az volt a cél, hogy valamennyire hasonlítson az eredeti kredit összesítőre. A táblázatos elrendezés megnyerő volt, viszont implementálás közben rá kellett jönnöm, hogy a táblázat kivitelezése nem olyan egyszerű feladat. Nem találtam olyat csomagot, könyvtárat, amivel egyszerűen, körülményes módosítások nélkül tudtam volna – például csak 1 függvénnyel – táblázatba illeszteni az adatokat, így maradt a manuális megoldás.

A pdf-lib könyvtárban lehetőség van téglalapok kirajzolására, ezt használtam ki. Egy téglalap kirajzolása a következőképpen néz ki:

```
firstPage.drawRectangle({x: 35, y: 515, width: 450, height: 25,
borderWidth: 1.5, borderColor: grayscale(0), color: rgb(1, 1, 1) })
```

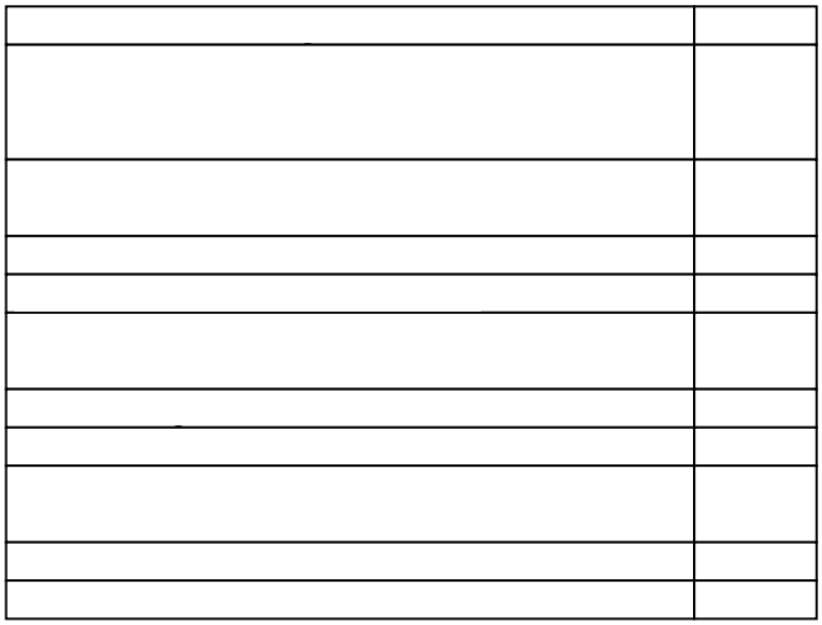
Pixel pontosan – összesen 22 darab – téglalappal tudtam kirajzolni a táblázatot. Aztán szintén pixel pontosan a táblázatban megjeleníteni külön-külön a mezők neveit és hozzájuk tartozó értékeket. A krediteket nem tudtam tökéletesen középre illeszteni, mivel számít a szöveg szélessége, és ez változik annak függvényében, hogy 1 vagy 2 számjegyről beszélünk.

Ez igen pepecselős munka volt, mivel a kivitelezés közben sokat kellett módosítgatni a méreteken, és ezáltal csúszott az egész táblázat. Minden módosításnál újra kellett számolni a pixeleket az összes téglalapnál és a hozzá tartozó szövegnél is.

A több értékű mezők téglalapjainál beállítottam egy plusz tulajdonságot, a *maxWidth*-et, ezzel megoldva, hogy sortöréssel több sorba kerüljenek a tárgyak, és kiküszöbölve a téglalapból való kicsúszást is.

Ezen kívül, pluszban még a keltezés is szerepel a lapon, dinamikusan az aktuális dátumból kapja az értékét.

```
const currentDate = new Date().toISOString().split('T')[0];
const miskolc = 'Miskolc, ';
const dateAndPlace = miskolc.concat(currentDate);
```



4.4. ábra: A téglalapokkal kirajzolt táblázat

Scope of Activities	Credits
Obligatory Subjects: PÉLDA-1, PÉLDA-2, PÉLDA-3, PÉLDA-4, PÉLDA-5, PÉLDA-6, PÉLDA-7	35
Optional Subjects: PÉLDA-8, PÉLDA-9, PÉLDA-10, PÉLDA-11	6
Research Seminar	0
Institutional/Departmental Research	0
Educational activity PÉLDA-10	5
Research Report	0
Complex Report	0
Second Language Exam: Intermediate	20
List of publications and professional presentations	0
Total Credit	66

