



# Szakdolgozati konzultációk támogatása webes rendszerrel

## **Készítette**

Vereczki Bálint Zoltán  
programtervező informatikus BSc

## **Témavezető**

Balla Tamás  
tanársegéd

EGER, 2021

# Tartalomjegyzék

<b>1. Téma elméleti kifejtése</b>	<b>5</b>
1.1. A web, mint platform . . . . .	5
1.2. Webes architektúrák . . . . .	6
1.3. Webalkalmazáshoz alkalmazható technikák . . . . .	8
<b>2. Felhasznált technológiák ismertetése</b>	<b>9</b>
2.1. Kliensoldali technológiák . . . . .	9
2.1.1. Angular . . . . .	10
2.1.2. React . . . . .	11
2.1.3. Vue.js . . . . .	14
2.2. Szerveroldali technológiák . . . . .	15
2.2.1. Express . . . . .	15
2.2.2. NestJS . . . . .	16
2.3. Adatbázis-kezelő rendszerek . . . . .	19
2.3.1. A relációs adatbázisok . . . . .	19
2.3.2. A NoSQL adatbázisok . . . . .	20
2.4. Összegzés . . . . .	21
<b>3. Specifikáció</b>	<b>22</b>
3.1. Feladatspecifikáció . . . . .	22
3.1.1. Felhasználói szerepkörök . . . . .	22
3.2. Adatbázisterv . . . . .	25
3.3. Autentikáció és autorizáció . . . . .	29
3.4. Kliensoldali alkalmazás felépítése . . . . .	32
<b>4. Fejlesztői dokumentáció</b>	<b>33</b>
4.1. Ügyintézői felület . . . . .	34
4.2. Oktatói felület . . . . .	37
4.2.1. Szakdolgozatok és témák . . . . .	37
4.2.2. Hallgatók . . . . .	39
4.2.3. Konzultációk . . . . .	40
4.2.4. Mérőfldkövek . . . . .	41

4.3.	Hallgatói felület . . . . .	44
4.3.1.	Szakdolgozat . . . . .	44
4.3.2.	Konzultációk . . . . .	44
4.3.3.	Mérföldkövek . . . . .	45
<b>5.</b>	<b>Összefoglalás</b>	<b>47</b>

# Bevezetés

Életünkben egyre nagyobb területén van jelen az internet. Bárhol, bármikor hozzáférhetünk a zsebünkben hordott mobilkészülékünkről, pár érintés után elérhetjük a legfrissebb információkat és híreket. Terméket és ételt rendelünk, tájékozódunk, hivatalos ügyeket intézünk, kapcsolatot tartunk, zenét és filmet nézünk, mindezt a világhálón. Ezen tevékenységek révén egyre interaktívabb weboldalak készültek az évek során, melyek ki tudják szolgálni ezeket az igényeket.

A felsőoktatásban elsődlegesen preferált kommunikációs forma a személyes konzultáción kívül az email. A legtöbb ügyintézés e-mail-es levelezéssel történik, ez pedig egyes esetekben problémákhoz vezethet. A rengeteg üzenet között előfordul, hogy egy-egy fontos üzenet elkerüli a figyelmet. A jelenlegi járványhelyzet akadályozza a személyes beszélgetés lehetőségét, így csupán az online kommunikációra hagyatkozhatunk.

A szakdolgozatom célja ezen okoknál fogva egy olyan webes platform létrehozása, amely mind az oktatók, mind a hallgatók munkáját nagyban megkönnyíti és elősegíti a szakdolgozat elkészültének ideje alatt. Lehetőséget nyújt a hallgatók, szakdolgozatók, konzultációs időpontok menedzselésére, határidőre megadott feladatok kiírására és azokon keresztül való visszajelzésre.

Szakdolgozatomban kitérek a webre, mint platformra és a webes alkalmazások mögött álló architektúrákra. Áttekintek néhány, a mai piaci trendek alapján leginkább használt keretrendszert, majd kiválasztom közülük azokat, amelyekkel az alkalmazásomat fogom elkészíteni. Ezt követően bemutatom a szoftver terveit és az elkészült alkalmazást.

# 1. fejezet

## Téma elméleti kifejtése

### 1.1. A web, mint platform

Az egész világot behálózó számítógépes hálózatot internetnek nevezzük. Az internetre kapcsolódó számítógépek a TCP/IP (Transmission Control Protocol / Internet Protocol) segítségével képesek az egymással való kommunikációra. Minden egyes csatlakozott eszköz egy egyedi azonosítót kap, amelyet IP címnek nevezünk. A weboldalakat az IP címek helyett könnyebben megjegyezhető domain címek alapján látogatjuk meg (például *www.google.com*), melyeket az internetre kapcsolódó speciális szerverek a DNS azaz névszerverek fordítanak le a hozzájuk tartozó IP címre.[1]

A fent említett protokollokon kívül számos jelentős protokoll létezik, melyeket más célokra tudunk használni. Jelentősebb protokollok közül ilyenek például[2]:

- UDP (User Datagram Protocol)
- FTP (File Transfer Protocol)
- SSH (Secure Shell)
- *HTTP* (Hypertext Transfer Protocol)

Az internet által használt TCP (Transmission Control Protocol) egy megbízható, kapcsolatorientált, végpont-végpont közötti kétirányú adatfolyamot biztosító protokoll. Megbízhatósága onnan ered, hogy az adatfolyam küldése közben nyugtázással biztosítja az adat megérkezését, hiba esetén pedig újrapróbálkozik az adat továbbításával.

Az UDP protokoll úgynevezett datagram alapon szállítja az adatot. A datagramban tárolódik a forrásport, célport, adat hossza, az ellenőrzőösszeg és maga a küldeni kívánt adat. Ellentétben a TCP-vel, nem biztosítja az üzenet megérkezését, olyan esetekben optimális a használata, ahol nem feltétlenül szükséges a csomag megérkeztéről való visszaigazolás.

Az FTP állományok számítógépek közötti átvitelére használatos protokoll, melynél két csatorna épül ki. Amikor a kliens csatlakozik az FTP szerverhez mindkét oldalon kiépül a vezérlő csatorna (Control Process), melyen keresztül a kliens parancsokat adhat a távoli gépnek. Amint állományátviteli kérést adunk ki, kiépül az adatcsatorna, amin keresztül az adatcsere folyamata zajlik.

Az FTP protokoll autentikáció szempontjából nem tekinthető biztonságosnak, sem a TLS (Transport Layer Security), sem az SSL (Secure Sockets Layer) titkosítási protollokat nem támogatja, a bejelentkezéshez használt felhasználónév és jelszó titkosítás nélkül kerül küldésre a kientől a szerverig. Ezen probléma megoldásaként nyújt alternatívát az SSH (Secure Shell) protokoll, amely biztonságos kapcsolatot épít ki a helyi és távoli számítógép között, és állományátvitelen kívül használható a távoli gép kezelésére, parancsok kiadására. Az SFTP protokoll SSH-t használ a kapcsolat kiépítéséhez, így biztonságos kapcsolatot biztosít az állományátvitelhez. Az autentikáció történhet jelszóval, illetve egy úgynevezett nyilvános kulccsal is.[3]

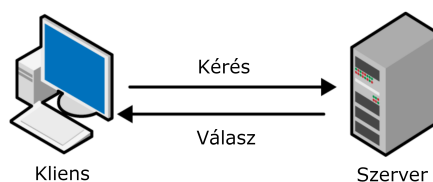
A HTTP protokoll segítségével képes kommunikálni a kliens (böngésző) és a szerver. Megbízható kapcsolatra van szükség, nem szabad üzenetvesztésnek történnie, ezért a kommunikációhoz TCP protokollt használ, amely kérés-válasz formájában történik. Ez a protokoll állapotmentes, ami azt jelenti, hogy a szerver nem emlékszik korábbi kérésekre, minden kérést egymástól függetlenül kezel.[4]

Minden kérés 3 részből áll össze: a kérés parancsából, ami tartalmazza a kérés metódusát, erőforrás útvonalát és a HTTP verzióját. Ezen kívül fejléceadatokat, mint például a Host, User-agent, Content-Type, és opcionálisan üzenetet tartalmaz. A válasz szintén 3 részre osztható: HTTP státusz kód, fejléceadatok és az üzenet.

Többféle HTTP metódussal lehet kérést küldeni, adat letöltéshez például a GET, feltöltéshez a POST, frissítéshez a PUT vagy PATCH, törléshez pedig a DELETE használható.[5]

## 1.2. Webes architektúrák

A webes alkalmazások tekintetében többféle módon is megtervezhetjük a mögöttes architektúrát. A HTTP kérés-válasz modellje miatt, legkézenfekvőbb a kliens-szerver architektúra (1.1. ábra).



1.1. ábra. A kliens-szerver architektúra<sup>1</sup>

A kliens-szerver architektúrának több fajtája létezik, az ezek közötti különbséget a rétegek száma határozza meg. Létezik kétrétegű (two-tier), háromrétegű (three-tier) architektúra, több réteg esetén pedig n rétegű (n-tier) architektúráról beszélhetünk.

A kétrétegű architektúra résztvevői a kliens és a szerver. A kliens megjeleníti a felhasználói interfészt, majd az azon történt interakció után kérést küld a szervernek, és az onnan kapott választ feldolgozza és megjeleníti. A szerver passzív szerepet tölt be, várja a kliensek kéréseit, melyeket teljesít és visszaküldi nekik a választ.

A fenti architektúrát egy újabb köztes réteggel bővítve háromrétegű (three-tier) architektúrát kapunk (1.2. ábra).



1.2. ábra. A háromrétegű architektúra<sup>2</sup>

A háromrétegű architektúrában megkülönböztetjük a megjelenítési (presentation), alkalmazás (application), és perzisztencia (data) rétegeket.[6]

A megjelenítési réteg közvetlenül elérhető a felhasználó számára, mely a szervertől kapott adatoknak biztosít grafikus interfészt. Ehhez használatos a HTML leírónyelv, CSS stílusok és a JavaScript, amely segítségével dinamikussá és lényegesen interaktívabbá tehető a felület.

Az alkalmazás rétegben található az összes szükséges üzleti logika, hidat alkotva a kliens és az adatbázis között. A kliensektől érkező kéréseket fogadja, kiértékel, logikai döntéseket hoz, adatokat validál és dolgoz fel, számolási feladatokat lát el, adatot továbbít a kérést indító kliens felé.

Az adatbázis réteg feladata az adattárolás, és adathozzáférés biztosítása, melyet általában egy relációsadatbázis-kezelő rendszer (RDBMS) végez, mint például Oracle, MySQL, PostgreSQL. Relációs adatbázisszervereken kívül történhet NoSQL adatbázis-kezelő rendszeren is az adattárolás, ezek közül a legnépszerűbb megoldás a MongoDB.

A kliens-szerver architektúra előnye, hogy nagyon könnyen skálázható újabb szerverek és kliensek beiktatásával, ezenfelül minden adat a szervereken tárolódik (centralizált

<sup>1</sup> Forrás: [https://madooei.github.io/cs421\\_sp20\\_homepage/assets/client-server-1.png](https://madooei.github.io/cs421_sp20_homepage/assets/client-server-1.png)

<sup>2</sup> Forrás: <https://docs.aws.amazon.com/whitepapers/latest/serverless-multi-tier-architectures-api-gateway-lambda/images/image2.png>

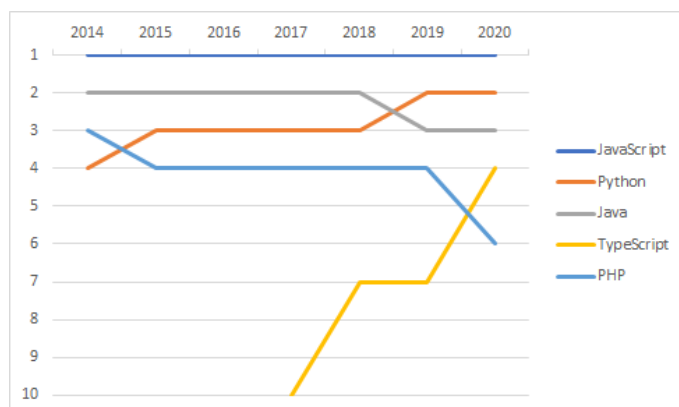
adattárolás), így sokkal biztonságosabb is, mert szabályozható az egyes erőforrásokhoz való hozzáférés.

Hátrányok közé sorolható, hogy gyakran eléggé drága szervereket kell vásárolni, illetve azok karbantartásához szakemberekre is szükség van.

A legtöbb webalkalmazás a háromrétegű architektúrát alkalmazza, ugyanis sokkal könnyebb ezzel a munka. A rétegek egymástól függetlenül működnek, ezáltal ha valamin – a rétegeket összekötő interfészekén kívül – változtatás vagy frissítés történik, az nem érinti egyben az egész alkalmazást, csupán az adott réteget. Ezen kívül a fejlesztés szempontjából is sokkal könnyebb egy ilyen rendszeren dolgozni az elkülönített részegységek révén.

