



# Webservice Menedzselhető Beléptetőrendszerhez

## Készítette

Dombi Tibor Dávid

Programtervező Informatikus Bsc.

## Témavezető

Dr. Tajti Tibor

Egyetemi Adjunktus

EGER, 2022

# Tartalomjegyzék

<b>1. Rendszer Bemutatása</b>	<b>5</b>
1.1. Példa üzem . . . . .	5
1.2. Telepítés . . . . .	6
1.2.1. Rendszerigény és környezeti követelmények . . . . .	6
1.2.2. Telepítő varázsló . . . . .	7
1.2.3. Tesztelés és CI/CD . . . . .	7
<b>2. API kommunikáció</b>	<b>10</b>
2.1. BREAD vagy CRUD műveletek . . . . .	10
2.2. HTTP metódusok . . . . .	11
2.2.1. GET . . . . .	11
2.2.2. POST . . . . .	12
2.2.3. PUT és PATCH . . . . .	13
2.2.4. DELETE . . . . .	13
2.3. Azonosítás . . . . .	13
2.3.1. JWT . . . . .	13
2.3.2. Bejelentkezés . . . . .	14
2.4. Hozzáférés szabályozás . . . . .	15
<b>3. Rest Service</b>	<b>17</b>
3.1. Laravel . . . . .	17
3.1.1. Eloquent . . . . .	17
3.1.2. Database . . . . .	19
3.2. Jogosultságok . . . . .	21
3.3. Erőforrások . . . . .	22
3.3.1. User . . . . .	22
3.3.2. Worker . . . . .	23
3.3.3. Lock . . . . .	23
3.3.4. Group . . . . .	23
3.3.5. Bouncer táblák . . . . .	23
3.4. AccessRule . . . . .	24
3.4.1. Speciális hozzáférési szabály . . . . .	24

3.4.2.	Általános hozzáférési szabály . . . . .	25
<b>4.</b>	<b>Admin Service</b>	<b>26</b>
4.1.	Angular . . . . .	26
4.2.	Ionic . . . . .	27
4.3.	Felület . . . . .	27
4.3.1.	Adattábla komponens . . . . .	28
<b>5.</b>	<b>Hardveres megvalósítási javaslat</b>	<b>30</b>
5.1.	Elosztott rendszerű megoldás . . . . .	30
5.2.	Csillagpontos megoldás . . . . .	31
<b>6.</b>	<b>Továbbfejlesztési lehetőségek</b>	<b>32</b>

# Bevezetés

Már egyetemi tanulmányaim előtt is legjobban a webes alkalmazások készítése foglalkoztatott, és ez az irány megmaradt végig a tanulmányaim során. Nem meglepő, hogy szakdolgozatomnak is ilyen témát választottam. Az utolsó két évemben elkezdtem komolyabban foglalkozni az IoT eszközökkel, és mivel rendkívül érdekesnek találtam a témát, szerettem volna belevenni a szakdolgozatomba.

Így született meg ez a dolgozati téma: Egy menedzselhető, online felületen elérhető beléptetőrendszer prototípusa. A feladat rendkívül aktuális, hiszen a legtöbb cég szabályozza, hogy ki léphet be a telephelyére, illetve valamilyen módon naplózza, hogy dolgozói mikor érkeztek és távoztak. Ez a rendszer mindkét feladatot automatizálja.

A projekt elkészítése alatt nagy hangsúlyt fektettem a költséghatékonyságra illetve arra, hogy minél felhasználóbarátabb legyen, mindezt úgy, hogy minden szükséges szolgáltatást megvalósítson.

A piacon számtalan hasonló rendszert találni, de ezek mind rendkívül komplexek és költségesek, többnyire csak a szolgáltató szerelői tudják beszerelni és üzemben tartani, valamint alig-, vagy egyáltalán nem szabhatók személyre. A célom az volt, hogy a projekt megoldást kínáljon ezekre a problémákra. Itt megjegyezném, hogy a dolgozat csak a webservice-al foglalkozik, a hardveres feladatokkal nem, viszont biztosít egy általános interfészt, amihez bármilyen hardver komponenst csatolhatunk.