Miskolc, 2024-05-12

4.5. ábra: Kitöltött táblázat

4.2.4. PDF-ek egyesítése

A savedFiles objektum segítségével a firstPage után hozzáadjuk a feltöltött PDF-eket a combinedPdfDoc-hoz az alábbi for ciklusban:

```
for (const fieldName of Object.keys(savedFiles)) {
    const pdfPaths = savedFiles[fieldName];

    for (const pdfPath of pdfPaths) {
        const pdfBytes = await fs.readFile(pdfPath);
        const externalPdfDoc = await PDFDocument.load(pdfBytes);

        const externalPages = await combinedPdfDoc.copyPages(
            externalPdfDoc,
            externalPdfDoc.getPageIndices()
        );

        externalPages.forEach((page) => {
            combinedPdfDoc.addPage(page);
        });
    }
}
```

Miután hozzáadtuk a saját PDF-enkhüz a feltöltötteket, elmentjük ezt egy byte tömbként:

```
const combinedPdfBytes = await combinedPdfDoc.save();
```

Aztán a combinedPdfPath változóban meghatározott útvonalon lévő fájlba írjuk a combinedPdfBytes byte tömb tartalmát. Az await kulcsszó miatt ez a művelet is aszinkron módon történik, így várakozik, amíg a fájl írása befejeződik.

4.2.5. PDF visszaküldése a kliensnek

Az előbb létrehozott combined_pdf.pdf fájlt beolvassuk a memóriába, és elmentjük ezt a *combinedPdfBytes* változóba.

Beállítjuk a HTTP válasz fejlécét – *res.setHeader()* az alábbiak szerint:

- A *Content-Type* tulajdonság *application/pdf* lesz, jelezve ezzel, hogy PDF fájlt küldünk vissza.
- A *Content-Disposition* tulajdonság *inline* értéket kap, ez azt jelenti, hogy a böngésző azonnal meg fogja jeleníteni a PDF-et. A *filename* tulajdonság pedig meghatározza a letöltött fájl nevét.

Végül a *res.send()* metódussal küldjük vissza a kliensnek a combinedPdfBytes változóban tárolt PDF tartalmat.

Hibakezelés

A *try-catch* blokk végzi a hibakezelést. Ha bármelyik fájlkezelő művelet (olvasás, írás) hibával tér vissza, akkor az *error* változóba kerül, és a catch blokkban kezeljük a hibát. Ebben az esetben a szerver 500-as hibakóddal és egy JSON válasszal jelzi az hibát (Internal Server Error).

5. fejezet

Tesztelés

5.1. Beüzemelés

1. Ahogy az implementációban már említettem, a szerver futtatásához szükséges feltelepíteni a Node.js-t.
2. Visual Studio Code-ban a terminál segítségével "cd" paranccsal a két fő mappába lépve (form-app és backend) az "npm install" paranccsal telepítjük a szükséges könyvtárakat.
3. Közvetlenül ezután – ugyanúgy a két fő mappában – az "npm run start", illetve a "node index.js" paranccsal tudjuk lefordítani, elindítani a két programot.
4. A böngészőben a *http://localhost:3000* címen elérhető a webalkalmazás (frontend része)

A szerver pedig a 3001-es porton üzemel.