### 1.3. Webalkalmazáshoz alkalmazható technikák

Webalkalmazások készítésére számtalan programozási nyelv és keretrendszer elérhető a fejlesztők számára. Bizonyos időközönként (havonta / évente) több szervezet is (például Tiobe, StackOverflow[8], GitHub[10]) statisztikát készít fejlesztők által kitöltött kérdőívek, és adott kifejezésekre való keresések száma alapján az adott időszak legkedveltebb és legtöbbet használt programozási nyelveiről és keretrendszerekről.



1.3. ábra. Programozási nyelvek ranglistája (2014-2020)

A State of the Octoverse[10] a GitHub által minden évben elkészített felmérés, melyben a projektek és felhasználók adatait felhasználva készítenek statisztikát az előző évre vonatkozóan. Az fenti grafikonon néhány, a webes alkalmazásokhoz is használható nyelvet emeltem ki a 2014-től 2020-ig tartó időszakban.

Az 1.3. ábra alapján egyértelműen látszik, hogy a JavaScript toronymagasan vezeti a ranglistát már 6 éve, míg a legtöbb weboldal által használt PHP népszerűsége az utóbbi időben csökkent. 2012-ben jelent meg a TypeScript, ami 2017-től kezdődően évről-évre népszerűbb a fejlesztők körében a statikus típusossága és objektumorientáltsága révén. Egyre nagyobb népszerűségnek örvend a Python is, habár elsődleges felhasználási területe nem a webfejlesztés, készült backendre alkalmazható keretrendszer.

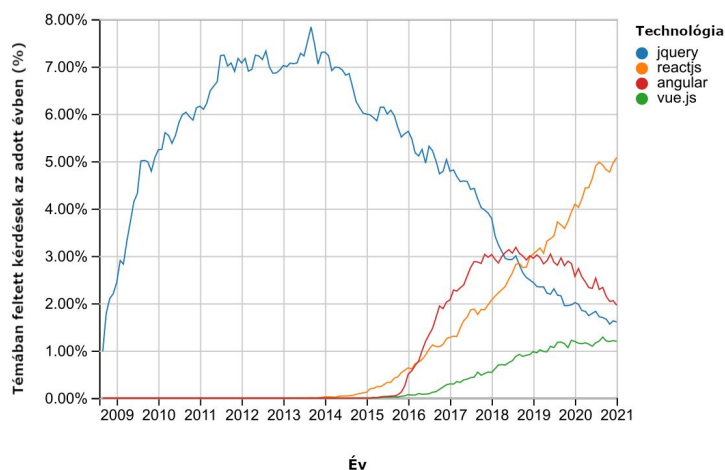


## 2. fejezet

# Felhasznált technológiák ismertetése

### 2.1. Kliensoldali technológiák

A 2006-ban megjelent jQuery[7] könyvtár lehetőségeket tárhat meg a web-fejlesztők felé. Segítségével rövidebb kóddal elérhető ugyanaz a működés, mint natív JavaScript esetén, és a különböző böngészők okozta kompatibilitási problémákra is megoldást nyújt. A könyvtárat a mai napig nagyon sok weboldal használja, ötvözve a fejlesztők által írt bővítményekkel pedig rengeteg problémára sokkal könnyebb megoldást találni, mint ha natív JavaScript-ben fejlesztené a programozó. Hátulütője, hogy sebességét tekintve jóval lassabb a JavaScript-nél és az SPA<sup>1</sup> webalkalmazások idejében egyre elavultabbá válik, például a Bootstrap CSS keretrendszer esetében a legújabb, 5. verzióban már kivezetésre került a könyvtár.



2.1. ábra. JavaScript keretrendszerek népszerűsége[9]

<sup>1</sup> Single Page Application

A 2.1. ábrán a Stackoverflow Trends adatai alapján készült grafikon látható, mely 2009-től 2021-ig bezárólag megmutatja, hogy egyes keretrendszerek mennyire népszerűek az oldalon feltett kérdések alapján. A piacot jelenleg 3 fő keretrendszer uralja, az Angular, a React és a Vue.js.

### 2.1.1. Angular[13]

Az Angular a Google által fejlesztett nyílt forráskódú keretrendszer, mely első kiadása 2016. szeptemberében jelent meg, az AngularJS második, továbbfejlesztett változataként. A keretrendszert TypeScript programozási nyelvben írták, és segítségével a HTML és TypeScript ötvöztetésével készíthetünk webalkalmazásokat, illetve natív mobilalkalmazásokat is.

Egy Angular alapokon épülő webalkalmazás úgynevezett NgModule-okból áll, melyek komponensekből, szervizekből, és egyéb az adott modulhoz tartozó fájlokból állnak össze. A komponensek nézeteket tartalmaznak az adatok manipulálására és megjelenítésére, és service-eket használnak különféle specifikus feladatokra (például szerver oldal felé történő kérések küldésére).

A view, azaz a nézet felelős az adatok megjelenítéséért. Ehhez az Angular-ban sablonokat (template) használhatunk, egy speciális Angular szintaxissal, amely kibővíti a HTML leírónyelvet, és több funkcionalitást biztosít, mint például a változók használata, eseménykezelés, és adatkötési technikák. Ezzel a szintaxissal definiált műveletek még a nézet megjelenítése előtt megváltoztatják a HTML elemeket, hogy azok már a kívánt adatokkal jelenjenek meg a böngészőben.

Révén, hogy a sablonokban nem használhatunk JavaScript programkódot, az Angular úgynevezett beépített direktívákat használ például az iterációhoz (*\*ngFor*) és feltételvizsgálathoz (*\*ngIf*), de írhatunk saját direktívákat is.

```
1 <tr *ngFor="let kep of kepek; let i = index;">
2   <td>{{ i }}</td>
3   <td>{{ kep.megnevezes }}</td>
4   <td>{{ kep.keszito }}</td>
5   <td>{{ kep.url }}</td>
6 </tr>
```

2.1. kód. \*ngFor direktíva használata

Service osztályokat azon adatoknak és programlogikának készíthetünk, melyek nem kifejezetten tartoznak egy nézethez, és a webalkalmazásunk akár több pontján is használni szeretnénk. Ezeket az osztályokat az *@Injectable()* dekorátorral annotálva más osztályokban is felhasználhatjuk a függőség befecskendezés (dependency injection) segítségével.

Az alkalmazásban történő útválasztás, átirányításokhoz a *@angular/router* csoma-

got használhatjuk, mely automatikusan beépül a generált projektbe, ha a projektet a „*ng new routing-app –routing*” paranccsal generáljuk le az Angular CLI-n keresztül.

Nagyobb alkalmazások esetén az alkalmazáson belüli állapotkezeléshez, az *ngrx*<sup>2</sup>, illetve *ngxs*<sup>3</sup> csomagokat használhatjuk, melyeket a React alkalmazásokhoz is használt Redux is nagyban inspirált.

### 2.1.2. React[11][14]

A React a Facebook által fejlesztett, nyílt forráskódú keretrendszer, mely 2013 májusában jelent meg. Webes alkalmazásokon kívül használható platformfüggetlen mobilalkalmazások készítésére is a React Native keretrendszer segítségével.

Webalkalmazásokban felhasználói interfész és azok komponenseinek készítésére alkalmazható. Ellentétben a többi keretrendszerrel, a program és megjelenítésért felelős logikát nem külön fájlokban, hanem egy komponensben központosítja. Ehhez a JSX<sup>4</sup>-et használhatjuk, ami egy JavaScript-et kibővítő szintaxis, mely a megjelenítéshez használatos HTML és egyéb komponens elemeken kívül JavaScript logikai programkódot is tartalmazhat.

```
1 import React from 'react';
2
3 const element = (
4     <h1 className='cimsor'>Ez egy cimsor!</h1>
5 );
```

2.2. kód. JSX elem létrehozása

Ezeket a komponenseket önállóan a böngésző nem tudja értelmezni, szükséges hozzá egy fordító is, a React a Babel JavaScript fordítót alkalmazza. A fordítás folyamán a JSX szintaxissal megírt kód minden egyes elemét átalakítja egy *React.createElement()* metódushívássá, amely ellenőrzések sora után a 2.2. kódban látható JSX elemből a 2.3. kódban látható JavaScript objektumot generálja le.

```
1 const element = {
2     type: 'h1',
3     props: {
4         className: 'cimsor',
5         children: 'Ez egy cimsor!'
6     }
7 };
```

2.3. kód. React.createElement() által generált objektum

---

<sup>2</sup><https://ngrx.io>

<sup>3</sup><https://ngxs.io>

<sup>4</sup>JavaScript XML

Ahhoz, hogy a létrehozott elemeket látni is lehessen a böngészőben, ki kell azokat rajzolni. A JavaScriptben a DOM<sup>5</sup> segítségével lehet az oldal tartalmához hozzáférni, illetve azt módosítani. A DOM fa alapú hierarchiában tárolja az oldal felépítését, minden csomópontban egy-egy elemmel és annak tulajdonságaival. Amikor megváltoztatásra kerül egy elem a DOM-ban, akkor az egész oldalt újra kell rajzolni a hozzá tartozó stílusokkal együtt. Ennek hátránya, hogy nagyobb módosítások esetén túl sok időt vesz igénybe a DOM újrarajzolása. Ezen probléma megoldására fejlesztették ki a React DOM-ot.

A React DOM a memóriában tárolja a felhasználói interfész másolatát, és biztosítja egy állapotváltozás esetén a szinkronizációt a DOM-mal. Folyamatosan vizsgálja az elemeket és azok gyerekeleit, csupán akkor rajzolja őket újra, ha különböznek az előző állapotuktól, ezzel elkerülve az oldal teljes újrarajzolását.

Az útválasztásra a React önmagában nem nyújt megoldást, ezt egy külső, a közösség által létrehozott és fejlesztett *react-router* csomaggal valósíthatjuk meg. Nagyobb méretű alkalmazások esetén az alkalmazás állapotkezelésére több könyvtár, illetve architektúra is létezik. A legnépszerűbb megoldások közé tartozik a Flux<sup>6</sup>, amelyet a Facebook fejlesztett ki, a React közösség által fejlesztett Redux, illetve a MobX<sup>7</sup>.

## Redux[15]

A Redux állapotkezelés alapelve, hogy az alkalmazás által használt adatokat egy központi helyen, a store-ban tároljuk. Az itt tárolt adatok önmagukban megváltoztathatatlan (immutable) objektumok.

A store-ban tárolt adatok felülírásához folyamatokat (action) kell létrehoznunk (2.4. kód). Minden folyamatnak meghatározott típusa van, ezen kívül szállíthat adatot is, amely a payload nevű mezőben található.

```
1 export const authRequested = () => {
2   return {
3     type: AUTH_REQUEST
4   }
5 }
6
7 export const successfulAuth = (token, user) => {
8   return {
9     type: AUTH_SUCCESS,
10    payload: {
11      token,
12      user
13    }
14 }
```

---

<sup>5</sup> Document Object Model

<sup>6</sup> <https://facebook.github.io/flux/>

<sup>7</sup> <https://mobx.js.org/>

```

14     }
15 }

```

#### 2.4. kód. Részlet az authActions.js fájlból

Az, hogy egyes folyamatok miként írják felül a store-ban tárolt értékeket a *reducer* dönti el (2.5. kód). Minden reducer rendelkezik egy *initialState* nevű objektummal, amely meghatározza az alapállapotot. Erről az objektumról készít másolatot a reducer, a megfelelő mezők módosításával, amely az új állapotává válik a storeban.

```

1  const initialState = {
2      loading: false ,
3      isAuthenticated: false ,
4      token: null ,
5      user: null ,
6      passwordChangeMessage: undefined
7  };
8
9  const authLoginReducer = (state = initialState , action
10     ) => {
11      const { type , payload } = action;
12
13      switch(type) {
14          case AUTH_REQUEST:
15          case AUTH_CHANGE_PASSWORD_REQUEST:
16              return {
17                  ...state ,
18                  loading: true ,
19                  passwordChangeMessage: undefined
20              }
21          case AUTH_SUCCESS:
22              return {
23                  ...state ,
24                  error: [] ,
25                  loading: false ,
26                  token: payload.token ,
27                  user: payload.user ,
28                  isAuthenticated: true
29              }
30      }
31  }

```

#### 2.5. kód. Részlet az authReducer.js fájlból

Store készítéséhez (2.6. kód) meg kell adnunk a használni kívánt reducer-eket, illetve megadhatunk opcionálisan kiegészítő funkciókat, melyekkel nyomon követhetjük a folyamatokat, és a store-ban történő adatmódosulásokat. Ezek segítségével könnyebb a fejlesztés során a felmerülő hibák megkeresése és javítása.

```

1  const middleware = [thunk];
2  const store = createStore(
3      rootReducer,
4      composeWithDevTools(applyMiddleware(...
5                               middleware))
  );

```

2.6. kód. Store létrehozása

### 2.1.3. Vue.js[11][12]

A Vue.js nyílt forráskódú keretrendszer Evan You nevéhez köthető. A Google-nál Angular alapokon fejlesztett projekteket, majd az onnan való kilépését követően kezdte el fejleszteni a Vue.js-t, melyet a mai napig karbantart.