A technológia kiválasztásánál az volt a célom, hogy minél általánosabb, és minél könnyebben hozzáférhető legyen. Az online elérés megvalósítására számtalan módszer van, de a legegyszerűbb, ha eleve az egész rendszer egy szerveren fut, és valamilyen webtechnológiával van megvalósítva. Ez magában hordozza azt az előnyt is, hogy így a program bármilyen platformról elérhető. Bár a PHP népszerűsége sokat zuhant az elmúlt években[1], még mindig az egyik legdominánsabb nyelv, ha szerver oldali programozásról van szó[2], így erre esett a választásom. Továbbá, a Node.js vagy Go lang alapú technológiákhoz képest jóval egyszerűbb olcsó, vagy kisebb adatforgalom esetén akár ingyenes tárhely szolgáltatót találni.

# 1. fejezet

## Rendszer Bemutatása

A rendszer rendkívül moduláris, így könnyen személyre szabható. De általánosítva, csupán egy webservice-ként üzemel, amihez Rest API-on keresztül csatlakozhatunk. Fontos megjegyezni, hogy a projekt nem arra szolgál, hogy programozási tudás nélkül beüzemelhető legyen, de nem is arra, hogy szolgáltatásként nyújtsuk. A projekt célja, hogy egy részletes, átfogó alapot adjon azoknak a fejlesztőknek, akik szeretnék a saját rendszerüket kifejleszteni.

### 1.1. Példa üzem

Legyen adott egy vállalkozás, ami szeretné a székhelyénél szolgáló iroda épületet biztosítani, továbbá nyomonkövetni, hogy a dolgozók mikor érkeznek és mikor távoznak az épületből. A cég eleve mikrokontrollerekkel foglalkozik, így a hardware-es kiépítés nem jelent problémát. Ebben az esetben nyúlhatnak olyan projektekhez, mint a jelen szakdolgozati munka.

*Telepítés* után a program egy egyszerű Rest API szolgáltatás. A kommunikáció JSON formátumban történik.

A hardware-es megvalósításra javaslat a 5. fejezetben található. Megvalósítástól függetlenül, a zárnak el kell küldenie a Webservice-nek a saját azonosítóját, illetve a dolgozó azonosítóját (ez lehet például az a szám amit egy RFID kártyáról leolvasott). A zárnak azonosítania kell magát. Ennek mikéntjét az *Azonosítás (2.3)* fejezetben részletezem. A webservice az ismert szabályok alapján eldönti, hogy az adott dolgozó ebben az időpontban használhatja-e a zárat, majd a döntésével válaszol a zárnak. A döntés naplózásra kerül.

Az adminisztrátori felületen követhetjük, mind a rendszerben történt eseményeket, mind a zárok használatát.

## 1.2. Telepítés

A projekt letölthető a saját github repository-jából[3]. Az alkalmazás futtatásához szükségünk lesz egy PHP szerverre (nginx, Apache, lighttpd, stb.)[4], illetve egy Laravel által támogatott adatbázis szerverre (MariaDB, MySQL, Postgres, SQLite, vagy SQL Server)[5]. Ezek lehetnek a felhőben, vagy egy lokális szerveren.

A példa vállalkozásunk kétféleképpen használhatja a programot. Felállíthatják maguk a szükséges rendszert, vagy **Docker** használatával előre beállított **container**-eket is használhatnak, köszönhetően a Laravel Sail támogatásnak[6]. Bár utóbbit telepíteni jóval gyorsabb, és szinte egyáltalán nem kell belenyúlnunk, saját teszteléseim alapján jelentős teljesítmény csökkenést jelent. Azt, hogy melyik módszert szeretné használni a vállalkozás, kiválaszthatja a Telepítő varázslóban.

### 1.2.1. Rendszerigény és környezeti követelmények

Ahhoz hogy az alkalmazást el tudjuk indítani biztosítanunk kell bizonyos előkövetelményeket. Ezekből két csoportot gyűjtöttem össze, attól függően, hogy **Docker**-el szeretnénk-e használni vagy sem.