5.2. Tesztfuttatások

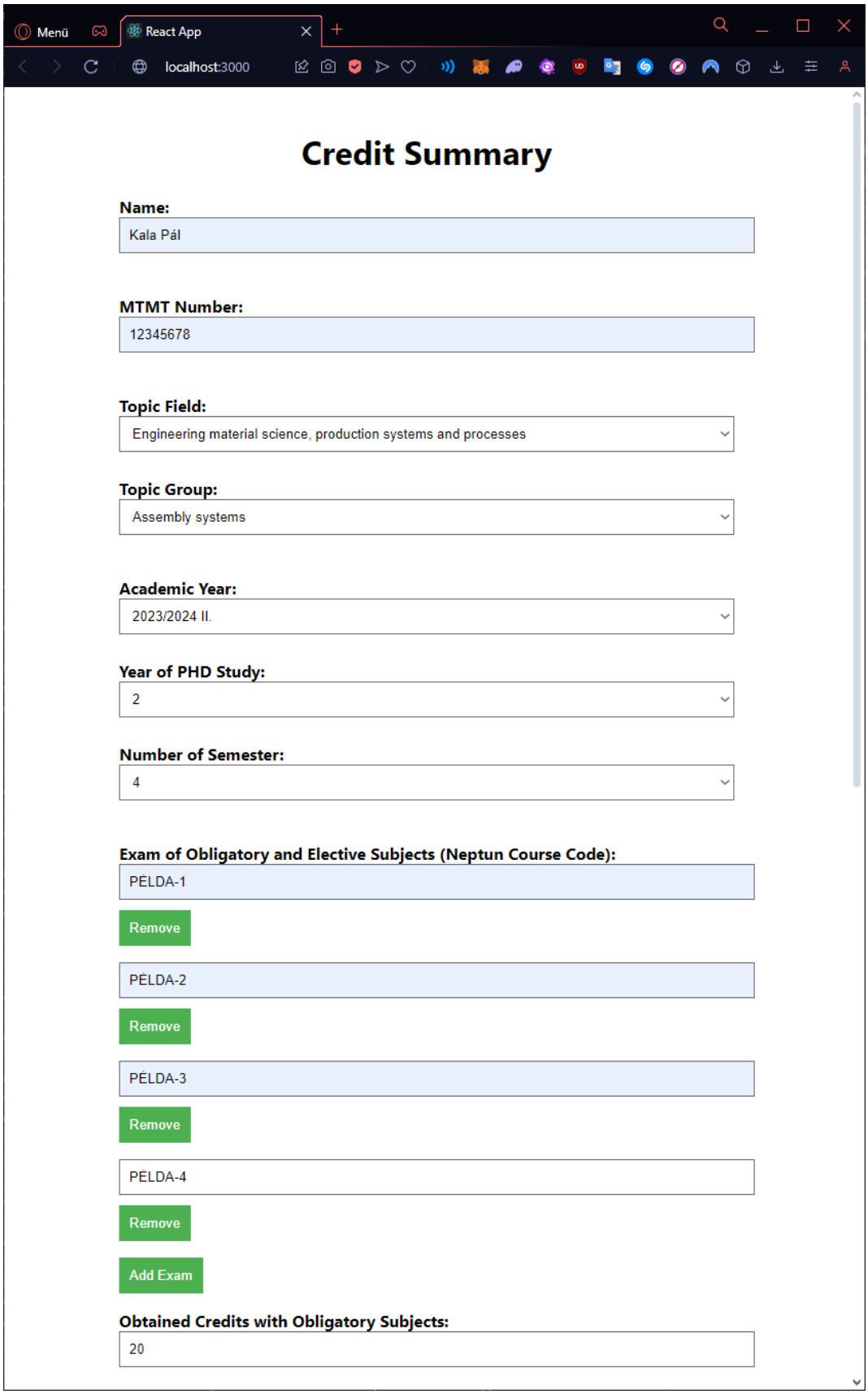
Kizárólag manuális tesztelés zajlott 2 készüléken, ezeket én végeztem el az implementáció közben és után.