A Vue akárcsak a React egy virtuális DOM-ot használ az állapotváltozások figyeléséhez, és a DOM-mal való szinkronizációhoz.

A Vue által megjelenített oldalak az Angular keretrendszerben is használt sablonokkal kerülnek definiálásra. Ehhez egy HTML alapú sablon szintaxist használ, mely segítségével adatkötést végezhetünk. Szöveges adatkötés esetén a „Mustache” szintaxis alkalmazhatjuk, azaz dupla kapcsos zárójelek közé kell a megjeleníteni kívánt adattagot tenni.

A komponensekben a már Angular esetén is látott direktívák alkalmazhatóak iterációhoz, feltételvizsgálathoz, adatkötéshez. A Mustache szintaxis például elhagyható a *v-html* direktíva használatával.

```

1  <p>Mustache szintaxissal: {{ uzenet }}</p>
2  <p>v-html directive: <span v-html="uzenet"></span></p>

```

2.7. kód. v-html direktíva használata

Számos más direktíva is elérhető, úgy mint a *v-bind*, amely segítségével HTML attribútumokat tudunk adatkötéssel beállítani, a *v-if* és a *v-else* a feltételvizsgálathoz használható, a *v-on:[eseményNév]* direktívával pedig az eseményekhez köthetünk funkciókat. Ha a megjelenítés során iterációt szeretnénk használni, ahhoz a *v-for* direktívát használhatjuk, amellyel tömbökön vagy egy objektum mezőin iterálhatunk végig.

Az alkalmazásbeli útválasztáshoz elérhető egy, a keretrendszer által is támogatott könyvtár a *vue-router*, mellyel kulcs-érték párokkal megadhatjuk, hogy egyes útvonalak esetén melyik komponensünket szeretnénk megjeleníteni.

Az alkalmazáson belüli állapotkezelésre több könyvtár is elérhető, többek között a React ihlette Flux-hoz nem csak névben, de működésében is hasonló *vuex*<sup>8</sup>.

<sup>8</sup><https://vuex.vuejs.org/>

## 2.2. Szerveroldali technológiák

### 2.2.1. Express[16][17]

A 2009-ben megjelent nyílt forráskódú, platformfüggetlen NodeJS által szerveroldalon is tudunk JavaScript nyelven írt programokat futtatni. A Google által fejlesztett V8 JavaScript motort használja, amely mellett egy alacsony-szintű I/O interfészt biztosít. A NodeJS telepítésekor megkapjuk az NPM<sup>9</sup> csomagkezelőt is, mellyel számtalan könyvtárat telepíthetünk az alkalmazásunkhoz.

Ilyen könyvtár az Express is, ami egy viszonylag minimalista webes keretrendszer. Önmagában az Express az útvonalkezelés mellett alkalmas szerver oldalról történő kliens renderelésre és különböző sablonokat (például Pug, EJS) is támogat. Az útvonalakhoz a kérés-válasz ciklus között különböző funkciókat (middleware) rendelhetünk, amelyek hozzáférnek mind a kérés, mind a válasz objektumhoz. Ezeket a middlewareket egész alkalmazásra, vagy egy-egy útvonalra vonatkozóan is alkalmazhatjuk, illetve hibakezelést, autentikációs ellenőrzést is megvalósíthatunk velük.

A keretrendszer tudását további NPM csomagok telepítésével tudjuk bővíteni. Számos könyvtár elérhető az NPM csomagkezelővel, melyekkel a sütiket (*cookie-parser*), munkameneteket (*cookie-session*), feltölteni kívánt fájlok kezelését (*multer*) is elvégezhethetjük.

```
1  const express = require('express');
2
3  const app = express();
4  const port = process.env.PORT || 3000;
5
6  app.get('/', (req, res) => {
7      res.send('It works!');
8  });
9
10 app.listen(port, () => {
11     console.log(`A szerver elindult a(z) ${port}
12     porton!`);
13 });
```

2.8. kód. Express szerver létrehozása

Egy Express szerver létrehozásához (2.8. kód) inicializálnunk kell a projektet az *npm init* parancs segítségével, majd telepíteni kell az Express csomagot az *npm install express* paranccsal. A feltelepített csomagokat (másnéven modulok) a *require()* metódus segítségével importálhatjuk különböző fájlokban.

Az *app* nevű változó értékének adott *express()* függvényhívás inicializál egy Express alkalmazást. Ezen a változón keresztül konfigurálhatjuk az útválasztást, a middleware

---

<sup>9</sup>Node Package Manager

funkciókat, és további beállításokat végezhetünk.

Az útválasztáshoz a HTTP metódusoknak megfelelő nevű metódusokat használhatjuk (get, post, put, delete), melynek első paraméterként meg kell adnunk az elérési utat, majd második paraméterben egy úgynevezett callback metódust kell megadnunk, amely három paraméterrel rendelkezik:

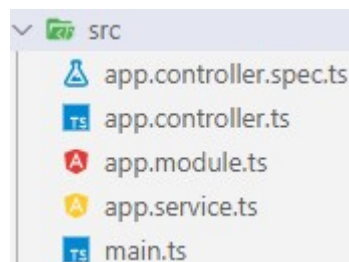
- req: tartalmazza a HTTP kérést, paramétereket, üzenetet
- res: a HTTP választ reprezentálja, amelyet az Express a kérésre válaszul visszaküld
- next: meghívásával átadjuk az irányítást a soron következő middleware-nek

Lévén, hogy az Express egy minimalista keretrendszer, csak az alapvető eszközöket biztosítja, nincsen egy előre felépített struktúra, melyre építkezve egy jól karbantartható és fejleszthető alkalmazást készíthetünk. Erre nyújt megoldást a NestJS.

### 2.2.2. NestJS[18]

A nyílt forráskódú NestJS keretrendszer az egyik legfeltörekvőbb TypeScript backend keretrendszer jelenleg, mely ötvözi az objektumorientált és funkcionális programozás elemeit. Az Express szerveroldali webes alkalmazások estén nincsen egy előre definiált architektúra, amelyre építhetjük az alkalmazásunkat, a NestJS erre fókuszál: jól skálázható és jól struktúrált szerveroldali alkalmazások írására fejlesztették ki. Fejlődését nagy mértékben befolyásolták a már megemlített frontend keretrendszerek, a React, Vue, de legnagyobb mértékben az Angular.

Projekt generálásához a keretrendszer által nyújtott Nest CLI<sup>10</sup>-t kell feltelepíteni `npm i -g @nestjs/cli` paranccsal, majd a „`nest new projekt_nev`” parancssal tudunk projektet létrehozni.



2.2. ábra. Nest projekt struktúrája

A Nest alkalmazások úgynevezett modulokból, más néven feature-ökből állnak. Az alkalmazás gyökérmodulja az `app.module.ts`, amely magába foglalja az alkalmazás összes többi modulját. Egy-egy modul több osztállyal kerül legenerálásra: saját

---

<sup>10</sup> Parancssoros felhasználói felület



kontroller osztállyal és az ahhoz tartozó egységteszt fájjal, saját service osztállyal és a modul egybeszervező module osztállyal. TypeScript keretrendszer révén a projektben több helyen is kötelező a dekorátorok használata, úgy mint modul, kontroller és service osztály létrehozása esetén is.

Minden modul osztályt kötelező a `@Module()` dekorátorral annotálni (2.9. kód). Modul generálásához a „*nest g module modul\_nev*” parancsot használhatjuk. Ebben az osztályban importálhatunk más modulokat, exportálhatunk az adott modulból a service-k közül azokat, amelyeket más modulban is el akarunk érni, illetve megadhatjuk az adott modul által használt kontroller és service osztályokat. A kontrollerek és service-k között megadott osztályokból automatikusan létrejön egy-egy példány, amelyet az objektumorientált programozásban használatos függőség befecskendezés technika segítségével használhatunk fel.

```
1 import { Module } from '@nestjs/common';
2 import { UsersController } from './users.controller';
3 import { UsersService } from './users.service';
4
5 @Module({
6     controllers: [UsersController],
7     providers: [UsersService]
8 })
9 export class UsersModule {}
```

2.9. kód. Module osztály tartalma

A „*nest g controller controller\_nev*” parancs segítségével generálhatunk egy új kontrollert. Ezeket az osztályokat a `@Controller()` dekorátorral kell annotálni, melyet felparaméterezve megadhatjuk az útvonal előtagját (prefix). Az osztály metódusait a HTTP kérés metódusainak megfelelő dekorátorokkal kell ellátni (`@Get()`, `@Post()`, `@Put`, `@Delete()`), innen tudja a keretrendszer, hogy az adott végpontot milyen metódussal lehet elérni. Ezeket a dekorátorokat paraméterezve megadhatjuk a végpontot. (2.10. kód).

```
1 import { Controller, Get } from '@nestjs/common';
2
3 @Controller('users')
4 export class UsersController {
5
6     @Get('/')
7     findAll(): string {
8         return 'Osszes felhasznalo listazasa';
9     }
10
11 }
```

2.10. kód. Kontroller osztály tartalma

Új service-t létrehozni a „*nest g service service\_nev*” paranccsal tudunk (2.11. kód). Ezeket az osztályokat a *@Injectable()* dekorátorral kell annotálni. Ez a dekorátor metaadatokkal látja el az osztályt, amely által az osztályt a keretrendszer IoC<sup>11</sup> konténere kezelni tudja. Az IoC konténer teszi lehetővé az automatikus függőség befecskendezést.

```
1 import { Injectable } from '@nestjs/common';
2
3 @Injectable()
4 export class UsersService {
5
6     findAll(): string[] {
7         return ['John Doe', 'Foo Bar'];
8     }
9
10 }
```

2.11. kód. Service osztály tartalma

Ahhoz, hogy a kontroller osztályban használni tudjuk a service metódusait injektálni kell azt a kontrollerbe a konstruktor paraméterei között (2.12. kód).

```
1 import { Controller, Get } from '@nestjs/common';
2 import { UsersService } from './users.service';
3
4 @Controller('users')
5 export class UsersController {
6     constructor(private readonly usersService:
7         UsersService) {}
8
9     @Get('/')
10     findAll(): string {
11         return this.usersService.findAll();
12     }
13 }
```

2.12. kód. Kontroller függőség befecskendezés

Az alkalmazásban történő hibák kezeléséhez készíthetünk filtereket, melyeket a kontrollerekhez kapcsolhatunk a *@UseFilters()* dekorátorral. Ezzel a megoldással ki kerülhetjük azt, hogy minden kontroller metódus törzsében try-catch blokkba kelljen helyezni a service hívását, ugyanis a kontroller globálisan kezeli, ha egy, a filterben a *@Catch()* dekorátorban definiált kivétel keletkezik.

---

<sup>11</sup> Inversion of Control

```

1 import { ExceptionFilter , Catch , ArgumentsHost } from
  '@nestjs/common';
2 import { Response } from 'express';
3 import { Error } from 'mongoose';
4
5 @Catch(Error.CastError)
6 export class CastErrorExceptionFilter implements
  ExceptionFilter {
7   catch(exception: Error.CastError , host :
    ArgumentsHost) {
8     const ctx = host.switchToHttp();
9     const response = ctx.getResponse<Response>();
10
11     response.status(500).json({
12       message: 'Hibas azonosito!'
13     });
14   }
15 }

```

2.13. kód. MongoDB ObjectId cast hibát kezelő filter

A NestJS alkalmazásokhoz készíthetünk úgynevezett Guard osztályokat is. Ezek a CanActive interfészt implementáló osztályok mindegyike egyetlen felelősséggel rendelkezik: eldöntik, hogy egyes kérések teljesítésre kerüljenek vagy sem. Felhasználásuk leginkább az autorizáció és a jogosultság alapú autentikációban merül ki.

Láthatóan a NestJS számtalan eszközt biztosít az alkalmazások fejlesztéséhez külön könyvtárak használata nélkül is, stabil struktúrát biztosít webalkalmazások készítéséhez, melyet később könnyedén bővíteni lehet.

## 2.3. Adatbázis-kezelő rendszerek

A webalkalmazások egyik legfontosabb rétege az adatbázis, ahol az alkalmazás által felhasznált adatok kerülnek tárolásra. Többféle megoldást is alkalmazhatunk, webalkalmazásonként eltérő, hogy egyes esetekben milyen rendszert érdemes alkalmazni. A mai adatbázisrendszerek közül megkülönböztetjük a relációs és nemrelációs, másnéven NoSQL rendszereket. A két rendszer merőben eltér egymástól, mindkettőnek megvan a maga előnye és hátránya, illetve az a felhasználási területe, ahol optimálisabb és kényelmesebb a használata.

