

VÁCI SZAKKÉPZÉSI CENTRUM
BORONKAY GYÖRGY
MŰSZAKI TECHNIKUM ÉS GIMNÁZIUM

VIZSGAREMEK

Certifio – Oklevelek online

SZABÓ DÁVID SZILÁRD
2022-2023.

Hallgatói nyilatkozat

Alulírott, ezúton kijelentem, hogy a vizsgaremek saját, önálló munkám, és korábban még sehol nem került publikálásra.

Szabó Dávid Szilárd

Konzultációs lap

Vizsgázó neve: Szabó Dávid Szilárd

Dolgozat címe: Certifio – Oklevelek online

A projekt nyújtotta szolgáltatások:

- Online oklevelek kiállítása
- Online oklevelek nyilvántartása
- Felhasználók nyilvántartása
- Felhasználók azonosítása
- Felhasználók emailben történő értesítése

Sorszám	A konzultáció időpontja	A konzulens aláírása
1.	2022. 10. 04.	
2.	2022. 11. 15.	
3.	2022. 12. 06.	
4.	2023. 01. 10.	
5.	2023. 02. 14.	
6.	2023. 03. 03.	

A vizsgaremeket beadható:

Vác, 2023.....

.....

Konzulens

A vizsgaremeket átvettem:

Vác, 2023.....

.....

A szakképzést folytató intézmény
felelőse

Tartalomjegyzék

Hallgatói nyilatkozat.....	2
Konzultációs lap	3
Tartalomjegyzék.....	4
Témaválasztás.....	7
Fejlesztői dokumentáció	8
Fejlesztői környezet.....	8
Visual Studio Code.....	8
MySQL Workbench 8.0 CE.....	8
Insomnia	8
Használt programozási nyelvek, technológiák, könyvtárak	8
HTML	8
CSS	8
JavaScript.....	8
TypeScript.....	9
Node.js.....	9
JSON Web Token	9
React.....	9
Ant Design	9
MySQL.....	10
ORM.....	10
Prisma	10
HTTP	10
REST API.....	10
Express.....	10
Adatbázis	11
Adatbázis alapadatai	11
Kapcsolatok	12
Táblák	13
A „people” tábla	13
Felépítés	13
SQL parancs	13
Az „auth_keys” tábla	14
Felépítés	14
SQL parancs	14
A „signatures” tábla.....	15

Felépítés	15
SQL parancs	15
A „certificate_bases” tábla	16
Felépítés	16
SQL parancs	16
A „certificates” tábla	17
Felépítés	17
SQL parancs	17
A „signers” tábla	18
Felépítés	18
SQL parancs	18
Használati esetmodell, szerepkörök.....	19
Az alkalmazás felépítése.....	21
Két részre osztás.....	21
Konfiguráció környezeti változókkal.....	21
A működés rövid ismertetése lépésenként.....	21
A backend rendszer részletes felépítése	22
A „.env” fájl.....	22
Az „index.ts” fájl	23
A „routers” könyvtár	24
A „middlewares” könyvtár	24
A „prisma” könyvtár	25
A „lib” könyvtár	25
Az „email-templates” könyvtár	25
A frontend rendszer részletes felépítése	25
A „.env” fájl.....	26
A „main.tsx” fájl.....	26
A „pages” könyvtár.....	28
A „components” könyvtár	28
A „styles” könyvtár	28
A „lib” könyvtár	28
Tesztelés	29
Backend tesztelés	29
Példa: A saját oklevelek listázására használt végpont tesztelése.....	30
Frontend tesztelése.....	31
Példa: Azonosító űrlap működése	31

Továbbfejlesztési lehetőségek	33
Adatszerkezettel és jogosultságokkal kapcsolatos revízió a SaaS architektúra előkészítéséhez ..	33
Testreszabható tulajdonságok hozzárendelése személyekhez, oklevelekhez	33
WYSIWYG szerkesztő	33
Felhasználói dokumentáció	34
A program rövid ismertetése	34
Üzembe helyezés.....	34
A frontend üzembe helyezése	34
A backend üzembe helyezése	35
Futtatás Dockerben	36
Használati útmutató	37
A program használata.....	37
A kezdőlap	37
Gyors megtekintés.....	37
A rendszerről	38
Azonosított megtekintés	38
Személyes oldal	40
Oklevél megtekintés	41
A program használata, mint adminisztrátor.....	42
Belépés az adminisztrátori felületre.....	42
Kezdőlap	42
Aláírások	43
Személyek.....	44
Oklevéltörzsek	45
Oklevelek	46
Mellékletek.....	47
Frontend applikáció.....	47
Backend applikáció.....	47
Ábrajegyzék	47
Irodalomjegyzék.....	48
Internetes tartalmak.....	48
Könyvek	48

Témaválasztás

Tavaly augusztus óta egy önkéntességen alapuló ifjúsági szervezet budapesti (Budapest és Pest megye) körzetében tevékenykedem, mint „Önkéntesekért felelős körzetkoordinátor”, a szervezet középiskolai csereprogramokat bonyolít le világszinten. Az elmúlt egy-két éves időszakban történt egy generációváltás, önkénteseink nagyobb része már nem tud aktívan részt venni a szervezet életében, mert felsőfokú akadémiai céljaikra vagy karrierjükre kell már fókuszálniuk, azonban az új érdeklődők bevonása folyamatosan történik. Ebben az időszakban kerültem a körzetvezetői pozícióba, és legfontosabb feladatomban az önkéntesek motiválása, elismerése. Az év végén oklevéllel történő elismerés ötlete más körzetek tapasztaltabb vezetőitől érkezett, egy-egy körzetnél ez már jól bevált hagyománynak mondható, azonban Budapest és Pest megye területén eddig nem volt bevett szokás.

Ezen ötlet megvalósításához szükségünk van egy módszerre, amely oklevelek tömeges kiállítását teszi lehetővé, emellett az önkénteseket emailben tájékoztatja oklevelük megszerzéséről, ha valamelyikükkel esetleg nem tudnánk személyesen találkozni. Továbbá szempont volt az is, hogy az önkéntesek könnyedén vissza tudják keresni már kiállított okleveleiket, illetve az is, hogy a körzetvezetés is egyszerűen nyilván tudja tartani az elismeréseket.

A projekt azonban túlmutat az önkéntesek elismerésén, hiszen ez a motivációs módszertan számos szituációban használható, legyen szó oktatási intézményekről vagy akár munkáltatói belső képzésekről. A projekt célja tehát egy olyan program, amely képes okleveleket kiállítani és azokat nyilvántartásba szervezni, amely a megfelelő jogosultságokkal böngészhető.

Az ilyesfajta elismerések azonban nemcsak motivációs erővel bírnak, hanem bizonyos helyzetekben a külvilág számára is információkat hordozhatnak, amelyből könnyen hasznót kovácsolhatunk. Gondoljunk csak bele – például egy nyelvtanfolyamról bemutatott tanúsítvány jól mutathat az önéletrajzunkban vagy LinkedIn profilunkon. Minél több ilyen apró teljesítményt rögzítünk szakmai önéletrajzunkban, annál nagyobb valószínűséggel érhetünk el sikereket a munkáltatók meggyőzésében.

Összegezve a projekt célja egy újfajta elismerési kultúra népszerűsítése, amely nemcsak a további tanulásra és sikerek elérésére ösztönzi az embereket, hanem a készségek és teljesítményeket kiemelését is lehetővé teszi a munkavállalókat keresők számára.

A mai technológiai adottságok mellett nem is volt kérdéses az alkalmazás szerkezete, mindenképpen webes technológiákban volt érdemes gondolkodni, hiszen ezek platformfüggetlenül nyújtanak azonos felhasználói élményt.

Fejlesztői dokumentáció

Fejlesztői környezet

Visual Studio Code

A Visual Studio Code egy nyílt forrású, ingyenesen elérhető kódszerkesztő alkalmazás, amelyet a Microsoft fejlesztett ki. Támogatja többek között a szintaxis kiemelést, a kódfordítást, a hibakeresést és a Git integrációt. Emellett rengeteg bővítmény érhető el hozzá, amelyekkel testre szabható és kiegészíthető a funkcionalitása. Az alkalmazás keresztplatformos, és támogatja a Windows, macOS és Linux rendszereket. A Visual Studio Code egyre népszerűbb fejlesztői eszköz a web- és az alkalmazásfejlesztés területén.

MySQL Workbench 8.0 CE

A MySQL Workbench 8.0 CE egy ingyenesen elérhető, grafikus felhasználói felülettel rendelkező adatbázis-kezelő eszköz, amelyet a MySQL fejlesztett ki. Segítségével könnyedén kezelhetők az adatbázisok, készíthetők lekérdezések, modellezhetők az adatbázisok és optimalizálhatóak az adatbázis-struktúrák. A MySQL Workbench támogatja többek között a Windows, macOS és Linux operációs rendszereket.

Insomnia

Az Insomnia egy ingyenes, nyílt forrású alkalmazás, amely lehetővé teszi a fejlesztők számára, hogy egyszerűen és hatékonyan teszteljék a REST API-kat. Az Insomnia támogatja az összes HTTP metódust, automatikus kódgenerálást biztosít, valamint lehetővé teszi a tesztesetek létrehozását és futtatását. A szoftver keresztplatformos, és elérhető Windowsra, macOS-re és Linuxra is.

Használt programozási nyelvek, technológiák, könyvtárak

HTML

Az HTML (HyperText Markup Language) egy leíró nyelv, amelyet a weboldalak létrehozásához használnak. Az HTML alapvetően a weboldalak strukturális elemeit definiálja, és a böngészők megfelelő megjelenítése érdekében használják. Az HTML használatával a fejlesztők képesek az oldal tartalmát, címsorait, fejlécét, láblécét, képeit, videóit és más multimédiás tartalmait definiálni.

CSS

A CSS (Cascading Style Sheets) egy stílusleíró nyelv, amely lehetővé teszi a weboldalak megjelenésének testreszabását. A CSS segítségével a fejlesztők definiálhatják a weboldalak eleminek stílusát, például a betűméretet, a betűtípust, a színt, a háttérszínt és a marginokat. A CSS megkönnyíti az oldalak egyedi megjelenésének létrehozását és a tartalom és a megjelenés elkülönítését, ami növeli a karbantarthatóságot és a skálázhatóságot.

JavaScript

A JavaScript egy dinamikus, interpretált, magas szintű programozási nyelv, amely weboldalakon történő interaktív műveletek végrehajtására használatos. A JavaScript elengedhetetlen az internetes alkalmazások és weboldalak fejlesztésében, és számos webes keretrendszer és könyvtár, mint például az Angular, a React és a Vue.js épül rá. A JavaScript lehetővé teszi a dinamikus tartalom létrehozását, az adatok lekérdezését, a felhasználói interakciók kezelését, az animációk és effektek létrehozását. A JavaScript nem csak a böngészőkben használható, hanem a szerveroldalon is, például a Node.js segítségével.

TypeScript

A TypeScript egy típusos, statikus típusellenőrzést használó nyelv, amely a JavaScriptre épül. A TypeScript lehetővé teszi a kód szervezését és karbantartását, nagyobb biztonságot nyújt a hibák ellen, és növeli az olvashatóságot és az érthetőséget. A TypeScript az alapvető JavaScript szintaxison túl kiterjesztésekkel rendelkezik, mint például az interfészek és a felsorolások, amelyek megkönnyítik a típusok és a struktúrák meghatározását. A TypeScript-et széles körben használják a nagyobb, komplexebb JavaScript projektekben, és az Angular, a React és a Vue.js keretrendszerek is támogatják.

Node.js

A Node.js egy nyílt forráskódú, platformfüggetlen szerveroldali JavaScript környezet, amely lehetővé teszi az aszinkron I/O műveleteket a JavaScript-ben. A Node.js a Chrome V8 JavaScript motorját használja, amely nagyon hatékony és gyors. A Node.js lehetővé teszi a fejlesztők számára, hogy azonos nyelvet használjanak a kliens- és a szerveroldali fejlesztéshez, amely gyorsabb és hatékonyabb fejlesztést tesz lehetővé. A Node.js alkalmas a nagy adatforgalmú és valós idejű alkalmazásokhoz, például az üzenetküldő alkalmazásokhoz és a valós idejű játékokhoz. A Node.js-ben széles körben elérhetők a modulok, amelyek könnyen telepíthetők és használhatók. A Node.js-t sok nagyvállalat, például a Netflix és a LinkedIn használja, és számos népszerű keretrendszer van hozzá, például az Express.

JSON Web Token

A JWT (JSON Web Token) egy nyílt szabvány, amely lehetővé teszi a biztonságos információcserét tokenek segítségével az internetes alkalmazások között. A tokenek tartalmazzák az azonosítást és az engedélyeket, és digitálisan alá vannak írva, hogy biztosítsák az adatok valóságát és érvényességét. A JWT-k nagyon elterjedtek az internetes alkalmazásokban, mivel könnyen használhatók és biztonságosak. A JWT-k több adatot tartalmazhatnak, mint például az időtúllépési időt vagy a felhasználó szerepkörét, amelyeket az alkalmazás használhat a felhasználói munkamenetek kezeléséhez és az engedélyek felügyeletéhez. A JWT-knek három része van: a fejléc, a token test és az aláírás. A fejléc tartalmazza a token típusát és az algoritmust, amelyet az aláírás létrehozásához használnak. A token test tartalmazza az információkat, amelyek azonosítják a felhasználót és az engedélyeit. Az aláírás az információk digitális aláírására szolgál, és biztosítja, hogy a token tartalma ne módosuljon vagy hamisításra kerüljön. A JWT-knek számos előnye van, például, hogy könnyen használhatók, biztonságosak és hordozhatók az alkalmazások között.

React

A React egy nyílt forráskódú, deklaratív és hatékony JavaScript könyvtár, amelyet a felhasználói felületek (UI-k) építésére használnak. A React segít az alkalmazások összetett felületeinek építésében, a komponensek újrafelhasználásában és a dinamikus UI-k kialakításában. A React az egyirányú adatáramláson alapuló architektúrát használja, amelyben a komponensek tiszta funkcionális egységek.

Ant Design

Az Ant Design egy React UI komponens könyvtár, amely segít a webfejlesztőknek hatékonyan és könnyen tervezni és fejleszteni modern felhasználói felületeket. A könyvtár széles körű komponenseket tartalmaz, beleértve a gombokat, az űrlapokat, az ikonokat, a navigációt és sok más elemet. Az Ant Design különös figyelmet fordít az összhangra és a konzisztenciára, így a felhasználói élmény egységes marad a teljes alkalmazásban. A könyvtár számos beépített funkcióval és testreszabási lehetőséggel rendelkezik, amelyek segítségével a fejlesztők könnyen testreszabhatják az UI komponenseket az alkalmazásuk igényeihez.