Amire mindenképpen szükség van:

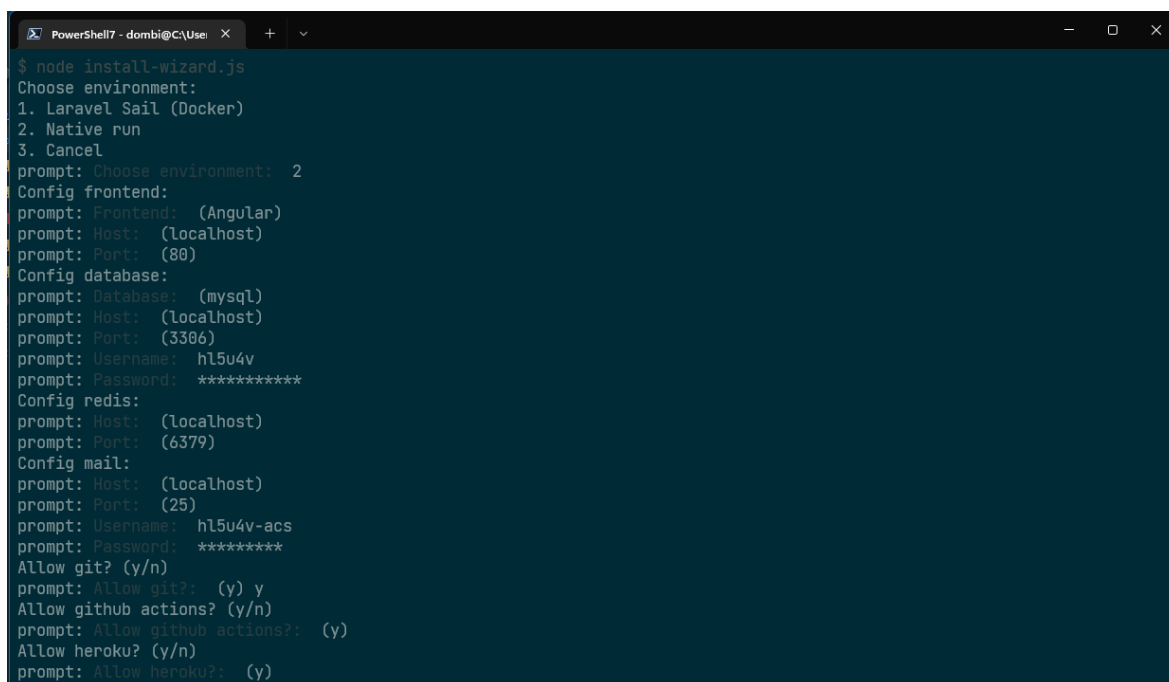
- **Internet** **elérés** (a package-ek telepítéséhez)
- **Node.js** **14.x** vagy újabb (az alapvető segéd script-ek futtatásához)
- **Yarn** **1.x.x**

Amennyiben **Docker**-el szeretnénk futtatni, úgy csak magát a Docker-t kell feltelepítenünk a fentebbiek mellett. Ellenkező esetben a következőkre van még szükségünk:

- **Web** **szerver** (Apache, NGINX, stb.)
- **PHP** **8.1**
- **Composer** **2.x**
- **Laravel** kompatibilis DBMS (MySQL, SQLite, MS SQL Server, Oracle SQL)[5]
- **Redis** **server** (Cache-eléshez, opcionális)
- **NG** **CLI** (Frontend-hez)
- **Ionic** **CLI** (Frontend-hez)

## 1.2.2. Telepítő varázsló

A telepítő varázsló forráskódja a `scripts/install-wizard.js` fájlban található[7]. Miután a projekt gyökérkönyvtárában feltelepítjük a package-eket (`yarn` parancs[9]) automatikusan elindul ez a fájl (`postinstall`). A script végig vezet bennünket a projekt konfigurációján. Ennek egy részletét látni a 1.1 ábrán.



```
PowerShell7 - dombi@C:\User> $ node install-wizard.js
Choose environment:
1. Laravel Sail (Docker)
2. Native run
3. Cancel
prompt: Choose environment: 2
Config frontend:
prompt: Frontend: (Angular)
prompt: Host: (localhost)
prompt: Port: (80)
Config database:
prompt: Database: (mysql)
prompt: Host: (localhost)
prompt: Port: (3306)
prompt: Username: hl5u4v
prompt: Password: *****
Config redis:
prompt: Host: (localhost)
prompt: Port: (6379)
Config mail:
prompt: Host: (localhost)
prompt: Port: (25)
prompt: Username: hl5u4v-ac
prompt: Password: *****
Allow git? (y/n)
prompt: Allow git?: (y) y
Allow github actions? (y/n)
prompt: Allow github actions?: (y)
Allow heroku? (y/n)
prompt: Allow heroku?: (y)
```