### 2.3.1. A relációs adatbázisok[19][20]

A relációs adatbázisok (RDBMS) táblákban tárolják a logikailag összetartozó adatokat. A táblák sorokból és oszlopokból állnak, egy-egy sort rekordnak, míg egy oszlopot

attribútumnak nevezünk.

Működését tekintve abban tér el a NoSQL adatbázisoktól, hogy a táblákban különböző megszorításokat (például elsődleges vagy idegen kulcs), illetve tárolt eljárásokat definiálhatunk. A kulcsok által definiált kapcsolatokkal a lekérdezések során több táblából gyűjthetjük össze a kívánt adatokat. A megszorításokkal definiált táblák előnye, hogy biztosítják a táblákban az adatok integritását és konzisztenciáját, megelőzik az adatok helytelen módon történő frissítését, illetve törlését. Az integritást ezen felül az ACID séma biztosítja, mely az *Atomitás*, *Konzisztencia*, *Izoláció* és *Tartósság* szavak kezdőbetűiből áll össze.

Az adatok lekérdezéséhez az SQL<sup>12</sup> lekérdezőnyelvet használhatjuk, mely kifejezetten a relációs adatbázisokhoz lett tervezve. Segítségével képesek vagyunk többek között az adatok lekérdezésére, felvitelére, frissítésére és törlésére.

Az adatok mennyiségének növekedése révén fontos szempont a rendszer skálázhatóság, mely a relációs adatbázisok esetében felfelé, másnéven vertikálisan kivitelezhető. A vertikális skálázás során a számítási teljesítmény növeléséhez új vagy több processzorral, több memóriával, illetve gyorsabb háttértárral bővíthetjük a kiszolgálót.

Legnépszerűbb relációs adatbázis rendszerek közé tartozik az SQL Server, MySQL, PostgreSQL, Oracle.

### 2.3.2. A NoSQL adatbázisok[21][22]

A NoSQL adatbázisok felépítése nagyban különbözik a relációs adatbázisok felépítésétől. Az adatok nem táblákban tárolódnak, és nincs lehetőség elsődleges vagy idegen kulcs megszorítások létrehozására sem. Olyan alkalmazások készítése esetén, melyekhez nem szükséges a relációs adatbázisok által nyújtott merev struktúra a NoSQL rendszerek nyújthatnak megoldást.

A nemrelációs adatbázisok alapvetően az írás/olvasás műveletekre vannak kihegyezve, a sebességkülönbség a relációs adatbázisokkal szemben az azokban használt műveletek elhagyása révén érhető el (például táblák összekapcsolása). Adattárolási módjuk alapján több csoportot is megkülönböztethetünk, ezek közül a négy legnépszerűbb a dokumentumtárolók, az oszlop-orientált, a kulcs-érték pár, és a gráfadatbázisok.

A dokumentumtárolók (például *MongoDB*) többféle formátumban is tárolhatják a dokumentumokat, ezek közül a leggyakoribb a JSON, BSON<sup>13</sup>, illetve XML[23]. JSON esetében alapvetően csupán szövegek, számok, logikai értékek és tömbök tárolhatóak, a BSON ezen támogatott típusok kibővítésében segít, általa dátumokat és nyers bináris adatokat is tudunk tárolni, illetve a bináris felépítés által gyorsabb feldolgozási időt biztosít. Lehetőségünk van a dokumentumok egymásba ágyazására, így a lekérdezéskor

---

<sup>12</sup> Structured Query Language

<sup>13</sup> Binary JSON

azokat együtt megkapjuk, nincs szükség az adatok összegyűjtésére, mint az SQL esetén, ezzel is meggyorsítva az adatlekérést. Ezenfelül flexibilis struktúrát biztosít, ha idővel változnának az alkalmazás igényei, akkor annak igénye szerint egyszerűen és könnyen módosítható az adatbázis felépítése.

A relációs adatbázisok soronként tárolják a rekordokat, ezzel szemben léteznek oszlop-orientált NoSQL rendszerek amelyek analitikai szempontból kínálnak gyors megoldást (például *Cassandra*). Az alapötletük az, hogy a lekérdezések során csak abból az oszlopból kerülnek lekérdezésre az adatok, amelyekre egy-egy számítás során szükségünk van, így a számítás szempontjából szükségtelen adatok ignorálásra kerülnek.

A NoSQL rendszerek legegyszerűbb megvalósítása a kulcs-érték párokat tároló adatbázisok (például *Redis*), ahol egy-egy attribútumhoz (kulcs) egy-egy értéket párosítunk.

Habár a NoSQL jelentése a nem-SQL vagy nemrelációs adatbázis, a gráfadatbázisokban (például *OrientDB*) fontos az adatok közötti kapcsolat. Minden adat egy csúcsban tárolódik, melyek közötti kapcsolatot az őket összekötő él jelöli. Minden élt két csúcspontja (kiindulási és vég), típusa és iránya írja le. Az ilyen módon tárolt adatokat a kapcsolatok révén könnyen bejárhatjuk, nincs szükség a relációs adatbázisokban használt táblakapcsolásokra.

## 2.4. Összegzés

Ebben a fejezetben összehasonlítottam a legnépszerűbb technológiákat a kliens, a szerver és adatbázis szempontjából. A webalkalmazást a MERN (MongoDB – Express – React – NodeJS) stack-ben helyet foglaló technológiákkal fogom megvalósítani.

A React által használt JSX szintaxis rengeteg lehetőséget rejt magában a különböző újrafelhasználható komponensek készítéséhez, az alkalmazás állapotkezeléséhez pedig a jól dokumentált, számos kiegészítővel rendelkező Redux-ot használhatjuk.

Szerveroldalon a NestJS keretrendszer mellett döntöttem az általa kínált eszközök, paradigmák és a TypeScript programozási nyelv támogatása miatt.

A MongoDB egyszerű, flexibilis, remek eszköztárral rendelkezik a dokumentumok kezeléséhez, a dokumentumok egybeágyazásának lehetősége pedig praktikus megoldás a kliensoldali program számára.

## 3. fejezet

# Specifikáció

### 3.1. Feladatspecifikáció

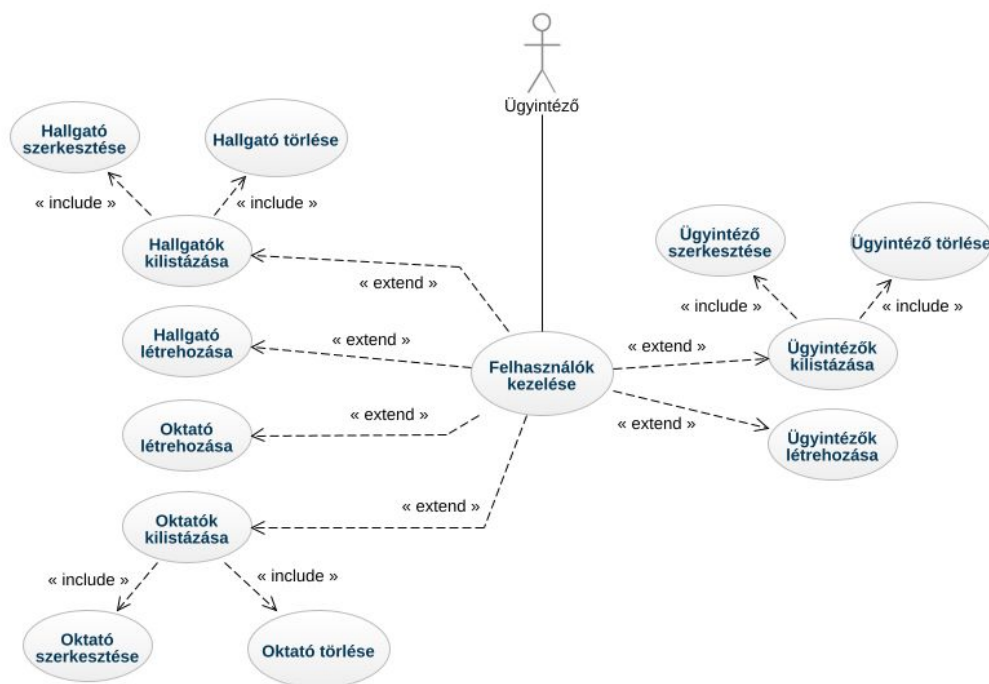
Egy olyan webalkalmazás készítését tűztem ki célul, amely elősegíti a szakdolgozatok elkészülésének ideje alatt mind az oktatók, mind a hallgatók munkáját és hatékonyabbá teszi a kommunikációt, illetve az elvégzendő feladatok nyomonkövetését.

#### 3.1.1. Felhasználói szerepkörök

Az alkalmazás felhasználóit jogosultságaik alapján három szerepkörre lehet felosztani: ügyintéző, oktató és hallgató. A felhasználók kizárólag a szerepkörük által elérhető funkciókhoz és adatokhoz férnek hozzá. Minden felhasználó képes jelszavát megváltoztatni, illetve saját adatlapját megtekinteni.

#### Ügyintézők

A webalkalmazásban az ügyintézői szerepkörrel rendelkező felhasználók kezelhetik a rendszerhez hozzáférő felhasználókat és azok adatait. Így többek között létrehozhatnak és kezelhetik a már rendszerben lévő hallgatókat, oktatókat és ügyintézőket.

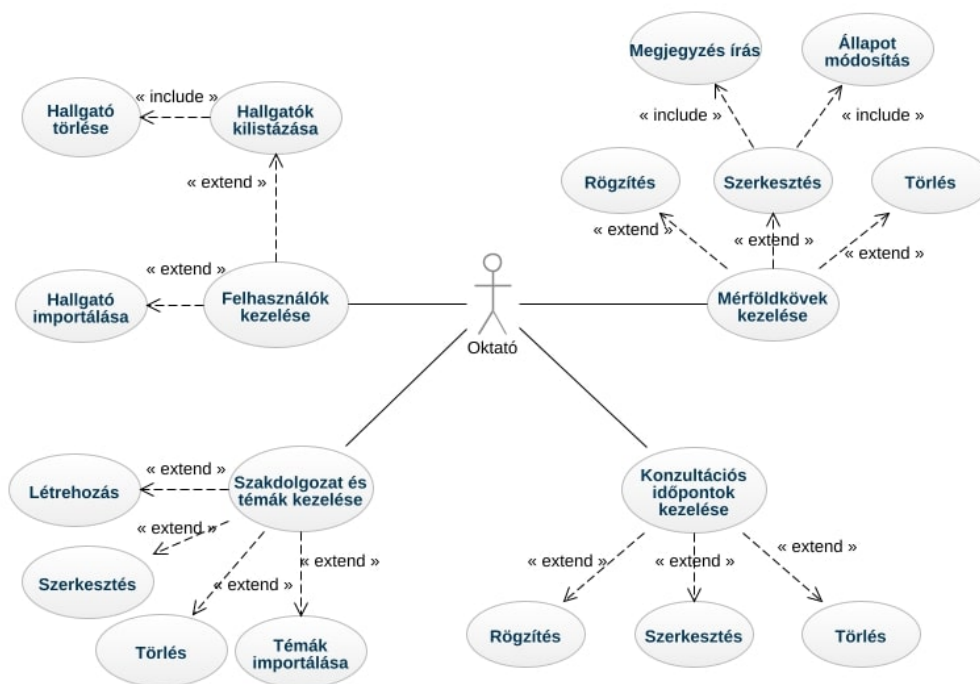


3.1. ábra. Ügyintéző használati esetei

## Oktatók

Az oktatói szerepkörrel rendelkező felhasználók szintén hozzáférnek a hallgatók adataihoz, számukra viszont nincs lehetőség a hallgatók adatainak szerkesztésére. Azért, hogy saját hallgatóikat kezelni tudják: új hallgatókat importálhatnak a rendszerbe, vagy törölhetnek már meglévő hallgatót a rendszerből. Ezen kívül képesek:

- szakdolgozati témákat létrehozni, szerkeszteni, törölni
- szakdolgozatokat rögzíteni, szerkeszteni, törölni
- konzultációs időpontokat rögzíteni, leírással ellátni, szerkeszteni, törölni
- szakdolgozathoz mérőfeladatokat rögzíteni, azokat szerkeszteni, törölni, szöveges megjegyzést fűzni (fájl csatolmánnyal is), állapotot megváltoztatni