MySQL

Az MySQL egy ingyenes, nyílt forráskódú relációs adatbázis-kezelő rendszer, amelyet a legtöbb operációs rendszerre telepíthetünk. A MySQL könnyen használható, széles körben elterjedt és nagyon skálázható, ami azt jelenti, hogy nagy adatmennyiségeket is képes hatékonyan kezelni. Az adatok egyszerű kezelése és a megbízhatósága miatt a MySQL széles körben használják az üzleti alkalmazásokban, weboldalakban, mobilalkalmazásokban, játékokban és még sok más területen. A MySQL támogatja az ACID-képességeket, amelyek biztosítják az adatok integritását és konzisztenciáját.

ORM

Az ORM (Object-Relational Mapping) egy programozási technika, amely lehetővé teszi, hogy az adatbázisokat objektumorientált nyelvekkel kezeljük. Az ORM segít összekapcsolni az adatbázisokat a programozási nyelvekkel, és lehetővé teszi az adatok kezelését, módosítását és lekérdezését az objektumorientált nyelvben. Az ORM keretrendszerek sok kódismétlődéstől és hibától mentesítik a fejlesztőket, és elősegítik az alkalmazások karbantarthatóságát. Az ORM használatával a fejlesztőknek nem kell ismerniük az SQL nyelvet, és az adatbázisok kódjának írása helyett az ORM segítségével egyszerűbb, olvashatóbb kód írható.

Prisma

A Prisma egy nyílt forráskódú ORM (Object-Relational Mapping) és adatbázis-kapcsoló könyvtár a Node.js-hez és a TypeScript-hez. A Prisma lehetővé teszi az alkalmazások számára, hogy egyszerűen kezeljék az adatbázisokat és a lekérdezéseket, miközben biztonságos és hatékony módosításokat hajtanak végre. A Prisma automatikusan generálja az interfészeket és az adatbázis-sémát, így nincs szükség a manuális adatbázis-migrációkra. A Prisma használata lehetővé teszi a típusbiztos lekérdezéseket és a teljesítmény-optimalizált adatbázis-hozzáférést.

HTTP

REST API

A REST (Representational State Transfer) egy elterjedt architektúra az alkalmazás-programozási interfészek (API-k) tervezéséhez és megvalósításához. A REST API-k az erőforrásokat (adatokat, szolgáltatásokat) ábrázolják, és az HTTP protokollt használják a kliens és a szerver közötti kommunikációhoz. A REST API-k állapotmentesek, azaz a kliens kérésének tartalmaznia kell az összes szükséges információt. A REST API-k népszerűségét az egyszerűség, az egységes interfész és a skálázhatóság adja, amely lehetővé teszi a nagyobb terhelések kezelését.

Express

Az Express.js egy nyílt forráskódú, könnyű és rugalmas webes alkalmazás-keretrendszer a Node.js-hez. Az Express segít az alkalmazások építésében, különösen az API-k és a webalkalmazások esetében. Az Express a middleware-k koncepciójára épül, amely lehetővé teszi a fejlesztők számára, hogy a kódot egyszerűen és hatékonyan strukturálják. Az Express kínál sok beépített funkciót, például routing, sablonok és statikus fájlok szolgáltatása, amelyek felgyorsítják az alkalmazások fejlesztését. Az Express széles körben használják a Node.js-alapú webalkalmazások és API-k készítéséhez, és nagyvállalatok, mint például az IBM és a PayPal is használják.

Adatbázis

Adatbázis alapadatai

Adatbázis szerver: MySQL

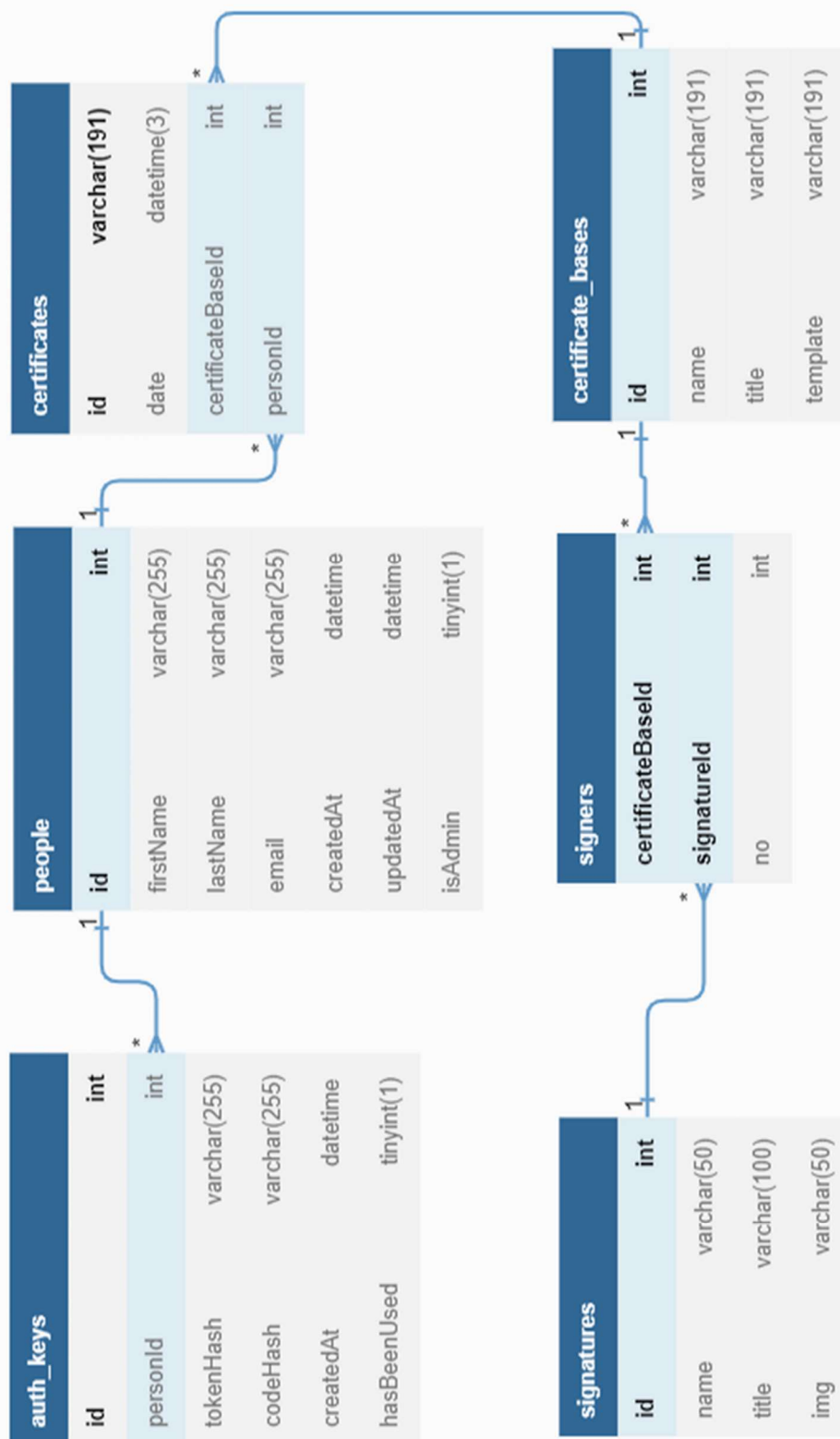
Adatbázis neve: certifio

Illesztés: utf8mb4_general_ci

Motor: InnoDB

SQL parancs a létrehozásához:

```
CREATE DATABASE IF NOT EXISTS `certifio`  
DEFAULT CHARACTER SET utf8mb4  
COLLATE utf8mb4_general_ci;
```



1. ábra Az adatbázis felépítése

Táblák

A „people” tábla

A „people” tábla rekordjai a rendszerben oklevelet szerző vagy adminisztrátori jogosultsággal rendelkező személyeket tárolnak.

Felépítés

Mező neve	Típusa	Leírás	Kulcs / Index
id	INT	A személy egyedi azonosítója.	Elsődleges kulcs
firstName	VARCHAR	A személy keresztnéve.	
lastName	VARCHAR	A személy vezetéknéve.	
email	VARCHAR	A személy email-címe.	Egyedi kulcs
createdAt	DATETIME	A személy rögzítésének időpontja.	
updatedAt	DATETIME	A személy frissítésének időpontja.	
isAdmin	TINYINT	A személy adminisztrátor-e.	

SQL parancs

```
CREATE TABLE `people` (  
  `id` int NOT NULL AUTO_INCREMENT,  
  `firstName` varchar(255) COLLATE utf8mb4_unicode_ci NOT NULL,  
  `lastName` varchar(255) COLLATE utf8mb4_unicode_ci NOT NULL,  
  `email` varchar(255) COLLATE utf8mb4_unicode_ci NOT NULL,  
  `createdAt` datetime NOT NULL,  
  `updatedAt` datetime NOT NULL,  
  `isAdmin` tinyint(1) NOT NULL DEFAULT '0',  
  PRIMARY KEY (`id`),  
  UNIQUE KEY `email` (`email`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

Az „auth_keys” tábla

Az „auth_keys” tábla a rendszerben tárolt személyek megbízható azonosításához szükséges adatokat tárolja. Az azonosítási folyamat kezdete után a rendszer egy email formájában a felhasználó rendelkezésére bocsájt egy tokent, amit egy hiperhivatkozás formájában használhat, illetve egy négyjegyű kódot is, amit manuális megadással hasznosíthat, ezen adatok hash értékeit tároljuk.

Felépítés

Mező neve	Típusa	Leírás	Kulcs / Index
id	INT	Az azonosító adat egyedi azonosítója.	Elsődleges kulcs
personId	INT	A kapcsolódó személy egyedi azonosítója.	Idegen kulcs
tokenHash	VARCHAR	Az egyedi token bcrypt hash értéke.	
codeHash	VARCHAR	Az egyedi kód bcrypt hash értéke.	
createdAt	DATETIME	Az azonosító adat létrehozása.	
hasBeenUsed	TINYINT	Az azonosító volt-e már használva.	

SQL parancs

```
CREATE TABLE `auth_keys` (  
  `id` int NOT NULL AUTO_INCREMENT,  
  `personId` int DEFAULT NULL,  
  `tokenHash` varchar(255) COLLATE utf8mb4_unicode_ci NOT NULL,  
  `codeHash` varchar(255) COLLATE utf8mb4_unicode_ci NOT NULL,  
  `createdAt` datetime NOT NULL,  
  `hasBeenUsed` tinyint(1) NOT NULL DEFAULT '0',  
  PRIMARY KEY (`id`),  
  KEY `personId` (`personId`),  
  CONSTRAINT `fk_authkeys_people` FOREIGN KEY (`personId`)  
  REFERENCES `people` (`id`) ON DELETE SET NULL ON UPDATE CASCADE  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

A „signatures” tábla

A „signatures” tábla a rendszerben tárolt aláírásokat tartalmazza, amelyek a kiállított okleveleken szerepelhetnek.

Felépítés

Mező neve	Típusa	Leírás	Kulcs / Index
id	INT	Az aláírás egyedi azonosítója.	Elsődleges kulcs
name	VARCHAR	Az aláíró neve.	
title	VARCHAR	Az aláíró titulusa.	
img	VARCHAR	Az aláírást tartalmazó képfájl neve.	

SQL parancs

```
CREATE TABLE `signatures` (  
  `id` int NOT NULL AUTO_INCREMENT,  
  `name` varchar(50) NOT NULL,  
  `title` varchar(100) DEFAULT NULL,  
  `img` varchar(50) DEFAULT NULL,  
  PRIMARY KEY (`id`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

A „certificate_bases” tábla

A „certificate_bases” táblában az „oklevéltörzs” adatait tároljuk. Az oklevéltörzs leírja egy kiadható oklevél felépítését és tulajdonságait. Ha objektumorientált perspektívából közelítjük meg, akkor azt mondhatjuk, hogy az oklevél kiállításához egy oklevéltörzs osztályból készítünk egy példányt.

Felépítés

Mező neve	Típusa	Leírás	Kulcs / Index
id	INT	Az oklevéltörzs egyedi azonosítója.	Elsődleges kulcs
name	VARCHAR	Az oklevéltörzs megnevezése.	
title	VARCHAR	Az oklevéltörzs nyilvános megnevezése.	
template	VARCHAR	Az oklevéltörzs felépítését leíró mappa.	

SQL parancs

```
CREATE TABLE `certificate_bases` (  
  `id` int NOT NULL AUTO_INCREMENT,  
  `name` varchar(191) COLLATE utf8mb4_unicode_ci NOT NULL,  
  `title` varchar(191) COLLATE utf8mb4_unicode_ci NOT NULL,  
  `template` varchar(191) COLLATE utf8mb4_unicode_ci NOT NULL,  
  PRIMARY KEY (`id`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```


A „certificates” tábla

A „certificates” tábla a kiállított okleveleket tárolja. Az egyedi azonosító egy speciális alfanumerikus karakterlánc. A kiállított oklevél azonosítóját, kiállításának dátumát, a szerkezetét és tulajdonságait leíró oklevéltörzs azonosítóját és az oklevelet megszerző személy azonosítóját tároljuk.