1.1. ábra. A telepítő varázsló felülete

Elsőként azt kell kiválasztanunk, hogy `Docker`-ben vagy anélkül (`native run`) szeretnénk futtatni az alkalmazást. Választásunktól függetlenül a következő lépés az alkalmazás elérési címének, és portjának beállítása, majd pedig az adatbázis kapcsolat beállítása lesz. A telepítő végén opcionálisan egyéb szolgáltatásokat is konfigurálhatunk, úgy mint `Redis` cache, `Mail service`[11], `Git`, illetve `CI/CD Github actions`[12]-el és `Heroku`-val.

Amennyiben a `Docker`-t választottuk további feladatunk nincs, elegendő elindítani a `container`-jeinket<sup>1</sup> és az alkalmazás készen áll, hogy felvigyük a megfelelő erőforrásokat (lásd 3.3 fejezet).

## 1.2.3. Tesztelés és CI/CD

A projekt tartalmazza a `PHP Unit` csomagot, amivel, nevével ellentétben nem csak unit teszteket fogunk tudni írni, hanem `Feature`, `Database` és egyéb teszteket is. Tesztjeink többsége unit tesztnél magasabb szintű teszt lesz, így ezeket fejtem ki bővebben.

<sup>1</sup> Releváns dokumentáció: <https://docs.docker.com/compose/reference/up/>

Feature teszt írásához örököltessük a teszt osztályunkat a `HttpTest` absztrakt osztálytól. Ekkor elérjük a Laravel beépített Http test funkcióit, illetve további segéd függvényeket. Ezeknek előnye, hogy a „beszédes” függvényekkel a tesztünk könnyen olvasható lesz.

```
1 <?php
2
3 namespace Tests\Feature;
4
5 use Tests\HttpTest;
6 use Utils\StatusCodes;
7
8 class ExampleTest extends HttpTest
9 {
10
11     public function testApiRequest()
12     {
13         $response = $this->actingAs(Admin())
14                     ->postJson(
15                         '/api/user',
16                         ['name' => 'John']
17                     );
18
19         $response->assertStatus(StatusCodes::CREATED)
20                 ->assertJson(['created' => true]);
21     }
22
23 }
```

1.1. kód. Feature teszt példa

A Console Test lényegében Feature teszt, ami a saját Artisan parancsainkat hivatott ellenőrizni. Erre nem írtam külön segéd osztályt, a teszt osztályunkat örököltessük a `TestCase` osztálytól (Ez a Laravel beépített teszt facade osztálya). Kiemelnék egy példát a Laravel dokumentációból[15]:

```
1 /** Az Artisan parancs: */
2 Artisan::command('question', function () {
3     $name = $this->ask('What is your name?');
4
5     $language = $this->choice('Which language do you
6         ↪ prefer?', [
7         'PHP',
8         'Ruby',
9         'Python',
10    ]);
11 }
```



```

11     $this->line('Your name is '.$name.' and you
12         ↳ prefer '.$language.'.');
    });

```

### 1.2. kód. Egy példa Artisan parancs

```

1  /** Test a console command. */
2  public function test_console_command()
3  {
4      $this->artisan('question')
5          ->expectsQuestion('What is your name?', '
6              ↳ Taylor Otwell')
7          ->expectsQuestion('Which language do you
8              ↳ prefer?', 'PHP')
9          ->expectsOutput('Your name is Taylor Otwell
10              ↳ and you prefer PHP.')
11          ->doesntExpectOutput('Your name is Taylor
12              ↳ Otwell and you prefer Ruby.')
13          ->assertExitCode(0);
14  }

```

### 1.3. kód. A fenti parancs tesztelése

Végül pedig az adatbázis tesztekhez elérhető egy `DatabaseTest` absztrakt osztály, melynek célja és használata is hasonló, mint az `HttpTest` osztálynak, csupán más függvényeket tesz elérhetővé. Kiemelném, hogy mind az `HttpTest` mind a `DatabaseTest` a Laravel beépített `TestCase` osztályától örököl, tehát a keretrendszer beépített függvényei mindenkor rendelkezésünkre állnak.