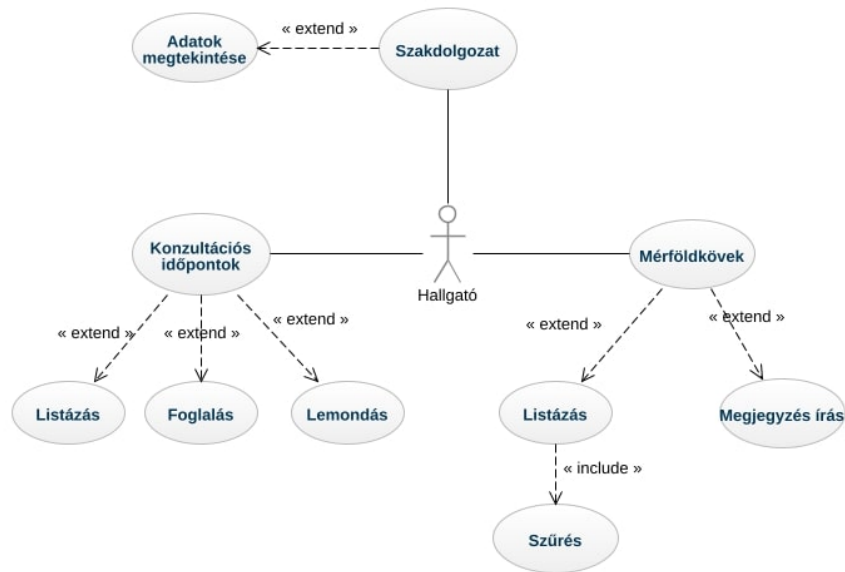
3.2. ábra. Oktató használati esetei

## Hallgatók

A hallgatói szerepkörrel rendelkező felhasználók rendelkeznek a legkevesebb jogosultsággal:

- megtekinthetik szakdolgozatuk adatlapját
- listázhatják az oktatójuk konzultációs időpontjait, lefoglalhatják azokat, és lemondhatják a már lefoglalt időpontokat
- megtekinthetik a szakdolgozatukhoz rendelt mérőldköveket
  - a mérőldkövek között szűrhetnek cím, állapot szerint, rendezhetik dátum szerint
  - megjegyzést fűzhetnek a mérőldkövekhez (fájl csatolmánnyal)





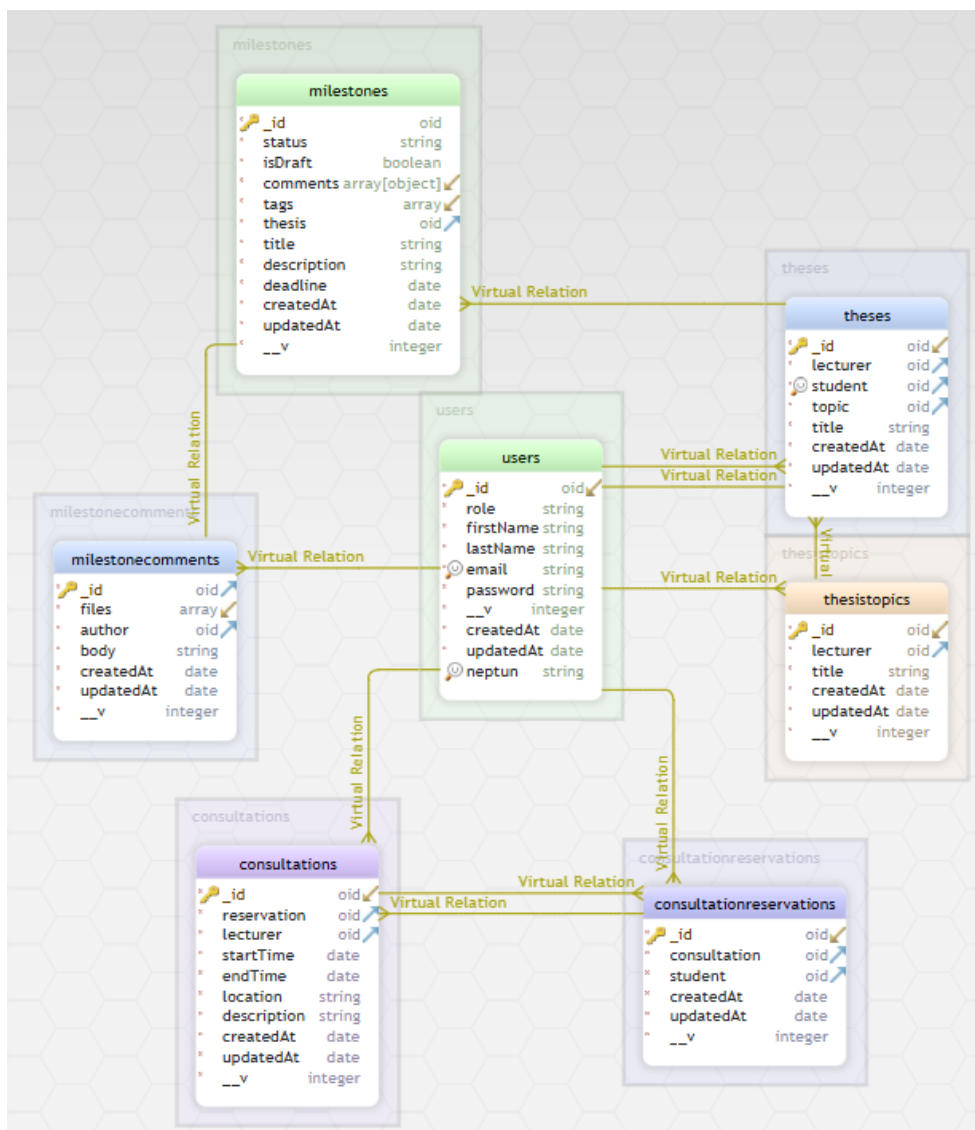
3.3. ábra. Hallgató használati esetei

## 3.2. Adatbázis-terv

Az alkalmazás által használt adatbázis megtervezése a legfontosabb lépések közé tartozik. Meg kell határozni az alkalmazásban használatos összes adatot, melyekből később a MongoDB-ben használt sémákat kell létrehozni. Ez fogja meghatározni az egyes kollekcióban tárolt adatok felépítését.

A dokumentumok közötti kapcsolatok esetén (ellentétben a relációs adatbázisokkal) nem kulcsokat hozunk létre, hanem az adott sémában egy referenciát tárolunk a kapcsolni kívánt dokumentumra, mely így egy beágyazott aldokumentuma lesz az eredeti dokumentumnak.

Az adatok validálásához a sémákban minden egyes mezőre validátor módszert használhatunk, amely megadott szempontok alapján vizsgálja az adatot. Ha az adat nem felel meg a kritériumoknak, akkor hibával leáll a dokumentum létrehozása.



3.4. ábra. A kollekcók közötti kapcsolatok

## Felhasználók

Minden felhasználóról csupán a legfontosabb adatok kerülnek eltárolásra. Ezek a vezetékek és keresztnév, NEPTUN azonosító, email cím, a belépéshez használt jelszó (hashelve) és a felhasználó jogosultsági szintje. Kikötés, hogy minden felhasználónak egyedi NEPTUN azonosítóval és email címmel kell rendelkeznie.

Mező	Típus	Hossz	Egyéb megkötés
neptun	string	6 karakter	UNIQUE
firstName	string	min. 4 karakter	-
lastName	string	min. 4 karakter	-
password	string	min. 8 karakter	-
email	string	-	UNIQUE
role	enum	-	[STUDENT, LECTURER, ADMIN]

3.1. táblázat. Felhasználó sémájának felépítése

### Szakdolgozati témák

Minden szakdolgozati téma beágyazott aldokumentumként tárolja az oktatót, aki a témát létrehozta és a téma megnevezését.

Mező	Típus	Hossz	Egyéb megkötés
lecturer	ObjectId	-	-
title	string	min. 5 karakter	-

3.2. táblázat. Szakdolgozati téma sémájának felépítése

### Szakdolgozatok

Minden szakdolgozat dokumentum tartalmazza beágyazott aldokumentumként az oktatót, a hallgatót, és a szakdolgozat témáját. Minden hallgatóhoz egyetlen szakdolgozat tartozhat, ezért kikötés, hogy a hallgató referenciájának egyedinek kell lennie.

Mező	Típus	Hossz	Egyéb megkötés
lecturer	ObjectId	-	-
student	ObjectId	-	UNIQUE
topic	ObjectId	-	-
title	string	min. 5 karakter	-

3.3. táblázat. Szakdolgozat sémájának felépítése

### Konzultációk

Minden egyes konzultációhoz eltárolásra kerül az oktató, akihez az időpont tartozik, a konzultáció kezdetének és végének ideje, a konzultáció helyszíne, és opcionálisan a leírás. Ezek mellett van egy mező, amely alapértelmezetten üres, viszont ha foglalás érkezik az időpontra, akkor a foglalás referenciája elmentésre kerül ebbe a mezőbe.

### Konzultáció foglalások

A foglalások dokumentumai tartalmazzák beágyazott aldokumentumként a konzultáció és a foglalo hallgató referenciáját.

Mező	Típus	Hossz	Egyéb megkötés
lecturer	ObjectId	-	-
startTime	Date	-	-
endTime	Date	-	-
location	string	-	-
description	string	-	-
reservation	ObjectId	-	-

3.4. táblázat. Konzultáció sémájának felépítése

Mező	Típus	Hossz	Egyéb megkötés
consultation	ObjectId	-	-
student	ObjectId	-	-

3.5. táblázat. Konzultáció foglalás sémájának felépítése

### Mérföldkövek és mérföldkő megjegyzések

A mérföldkövek dokumentumai magában foglalják a szakdolgozat referenciáját, a mérföldkő címét és leírását, a feladat határidejét, státuszát, hogy piszkozatként van-e elmentve az adott feladat, a mérföldkőhöz tartozó megjegyzések referenciáit és az opcionálisan megadható címkék listáját.

A megjegyzések dokumentumai referenciaként tárolják annak íróját, a szöveges tartalmát és az esetlegesen csatolt fájl(ok) relatív elérési útjait.

Mező	Típus	Hossz	Egyéb megkötés
thesis	ObjectId	-	-
title	string	min. 3 karakter	-
description	string	min. 5 karakter	-
deadline	Date	-	-
status	enum	-	[PENDING,ACCEPTED,REJECTED]
isDraft	boolean	-	-
comments	[ObjectId]	-	-
tags	string[]	-	-

3.6. táblázat. Mérföldkő sémájának felépítése

Mező	Típus	Hossz	Egyéb megkötés
author	ObjectId	-	-
body	string	min. 3 karakter	-
files	string[]	-	-

3.7. táblázat. Mérföldkő megjegyzés sémájának felépítése

### 3.3. Autentikáció és autorizáció

A webalkalmazás fontos és egyben legkritikusabb pontja az autentikáció. Számtalan módon lehetséges a felhasználó azonosítása, mely történhet egyedileg fejlesztett autentikációs folyamattal, de léteznek erre is előre elkészített csomagok amelyeket használhatunk. Egyik ilyen csomag a Passport.js<sup>1</sup>, melyben 500-nál is több autentikációs stratégia közül választhatunk, mint például: Google OAuth, Facebook, Auth0 vagy JWT<sup>2</sup>.

A webalkalmazás token alapú autentikációt fog használni, nevezetesen JSON Web Token-t. A bejelentkezés során helyes email cím és jelszó párosítást megadva egy, a szerver által aláírt, és csak bizonyos időtartamig érvényes token-t kapunk válaszként, melyet minden szerver felé irányuló kérés során csatolunk a kéréshez. Minden kérés beérkezése során a szerver ellenőrzi a tokent, és ha az érvényes, akkor teljesíti a kérést.

Ahogy a 2.2.2. alfejezetben is említésre került, hogy a keretrendszerben készíthetünk Guard osztályokat, amelyeket használhatunk az autentikáció során. Az implementációhoz szükséges *@nestjs/passport* csomag rendelkezik egy beépített AuthGuard nevű Guard-al, melynek paraméterben megadhatjuk az autentikáció típusát (jelen esetben jwt). Mindegyik kontroller metóduson, amelyeket csak autentikált felhasználók érhetnek el a *@UseGuards()* dekorátorral alkalmaznunk kell az AuthGuard-ot, ezzel biztosítva, hogy csak és kizárólag érvényes tokennel rendelkező kérések kerüljenek kiszolgálásra.

Ezen felül egyes kérések esetén szükséges vizsgálni a kérést indító felhasználó jogosultsági szintjét is. Ehhez egy saját dekorátorra (*@Roles()*) és Guard-ra van szükség (*RoleGuard*). A dekorátor használatával minden kontroller metódus esetén egyesével megszabhatjuk, hogy az adott erőforrást milyen jogosultsági szinttel rendelkező felhasználók érhessek el. A dekorátor egy saját metaadatban tárolja a megadott jogosultsági szint(ke)t (3.1. kód).

```
1 import { SetMetadata } from '@nestjs/common';  
2  
3 export const Roles = (...roles: string[]) =>  
    SetMetadata('roles', roles);
```

3.1. kód. A jogosultsági szintet tároló metaadat dekorátor

A RoleGuard ebből a metaadatból kiolvastva dönti el, hogy a kérést indító felhasználó jogosult-e a kért erőforrás elérésére (3.2. kód).

```
1 import { CanActivate, ExecutionContext, Injectable }  
    from '@nestjs/common';  
2 import { Reflector } from '@nestjs/core';  
3 import { Observable } from 'rxjs';  
4
```

<sup>1</sup><http://www.passportjs.org/>

<sup>2</sup>JSON Web Token

```

5  @Injectable()
6  export class RoleGuard implements CanActivate {
7      constructor(private reflector: Reflector) {}
8
9      canActivate(context: ExecutionContext): boolean |
        Promise<boolean> | Observable<boolean> {
10         const roles = this.reflector.get<string[]>('roles
            ', context.getHandler());
11         if(!roles) {
12             return true;
13         }
14         const request = context.switchToHttp().getRequest
            ();
15         const user = request.user;
16         const hasRole = () =>
17             !!roles.find(role => role === user.role);
18
19         return user && user.role && hasRole();
20     }
21 }

```

### 3.2. kód. A RoleGuard felépítése

A szerveroldalon kívül a kliensoldali programban is szükséges az erőforrások elérhetőségét jogosultsági szint alapján vizsgálni. Ehhez saját komponenst alkalmazhatunk, amely a store-ban eltárolt bejelentkezett felhasználó jogosultsági szintjét ellenőrzi.