Felépítés

Mező neve	Típusa	Leírás	Kulcs / Index
id	VARCHAR	Az oklevél egyedi azonosítója.	Elsődleges kulcs
date	DATETIME	Az oklevél kiadásának dátuma.	
certificateBaseId	INT	Az oklevéltörzs azonosítója.	Idegen kulcs
personId	INT	A személy azonosítója.	Idegen kulcs

SQL parancs

```
CREATE TABLE `certificates` (  
  `id` varchar(191) COLLATE utf8mb4_unicode_ci NOT NULL,  
  `date` datetime(3) NOT NULL,  
  `certificateBaseId` int NOT NULL,  
  `personId` int NOT NULL,  
  PRIMARY KEY (`id`),  
  KEY `certificates_certificateBaseId_fkey` (`certificateBaseId`),  
  KEY `certificates_personId_fkey` (`personId`),  
  CONSTRAINT `certificates_certificateBaseId_fkey` FOREIGN KEY  
  (`certificateBaseId`) REFERENCES `certificate_bases` (`id`) ON  
  DELETE RESTRICT ON UPDATE CASCADE,  
  CONSTRAINT `certificates_personId_fkey` FOREIGN KEY (`personId`)  
  REFERENCES `people` (`id`) ON DELETE RESTRICT ON UPDATE CASCADE  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

A „signers” tábla

A „signers” tábla az egyes oklevéltörzsek aláíróit rögzíti az aláírások sorrendjének meghatározásával.

Felépítés

Mező neve	Típusa	Leírás	Kulcs / Index
certificateBaseId	INT	Az oklevéltörzs azonosítója.	Elsődleges, ldegen kulcs
signatureId	INT	Az aláírás azonosítója.	Elsődleges, ldegen kulcs
no	INT	Az aláírás sorrendjét leíró szám.	

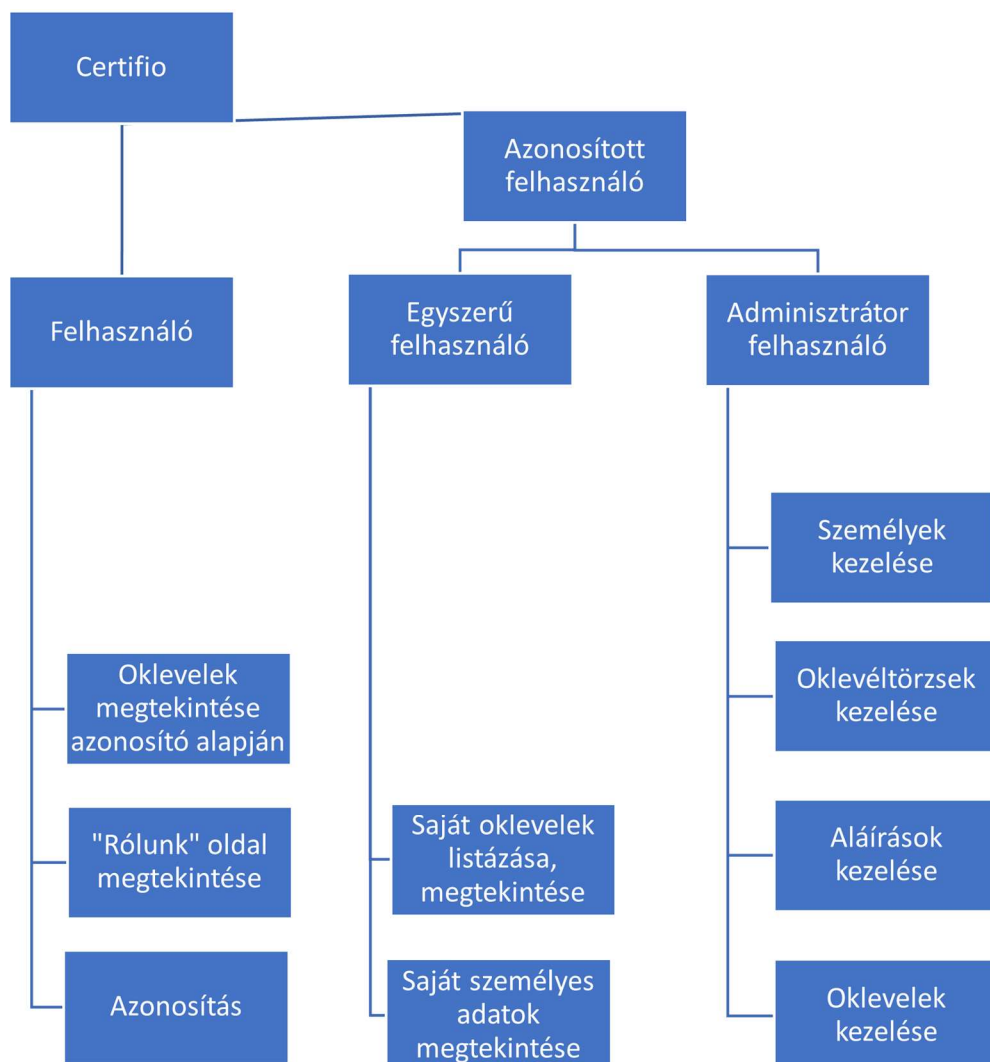
SQL parancs

```
CREATE TABLE `signers` (  
  `certificateBaseId` int NOT NULL,  
  `signatureId` int NOT NULL,  
  `no` int NOT NULL,  
  PRIMARY KEY (`certificateBaseId`,`signatureId`),  
  KEY `FK_signers_signatures` (`signatureId`),  
  CONSTRAINT `FK_signers_certificate_bases` FOREIGN KEY  
  (`certificateBaseId`) REFERENCES `certificate_bases` (`id`) ON  
  DELETE RESTRICT ON UPDATE RESTRICT,  
  CONSTRAINT `FK_signers_signatures` FOREIGN KEY (`signatureId`)  
  REFERENCES `signatures` (`id`) ON DELETE RESTRICT ON UPDATE RESTRICT  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

Használati esetmodell, szerepkörök

A programban a felhasználók szerepkörei meghatározott jogosultságokkal rendelkeznek. Az alapvető felhasználói szerepkör lehetővé teszi a felhasználó számára az oklevél lekérését azonosító alapján, valamint egy a rendszerről szóló ismertető elolvasását. Az azonosított felhasználó ehhez képest további jogosultságokhoz jut, így megtekintheti saját okleveleinek listáját és saját személyes adatait. Az adminisztrátor szerepkörében a felhasználó képes az oklevéltörzsek, azaz sablonok készítésére és módosítására, oklevelek kiállítására, személyek felvételére és módosítására. Az ilyen szerepkörök megfelelően strukturálják a felhasználói jogosultságokat, így biztosítva az adatbiztonságot és a jogosulatlan hozzáférés megakadályozását.

A jogosultságok hierarchikusan egymásra épülnek, azaz a legalább második szintű jogosultsági szinttől elmondható, hogy minden jogosultsági szint rendelkezik az őt megelőző szint tulajdonságaival. A program esetén konkrétan a következő jogosultsági szinteket különböztetjük meg: felhasználó, azonosított felhasználó és adminisztrátor. Az egyes engedélyeket a következő ábrák segítségével szemléltethetjük.



2. ábra A jogosultságok

Ennél az ábrán fontos kiemelni, hogy ha jobbról balra haladunk, akkor a szerepkörre az attól balra található tulajdonságok is jellemzők.

Egy másik ábra segítségével, halmazszerűen talán jobban szemléltethető ez a felépítés. A következő ábrán a lekerekített téglalapok jelzik az egyes szerepköröket, míg a világoskék téglalapok az egyes engedélyezett tevékenységeket. Jól látható, hogy a legmagasabb szinten az „Adminisztrátor” áll, aki nem csak „Adminisztrátor”, de „Azonosított felhasználó”, sőt „Felhasználó”, tehát képes minden olyan művelet elvégzésére, mint ezen jogosultsági csoport tagjai.



3. ábra A jogosultságok

Az alkalmazás felépítése

Két részre osztás

Az alkalmazás két külön programból áll, melyek egységesen TypeScript nyelven íródtak. Az alkalmazás két része a backend és frontend, a szétválasztás tudatosan történt az ideális működés elérése érdekében. A frontend egy olyan React.js applikáció, amely működéséhez kizárólagosan csak statikus fájlokat igényel, ezért a kliensek irányába történő kiszolgálása történhet egy statikus fájlok hostolására specializált nagy teljesítményű globális szolgáltató rendszeréről, amely a CDN hálózatok miatt gyors és optimális élményt nyújt a felhasználónak. A backend egy Node.js applikáció, amely használható közvetlenül az internetre kapcsolva is, vagy egy reverse-proxy szerveren keresztül is, nagyobb terhelés esetén akár párhuzamosan több példányt futtatva terheléskegyenlítéssel. A kommunikáció a projekt két része között REST architektúrát követ, azaz HTTP kérések végzik az adat áramoltatását.

Konfiguráció környezeti változókkal

A konfigurációhoz a fordítást megelőzően környezeti változókkal kell megadni a szükséges adatokat, ez a mondat elsőre furcsán hathat egy magasszintű programozási nyelvnél, azonban fontos kiemelni, hogy a TypeScript programot futtatás előtt JavaScriptre, egy interpretált nyelvre kell fordítani. A frontend applikációnak egyedül a backend applikáció elérési útját kell megadnunk, míg a backend applikáció számára meg kell adnunk az adatbázis adatait, a frontend elérési útját, valamint az email küldéshez szükséges információkat.

A működés rövid ismertetése lépésenként

1. A felhasználó ellátogat a frontendre (továbbiakban FE).
2. A FE betöltés után egy API kéréssel lekéri a backenden (továbbiakban BE) megadott üzemeltetéssel kapcsolatos adatokat, addig egy folyamat jelző animációt mutat. Ebben kötelezően szerepel az üzemeltető személy vagy szervezet neve, és opcionálisan a címe, honlapja és kapcsolattartójának neve és email-címe. Ha a BE-n nincs megadva legalább az üzemeltető neve, akkor a „DEMO” felirat fog megjelenni.
3. Megjelenik a kezdőlap a FE-n, ahol a felhasználó választhat az azonosító alapú lekérdezés, az azonosítás és az ismertető oldal meglátogatása között.
 - a. A gyors lekérdezésnél a felhasználó megadja a tízjegyű azonosítót, majd a rendszer egy API kérést intéz a BE irányába. Ha az azonosító hibás, a program jelzi azt a felhasználónak. Ha az azonosító helyes, akkor a BE betölti az oklevél és annak törzsének adatait az adatbázisból, valamint a vizuális felépítését a fájlrendszerből és elküldi azt a FE számára. A FE ez alapján felépíti az oklevelet a megfelelő HTML elemekből, a megfelelő adatokkal.
 - b. Az azonosításhoz meg kell adni a felhasználó email-címét. A FE API kérést küld a BE-nek, ha az email-cím nem szerepel az adatbázisban a FE tájékoztatja a felhasználót. Ha szerepel, akkor a BE generál egy tokent, illetve egy négyjegyű számot, amelyeket emailben elküld a megadott címre, majd tárolja azok hash értékeit az adatbázisban. A FE az email küldés sikeréről tájékoztatja a felhasználót. A felhasználó a tokent tartalmazó hivatkozás vagy a kód segítségével azonosítja magát. A sikeres azonosítás követően a FE lekéri a felhasználó okleveleinek listáját és személyes adatait, majd megjeleníti azokat. Ha a felhasználó valamelyik oklevelére kattint, akkor az előző pontban tárgyaltak szerint fut végig a folyamat.
 - c. A felhasználó a rendszert ismertető oldalra látogat, ahol lenyíló dobozokban találhatóak kérdések a rendszer működésével kapcsolatban.

A backend rendszer részletes felépítése

A továbbiakban a backend felépítését fogom részletezni, az átlátható és strukturált felépítés fontos, hiszen megteremti a későbbi fejlesztések könnyű és gyors végrehajtásához szükséges feltételeket.

A „.env” fájl

Ez a fájl tartalmazza a környezeti változók listáját, amelyeket a program induláskor betölt. Az adatbázis elérhetőségeit, az email küldéshez szükséges adatokat, valamint a frontend elérési útvonalát is itt kell beállítani. A program egy egyedi eljárás keretébe vizsgálja, hogy a kötelező környezeti változók megadásra kerültek-e, valamint, hogy helyes értékekkel rendelkeznek-e. Az ilyen jellegű fájlok verziókövető rendszerbe nem kerülhetnek be, hiszen érzékeny adatokat is tartalmaznak, azonban van egy „.env.template” fájl, ami mintaként szolgál.

A következő oldalon található az „.env.template” fájl tartalma, valamint annak értelmezése.

```
# A connection URL for the database. See
https://www.prisma.io/docs/reference/database-reference/connection-urls#mysql
DATABASE_URL=""

# The base URL for the frontend application without the trailing slash. Required
for the authentication emails.
FRONTEND_URL=""

# The email address from where emails will be sent out.
SYSTEM_EMAIL=""

# The secret used to sign the authenticational tokens. Preferably use a random
string. e.g.:
https://www.random.org/strings/?num=1&len=20&digits=on&upperalpha=on&loweralph
a=on&unique=on&format=plain&rnd=new
JWT_SECRET=""

# Uncomment this part if you want to use SMTP.
# Set the SMTP connection URL accordingly. See https://nodemailer.com/smtp/

# EMAIL_MODE="SMTP"
# SMTP_CONNECTION_URL=""

# Uncomment this part if you want to use AWS SES.
# Set the AWS region, access key id and secret access key.

# EMAIL_MODE="AWS"
# AWS_REGION=""
# AWS_ACCESS_KEY_ID=""
# AWS_SECRET_ACCESS_KEY=""
```

A mintában eredetileg nem kommentben található három környezeti változó („DATABASE_URL”, „FRONTEND_URL”, „SYSTEM_EMAIL”, „JWT_SECRET”) megadása kötelező, tetszőleges szöveges értéket vesznek fel, a program hibaüzenettel megáll amennyiben mégsem adjuk meg őket. Az „EMAIL_MODE” változó megadása szintúgy kötelező, azonban értékészlete az $R = \{ "AWS", "SMTP" \}$ halmaz. Ha az SMTP értéket adtuk meg, azaz SMTP protokollal szeretnénk levelezést folytatni, akkor szükséges megadni az „SMTP_CONNECTION_URL” változót is, ha az AWS értéket adtuk meg, azaz email küldéshez az Amazon Web Services Simple Email Service szolgáltatását szeretnénk

használni, akkor meg kell adni az „AWS_REGION”, „AWS_ACCESS_KEY_ID” és „AWS_SECRET_ACCESS_KEY” változókat. Ha a megadott konfiguráció nem megfelelő, hiányos, akkor a program azt egyértelmű hibaüzenettel jelzi, majd leáll.

Az „index.ts” fájl

Az index.ts fájl a backend applikáció belépőpontja, itt történik a függvényhívás a környezeti változók helyességének ellenőrzésére, valamint az Express.js HTTP szerver viselkedése is itt kerül beállításra, végül a szolgáltatás elindítása is itt történik meg.

Részlet az index.ts fájlból:

```
//A környezeti változók ellenőrzése.
checkEnv();

//Az Express.js szerver példányosítása, majd konfigurációja.
const app = express();
//Szerszintű middleware állítás, amely a JSON body értékeket fogja
objektumokká alakítani.
app.use(express.json());
//Szerszintű middleware állítás, amely a CORS headerek megfelelő beállítását
végzi.
app.use(cors({origin: frontendURL}));
//Egyes elérési utak viselkedését állító routerek megadása.
app.use("/auth", AuthRouter);
app.use("/certificates", CertificatesRouter);
app.use("/info", InfoRouter);
//Elérési út szintű middleware állítás, amely a statikus képek kiszolgálását
teszi lehetővé.
app.use("/storage/images", express.static(createPath("images")));

//Az Express.js szerver elindítása.
app.listen(port, () => {console.log(`Service running on PORT ${port}`)});
```

A „routers” könyvtár

Ez a könyvtár tartalmazza az Express.js routereket, amelyek egy-egy HTTP végpont viselkedését írják le. Az Express.js router segítségével könnyen megadhatjuk az útvonalakat, a végpontokat, és a hozzájuk tartozó műveleteket.