5.2.1. Számítógép

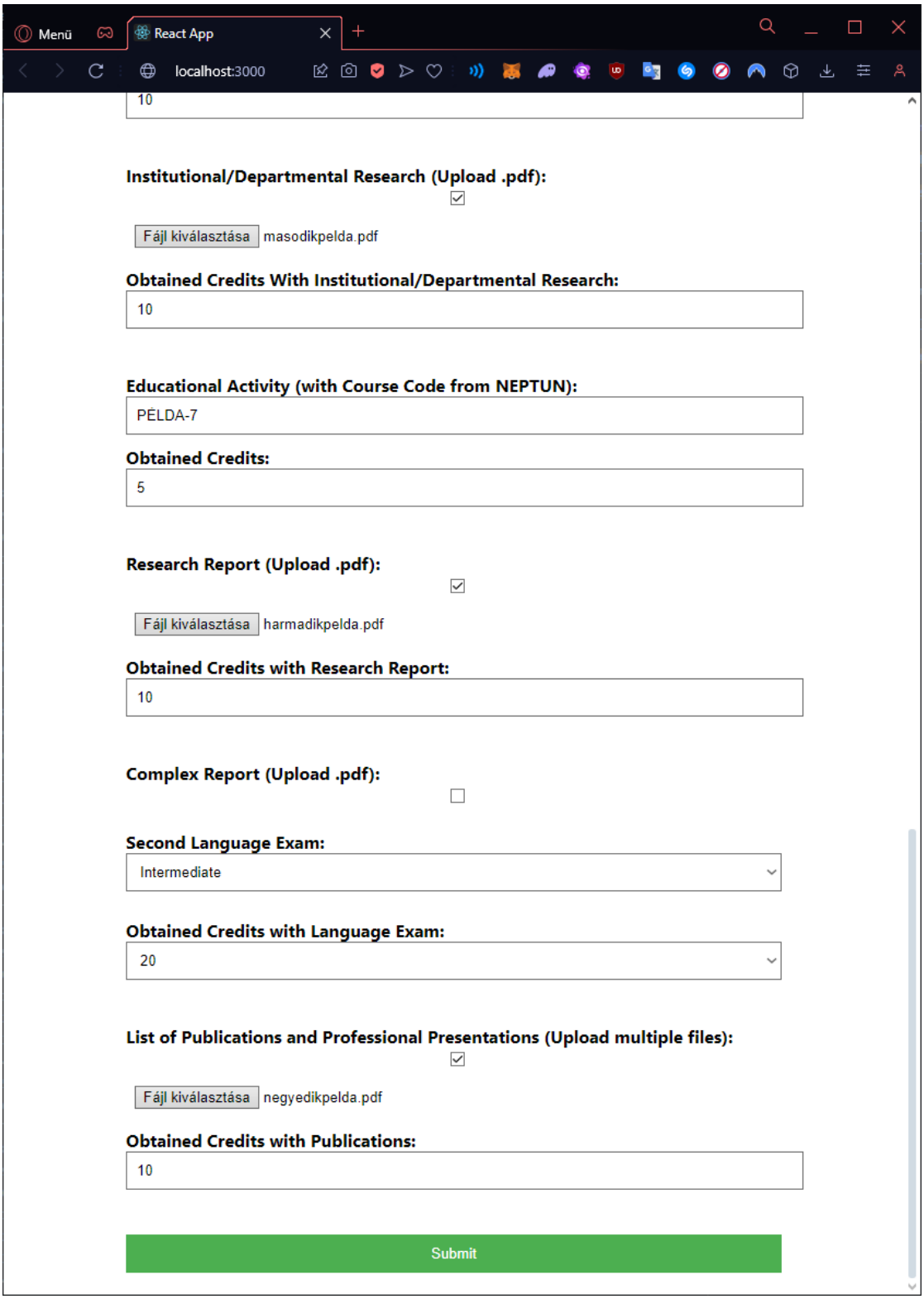
Windows operációs rendszeren teszteltem az alábbi 2 böngészőben:

- Microsoft Edge – A Windowson futó számítógépeken elérhető
- Opera GX – Személyes kedvencem, ezt a böngészőt használom a leggyakrabban

Utóbbiban teszteltem a fejlesztés közben, Edge-ben csak a kész alkalmazást.



5.1. ábra: A webalkalmazás első fele PC-n



5.2. ábra: A webalkalmazás második fele PC-n

Credit Summary

Name: Kala Pál
MTMT Number: 12345678
Topic Field: Design of machines and structures
Topic Group: Design of machine tools
Academic Year: 2023/2024 II.
Year of Ph.D. Study: 4
Number of Semester: 8