Ha egy bizonyos útvonalat jogosultsági szinttől függetlenül, viszont csak és kizárólag autentikált felhasználók tekinthetnek meg, úgy elegendő ellenőrizni a store-ban, hogy bejelentkezett-e a felhasználó (3.3. kód). Ha be van jelentkezve, akkor megjelenítjük a komponenst annak paramétereivel, ha nincs akkor átirányítjuk a felhasználót.

```

1  import React from 'react';
2  import { Redirect, Route } from 'react-router-dom';
3  import { useSelector } from "react-redux";
4
5  const ProtectedRoute = ({
6      component: Component,
7      ...rest
8  }) => {
9      let isAuthenticated = useSelector(state => state.
        auth.isAuthenticated);
10
11      if(!isAuthenticated) {
12          return <Redirect to="/user" />;
13      }
14      return (
15          <Route

```

```

16         {... rest}
17         render={props =>
18             <Component {... props} />
19         }
20     />
21 );
22 }
23
24 export default ProtectedRoute;

```

### 3.3. kód. ProtectedRoute felépítése

Amennyiben az útvonalat autentikált, és bizonyos jogosultsági szinttel rendelkező felhasználók érhetik csak el, úgy a bejelentkezés vizsgálata mellett ellenőrizni kell a belépett felhasználó jogosultsági szintjét. A 3.4. kódban látható komponens csak azon felhasználóknak engedélyezi az útvonal elérést, amelyek oktatói jogosultsággal rendelkeznek.

```

1  import React from 'react';
2  import { Redirect, Route } from 'react-router-dom';
3  import { useSelector } from "react-redux";
4
5  const LecturerRoute = ({
6      component: Component,
7      ...rest
8  }) => {
9      let isAuthenticated = useSelector(state => state.auth.isAuthenticated);
10     let user = useSelector(state => state.auth.user);
11     if(!isAuthenticated || !user || (user && user.role !== 'LECTURER')) {
12         return <Redirect to="/user" />;
13     }
14     return(
15         <Route
16             {... rest}
17             render={props =>
18                 <Component {... props} />
19             }
20         />
21     );
22 }
23
24 export default LecturerRoute;

```

### 3.4. kód. LecturerRoute felépítése

### 3.4. Kliensoldali alkalmazás felépítése

Révén, hogy az alkalmazásnak többféle jogosultsági szinttel rendelkező felhasználókat kell kiszolgálnia, így a kliensoldali React alkalmazást két kisebb alkalmazásra bontom föl. Egyik az ügyintézői, a másik pedig az oktatói és hallgatói modul. Ennek előnye, hogy a két rendszer elkülönül egymástól, mindkettő saját útválasztással és Redux állapotkezeléssel rendelkezik. Ezáltal egyszerűbb a külön-külön modulokban való fejlesztés, nem lesz túlságosan összetett és bonyolult a mögöttes Redux állapot felépítése.

A React alkalmazás fő App komponensének szerepe mindössze annyi, hogy elkülönítse az ügyintézői, oktatói, hallgatói, és a hibaútvonalakat (például „404 - Az oldal nem található”). Az ezeken belüli navigálás a továbbiakban már az adott modul feladata.

A mindkét modul által használt komponensek, Redux által használt action és reducer fájlok a projekt gyökérmappájában található közös mappában helyezkednek el. Innen mindkét modulból elérhetőek, újrafelhasználhatóak, ezáltal elkerülve a fölösleges kódismétléseket.

A Redux store minden oldalfrissítéssel kiürül, ez az autentikáció szempontjából hátrányos, ugyanis minden oldalfrissítéssel újra és újra le kellene kérni a szervertől a felhasználó tokenje alapján a felhasználó adatait. Hogy ezt a problémát kiküszöböljem a *redux-persist*<sup>3</sup> csomagot fogom használni. Ez egy olyan köztes funkció szerepét tölti be, amely megadott reducer-ek esetében a store-ban tárolt adatokról egy másolatot tárol a böngésző lokális tárolójában (localStorage vagy sessionStorage). Oldalfrissítés esetén pedig innen visszatöltődnek az adatok a store-ba.

---

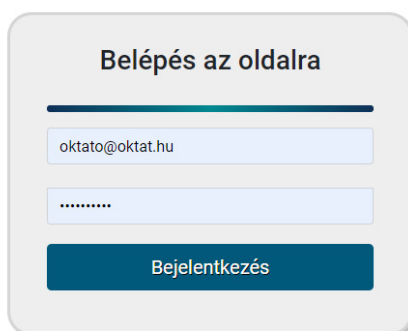
<sup>3</sup><https://github.com/rt2zz/redux-persist>



## 4. fejezet

# Fejlesztői dokumentáció

A 3. fejezetben leírt tervek alapján a webalkalmazás a belépett felhasználó jogosultsági szintjének megfelelő felhasználói interfészt biztosít. Az alkalmazás háromféle jogosultsági szint kezelésére képes: ügyintéző, oktató és hallgató. A felületek hozzáférése előtt a felhasználónak be kell jelentkeznie a fiókjához tartozó email címmel és jelszavával.



The login form is titled "Belépés az oldalra". It contains two input fields: the first is for the email address, with the placeholder text "oktato@oktat.hu"; the second is for the password, represented by a series of dots. Below the input fields is a blue button labeled "Bejelentkezés".

4.1. ábra. Felhasználói belépő felület




Two screenshots of the login form are shown side-by-side. The left screenshot shows an error message in a red box: "Nincs felhasználó ilyen email címmel!". The right screenshot shows an error message in a red box: "Sikertelen bejelentkezés!". Both screenshots show the same input fields and "Bejelentkezés" button as in the previous figure.

4.2. ábra. Sikertelen belépés esetén hibaüzenet kerül kiírásra

Sikeres bejelentkezést követően a menüsor jobb szélén megjelenő „Teljes név (Jogosultság)” lenyíló menüre kattintva megtekintheti a felhasználó a profil beállításait

(profil adatlap és jelszóváltoztatás), illetve kijelentkezhet a felületről, amennyiben nem szeretné azt tovább használni.

## Felhasználó adatlapja

 <b>I Am Admin</b>	<table><tr><td>Vezetéknév</td><td>I Am</td></tr><tr><td>Keresztnev</td><td>Admin</td></tr><tr><td>NEPTUN</td><td>-</td></tr><tr><td>Email</td><td>admin@admin.com</td></tr><tr><td>Jogosultság</td><td>Ügyintéző</td></tr><tr><td>Létrehozás ideje</td><td>2021.01.28 11:15</td></tr><tr><td>Utolsó frissítés ideje</td><td>2021.03.28 17:41</td></tr></table>	Vezetéknév	I Am	Keresztnev	Admin	NEPTUN	-	Email	admin@admin.com	Jogosultság	Ügyintéző	Létrehozás ideje	2021.01.28 11:15	Utolsó frissítés ideje	2021.03.28 17:41
Vezetéknév	I Am														
Keresztnev	Admin														
NEPTUN	-														
Email	admin@admin.com														
Jogosultság	Ügyintéző														
Létrehozás ideje	2021.01.28 11:15														
Utolsó frissítés ideje	2021.03.28 17:41														

## Jelszó megváltoztatása

☐ Jelszavak mutatása

Jelenlegi jelszó\*:

Új jelszó\*:

Legalább 8 karakter.

Új jelszó megerősítése\*:

Mentés







4.3. ábra. Felhasználó adatlap és jelszómódosítás

### 4.1. Ügyintézői felület

A rendszer használatára jogosult felhasználókat az ügyintézők saját felületükön tudják kezelni. A sikeres bejelentkezés után alapértelmezetten a hallgatók kerülnek kilistázásra táblázatos formában, a menüsorban elérhető „Oktatók” és „Ügyintézők” menüpontokban lehetőség van az ezekkel a jogosultságokkal rendelkező felhasználók menedzselésére

is.

A 4.4. ábrán látható táblázat a szűrésen kívül két további funkcióval rendelkezik: az adott felhasználó adatlapjának megjelenítésére szolgáló kék ikonnal, illetve a felhasználó törlésére szolgáló piros ikonnal. A törlés ikonra kattintva egy felugró ablakban kell megerősítenie az ügyintézőnek a felhasználó törlésének szándékát.

Hallgatók kezelése			
Új felvétele			+ Importálás
Név alapján	Keresendő érték		
Teljes név	NEPTUN kód	Email	Műveletek
Vereczki Bálint Zoltán	EPQF7Q	vereczkibalint@gmail.com	 
Teszt Pista	TESZTP	teszt@pista.hu	 
Tóth János	TTHJNS	tthjns@gmail.com	 

4.4. ábra. Hallgatók listája

A táblázatban szereplő felhasználók szűrése a lenyíló menüben található szempontok (név, email, NEPTUN azonosító) alapján lehetséges, a „Keresendő érték” mezőbe írt érték alapján. A webalkalmazás többféle módot is biztosít az ügyintézők számára, hogy új felhasználókat vigyenek fel a rendszerbe: az „Új felvétele” feliratú gombra kattintva a 4.5. ábrán látható űrlap kitöltésével egy felhasználót, az „Importálás” feliratú gombra kattintva CSV formátumú adatforrásból képes egyszerre több felhasználót felvinni a rendszerbe.

Vissza

Új felhasználó felvétele

☒ Hallgató létrehozása ☐ Oktató létrehozása ☐ Ügyintéző létrehozása

NEPTUN azonosító

6 karakter, betűk és számok.

Vezetéknév

Legalább 4 karakter.

Keresztnév

Legalább 4 karakter.

Email cím

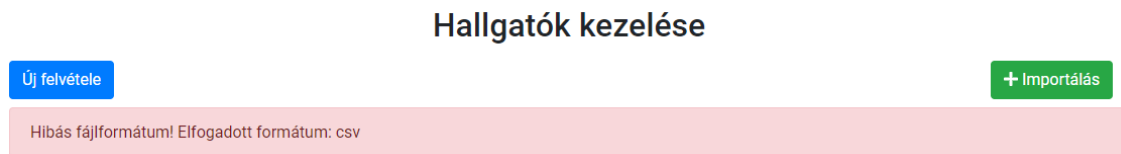
Érvényes email cím. (ajánlott: @gmail.com)

Mentés

4.5. ábra. Új felhasználó felvitele űrlap segítségével

Felhasználók importálására kizárólag pontosvesszővel tagolt, maximum 1 megabájt

méretű CSV formátumból van lehetőség, hiba esetén a 4.6. ábrán látható hibaüzenet jelenik meg.




4.6. ábra. Sikertelen felhasználó importálás

A táblázatban a kék ikonra kattintva megnyílik a hallgató adatlapja és megjelenik minden fontos információ az adott felhasználóról, úgy mint a neve, NEPTUN azonosítója, email címe, jogosultsági szintje, a profil létrehozásának és legutolsó frissítésének pontos ideje. Az adatlap fölött jobb oldalon található „Szerkesztés” feliratú gomb megnyomásával átvált az oldal szerkesztői módba, a szerkeszthető mezők beviteli mezővé alakulnak át. A szerkesztést az adatlap fölött középen található „Mentés” feliratú gombbal lehet véglegesíteni, ha nem kívánjuk menteni az adatokat, akkor a szerkesztést a „Mégsem” feliratú gombra kattintva fejezhetjük be.

4.7. ábra. Felhasználó szerkesztése

A szerkesztés során felmerülő hibákról (mint a már regisztrált NEPTUN kód és email cím) a Mentés gombra kattintás után kap visszajelzést a felhasználó (4.8. ábra), ilyenkor a beviteli mező piros körvonallal jelenik meg, benne egy felkiáltójel ikonnal, a mező alatt pedig piros felirattal jelenik meg a mezőre vonatkozó hibaüzenet.