Részlet a „/certificates” elérési út alatt található routerből:

```
//A router az indexre (/certificates) a következő műveleteket végzi:  
//Meghívásra kerül az AuthRequired middleware, ami csak akkor engedi folytatni  
a végrehajtást, ha a felhasználó azonosítva van.  
router.get("/", AuthRequired, async (req, res) => {  
  //Az adatbázisból lekérjük azokat az okleveleket, amelyek a kérést küldő  
felhasználóhoz tartoznak, az oklevéltörzsekkel együtt.  
  const certificates = await prisma.certificate.findMany({where: {personId:  
req.user?.id}, include: {base: true}}});  
  //A kapott értékeket optimális formára hozzuk, majd elküldjük a kliensnek.  
  const results = certificates.map((e: any) => {  
    return {  
      id: e.id,  
      title: e.base.title,  
      date: e.date  
    }  
  })  
  res.send(results);  
});
```

A „middlewares” könyvtár

Ez a könyvtár az Express.js middlewareket tartalmazza, amelyek olyan többször használatos eljárások, amelyeket egyes HTTP végpontok műveletei elé szűrhatunk be. Az autentikáció és adminisztrátori végpontokhoz szükséges autorizáció is ilyen middlewarekkel történik.

Az autentikációt végző middleware:

```
export default function AuthRequired(req: Request, res: Response, next: NextFunction){
    const jwt = req.headers["x-auth-key"];
    //Ha a kérés nem rendelkezik a JWT-t tároló HTTP headerrel, akkor a végrehajtás leáll HTTP 403-as státusszal.
    if(!jwt)
        return res.status(403).send({respCode: "AUTH_REQUIRED"});
    try{
        //A kapott értéket kriptográfiailag ellenőrizzük, hiba, azaz kivétel esetén a végrehajtás a catch ágba kerül.
        //@ts-ignore
        const data = verify(jwt, "secret");
        let user = data.person;
        user.createdAt = new Date(user.createdAt);
        user.updatedAt = new Date(user.updatedAt);
        //A végrehajtás továbbadása előtt a kérést tartalmazó objektumot populáljuk a felhasználó adatival.
        req.user = user;
        //A végrehajtást továbbadjuk a következő middleware-nak vagy a routernek.
        next();
    }
    catch(err){
        //Ha a kriptográfiai megerősítés közben kivételt kaptunk, akkor a token hamis, a végrehajtás leáll HTTP 403-as státusszal.
        return res.status(403).send({respCode: "AUTH_REQUIRED"});
    }
}
```

A „prisma” könyvtár

A Prisma könyvtár tartalmazza a Prisma ORM rendszer működéséhez szükséges „schema.prisma” fájlt, amely az adatbázisban található táblák, illetve az abból gyártott osztályok-objektumok szerkezetét írja le.

A „lib” könyvtár

A sajátfejlesztésű eljárásokat tartalmazza, amelyek a program működéséhez szükséges részfeladatok végrehajtását írják le, pl.: email küldés, fájl beolvasás, autentikációs tokenek létrehozása, adatbázis manipuláció. Lényegében ide kell elhelyezni minden olyan kódot, amely funkcionalitásából adódóan az előbb tárgyalt mappák egyikébe sem illik.

Az „email-templates” könyvtár

Az emailek küldéséhez használt sablonok ebben a könyvtárban találhatóak, a Handlebars.js templating engine felhasználásával történik az emailek kialakítása.

A frontend rendszer részletes felépítése

A továbbiakban a frontend felépítését fogom részletezni, az átlátható és strukturált felépítés fontos, hiszen megteremti a későbbi fejlesztések könnyű és gyors végrehajtásához szükséges feltételeket.

A „.env” fájl

Ez a fájl tartalmazza a „production-ready” statikus fájlokból álló weboldal létrehozásakor szükséges környezeti változók listáját. Jelenleg csak egy „PUBLIC_BACKEND_URL” nevű változó létezik, amely tartalmazza a backend rendszer elérési útvonalát. A „PUBLIC” előtagra a használt buildtool, a Vite miatt van szükség, mert biztonsági okokból megköveteli, hogy a publikus használatra szánt változókat prefixáljuk, ezzel elkerülve szenzitív adatok kiszivárgását pl.: egy monolitikus applikáció esetén.

A „main.tsx” fájl

Ez az alkalmazás belépőpontja, itt adjuk meg a React applikáció gyökerét, az alkalmazásszintű adatszolgáltatókat („provider”). Egy-egy provider végzi az autentikáció kezelését, az üzemeltető adatainak letöltését, és magát a kliens oldali routingot is. A kliens oldali routing azt jelenti, hogy habár a felhasználó számára úgy tűnik, mintha különböző elérési úttal rendelkező oldalak között navigálna böngészője, mint a hagyományos webes alkalmazások esetén, a valóság ezzel szemben az, hogy ugyanazon weboldalon marad újabb tartalom letöltése nélkül, és ezen oldal tartalma változik a címsorban megjelenő elérési út alapján.

```

//A routing folyamatot leíró objektum.
//(Csak példa, nem tartalmazza az összes oldalt.)
const router = createBrowserRouter([
  {
    path: "/",
    element: <NestedLayout />,
    //Jól megfigyelhető, hogy az oldalak egymásba épülnek.
    //A NestedLayout komponens tartalmazza a minden oldalon használt
    elemeket: fejléc, lábléc.
    children: [
      {
        //elérési út: /
        path: "",
        element: <Home />
      },
      {
        //elérési út: /authentication
        path: "authentication",
        element: <Authentication />
      },
      {
        //elérési út: /dashboard
        path: "dashboard",
        //A ProtectedRoute komponens csak akkor engedi át a felhasználót,
        ha azonosítva van.
        element: (
          <ProtectedRoute>
            <Dashboard />
          </ProtectedRoute>
        )
      }
    ]
  }
]);

//A React gyökér létrehozása.
ReactDOM.createRoot(document.getElementById("root") as HTMLElement).render(
  //Az AuthProvider felel az autentikáció követéséért, értesíti az app többi
  részét, ha változás történt.
  <AuthProvider>
    {/* Az InfoProvider az alkalmazás indulásakor letölti az üzemeltető
    adatait, és elérhetővé teszi azt az app többi részén. */}
    <InfoProvider>
      {/* A RouterProvider a router objektum alapján elvégzi az elérési
      útnak megfelelő állapot beállítását. */}
      <RouterProvider router={router} />
    </InfoProvider>
  </AuthProvider>
);

```

A „pages” könyvtár

Olyan React komponenseket tartalmaz, amelyek oldalszintű funkciókat látnak el. Gondolhatunk itt például a kezdőlapra, „A rendszerről” oldalra vagy akár a „Személyes oldalra” is.

A „components” könyvtár

Ebben a könyvtárban olyan React komponenseket találhatók, amelyeket a projekt során akár többször is felhasználunk, de nem oldalszintű kompozíciók.

Példa egy komponensre:

```
export default function Signatures({ context }) {
  //A környezeti változóból build idején string literálra fog cserélődni ez a
  kifejezés.
  const backendURL = import.meta.env["PUBLIC_BACKEND_URL"];
  return (
    <div className="certificate-signatures">
      {/* A kontextusban található aláírások mindegyikéhez egy HTML
      elemhalmazt rendelünk. */}
      {/* A divben található egy kép, amely az aláírás, illetve az aláíró
      neve és titulusa. */}
      {context.signatures
        .sort((a, b) => a.order - b.order)
        .map((e, i) => (
          <div key={i}>
            <img
src={` ${backendURL}/storage/images/signatures/${e.img}`} alt="" />
            <h3>{e.name}</h3>
            <h4>{e.title}</h4>
          </div>
        ))}
    </div>
  );
}
```

A „styles” könyvtár

A könyvtárban CSS fájlokat találhatók, amelyeket az oklevelek formázását, illetve a felhasználói felületeken használt design rendszer felülírását teszik lehetővé.

A „lib” könyvtár

A sajátfejlesztésű eljárásokat tartalmazza, amelyek a program működéséhez szükséges részfeladatok végrehajtását írják le, pl.: API kérések lebonyolítása, egyedi adatszolgáltatók („provider”). Lényegében ide kell elhelyezni minden olyan kódot, amely funkcionalitásából adódóan az előbb tárgyalt mappák egyikébe sem illik.

Tesztelés

A szoftverfejlesztési folyamat során az egyik legfontosabb lépés a tesztelés, mivel ez segít garantálni, hogy a késztermék megfeleljen a tervezett követelményeknek és elvárásoknak. A tesztelés révén a fejlesztők képesek észlelni és korrigálni a szoftverhibákat, mielőtt azok kárt okoznának a végfelhasználók számára, ezzel javítva a szoftver megbízhatóságát és teljesítményét. A tesztelési folyamat során a fejlesztők az összes lehetséges forgatókönyvet végigjátszva kiderítik, hogy a szoftver működik-e a tervezett módon, és hogy nem okoz-e nem várt hibákat. A tesztelés lehetővé teszi a fejlesztők számára, hogy azonosítsák a szoftver sebezhetőségeit és korrigálják azokat, mielőtt a szoftver nyilvánosan elérhetővé válna. A tesztelési folyamat a fejlesztési költségek csökkentéséhez is hozzájárul, mivel a hibák korai felfedezése segít elkerülni a későbbi, magasabb költségeket igénylő javításokat. A tesztelés lehetővé teszi a szoftvertermék minőségének folyamatos javítását és ellenőrzését a fejlesztési ciklus minden szakaszában. A tesztelési folyamat által megszerzett visszajelzések segíthetnek a fejlesztőknek az új funkciók, javítások vagy változtatások prioritásainak meghatározásában.

A projekt fejlesztése során manuális tesztelést végeztem, ez azt jelenti, hogy az alkalmazások elvárt működését gondosan ellenőriztem szimulálva az összes lehetséges esetet. A továbbiakban egy-egy példa segítségével bemutatom ennek lépéseit külön a backend, külön a frontend rendszer esetén.

Backend tesztelés

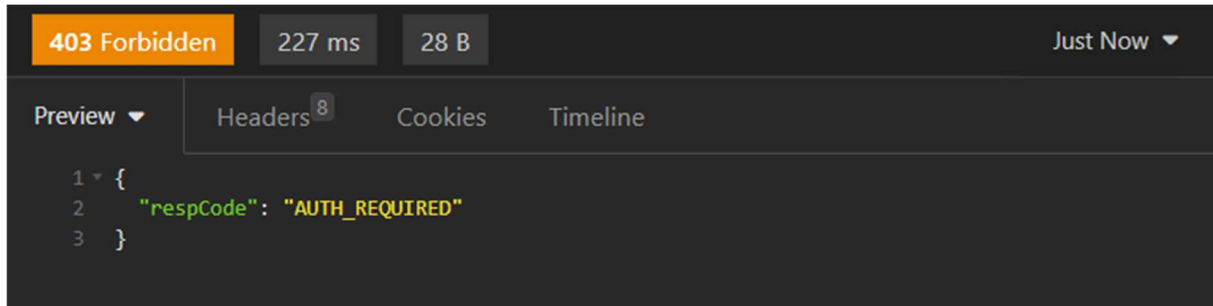
A backend tesztelésének legfontosabb lépése az, hogy ellenőrizzük az egyes API végpontok megfelelő, stabil és elvárt működését. A stabilitás mellett kulcs szerepet játszik a biztonság is, azaz megfelelően kell tesztelni az autentikációt és az autorizációt végző kódrészeket is. Az ezen feladatokat ellátó kód úgynevezett „middleware” architektúrával lett elkészítve, így ezeket egy új végpont kialakításakor könnyen újrafelhasználhatjuk, amely nem csak a kódolási folyamatot gyorsítja, hanem a tesztelésit is, hiszen ezen részeket elég egyszer tesztelni.

Egy azonosítást igénylő végpont végrehajtása a következő módon írható le:

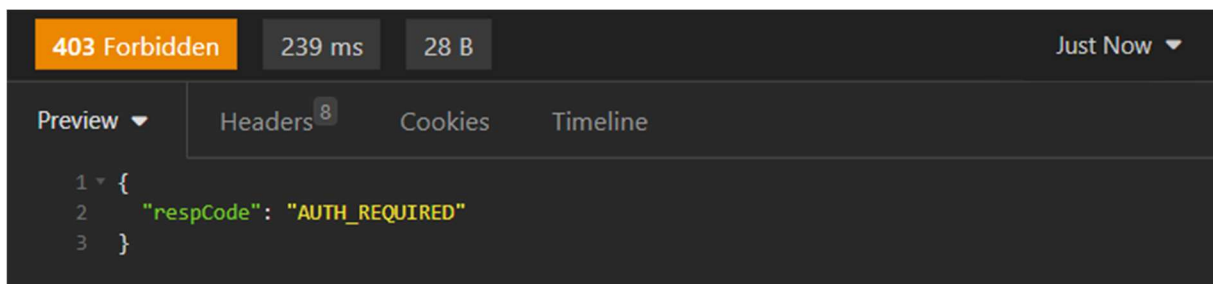
1. A rendszer elkezd feldolgozni a beérkező HTTP kérést.
2. Az „AuthRequired” middleware végrehajtása elkezdődik.
 - a. Ha a kérés nem tartalmazza a megfelelő token-t a HTTP kérés fejlécében, akkor 403-as „Hozzáférés megtagadva” státusszal lezárja a kérést.
 - b. Ha a kérés tartalmaz egy token-t a HTTP kérés fejlécében, azonban az nem érvényes, akkor az előzőhöz hasonló választ küld a szerver.
 - c. Ha érvényes a token, akkor a kérés objektum populálásra kerül az azonosított személy adataival és a végrehajtás folytatódik.
3. Ha a kérés nem került eddig lezárásra, azaz az autentikáció sikeres volt, akkor elkezdődik a végponthoz tartozó kód végrehajtása.