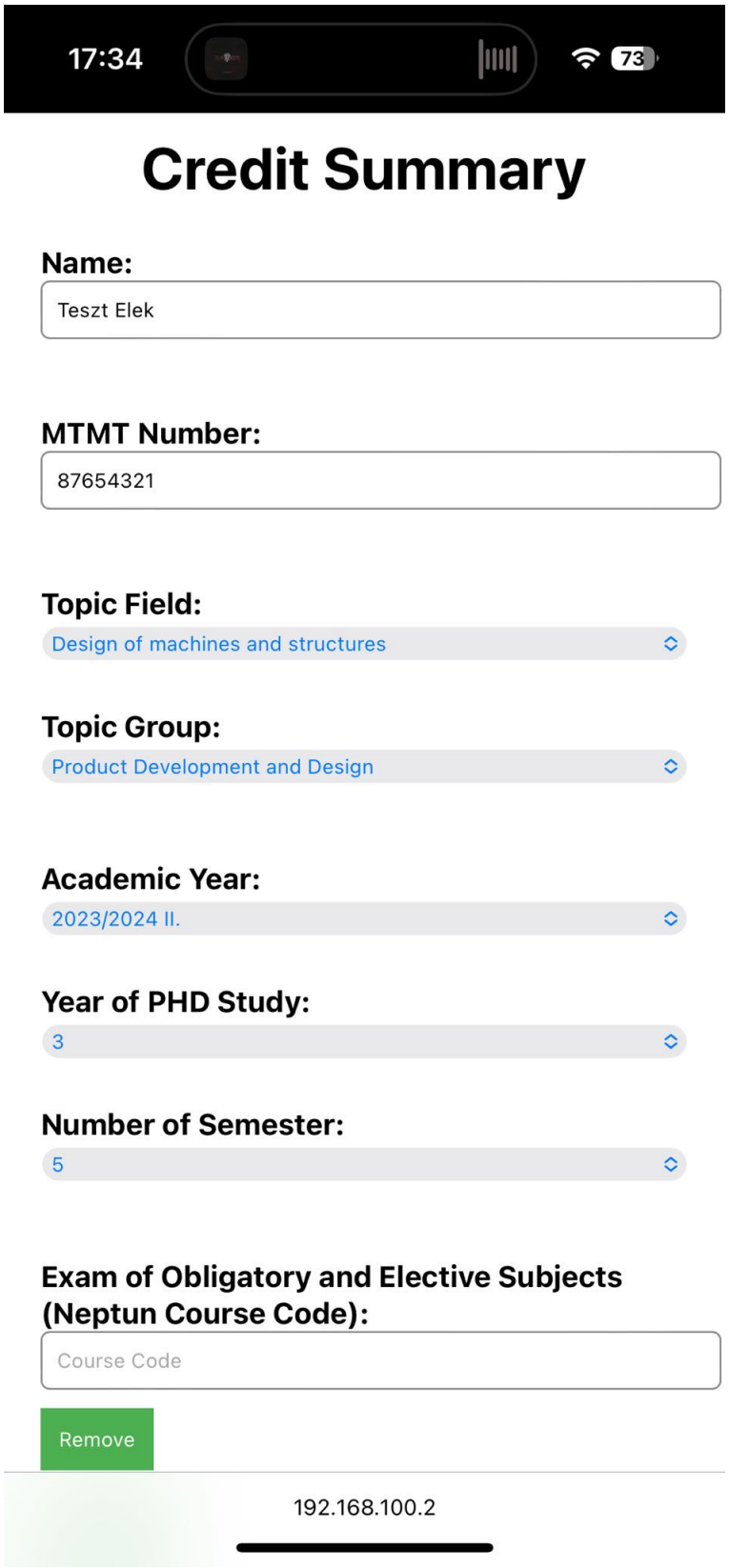
Scope of Activities	Credits
Obligatory Subjects: PÉLDA-1, PÉLDA-2, PÉLDA-3, PÉLDA-4, PÉLDA-5, PÉLDA-6, PÉLDA-7	35
Optional Subjects: PÉLDA-8, PÉLDA-9, PÉLDA-10, PÉLDA-11	6
Research Seminar	0
Institutional/Departmental Research	0
Educational activity PÉLDA-10	5
Research Report	0
Complex Report	0
Second Language Exam: Intermediate	20
List of publications and professional presentations	0
Total Credit	66

Miskolc, 2024-05-12

5.3. ábra: Az kapott PDF dokumentum első oldala

5.2.2. Mobiltelefon

Az operációs rendszer itt iOS volt, a böngésző pedig Safari. Ugyebár a számítógépen futtattam localhost-ban az alkalmazást, és ahhoz, hogy tesztelni lehessen mobilon is, meg kellett tudnom a számítógépem IP címét. Parancssorban az *ipconfig* paranccsal kiderítettem az IPv4 címemet. Jelen esetben ez a 192.168.100.2 volt, így a mobiltelefonomon a 192.168.100.2:3000 címen értem el az alkalmazást.