Azért, hogy a felhasználó minél hamarabb megkapja a programját, illetve, hogy a hibák minél hamarabb kiderüljenek és minél hamarabb orvosolhatók legyenek, a modern programozási feladatok legnagyobb része használ valamilyen Continuous Integration-t és Continuous Deployment-et biztosító szolgáltatást (CI/CD). Amennyiben nagyobb mértékben szeretnénk tovább fejleszteni a projektet, mindenképpen érdemes elgondolkodnunk ezek használatán.[16]

A projekt majdnem teljesen konfigurálva érkezik Github actions (CI) illetve Heroku (CD) számára, így elég a projekt specifikus adatokat megadni (felhasználó nevek, API kulcsok, stb.), és már is használhatóak ezek a technológiák is. Hogy pontosan mit kell beállítanunk, az attól függ, hogy milyen szolgáltatást veszünk igénybe, de a környezeti változókat minden esetben fel kell vinnünk (adatbázis hozzáférés, levelező szerver hozzáférés, stb.).

## 2. fejezet

# API kommunikáció

A webservice egy RESTful API-t biztosít, hogy a saját eszközeink könnyedén tudjanak kommunikálni vele. Az erőforrásaink ebben az esetben a *zárak*, *dolgozók*, *ezek csoportjai*, valamint a *szabályok*. Ezekre a BREAD műveleteket (2.1) (Browse, Read, Edit, Add, Delete) úgy tudjuk alkalmazni, hogy az API megfelelő végpontjait érjük el, megfelelő HTTP metódussal (GET, POST, PUT/PATCH, DELETE).[17]

Az összes végpont, és azok válaszainak a leírását egy OpenAPI definíciós fájlban megadtam, és a szakdolgozathoz csatoltam[18], így ezeket itt nem fejtem ki, továbbá megtekinthető online is a Stoplight.io felületén is[19].

### 2.1. BREAD vagy CRUD műveletek

Az erőforrásokkal végezhető műveletekről beszélve gyakran használjuk a CRUD vagy BREAD mozaikszavakat. A CRUD a *Create*, *Read*, *Update*, *Delete* szavak rövidítése. Gyakran használják mivel közvetlenül megfeleltethető mind az SQL nyelv metódusainak, mind az HTTP metódusainak, ezt látjuk a 2.1 táblázaton.

SQL	CRUD	HTTP
INSERT	CREATE	POST
SELECT	READ	GET
UPDATE	UPDATE	PUT / PATCH
DELETE	DELETE	DELETE

2.1. táblázat. HTTP, SQL, és CRUD műveletek megfeleltetése

Egy másik megközelítés, a BREAD (Browse, Read, Edit, Add, Delete). Ez lényegében megegyezik a CRUD metódusokkal, annyi különbséggel, hogy a BREAD külön veszi azt a metódust, amikor minden erőforrásra kíváncsiak vagyunk (Indexelés), illetve azt a metódust amikor csak egy erőforrás részleteire vagyunk kíváncsiak. Ez azért előnyösebb a számomra, mert ezek jobban megfeleltethetők a Laravel Controller

metódusainak. Ha ezeket is meg szeretnénk feleltetni egymással, akkor a táblázatunk módosul[20], ez látszik a 2.2 táblázaton.