Példa: A saját oklevelek listázására használt végpont tesztelése

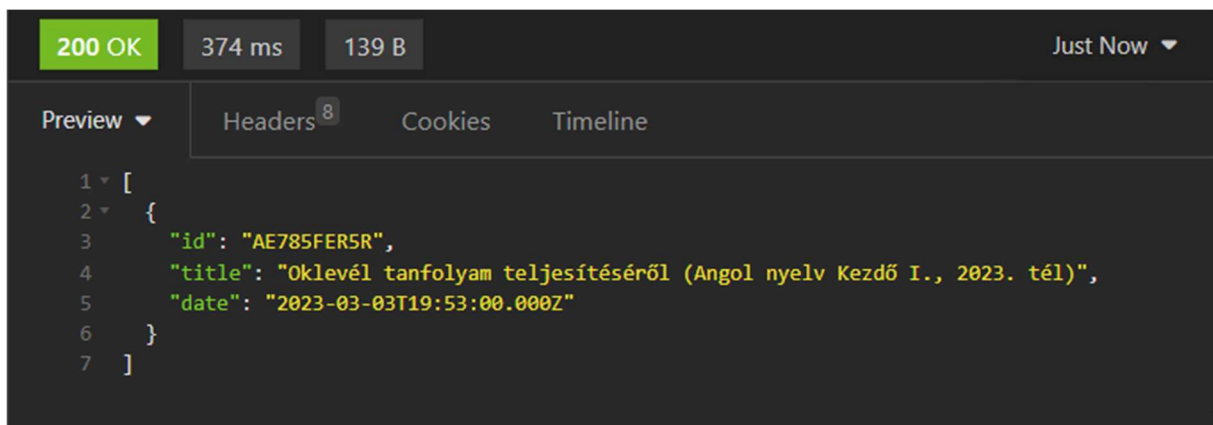
A következőkben az azonosított személy saját okleveleinek listázására használt API végpontot fogom tesztelni, egyúttal az autentikációs kódrészletet is. Az első két esettel csak az autentikációs folyamatot tesztelem, míg a harmadik esettel a végpont implementálását is. A teszteléshez az Insomnia REST API kliensprogramot használtam, a képernyőképeken látszik a HTTP státusz narancssárga, illetve zöld keretben, a válaszdíő, a válasz mérete, illetve maga a backend válasza is.



4. ábra 1. eset: A végpont token nélküli hívása



5. ábra 2. eset: A végpont hívása érvénytelen token használatával



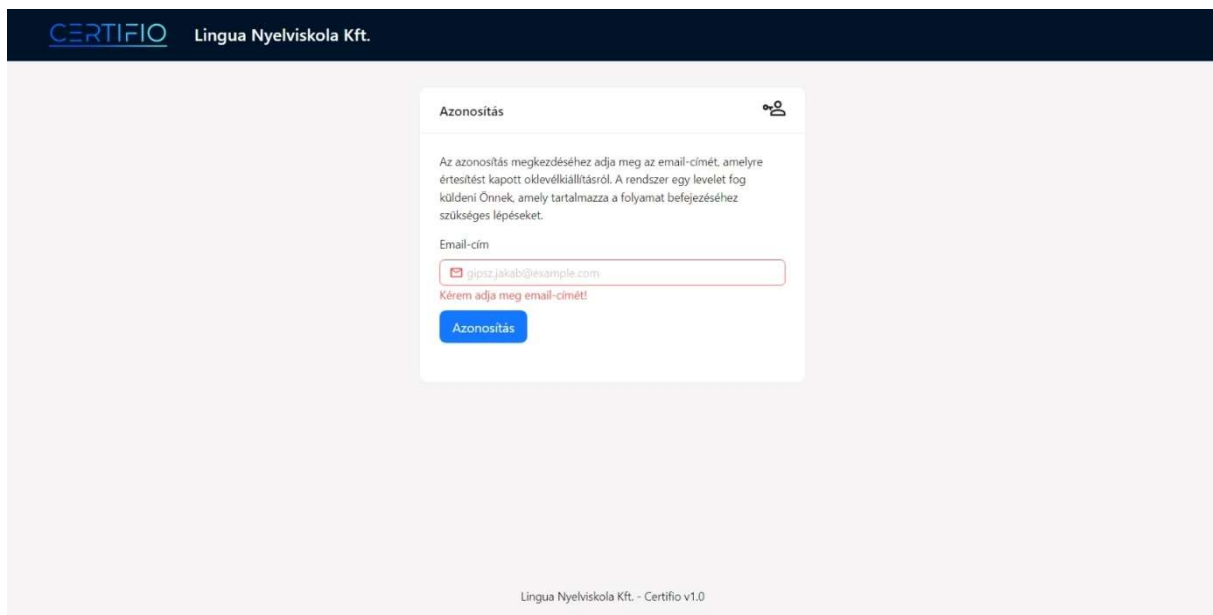
6. ábra 3. eset: A végpont hívása érvényes token használatával

Frontend tesztelése

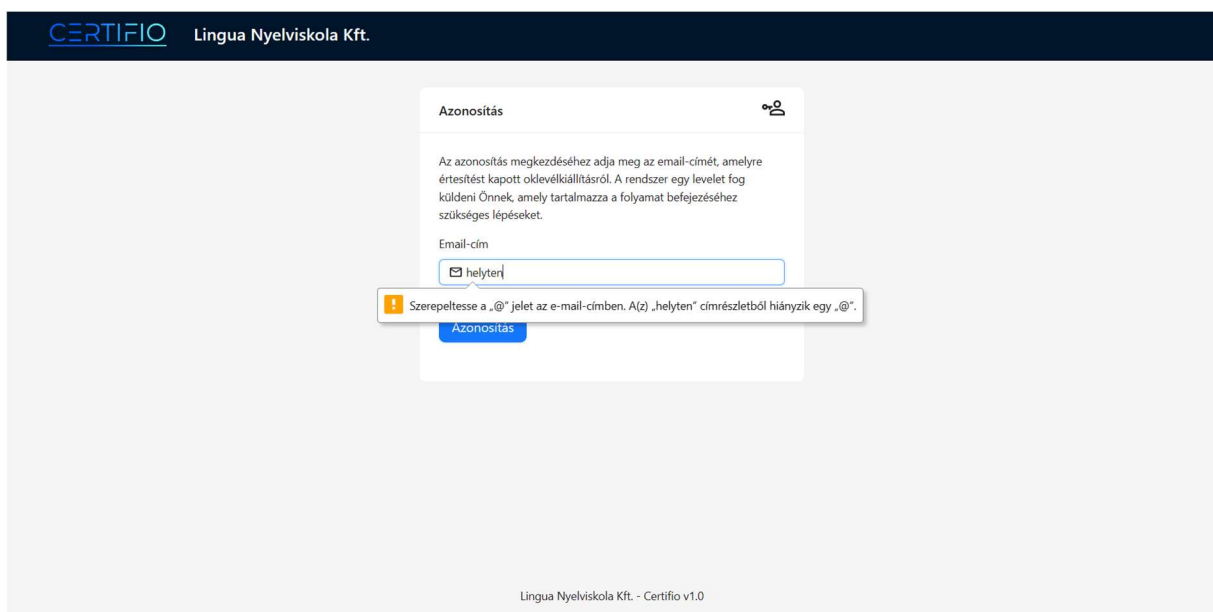
A frontend tesztelése manuálisan történt, az egyes komponensek elkészítése után, ezzel megteremtve egy jól nyomonkövethető munkafolyamatot. A frontend tesztelése során az elsődleges cél az űrlapok működésének ellenőrzése, továbbá másodrendű feladatnak tekinthető a védett oldalakról történő elnavigálás azonosítás hiányában, valamint az űrlapok tartalmi helyességének kliens oldali ellenőrzése. Az utóbb megemlített feladatok, habár fontosak, és nagyban hozzájárulnak a felhasználói élmény javításához, működésük nem kritikus, hiszen az adatok tényleges validációja, illetve a hozzáférés engedélyezése a backenden történik.

Példa: Azonosító űrlap működése

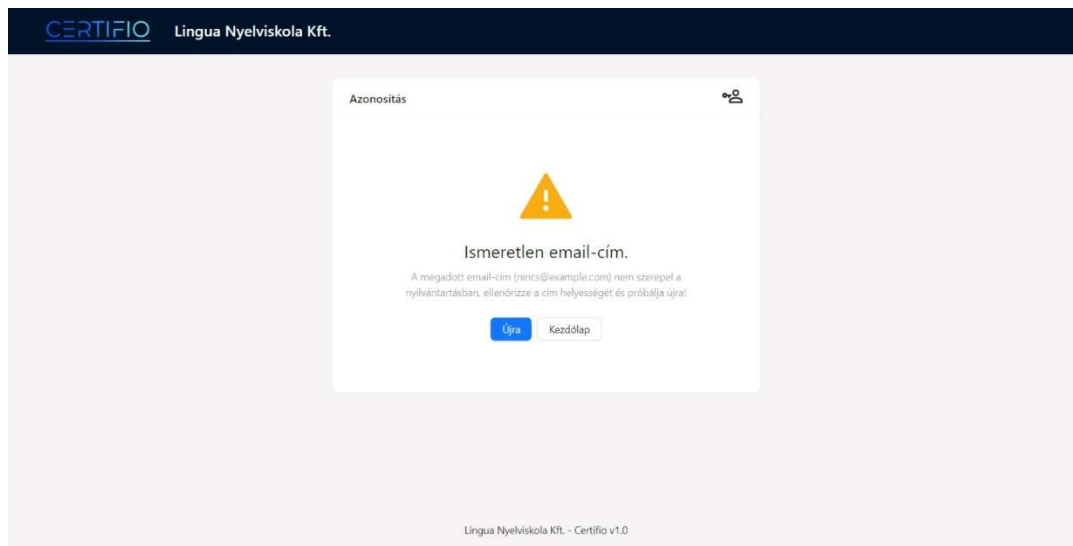
A következőkben az azonosításhoz használt űrlap viselkedését fogjuk vizsgálni. Az első két esetben az űrlapnak csak a kliens oldali viselkedését fogjuk vizsgálni, míg a további esetekben a backend által adott válaszokra produkált viselkedést fogjuk ellenőrizni.



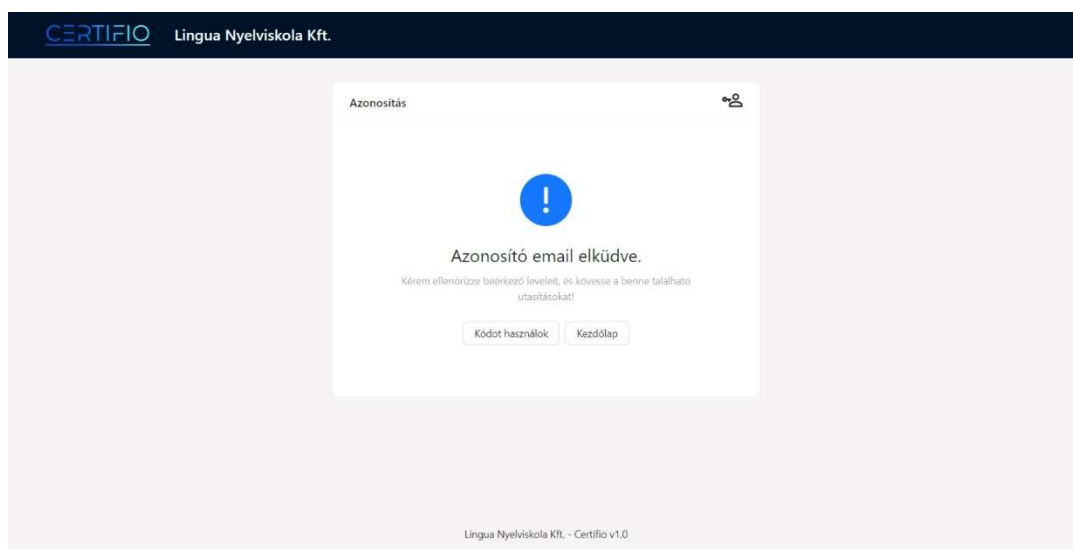
7. ábra 1. eset: Az űrlap nem küldhető be üresen



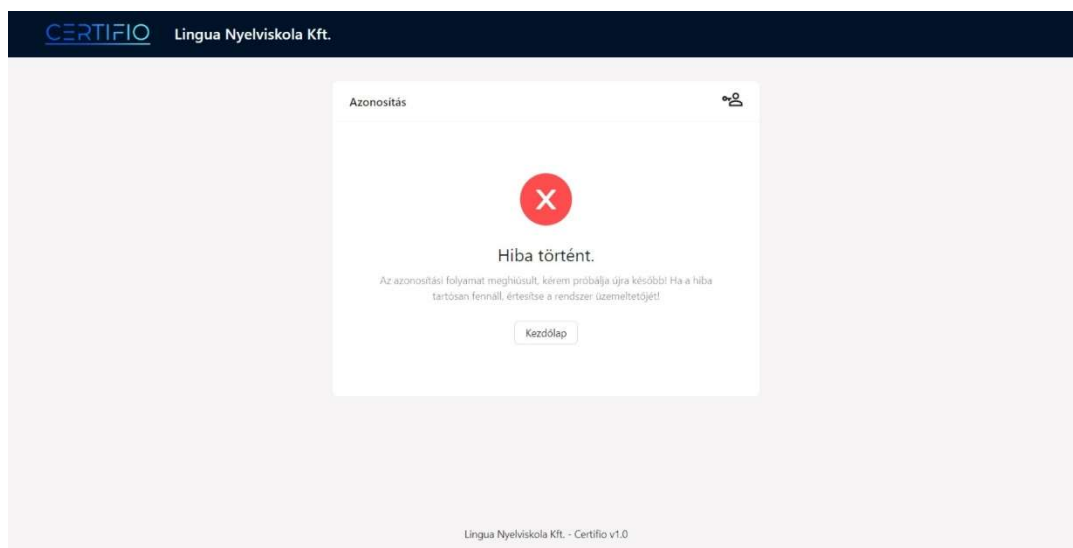
8. ábra 2. eset: Az űrlapnak formailag helyes adatot kell megadni



9. ábra 3. eset: Az űrlapon megadott adat formailag helyes, azonban nem szerepel az adatbázisban



10. ábra 4. eset: Az űrlapon megadott adat helyes



11. ábra 5. eset: Az űrlapon megadott adat helyes, azonban más hibából kifolyólag a művelet nem folytatható

Továbbfejlesztési lehetőségek

Adatszerkezettel és jogosultságokkal kapcsolatos revízió a SaaS architektúra előkészítéséhez

A projekt hosszútávon úgy lehetne profitábilis, hogy ha a napjainkban is népszerű nyílt forráskódú, de előfizetési opcióval is rendelkező üzleti modellt választanánk. Lényegében arról lenne szó, hogy a szoftver maga ugyanúgy ingyenesen használható marad, azonban ilyenkor az üzemeltetésért a felhasználó felel, garanciát a készítő nem adnak a program helyes működésére, illetve lenne lehetőség Software-as-a-Service konstrukcióban akár exkluzív többlet funkcionalitással rendelkező igénybevételre is. Ilyenkor a felhasználó egy előre meghatározott havi díj ellenében használhatja a központilag üzemeltetett szoftvert, amire a készítő garanciát és felelősséget is vállalnak. Az ilyen architektúrára való áttéréshez a jelenlegi adatszerkezet átdolgozása lenne szükséges, mondjuk érdemes lenne egy új absztrakciós réteget bevezetni, amely egy program példányban több különböző szervezet tárolását tenné lehetővé több különböző szintű felhasználóval. A megvalósítás során arra kell ügyelni, hogy egy szervezet csak a saját adatait (okleveleit, oklevéltörzseit és személyeit) tudja lekérdezni, módosítani. Továbbá szükségesnek látom újabb szerepkörök bevezetését a jelenlegi adminisztrátori szerepkör megbontásával, így biztosítva a biztonságosabb működést.