Vissza
Mentés
Mégsem



Vereczki Bálint Zoltán

Azonosító	5fd882b7eeee2e2affd62f00
Vezetéknév	Vereczki
Keresztnév	Bálint Zoltán
NEPTUN	TESZTP <span style="color: red;">❗</span> <small>A(z) TESZTP már használatban van!</small>
Email	teszt@pista.hu <span style="color: red;">❗</span> <small>A(z) teszt@pista.hu már használatban van!</small>
Jogosultság	Hallgató
Létrehozás ideje	2020. 12. 15. 10:32:40
Utolsó frissítés ideje	2021. 03. 07. 19:41:42

4.8. ábra. Szerkesztés során felmerülő hibák

## 4.2. Oktatói felület

### 4.2.1. Szakdolgozatok és témák

Sikeres bejelentkezést követően alapértelmezetten a „*Szakdolgozatok kezelése*” oldal fogadja az oktatót. Amennyiben még az adott oktatóhoz nem került rögzítésre szakdolgozat akkor a „*Nincsen szakdolgozat az adatbázisban*” hibaüzenet jelenik meg (4.9. ábra), ellenkező esetben pedig táblázatos formában az összes szakdolgozat (4.12. ábra). A táblázat fölött található szűrők segítségével lehet keresni a szakdolgozatok között, hallgató neve, téma és a szakdolgozat címe alapján.

#### Szakdolgozatok kezelése

Nincsen szakdolgozat az adatbázisban!

4.9. ábra. Hibaüzenet, ha nincsen szakdolgozat rögzítve

Szakdolgozat rögzítése előtt rögzíteni kell a rendszerben a szakdolgozati témákat, melyet a „*Szakdolgozati témák*” menüpontja alatt lehet megtenni többféle módon (4.10. ábra). Ha csak egy-egy téma felvitelére van szükség, akkor a bal oldali beviteli mezőbe begépelve felvihető, több téma esetén pedig a jobb oldali „*Importálás*” gombra kattintva pontosvesszővel tagolt CSV fájlból lehet importálni. A téma szerkesztésére és törlésére az adott téma sorának végén található ikonokra kattintva van lehetőség.

### Szakdolgozati témák kezelése

Téma megnevezése	Létrehozás dátuma	Műveletek
Alkalmazásfejlesztés web alapokon	2021.03.22 18:10	<input type="button" value="✎"/> <input type="button" value="🗑️"/>

4.10. ábra. Szakdolgozati témák kezelése

Szakdolgozat rögzítésére a „*Szakdolgozat*” lenyíló menüben található „*Új létrehozása*” menüpont alatt van lehetőség (4.11. ábra). Ha a rendszerben még nem szerepelnek hallgatók vagy az oktatóhoz tartozó szakdolgozati témák, úgy a beviteli mezők letiltásra kerülnek. Ellenkező esetben a mezőbe kattintva megjelennek az elérhető értékek, melyek között a mezőbe való gépeléssel lehet keresni. Ha a kiválasztott hallgatóhoz már hozzá van rendelve egy szakdolgozat, akkor a beviteli mező alatt „*Ehhez a hallgatóhoz már tartozik szakdolgozat!*” hibaüzenet kerül megjelenítésre.

## Új szakdolgozat felvétele

Hallgató

Szakdolgozat témája

Szakdolgozat címe

Legalább 5 karakter.

4.11. ábra. Szakdolgozat létrehozása

### Szakdolgozatok kezelése

Hallgató neve	Téma	Cím	Műveletek
Vereczki Bálint Zoltán	Web alapú alkalmazásfejlesztés	Webfejlesztés modern technológiákkal	<input type="button" value="👁️"/> <input type="button" value="🗑️"/>

4.12. ábra. Szakdolgozatok táblázatos formában

A táblázat két művelet elvégzésére nyújt opciót: a szakdolgozat adatlapjának és szerkesztő felületének megnyitására (4.13. ábra), és a szakdolgozat törlésére, amely egy felugró ablakban történő megerősítés után történik meg.

Viszsa	Szerkesztés
<b>Webfejlesztés modern technológiákkal</b>	
Hallgató	Vereczki Bálint Zoltán Email: <a href="mailto:vereczkibalint@gmail.com">vereczkibalint@gmail.com</a>
Szakdolgozat témája	Web alapú alkalmazásfejlesztés
Szakdolgozat címe	Webfejlesztés modern technológiákkal
Létrehozás ideje	2021.03.22 17:31
Utolsó frissítés ideje	2021.03.22 17:31

4.13. ábra. Szakdolgozat adatlapja

Az adatlapon a jobb felső sarokban található „Szerkesztés” feliratú gombra kattintva átvált az oldal szerkesztői módba (4.14. ábra), ahol lehetőség van a szakdolgozat hallgatóját, témáját és címét megváltoztatni.

Viszsa	X Szerkesztés elvetése
<b>Webfejlesztés modern technológiákkal</b>	
Hallgató	Vereczki Bálint Zoltán (EPQF7Q)
Szakdolgozat témája	Web alapú alkalmazásfejlesztés
Szakdolgozat címe	Webfejlesztés modern technológiákkal
	Legalább 5 karakter.
Mentés	

4.14. ábra. Szakdolgozat szerkesztése

## 4.2.2. Hallgatók









A „Hallgatók” menüpont alatt elérhető felületen tudják az oktatók menedzselni a rendszerbe rögzített hallgatókat. Ha valamilyen adatra van szükség, akkor a felhasználó adatlapját megtekinthetik, illetve törölhetik is az adott hallgató profilját a rendszerből. Lehetőség van név, NEPTUN azonosító és email cím szerint szűrni az adatokat, illetve pontosvesszővel tagolt CSV fájlból új hallgatókat felvinni a rendszerbe.

+ Importálás

## Hallgatók kezelése

Név alapján

Keresendő érték

Teljes név	NEPTUN kód	Email	Műveletek
Kerekes Gábor	KRKSGB	kerekes.gabor@gmail.com	 
Pénzes József	PNZSJZ	penzes.jozsef@gmail.com	 
Álmos Elemér	LMSLMR	almos.elemer@gmail.com	 
Vereckzi Bálint Zoltán	EPQF7Q	vereczkibalint@gmail.com	 

4.15. ábra. Hallgatók kezelése

### 4.2.3. Konzultációk

A „Konzultációk” menüpont alatt kezelhetők a kiírt konzultációk, és megtekinthető, hogy egyes időpontokat mely hallgatók foglalták le.

Amennyiben az oktató még nem rögzített konzultációs időpontot a rendszerbe, úgy a „Nem található konzultáció!” hibaüzenet jelenik meg az oldalon.

A „Konzultációk” lenyíló menüben az „Új létrehozása” menüpont alatt hozható létre egy új konzultációs időpont (4.16. ábra). A létrehozás során kötelező a konzultációs időpont kezdetét és végét, illetve a helyszínét megadni. A „Leírás” mező opcionálisan kitölthető, melyben kiegészítő információk adhatóak meg az eseményhez (például online konzultáció esetén Zoom meghívó / konferencia csatlakozáshoz szükséges link).

Vissza

### Új konzultáció felvétele



Konzultáció kezdete

Konzultáció vége

Konzultáció helyszíne

Leírás (opcionális)

**B** *I* U ~~S~~ `{ }`  $x^2$   $x_2$  16 Font

Mentés

4.16. ábra. Új konzultációs időpont létrehozása

A konzultációk listázása oldalon megtekinthető az összes rögzített időpont azok



leírásával, és elérhetőségének állapotával (elérhető állapot esetén egy zöld pipa ikonnal, foglalt állapotban pedig a foglaltó hallgató nevével és NEPTUN azonosítójával, 4.17. ábra). Ezen kívül a sorok végén található ikonokkal szerkeszthetők és törölhetők az időpontok.

**Saját konzultációk kezelése**

Konzultáció kezdete	Konzultáció vége	Konzultáció helyszíne	Elérhetőség	Leírás	Műveletek
2021.04.30. 16:47	2021.04.30. 17:17	online	Hallgató: Vereczki Bálint Zoltán(EPOF7Q)		
2021.05.03. 10:30	2021.05.03. 11:00	online (zoom)			

4.17. ábra. Konzultációk listája

#### 4.2.4. Mérföldkövek

A „*Mérföldkövek*” menüpont alatt kezelhető minden szakdolgozathoz rendelt mérföldkő. A listázó oldalon a „*Szakdolgozat*” lenyíló menüben választható ki, hogy melyik szakdolgozathoz tartozó mérföldkövek kerüljenek kilistázásra. Amennyiben az adott szakdolgozathoz nem tartozik mérföldkő, úgy a „*A megadott kritériumokkal nem található mérföldkő!*” hibaüzenet kerül kiírásra (4.18. ábra).

**Mérföldkövek kezelése**

Szakdolgozat:

Vereczki Bálint Zoltán - Webfejlesztés modern technológiákkal

Cím: Mérföldkő címe      Dátum: Növekvő      Állapot: Összes

A megadott kritériumokkal nem található mérföldkő!

4.18. ábra. Hibaüzenet, amennyiben nincs rögzített mérföldkő

A kilistázott mérföldkövek (4.19. ábra) rendezése és az azok közötti szűrés a szakdolgozat választó menü alatti opciókkal tehető meg. Az adatok szűrhetők cím alapján, és állapotuk alapján (Folyamatban, Elfogadott, Elutasított), illetve határidejük alapján rendezhetők növekvő és csökkenő sorrendben (alapértelmezetten csökkenő sorrendben jelennek meg).

Azon mérföldkövek határideje, melyek még nem jártak le, vastag, piros betűszínnel jelennek meg. A feladat állapota az annak megfelelő ikonnal jelenik meg (Folyamatban esetén homokóra, Elfogadott esetén zöld pipa, Elutasított esetén piros x ikonnal).

A sor végén található kék szem ikonnal nyitható meg a mérföldkő adatlapja, ahol elolvasható a feladat részletes leírása, módosítható a feladat állapota, illetve hozzászólások formájában kommunikálhat az oktató a hallgatóval, ahol csatolmány formájában segédanyagokat, dokumentumokat és egyéb fájlokat oszthatnak meg egymással.

## Mérföldkövek kezelése

Szakdolgozat:

Vereczki Bálint Zoltán - Webfejlesztés modern technológiákkal

Cím

Dátum

Állapot

Mérföldkő címe

Növekvő

Összes

Mérföldkő címe	Határidő	Állapot	Műveletek
Autentikáció implementálása	2021.04.30. 23:59 (34 nap)		

4.19. ábra. Mérföldkövek kilistázva

A mérföldköő adatlapjának (4.20. ábra) jobb oldalán elhelyezkedő „*Szerkesztés*” feliratú gombra kattintva váltható az oldal szerkesztő módba. Ekkor az oldal aljáról eltűnik a „*Megjegyzések*” szekció, a részletes leírás helyén pedig megjelenik a szerkesztési űrlap, ahol megváltoztatható a feladat címe, határideje, leírása és a címkéi.

A szerkesztés gomb alatt változtatható meg a feladat állapota, a háttérszínnel kijelölt gomb mutatja a feladat jelenlegi állapotát. Állapotváltoztatás esetén egy új komment jön létre a „*Megjegyzések*” szekcióban, amely rögzíti az új állapotot, és leolvasható a komment dátumáról a változtatás időpontja.

Vissza

# Autentikáció implementálása

Határidő: 2021.04.30. 23:59FolyamatbanLétrehozva: 2021.03.27. 20:14

## Leírás:

Kedves Hallgató!

A fent megadott határidőig az alábbi feladatok implementálása szükséges:

- JWT autentikáció backend

Köszönettel,

Oktató István

Utolsó frissítés: 2021.03.27. 20:14

Címkék:

backendautentikációjwt

Szerkesztés

Elfogadás

Folyamatban

Elutasítás

## Megjegyzések

B*I*U&{}x²x₂16Font

Csatolmányok

Elfogadott formátumok: png, jpg, pdf, docx, xlsx

Fájlok kiválasztásaNincs fájl kiválasztva

Küldés

Nincsenek hozzászólások!

4.20. ábra. Egy mérföldkő adatlapja

Mérföldkövet létrehozni a „*Mérföldkövek*” lenyíló menüben található „*Új létrehozása*” menüpont alatt elérhető űrlap kitöltésével lehet (4.21. ábra). A „Szakdolgozat” lenyíló menüből kiválasztható, hogy mely hallgató mely szakdolgozatához szeretnének a mérföldkövet rögzíteni. Ha még nincsen az oktatóhoz rendelve szakdolgozat, akkor a menü letiltásra kerül (nem lesz kattintható), és a „*Nincs elérhető szakdolgozat!*” felirat jelenik meg a menüben.

A mérföldkő címén és határidején kívül opcionálisan megadhatóak kulcsszavak, másnéven címkék is, melyekhez segítséget a „*Címkék*” felirat melletti kérdőjel ikonra kattintva lehet elolvasni. Ezek a kulcsszavak segítenek a hallgatónak az adott témában való keresésben. Címke rögzítése az Enter billentyű lenyomása után történik meg, törlésre pedig használható a címke mellett megjelenő „X” ikon, vagy kétszer egymás után leütött Backspace billentyű is.