SQL	CRUD	HTTP	BREAD	Laravel Controller
SELECT	READ	GET	BROWSE	INDEX
SELECT	READ	GET	READ	SHOW
UPDATE	UPDATE	PUT / PATCH	EDIT	UPDATE
INSERT	CREATE	POST	ADD	STORE
DELETE	DELETE	DELETE	DELETE	DESTROY

2.2. táblázat. HTTP, SQL, CRUD műveletek a BREAD-hez képest

## 2.2. HTTP metódusok

Az HTTP szabvány úgynevezett metódusokat (*methods*) definiál nekünk[21], hogy ezzel közöljük a szerverrel, hogy mi a szándékunk az URI-ben megnevezett erőforrással.[22] Vegyük például a **worker** erőforrást. Ebben az esetben az implementált metódusok a következők:

### 2.2.1. GET

Ezen HTTP metódussal két funkció is elérhető. Ha nem definiáljuk hogy pontosan melyik rekordot (példákban: melyik dolgozót) szeretnénk részletezni egy erőforráson belül, akkor vissza kapjuk az összes ezen erőforrásban tárolt rekordot (tehát az összes dolgozót). A Laravelben megegyezés szerint ez a függvény az `index()`.

```
1
2 Request:
3 GET www.example.com/worker
4
5 Response body:
6 [
7   {
8     "name": "Shelia Lakin",
9     "created_at": "2021-06-30T12:09:20.478Z",
10    "phone": "938-402-3157 x810",
11    "id": "4b3403665fea6"
12  },
13  {
14    "name": "Whitney McClure",
15    "created_at": "2021-06-18T05:26:38.641Z",
16    "phone": "(765) 538-9079",
17    "id": "4b3407665bcd5"
18  }
```

## 2.1. kód. Példa a BROWSE metódusra

Amennyiben az URI-ben specifikáljuk, hogy melyik rekordot szeretnénk elérni, úgy részletes információkat kapunk arról (tehát egy dolgozó adatait kapjuk vissza). Laravelben ez a `show()` függvény, konvenció szerint.

```

1
2 Request:
3 GET www.example.com/worker/4b3403665fea6
4
5 Response body:
6 {
7   "name": "Whitney McClure",
8   "created_at": "2021-06-18T05:26:38.641Z",
9   "last_logged_in": "2021-06-30T05:26:38.641Z",
10  "phone": "(765) 538-9079",
11  "id": "4b3407665bcd5",
12  "teams": [
13    {
14      "id": 1,
15      "name": "Team1"
16    },
17    {
18      "id": 2,
19      "name": "Team2"
20    }
21  ]
22 }
```

## 2.2. kód. Példa a READ metódusra

## 2.2.2. POST

Ezzel a metódussal új erőforrást hozhatunk létre. Ehhez a kérésünk törzsében meg kell adnunk minden szükséges tulajdonságot, illetve az opcionális tulajdonságokat vagy megadjuk, vagy nem. Ezen tulajdonságok sorrendje tetszőleges. A kérésünk törzsében lehet egy JSON objektum vagy egy tömb, ami több objektumot is tartalmazhat. Ha ez utóbbi áll fent, akkor a webservice felveszi nekünk az összes rekordot. Ez a felvétel tranzakció szerűen működik, tehát ha bármelyik adatfelvitel sikertelen, akkor az összes visszavonásra kerül. Ha az adatfelvitel sikeres a szerver 201-es státuszkóddal válaszol. Ez a Laravel-ben a `store()` függvény.

### 2.2.3. PUT és PATCH

Ez a két metódus nagyon hasonló egymáshoz, azonban van két nagyon fontos különbség: A PUT metódust használva az általunk megadott adatok *felülírják* azt a rekordot, amit az URI-ben azonosítottunk. Ez azt jelenti, hogy ha részleges adatot küldünk, akkor csak ezek tárolódnak le, minden más elveszik. A PATCH ehhez képest kifejezetten részleges adatot vár, és csak a kapott tulajdonságokat módosítja.

A másik különbség abban van, hogy hogyan kezelik azt az esetet, amikor nem létezik a rekord, amit el szeretnénk érni: A PUT ilyenkor egyszerűen létrehozza azt, a PATCH viszont hibát ad 404-es státuszkóddal.

A Laravel alap beállításai alapján nem tesz különbséget a PUT és a PATCH között, de ezt megváltoztathatjuk. Mivel gyakorlatban többnyire csak a PATCH metódus lesz használva, csak ezt implementáltam, és megtartottam a Laravel alap beállítását. Tehát mind a két metódus az `update()` függvényt éri el.

### 2.2.4. DELETE

Laravelben ez a `destroy()` függvény. A metódus neve elárulja a funkcióját is. Ezt a végpontot elérve, a webservice eltávolítja a URI-ben azonosított rekordot, majd a válaszban visszaküldi a kliensnek az eltávolított rekord részleteit, mint a `show()` függvény, azonban ez a rekord már nem szerepel az adatbázisban.