Testreszabható tulajdonságok hozzárendelése személyekhez, oklevelekhez

A jövőben érdemes lenne az adatszerkezeteket úgy bővíteni, hogy a személyekről ne csak az előre meghatározott adatokat lehessen tárolni, hanem a szervezet munkájától vagy az oklevelektől függően indokolt tulajdonságokat is.

Az oklevéltörzsekhez érdemes lenne egyedi kontextus kialakítást biztosítani, ezáltal szélesebb körben lehetne használni az oklevélkiállító funkciót. Például, ha lehetőség lenne, mondjuk egy egyedi érdemjegy mezőt létrehozni, akkor a programmal már bizonyítvány jellegű dokumentumokat is lehetséges volna kiállítani.

WYSIWYG szerkesztő

Egy „What You See Is What You Get” szerkesztő implementálása nagyban megkönnyítené az újabb oklevéltörzsek létrehozását. Mivel jelenleg egy oklevéltörzs szerkezetét JSON fájlban tároljuk, és az abból keletkezett JavaScript objektumokból építjük fel az okleveleket, ezért egy ilyen szerkesztő implementálásakor csak a szerkesztő felületre kellene fókuszálni, az adatszerkezetre nem. A React.js komponens jellegű filozófiája miatt az oklevelek megjelenítéséhez használatos kódrészek újrahasználatosak a szerkesztő kialakítása során.

Felhasználói dokumentáció

A program rövid ismertetése

A Certifio egy webes technológiákon alapú program, amelynek célja digitális oklevelek kibocsájtása és nyilvántartása, az elnevezés az angol „certificate” (oklevél, tanúsítvány) és a latin „fio” (készítek, csinállok) szavakból ered. A program ideális lehet oktatási intézményeknek a tanulók motiválására, elismerésére. A program használata ingyenes, azonban az üzemeltetésről a felhasználónak kell gondoskodnia, a jövőben Software-as-a-Service konstrukcióban, havi díj ellenében elérhetővé fog válni egy központilag menedzselt változat is.

A dokumentáció során egy képzeletbeli intézmény, a Lingua Nyelviskola Kft. rendszere kerül üzembe helyezésre, illetve felhasználásra.

Üzembe helyezés

Az applikáció két részből áll, egy frontend és egy backend részből, amelyeket az adott környezetnek megfelelően kell paraméterezni, majd lefordítani. A futtatáshoz Node.js-re van szükség, amely elérhető mind Windowson, Linuxon és Macintoshon.

A frontend üzembe helyezése

A frontend applikáció forráskódjából egy tömörített, optimalizált statikus állományokból álló kiadást kell készítenünk, amely már tartalmazza a backend rendszerünk internetes elérési útját. A frontend applikáció elkészítéséhez szükségünk lesz aktív internetkapcsolatra, valamint a Node.js legfrissebb LTS kiadására.

1. A frontend munkakönyvtárban adjuk ki az „npm run install” parancsot, amely automatikusan telepíti a projekt által használt harmadik féltől származó nyílt forráskódú könyvtárakat, majd az „npm run build” parancsot.
2. A munkakönyvtárban hozzunk létre egy „.env” nevű, kiterjesztés nélküli szöveges fájlt. Ebben adjuk meg a backend rendszer internetes elérési útját a „PUBLIC_BACKEND_URL=” szöveg után. Ügyeljünk arra, hogy az elérési út végén ne legyen „/”!
3. Adjuk ki az „npm run build” parancsot, amely egy optimalizált, „production-ready” kiadást fog létrehozni az applikációból a „dist” mappába.
4. A „dist” mappa tartalmát kell egy tetszőleges webservert segítségével szolgáltatni, arra ügyelve, hogy a webservice fel legyen készítve a kliens oldali forgalomirányításra, azaz a „nem található” elérési utak estén az „index.html” állományt szolgáltatassa.
5. Ha megváltozna a backend rendszer elérési útvonala, akkor a 2. lépéstől kezdve meg kell ismételni a folyamatot.

A backend üzembe helyezése

A backend használatához a TypeScript forráskódot JavaScriptre kell fordítanunk, amelyet már tud értelmezni a Node.js futtató környezet. A futtatáshoz szükségünk lesz aktív internetkapcsolatra, a Node.js legfrissebb LTS kiadására, valamint egy MySQL szerverre és egy SMTP szerverre, vagy egy Amazon Web Services Simple Email Service fiókra hitelesítő adatokkal.

1. A backend munkakönyvtárban adjuk ki az „npm run install” parancsot, amely automatikusan telepíti a projekt által használt harmadik féltől származó nyílt forráskódú könyvtárakat.
2. A munkakönyvtárban található „.env.template” fájl alapján hozzunk létre egy „.env” fájlt, amelyben a konfigurációt fogjuk tárolni.
3. Adjuk meg a szükséges adatokat
 - a. DATABASE_URL: A MySQL kapcsolat létrehozásához szükséges adatok, a pontos szintaxis a kommentben található linken elérhető.
 - b. FRONTEND_URL: A frontend rendszer internetes elérési útvonala, az emailés azonosítás hivatkozásaihoz szükséges.
 - c. SYSTEM_EMAIL: A megadott címről fogja küldeni a rendszer az email üzeneteket.
 - d. JWT_SECRET: Az azonosításhoz szükséges tokenek aláírásához használt titkos kulcs, használjon egy véletlenszerű karakterláncot. A kommentben található link segítségével generálhat ilyet!
 - e. EMAIL_MODE: „SMTP” vagy „AWS” attól függően, hogy hogyan szeretnénk emaileket küldeni.
SMTP használata esetén
 - f. SMTP_CONNECTION_URL: Ha SMTP protokollt szeretnénk használni, akkor itt kell megadni az SMTP szerver adatait, a pontos szintaxis a kommentben található linken elérhető.**AWS Simple Email Service használata esetén**
 - g. AWS_REGION: A fiókunk által használt AWS felhős régió. pl.: eu-central-1
 - h. AWS_ACCESS_KEY_ID: A hozzáférési kulcs azonosítója.
 - i. AWS_SECRET_ACCESS_KEY: A hozzáférési kulcs titkos azonosítója.
4. A MySQL szerveren futtassuk a „db_install.sql” fájlban található utasításokat, majd az „admin.sql” fájlban található parancsot úgy, hogy a saját vezetéknévünket, keresztnévünket és email-címünket helyezzük bele.
5. A „storage” mappában hozzuk létre az „owner.json” fájlt, amely az üzemeltető szervezetet adatait fogja tartalmazni. Itt egy minta a fájl felépítésére:

```
{
  "name": "Lingua Nyelviskola Kft.",
  "address": {
    "country": "Magyarország",
    "postalCode": "1014",
    "city": "Budapest",
    "street": "Valami utca 5. I/1."
  },
  "contact": {
    "name": "Pelda Peter",
    "email": "peter.pedla@lingua.hu"
  },
  "website": "https://example.com"
}
```

6. Adjuk ki az „npm run start” parancsot, a szolgáltatás a 8080-as porton fog futni.

Futtatás Dockerben

Tapasztaltabb felhasználók figyelmébe ajánljuk a Dockerben történő futtatást, amelyhez a projekt rendelkezik Dockerfile állománnyal, amely a konténerizációs utasításokat tartalmazza. Amennyiben konténerben futtatja a backendet, akkor hasonlóan kell eljárni, azonban a konfigurációt környezeti változók segítségével kell megadni, illetve érdemes felcsatolni egy mappát a konténer „/app/storage” elérési útjára.

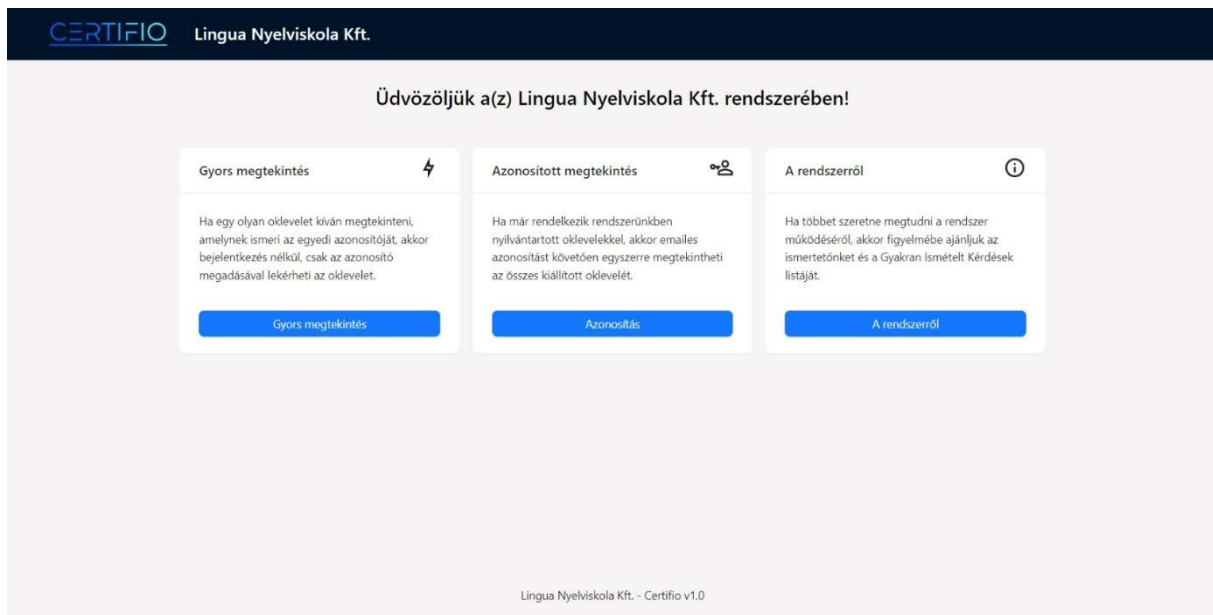
Használati útmutató

A program használata

Felhasználóként lehetőségünk van megtekinteni a kezdőlapot, lekérdezni okleveleket, ha ismerjük azok számát, valamint azonosítási folyamatot indítani.

A kezdőlap

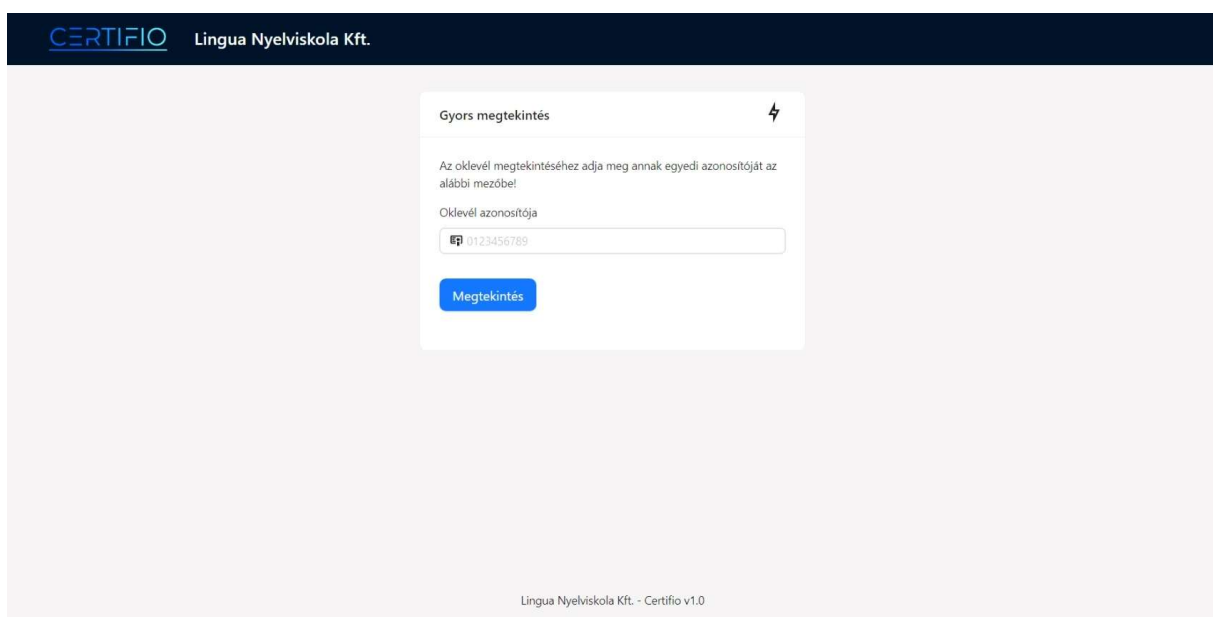
A kezdőlapon három funkció közül választhatunk, a „Gyors megtekintés”, „Azonosított megtekintés” és „A rendszerről” oldalak érhetőek el. A fejlécen, középen és a láblécen megjelenik az üzemeltető neve.



12. ábra A kezdőlap

Gyors megtekintés

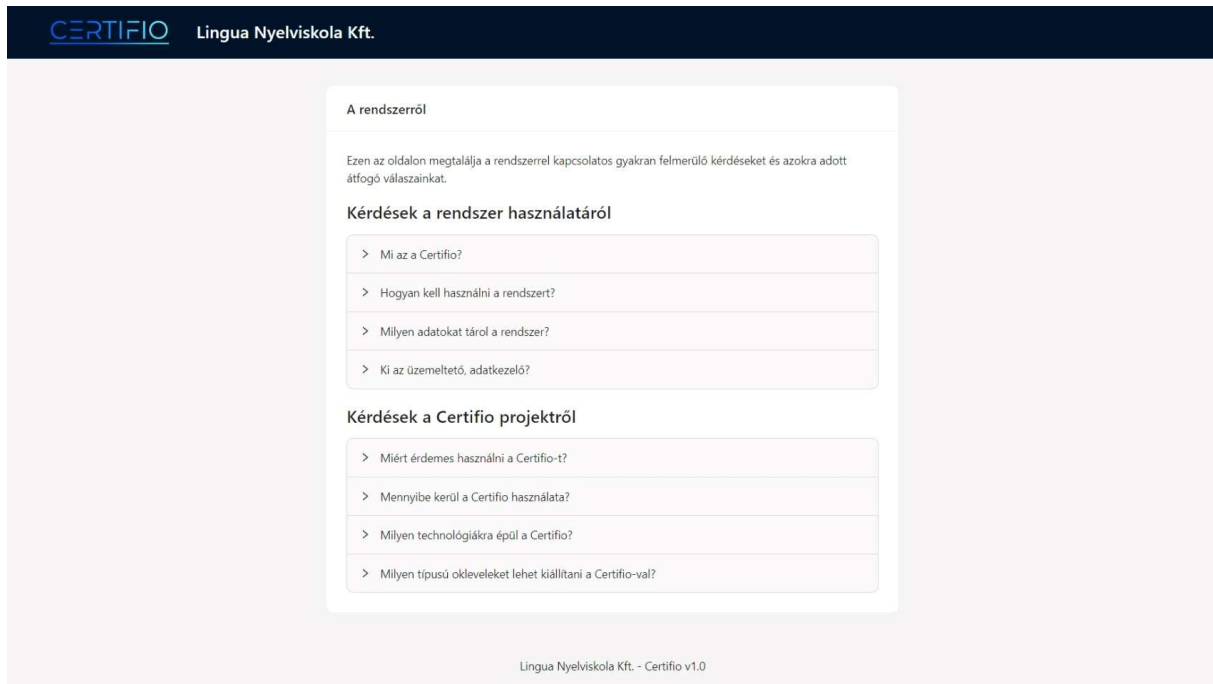
Ha ismerjük a megtekintendő oklevél azonosítóját, akkor annak begépelésével lekérdezhetjük azt. Ha helytelen azonosítót adunk meg, akkor a program hibaüzenettel jelzi azt.