5.4. ábra: A webalkalmazás mobilon

6. fejezet

Összefoglalás

A szakdolgozat feladatom egy webalkalmazás fejlesztése volt, amely lehetővé teszi a doktorandusz hallgatók féléves kredit összesítő dokumentumának digitális előkészítését, megkönnyítve az űrlap kitöltését és a szükséges igazolások egy PDF fájllá való összefűzését.

A megvalósítás során megtervezésre került a frontend (melynek alapja a Doktori Iskola jelenleg használt kredit összesítő formanyomtatványa). Azaz a felhasználói felület egy űrlap kitöltő weboldal, amely különböző funkciókat megvalósító mezőkből áll. Ezáltal lehetővé teszi a benyújtandó dokumentum fedőlapjának digitális kitöltését, a szükséges igazolások csatolását és a végső dokumentum egyetlen PDF fájlban történő elkészítését.

Az alkalmazás háttér műveleteit a backend részben valósítottam meg. A felhasználó számára rejtett műveletek az adatkezelésből és fájlműveletek elvégzéséből állnak. A bevitt adatok ellenőrzése, szükséges feltételek megléte (csatolások kikényszerítése) és a végső pdf fájl létrehozása tartozik ide.

A szakdolgozatomban megvalósítottam a kiírt feladatot, elkészült egy webalkalmazás, amely teljesíti az előírt követelményeket. Ha üzembe lenne helyezve az alkalmazás, akkor megkönnyítené a doktoranduszok féléves kredit összesítő lapjának előkészítését és az ügyintézők dolgát is a benyújtási folyamat során.

Azonban a program még nem tökéletes, találhatóak benne kisebb hibák és pontatlanságok. Ezeknek a kijavítása folyamatban van, terveim szerint már a következő félévben használható lesz a program éles helyzetben is. Az éles felhasználás előtt az alábbi fejlesztéseket szeretném még megvalósítani.

Frontend terén az alábbi dolgok kerültek fel a teendőlistára: a fájl kiválasztó mezőknél jelenleg lévő checkboxok elég primitívek, ezeket mindenképp lecserélném vagy továbbvíve a gondolatot, akár lehetne azt is, hogy csak teljesített félév függvényében jelenjenek meg olyan dolgok, amelyeknek teljesítése csak bizonyos esetekben lehetséges – így egyáltalán nem lenne szükség a checkboxokra. Például a Research Seminar csak az 5., 6. és 7. félévekben kellenek, és ez csak akkor jelenne meg, ha ezek közül az egyik

van kiválasztva a Number of Semester mezőben. Továbbá az alkalmazás kinézetét is lehetne még fejleszteni egy teljes UX designnal vagy akár csak különféle CSS elemekkel, egész biztos vagyok benne, hogy sokat javítana a felhasználói élményen.

Backend részen is van még min finomítani, például a generált PDF első oldalának elrendezésén, ennek a teljes újratervezését is esélyesnek látom, mivel a manuális táblázat és szöveg kirajzolás túl sok negatívummal jár. Ezen kívül a létrehozott PDF fájl nevét szeretném majd a Name mező értékének felhasználásával dinamikusan beállítani, hogy ezzel ne a letöltés után legyen dolga a felhasználónak.

Felmerült bennem, hogy a React használata lehet kissé túlzott volt, akár egyszerűen HTML-ben is véghez lehetett volna ezt vinni, úgy viszont nem hagytam volna ekkora teret a továbbfejlesztésnek.

7. fejezet

Summary

My thesis task was to develop a web application that allows the digital preparation of a semester credit summary document for doctoral students, facilitating the completion of the form and the compilation of the required certificates into a PDF file.

The implementation involved the design of the frontend (based on the credit summary form currently used by the Doctoral School). The user interface is a form-filling web page consisting of fields implementing different functionalities. It allows the digital completion of the cover page of the document to be submitted, the attachment of the necessary supporting documents and the creation of the final document in a single PDF file.

The background operations of the application are implemented in the backend section. The operations hidden to the user consist of data management and performing file operations. This includes checking the input data, necessary conditions (forcing attachments) and creating the final PDF file.