A rendszer lehetőséget nyújt a mérföldkövek piszkozatként való mentésére is, amely a „*Mentés piszkozatként*” gombra kattintva tehető meg. Ekkor a piszkozat megtekinthető, a továbbiakban szerkeszthető és törölhető is az oktató által, viszont a hallgató számára nem jelenik meg. Az oktatói felületen a piszkozatként mentett mérföldkövek neve pirossal jelenik meg, zárójelben a „piszkozat” szóval megjelölve. Ahhoz, hogy egy piszkozatnak jelölt mérföldkő megjelenjen a hallgató számára is a mérföldkő szerkesztői módjában a „*Közzététel*” gombra kattintva kell elmenteni a feladatot.

4.21. ábra. Mérföldkő létrehozása

## 4.3. Hallgatói felület

### 4.3.1. Szakdolgozat

A hallgató bejelentkezése után alapértelmezetten a „*Szakdolgozat*” oldal jelenik meg. Amennyiben az adott hallgatóhoz még nincsen szakdolgozat rendelve úgy a „*Nincsen szakdolgozat az Ön felhasználójához rendelve!*” hibaüzenet jelenik meg (4.22. ábra).

Nincsen szakdolgozat az Ön felhasználójához rendelve!

4.22. ábra. A hallgatóhoz nincsen szakdolgozat rendelve

Amennyiben a hallgatóhoz már van szakdolgozat rendelve, úgy annak adatlapja jelenik meg (4.23. ábra). Az adatlapon az oktató nevén és email elérhetőségén kívül leolvasható a szakdolgozat témája, címe, a mérföldkövek száma státuszokkénti lebontásban, és a szakdolgozat rögzítésének, illetve legutóbbi frissítésének pontos ideje.

Szakdolgozat adatlapja	
Oktató	Oktató István Email: <a href="mailto:oktato@oktat.hu">oktato@oktat.hu</a>
Szakdolgozat témája	Alkalmazásfejlesztés web alapokon
Szakdolgozat címe	Webfejlesztés modern technológiákkal
Mérföldkövek száma	<div><div> Összesen 0 darab</div><div> Folyamatban 0 darab</div><div> Elfogadott 0 darab</div><div> Elutasított 0 darab</div></div>
Létrehozás ideje	2021.03.29 20:29
Utolsó frissítés ideje	2021.03.29 20:29

4.23. ábra. Szakdolgozat adatlapja

### 4.3.2. Konzultációk

A „*Konzultációk*” menüpont alatt lekérdezhető a hallgatók által a szakdolgozati oktatójuk összes konzultációs időpontja, illetve megtekinthető azok elérhetősége.

Ha a hallgatóhoz nincsen még szakdolgozat rögzítve, akkor a „*Konzultációk lekérése nem lehetséges!*” hibaüzenet jelenik meg az oldalon (4.24. ábra). Ha van a hallgatóhoz szakdolgozat rendelve, de az oktátónak nincsen konzultációs időpontja a rendszerbe rögzítve, akkor a „*A megadott feltételekkel nincs elérhető konzultáció!*” hibaüzenet jelenik meg.

A konzultációs időpontok megjelenítési módjára két opció közül lehet választani: az összes időpont kilistázása, illetve a hallgató által lefoglalt időpontok megjelenítése

(4.25. ábra). Ha a hallgató még nem foglalt le egyetlen időpontot sem, akkor a „A megadott feltételekkel nincs elérhető konzultáció!” hibaüzenet jelenik meg az oldalon.

Konzultációk lekérése nem lehetséges!

4.24. ábra. Hibaüzenet, ha a hallgatóhoz nem tartozik szakdolgozat

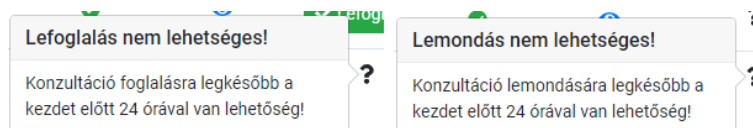
Elérhető konzultációk						
Megjelenítési mód:	Összes mutatása					
Oktató	Konzultáció kezdete	Konzultáció vége	Konzultáció helyszíne	Elérhetőség	Leírás	Műveletek
Oktató István	2021.04.19. 10:30	2021.04.19. 11:00	online (zoom)	✓	👁	✓ Lefoglalás
Oktató István	2021.04.19. 13:00	2021.04.19. 13:30	online (zoom)	✓	👁	✓ Lefoglalás

4.25. ábra. Konzultációs időpontok listája

Minden konzultációhoz megjelenik a kezdet és befejezés időpontja, a konzultáció helyszíne, és az időpont elérhetősége. Ha az időpont még nincsen lefoglalva, akkor egy zöld pipa jelzi annak szabad állapotát, a foglalt állapotot pedig piros x ikon jelöli. Ha a konzultációhoz tartozik leírás, akkor a „Leírás” oszlopban megjelenő kék szem ikonra kattintva elolvasható a megjelenő felugró ablakban. A szabad időpontot a sor végén megjelenő „Lefoglalás” feliratú gombra kattintva lehet lefoglalni, legkésőbb a konzultáció kezdete előtt 24 órával.

A lefoglalt időpontokat később le lehet mondani, ezekre ugyanaz a szabály vonatkozik, mint a lefoglalásra: lemondani egy időpontot legfeljebb a konzultáció kezdete előtt 24 órával a „Lemondás” feliratú gomb megnyomásával lehetséges.

Ha már nem lehet lefoglalni vagy lemondani az időpontot, akkor a gomb helyén megjelenő kérdőjel ikonra kattintva olvasható el a hibaüzenet (4.26. ábra).



4.26. ábra. Foglaláskor és lemondáskor megjelenő hibaüzenetek

### 4.3.3. Mérföldkövek

A „Mérföldkövek” menüpont alatt tekinthető meg az összes a hallgató szakdolgozatához tartozó mérföldkő. Amennyiben a hallgatóhoz még nincsen szakdolgozat társítva, vagy nincsen a szakdolgozatához kapcsolódó mérföldkő, úgy a „A keresett feltételekkel nem található mérföldkő!” hibaüzenet jelenik meg az oldalon (4.27. ábra).

A keresett feltételekkel nem található mérföldkő!

4.27. ábra. Hibaüzenet, ha a hallgatónak nincs szakdolgozata, vagy mérföldköve

A kilistázott mérföldkövek között a táblázat fölötti szűrési lehetőségek használatával lehet keresni név, és állapot alapján, illetve lehetőség van határidő szerint növekvő és csökkenő sorrendbe rendezve megtekinteni azokat (4.28. ábra). A még nem lejárt feladatok határideje piros, vastagított betűvel vannak jelezve. A műveletek oszlopban található kék ikonnal megnyitható a mérföldkő adatlapja (4.29. ábra), ahol a hallgató elolvashatja a feladat leírását, a feladathoz tartozó címkéket, és lehetősége van a feladathoz megjegyzést hozzáfűzni, esetlegesen a megjegyzéshez csatolmányként fájlokat küldeni.

## Mérföldköveim

Cím

Dátum

Állapot

Növekvő

Összes

Mérföldkő címe	Határidő	Állapot	Műveletek
Autentikáció implementálása	2021.04.30. 23:59 (31 nap)		

4.28. ábra. Mérföldkövek listája

Vissza

# Autentikáció implementálása

Határidő: 2021.04.30. 23:59

Folyamatban

Létrehozva: 2021.03.30. 13:00

## Leírás:

Kedves Hallgató!

A kitűzött határidőig szükséges feladatmegoldás:

- backend autentikáció implementálása JWT-nel

Utolsó frissítés: 2021.03.30. 13:00

Címkek:

backendautentikációjwt

Megjegyzések

B I U S X<sup>2</sup> x<sub>2</sub> 16 Font

Csatolmányok

Elfogadott formátumok: png, jpg, pdf, docx, xlsx

Fájlok kiválasztásaNincs fájl kiválasztva

Küldés

Nincsenek hozzászólások!

4.29. ábra. Mérföldkő adatlapja

## 5. fejezet

# Összefoglalás

Szakdolgozatomban a webes platform és az ott használatos protokollok, architektúrák áttekintése után megvizsgáltam és összehasonlítottam a piacon manapság legnépszerűbb webes technológiákat. Az alkalmazás specifikációjának ismertetése után pedig bemutattam a fejlesztett alkalmazás használatát.

A jövőre vonatkozóan többek között az alábbi terveket fogalmaztam meg:

- az ügyintézők által kezelhető ügyek bővítése
- egy email visszajelző rendszer kiépítése, amely mérföldkövek vagy konzultációs időpontokkal kapcsolatos változások esetén üzenetet küldene az érintett oktató és hallgató számára
- a konzultációs időpontok jobb átláthatósága érdekében egy naptár nézetet létrehozni, Google Naptár vagy akár telefonos naptárba való exportálás lehetőségével
- az oldal többnyelvűsítése, hogy azt külföldi személyek is használhassák.

Úgy gondolom, hogy a szakdolgozat elején megfogalmazott célokat sikerült elérnem. Reményeim szerint sikerült egy olyan webes alkalmazást készítenem, amelynek tényleg hasznát vehetik az oktatók és hallgatók egyaránt, és hatékonyabbá teheti a munkát és az ahhoz kapcsolódó visszajelzést.

# Irodalomjegyzék

- [1] How the Web works (2021.02.08.)  
[https://developer.mozilla.org/en-US/docs/Learn/Getting\\_started\\_with\\_the\\_web/How\\_the\\_Web\\_works](https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/How_the_Web_works)
- [2] Internet Protocols (2021.02.08.)  
[https://www.tutorialspoint.com/internet\\_technologies/internet\\_protocols.htm](https://www.tutorialspoint.com/internet_technologies/internet_protocols.htm)
- [3] SSH Protocol – Secure Remote Login and File Transfer (2021.02.08.)  
<https://www.ssh.com/ssh/protocol/>
- [4] HTTP (2021.02.08.)  
<https://developer.mozilla.org/en-US/docs/Web/HTTP>
- [5] Client-Server Overview (2021.02.08.)  
[https://developer.mozilla.org/en-US/docs/Learn/Server-side/First\\_steps/Client-Server\\_overview](https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Client-Server_overview)
- [6] Three tier architecture (2021.02.08.)  
<https://www.quora.com/p/29000/explain-three-tier-architecture-with-advantages-di/>
- [7] JQuery (2021.02.15.)  
<https://jquery.com/>
- [8] Stack Overflow Developer Survey 2020 (2021.02.10.)  
<https://insights.stackoverflow.com/survey/2020#most-popular-technologies>
- [9] Stackoverflow Trends (2021.02.10.)  
<https://insights.stackoverflow.com/trends>
- [10] The State of the Octoverse (2021.02.10.)  
<https://octoverse.github.com/>



- [11] Comparison with Other Frameworks (2021.02.25.)  
<https://vuejs.org/v2/guide/comparison.html>
- [12] Vue dokumentáció (2021.04.15.)  
<https://v3.vuejs.org/guide/>
- [13] Introduction to the Angular Docs (2021.03.01.)  
<https://angular.io/docs>
- [14] React (2021.02.10.)  
<https://reactjs.org/>
- [15] Redux (2021.04.18.)  
<https://redux.js.org/introduction/getting-started>
- [16] Express/Node introduction (2021.04.08.)  
[https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express\\_Nodejs/Introduction](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction)
- [17] Express.js (2021.04.08.)  
<https://expressjs.com/>
- [18] NestJS Documentation (2021.04.08.)  
<https://docs.nestjs.com/>
- [19] Relational vs. non-relational databases (2021.03.04.)  
<https://www.pluralsight.com/blog/software-development/relational-vs-non-relational-databases>
- [20] SQL vs. NoSQL Databases: What's the Difference? (2021.03.04.)  
<https://www.ibm.com/cloud/blog/sql-vs-nosql>
- [21] Understanding the Different Types of NoSQL Databases (2021.03.04.)  
<https://www.mongodb.com/scale/types-of-nosql-databases>
- [22] What Is a Graph Database? (2021.03.04.)  
<https://aws.amazon.com/nosql/graph/>
- [23] JSON and BSON | MongoDB (2021.03.04.)  
<https://www.mongodb.com/json-and-bson>

## NYILATKOZAT

Alulírott Verecki Balint Zoltan, büntetőjogi felelősségem tudatában kijelentem, hogy az általam benyújtott, Szakdolgozati konzultációk támogatása webes rendszerrel című szakdolgozat (diplomamunka) önálló szellemi termékem. Amennyiben mások munkáját felhasználtam, azokra megfelelően hivatkozom, beleértve a nyomtatott és az internetes forrásokat is.

Tudomásul veszem, hogy a szakdolgozat elektronikus példánya a védés után az Eszterházy Károly Egyetem könyvtárába kerül elhelyezésre, ahol a könyvtár olvasói hozzájuthatnak.

Kelt: Eger, 2021. év április hó 26. nap.

Verecki Balint Zoltan

aláírás