13. ábra Gyors megtekintés

A rendszerről

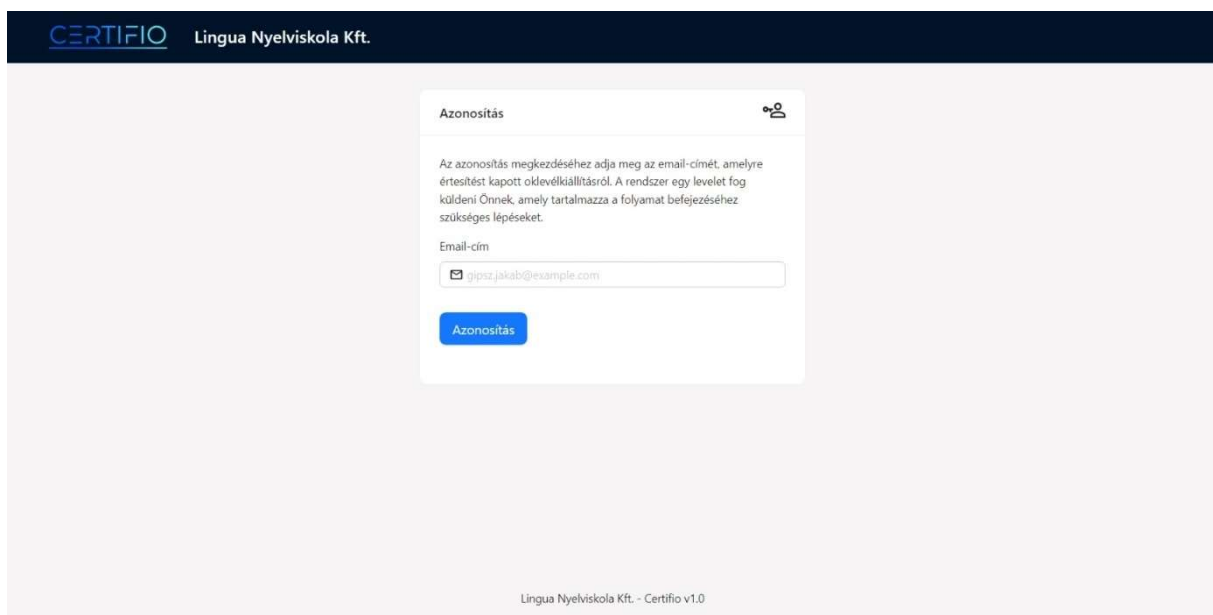
Ezen az oldalon a projektről, illetve a program példány üzemeltetőjéről találhatunk információkat lenyitható dobozokban tárolt kérdések formájában.



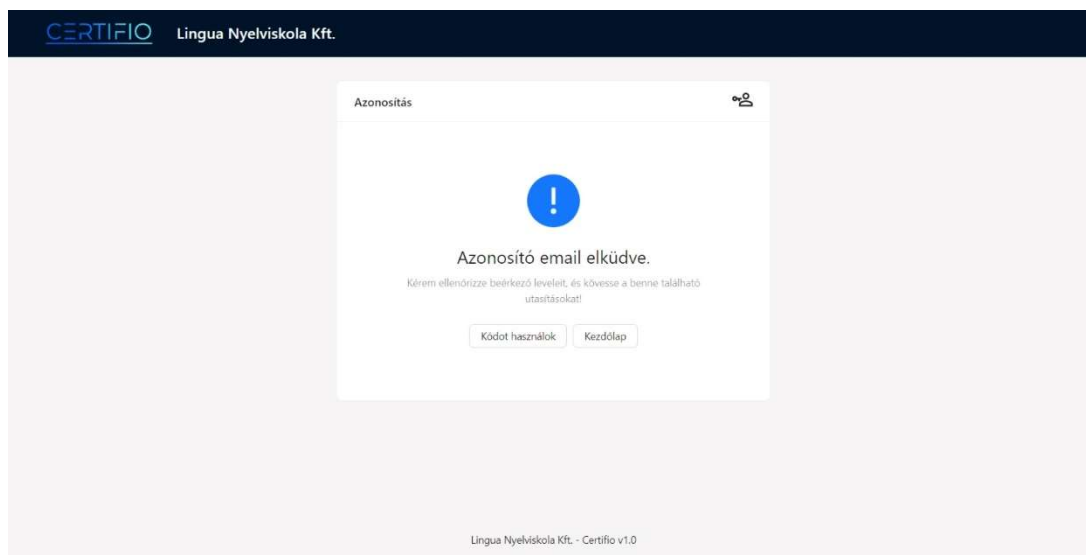
14. ábra A rendszerről

Azonosított megtekintés

Az azonosítási folyamat megkezdéséhez meg kell adnunk az email-címünket. Ha a nyilvántartásban nem szerepel az email-cím, akkor a program jelzi azt számunkra. Ha szerepel, akkor egy emailt fogunk kapni, amelyben az azonosítás befejezéséhez szükséges lépéseket találjuk. Ha másik eszközön olvassuk a levelezést, akkor a négyjegyű kód használatával is beléphetünk.



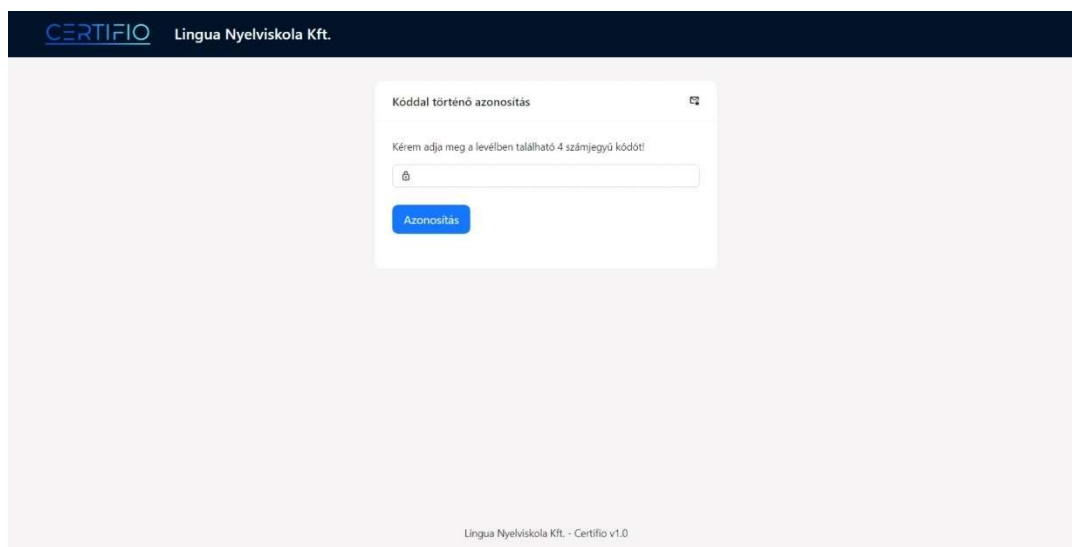
15. ábra Azonosítási folyamat



16. ábra A rendszer tájékoztatja a felhasználót az azonosító levél kiküldéséről



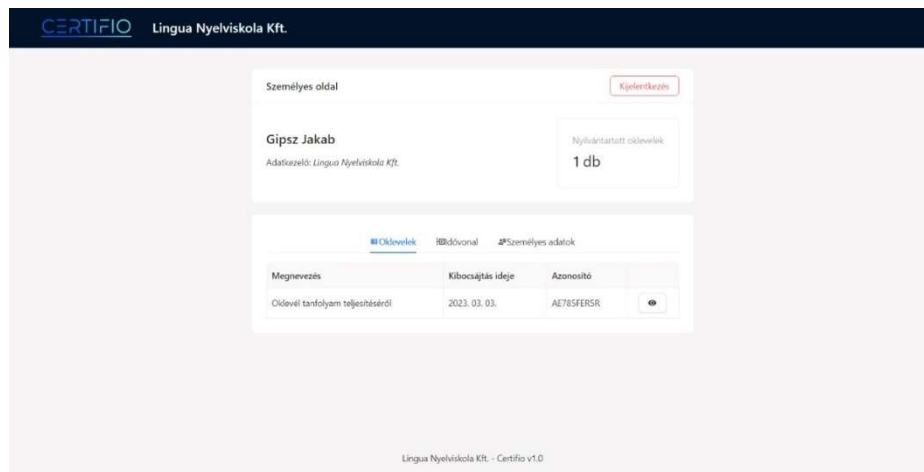
17. ábra Az azonosításra szolgáló levél



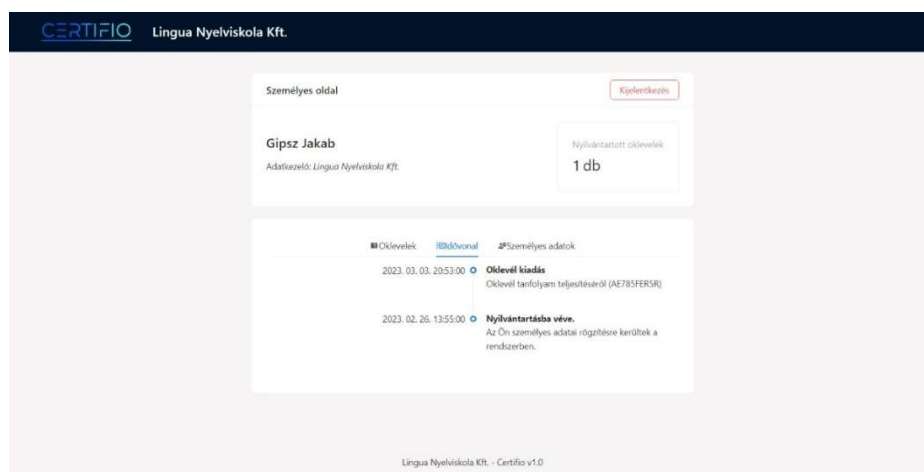
18. ábra Kóddal végzett azonosítás

Személyes oldal

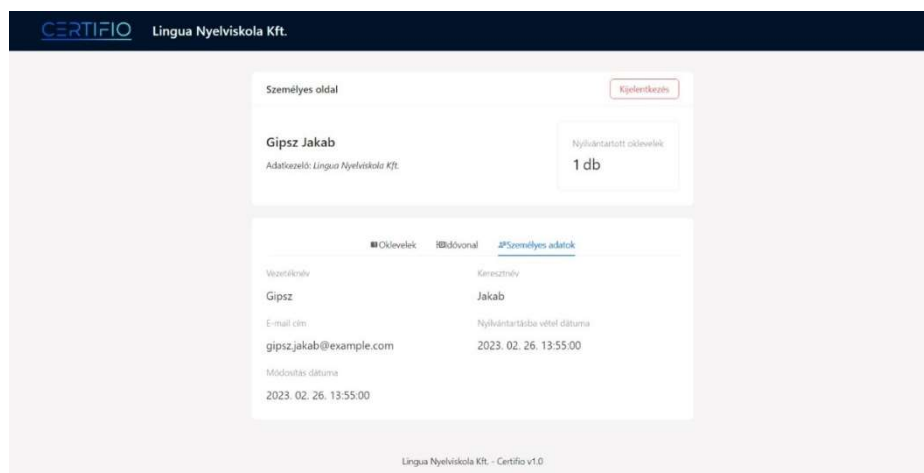
A sikeres azonosítás követően a felhasználó a „Személyes oldal” elnevezésű programrészbe kerül. Itt megtekintheti okleveleit, egy idővonalat a rendszerben tárolt eseményeiről, valamint személyes adatait is. A felhasználó munkamenete a „Kijelentkezés” gomb megnyomásával vagy a böngészőablak bezárásával fejezhető be.



19. ábra Személyes oldal: Oklevelek menüpont



20. ábra Személyes oldal: Idővonal menüpont



21. ábra Személyes oldal: Személyes adatok menüpont

Oklevél megtekintés

Ha a „Gyors megtekintés” funkció használatával megadtuk egy oklevél azonosítóját, vagy „Személyes oldalról” a megtekintés gombra kattintottunk, akkor az alkalmazás betölti a kért oklevelet. Ha kisseretnénk nyomtatni az oklevelet, akkor a beállítások között érdemes engedélyezni a háttérgrafikák nyomtatását.