In my thesis, I have implemented the assigned task and created a web application that fulfils the requirements. If the application was put in place, it would facilitate the preparation of the doctoral student's semester credit summary sheet and also the administrators' work during the submission process.

However, the program is not yet perfect, and there are some minor bugs and inaccuracies. These are being corrected and I plan to have the program ready for live use until the next semester. I would like to implement the following improvements before going live.

On the frontend side, the following things have been added to the to-do list: the checkboxes in the file selection fields are currently quite primitive, I would definitely replace them or, taking the idea further, I could even make it so that things that can only be completed in certain cases appear only depending on the completed semester – so the checkboxes would not be needed at all. For example, Research Seminar would only be required in semesters 5, 6, and 7, and it would only appear if one of those is selected in the Number of Semester field. Also, the look of the app could be improved

even more with a full UX design or even just different CSS elements, I'm quite sure it would improve the user experience a lot.

There is still room for improvement on the backend part too, as for the the layout of the first page of the generated PDF, I see a complete redesign of this as a possibility, as manual table and text plotting has too many disadvantages. In addition, I would like to dynamically set the name of the generated PDF file using the value of the Name field, so that the user does not have to deal with it after downloading.

It came up to me that the use of React might have been a bit excessive, I could have just done it in HTML, but then I wouldn't have left so much room for further development.

Források

[1] Anuj Adhikari. „Full stack javascript: Web application development with mean”. (2016).

[2] BEEKAI. *React Hook Form*. URL: https://react-hook-form.com.

[3] Scott Chacon és Ben Straub. *Pro Git*. URL: https://git-scm.com/book/en/v2.

[4] Visual Studio Code. *Learn to code with Visual Studio Code*. URL: https://code.visualstudio.com/learn.

[5] MDN Web Docs. *What is JavaScript?* URL: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript.

[6] Sályi István Gépészeti tudományok Doktori Iskola. *Kredit összesítő benyújtási rendje*. URL: https://geik.uni-miskolc.hu/intezetek/SALYI/dokumentumok.

[7] Sályi István Gépészeti tudományok Doktori Iskola. *Kredit összesítő lap*. URL: https://geik.uni-miskolc.hu/intezetek/SALYI/dokumentumok.

[8] Sályi István Gépészeti tudományok Doktori Iskola. *Kreditrendszer*. URL: https://geik.uni-miskolc.hu/intezetek/SALYI/kredit.

[9] Matteo Frana. *React Hook Form: the best form library?* URL: https://dev.to/matfrana/react-hook-form-the-best-form-library-1o1l.

[10] iLovePDF. *PDF merger*. URL: https://www.ilovepdf.com/merge_pdf.

[11] Kinsta. *What Is GitHub?* URL: https://kinsta.com/knowledgebase/what-is-github/.

[12] Moritz Mähr. „Working with batches of PDF files”. *The Programming Historian* (2020).

[13] NAV. *Adóbevallás Kitöltő*. URL: https://webnyk.nav.gov.hu/app/public.action#tform.

[14] Minh Nguyen Nhat. „Building a component-based modern web application: full-stack solution”. (2018).

[15] Node.js. *Introduction to Node.js*. URL: https://nodejs.org/en/learn/getting-started/introduction-to-nodejs.

[16] W3Schools. *CSS Introduction*. URL: https://www.w3schools.com/css/css_intro.asp.

[17] W3Schools. *What is React?* URL: https://www.w3schools.com/whatis/whatis_react.asp.

Adathordozó használati útmutató

A program futtatásához szükség van egy integrált fejlesztői környezetre (IDE), például a Visual Studio Code-ra, amit én is használtam, valamint telepíteni kell még a Node.js-t is.

Az adathordozón megtalálható a dolgozat.pdf, ami maga a szakdolgozat PDF formátumban, emellett a Szakdolgozat mappában megtalálható a PDF előállításához használt LaTeX fájlok is a felhasznált képekkel együtt. A programok a *webapp* mappában találhatóak meg, ezt kell megnyitni a Visual Studio Code-ban ha a program futtatása a cél. Megnyitás után az 5. fejezet elején leírtak alapján lehet beüzemelni az alkalmazást.