22. ábra Oklevél

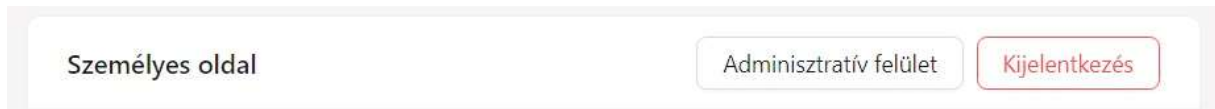


23. ábra Az oklevél nyomtatás Chrome böngészőben

A program használata, mint adminisztrátor

Belépés az adminisztrátori felületre

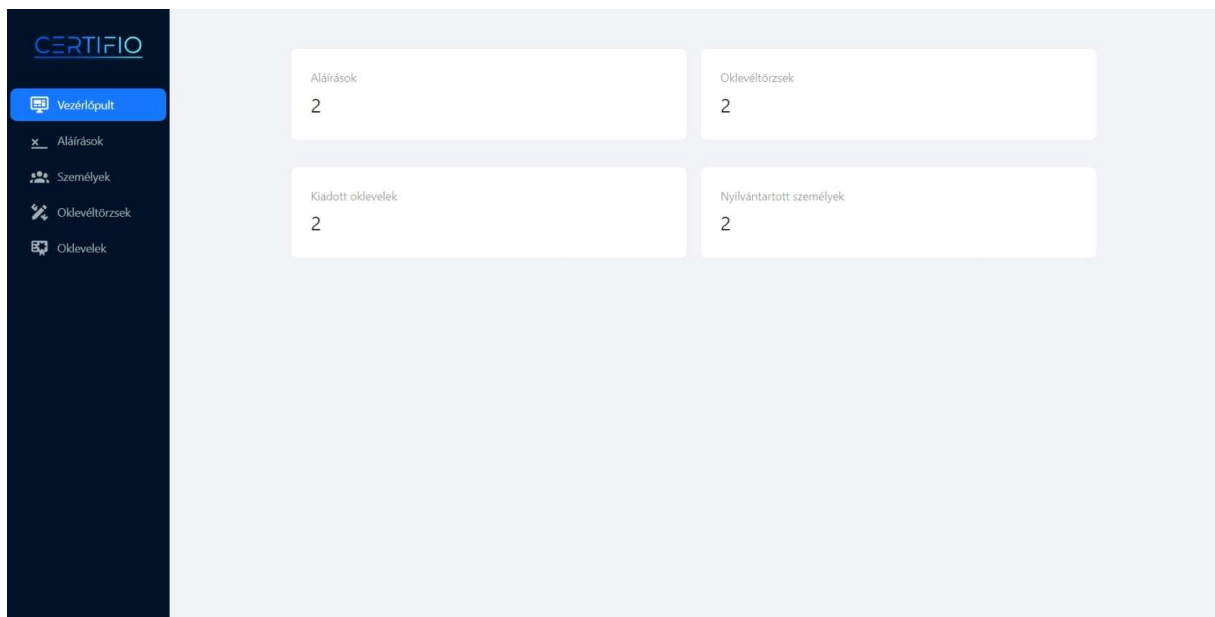
Ha rendelkezünk adminisztrátori jogosultsággal, akkor a „Személyes oldalon” a kijelentkezésre szolgáló gomb mellett megjelenik egy „Adminisztrációs felület” elnevezésű gomb, amellyel beléphetünk.



24. ábra Belépés az adminisztrációs felületre

Kezdőlap

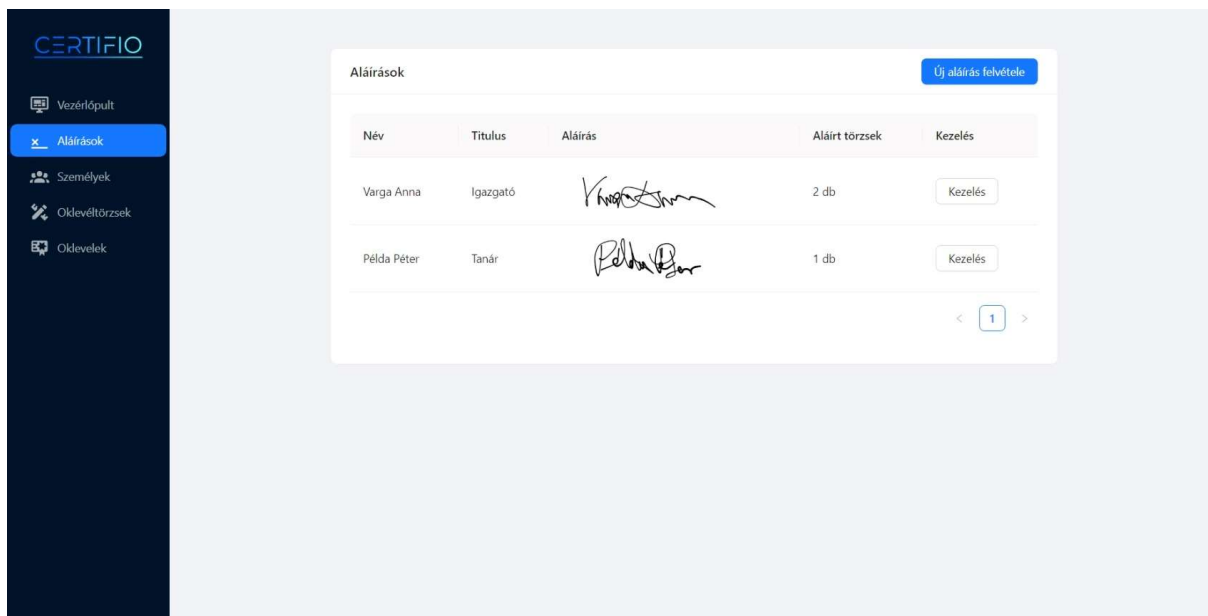
A kezdőlapon néhány számadatot láthatunk, amely tájékoztatást ad a rendszerben nyilvántartott személyekről, aláírásokról, oklevéltörzsekről és oklevelekről.



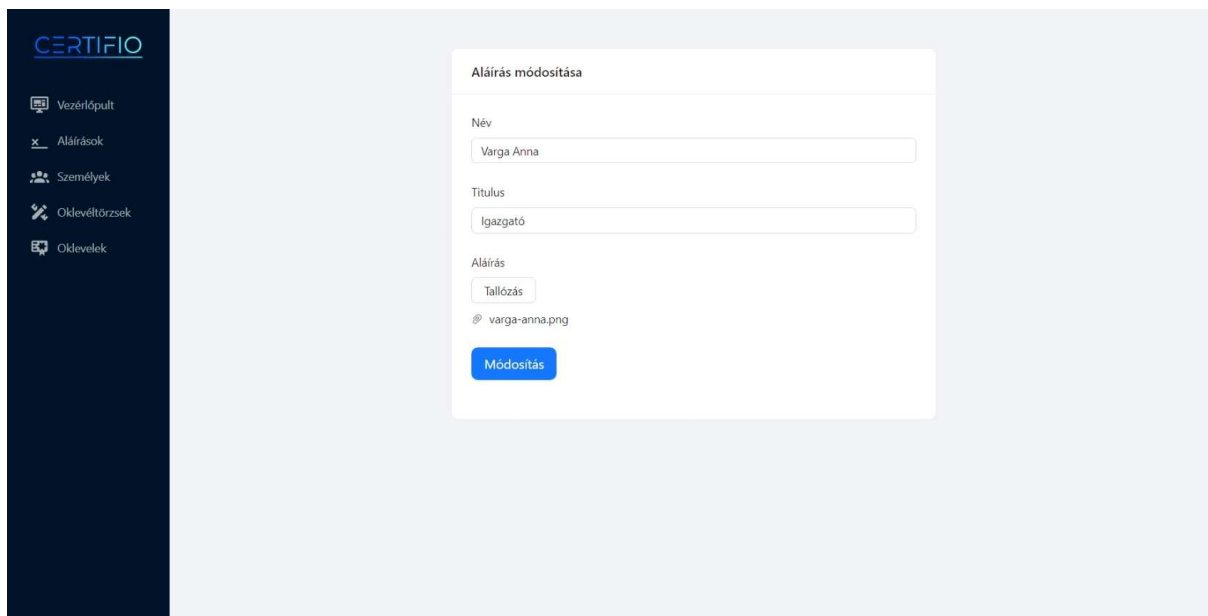
25. ábra Az adminisztrációs kezdőlap

Aláírások

Ebben a menüpontban megtekintheti a rendszerben található aláírásokat, módosíthatja azokat, illetve új aláírást is rögzíthet. A módosításra és rögzítésre használt űrlap megegyezik, ha már létező aláírásnál kattint a „Kezelés” gombra, akkor módosítani fog és az űrlap az aláírás jelenlegi adataival töltődik be, ha a jobb felső sarokban található gombot használja, akkor új aláírást fog rögzíteni, így az űrlap üresen jelenik meg.



26. ábra Aláírások oldal



27. ábra Egy meglévő aláírás módosítása

Személyek

Ebben a menüpontban megtekintheti a rendszerben található személyeket, módosíthatja azokat, illetve új személyt is rögzíthet. A módosításra és rögzítésre használt űrlap itt is megegyezik, és ugyanúgy működik, mint az aláírásoknál. Biztonsági okokból adminisztrátori jog nem adható, ehhez közvetlenül a MySQL szerveren kell kiadnunk a megfelelő parancsot.

ID	Vezetéknév	Keresztnév	Email	Adminisztrátor	Felvétel	Kezelés
1	Gipsz	Jakab	gipsz.jakab@example.com	igen	2023. 02. 26. 13:55:00	Kezelés
2	Kerekes	Géza	kerekes.geza@example.com	nem	2023. 03. 10. 19:48:00	Kezelés

28. ábra Személyek

Személy módosítása

Vezetéknév
Gipsz

Keresztnév
Jakab

Email-cím
gipsz.jakab@example.com

Módosítás

29. ábra Egy személy adatainak módosítása

Oklevéltörzsek

Ebben a menüpontban megtekintheti a rendszerben található oklevéltörzseket, módosíthatja azokat, illetve újat is készíthet. Az oklevéltörzsek sablonként szolgálnak a jövőben kiadni kívánt oklevelek számára. A sablon elkészítéséhez meg kell adni egy az adminisztrációhoz használt nevet, egy az oklevelet megkapók részére látható nevet, az oklevelet aláírók listáját, valamint magát az oklevél szerkezetét leíró „.json” állományt. A módosításra és rögzítésre használt űrlap itt is megegyezik.

Név	Publikus név	Aláírásk	Kiadott oklevelek	Kezelés
23TÉL-CSOP1-ANGOL-K1	Oklevél tanfolyam teljesítéséről (Angol nyelv Kezdő I.)	2 db	2 db	Kezelés
23TAV-CSOP1-ANGOL-K2	Oklevél tanfolyam teljesítéséről (Angol nyelv Kezdő II.)	2 db	0 db	Kezelés

30. ábra Oklevéltörzsek listája

Oklevéltörzs létrehozása

Név

23TAV-CSOP2-NÉMET-K1

Publikus név

Oklevél tanfolyam teljesítéséről (Német nyelv Kezdő I., 2023. tavasz)

Aláírók

Példa Péter (Tanár) × Varga Anna (Igazgató) ×

Szerkezet

Tallózás

okleveljson

Módosítás

31. ábra Oklevéltörzs létrehozása

Oklevelek

Az oklevelek menüben lehetőségünk van új okleveleket kiállítani, ehhez az oklevéltörzset kell megadni, illetve a személyek email-címét. Lehetőségünk van .csv állományok importálására is.

CERTIFIO

- Vezérlőpult
- Aláírások
- Személyek
- Oklevéltörzsek
- Oklevelek**

Oklevélkiadás

Oklevéltörzs
23TAV-CSOP1-ANGOL-K2

Személyek
Gipsz Jakab (gipsz.jakab@example.com)

Keltetés dátuma
2023-03-06 16:00:00

Email értesítő küldése
☒

Kiadás

32. ábra Oklevélkiadás

Mellékletek

Frontend applikáció

Az applikáció letölthető az alábbi publikus GitHub repositoryból:
<https://github.com/davidszab/certifio-frontend>

Backend applikáció

Az applikáció letölthető az alábbi publikus GitHub repositoryból:
<https://github.com/davidszab/certifio-backend>

Ábrajegyzék

1. ábra Az adatbázis felépítése.....	12
2. ábra A jogosultságok	19
3. ábra A jogosultságok	20
4. ábra 1. eset: A végpont token nélküli hívása.....	30
5. ábra 2. eset: A végpont hívása érvénytelen token használatával	30
6. ábra 3. eset: A végpont hívása érvényes token használatával	30
7. ábra 1. eset: Az űrlap nem küldhető be üresen	31
8. ábra 2. eset: Az űrlapnak formailag helyes adatot kell megadni	31
9. ábra 3. eset: Az űrlapon megadott adat formailag helyes, azonban nem szerepel az adatbázisban	32
10. ábra 4. eset: Az űrlapon megadott adat helyes	32
11. ábra 5. eset: Az űrlapon megadott adat helyes, azonban más hibából kifolyólag a művelet nem folytatható.....	32
12. ábra A kezdőlap	37
13. ábra Gyors megtekintés	37
14. ábra A rendszerről	38
15. ábra Azonosítási folyamat	38
16. ábra A rendszer tájékoztatja a felhasználót az azonosító levél kiküldéséről	39
17. ábra Az azonosításra szolgáló levél	39
18. ábra Kóddal végzett azonosítás.....	39
19. ábra Személyes oldal: Oklevelek menüpont	40
20. ábra Személyes oldal: Idővonal menüpont	40
21. ábra Személyes oldal: Személyes adatok menüpont	40
22. ábra Oklevél.....	41
23. ábra Az oklevél nyomtatás Chrome böngészőben	41
24. ábra Belépés az adminisztrációs felületre	42
25. ábra Az adminisztrációs kezdőlap	42
26. ábra Aláírások oldal	43
27. ábra Egy meglévő aláírás módosítása	43
28. ábra Személyek.....	44
29. ábra Egy személy adatainak módosítása.....	44
30. ábra Oklevéltörzsek listája.....	45
31. ábra Oklevéltörzs létrehozása	45
32. ábra Oklevélkiadás	46

Irodalomjegyzék

Internetes tartalmak

React: <https://reactjs.org/docs/getting-started.html>

NPM: <https://www.npmjs.com/>

TypeScript: <https://www.typescriptlang.org/docs/>

Node.js: <https://nodejs.org/en/docs/>

StackOverflow: <https://stackoverflow.com/>

MDN Web Docs: <https://developer.mozilla.org/en-US/>

Ant Design: <https://ant.design/docs/react/introduce>

Nodemailer: <https://nodemailer.com/usage/>

Amazon Simple Email Service Documentation: <https://docs.aws.amazon.com/ses/index.html>

Express.js API Reference: <https://expressjs.com/en/4x/api.html>

Prisma: <https://www.prisma.io/docs>

Handlebars: <https://handlebarsjs.com/>

Prettier: <https://prettier.io/docs/en/index.html>

Vite: <https://vitejs.dev/guide/>

Iconify: <https://docs.iconify.design/>

Axios: <https://axios-http.com/docs/intro>

JWT-decode: <https://github.com/auth0/jwt-decode#readme>

Json Web Tokens: <https://jwt.io/introduction>

Node-Bcrypt: <https://github.com/kelektiv/node.bcrypt.js#readme>

Dotenv: <https://github.com/motdotla/dotenv#readme>

Könyvek

Robert C. Martin: Tiszta kód

Michael J. Hernandez: Database Design for Mere Mortals: A Hands-On Guide to Relational Database Design

Jeffrey D. Ullman, Jennifer Widom: Adatbázisrendszerek – Alapvetés

Alex Banks, Eve Procello: Learning React: Modern Patterns for Developing React Apps