## 2.3. Azonosítás

Mint minden webszolgáltatásnál itt is szükséges szabályozni a hozzáférést. Erre a Laravel Sanctum<sup>1</sup> csomagot kombináltam a `php-open-source-saver/jwt-auth` csomaggal<sup>2</sup>. Utóbbi csomag az egyik legkedveltebb, és legrobosztusabb JWT csomag ami csak elérhető PHP-hoz. A Sanctum-ra azért van szükség, mert így a hitelesítést el lehet végezni a Laravel beépített Policy rendszerén keresztül[23]. A JWT token elküldhető fejlécben, sütiként, vagy akár az üzenet törzsében is.

### 2.3.1. JWT

A JWT (JSON Web Token) egy szabvány (RFC 7519), aminek a célja, hogy biztonságosan lehessen üzeneteket küldeni egyszerű JSON objektumokként. Az üzenet minden esetben digitálisan alá is van írva. Bár az üzenet lehet titkosított is, ebben az alkalmazásban csak az aláírást használtam.[24]

A JWT kompakt formája három darab Base64-URL kódolt szöveg. [24]

<sup>1</sup> Forráskód: <https://github.com/laravel/sanctum>

<sup>2</sup> Forráskód: <https://github.com/PHP-Open-Source-Saver/jwt-auth>

Az első rész a fejléc, egy JSON objektum, aminek két mezője van. Az első az üzenet típusa (*typ*). Ennek az értéke mindig *JWT*. A második az algoritmus (*alg*), amit az aláíráshoz használtunk (*HMAC*, *SHA256* vagy *RSA*, utóbbi esetén publikus-privát kulcs generálandó, így a kliens is meggyőződhet róla, hogy a szerver küldte az üzenetet, és nem egy harmadik fél).[25]

Ebben a projektben az *HMAC* algoritmust használtam, mivel a projekt volumenéhez mérten elegendőnek találtam.

A második rész a *payload*. Ide adhatunk hozzá minden olyan mezőt, amire a kliensnek szüksége van. Bár semmi sem akadályoz minket abban, hogy kedvünkre bármilyen kulcsot (*claim*) hozzáadjunk, vannak bizonyos mezők, amiknek a neveit és tartalmukat szabvány írja elő. Ezek megtekinthetők az IETF által nyilvántartott RFC-ben (*Registered Claims*, ezek a JWT szabvány részei), illetve a IANA nyilvántartásában is (*Public Claims*, ezek regisztrációját kérték). Minden más *claim* úgy nevezett *Private Claim*. [24]

Ebben a projektben ilyen *Private Claim* a *user*, amiben az aktuális user alapvető adatai vannak. Ezek az adatok használhatók frontend-en történő megjelenítésre, illetve, amikor egy későbbi kérésnél a kliens elküldi ezt a token-t, a webservice-nek nem kell kikeresnie a felhasználót az adatbázisból. Mivel a token-ek nem titkosítottak privát jellegű információt nem csomagolok ebbe az üzenetbe.

A JWT utolsó része egy digitális aláírás. Az aláírást a webservice végzi, egy szerveren tárolt 32 bájts hosszú titkos kulcs segítségével. Mivel a kulcs titokban van tartva, így a webservice bejövő üzenetnél ellenőrizheti, hogy módosították-e a *payload* tartalmát. Amennyiben az üzenet módosult, a webservice eldobja azt. (Amennyiben *RSA* algoritmust használtunk a kliens ezt az aláírást ellenőrizheti.) [25]

### 2.3.2. Bejelentkezés

A felhasználó bejelentkeztetése úgy történik, hogy a felhasználó egy *POST* kérést indít az *api/auth/signin* végpontra. Az üzenet tartalma (*Request body*) az email címe és a jelszava. A backend ezt validálja. Amennyiben ez a validáció sikeres, úgy a backend az azonosított felhasználó adatait a válasz JWT *payload* részébe csomagolja, *private claim*-ként, majd az egészet egy titkos kulcs alapján aláírja. [26] Az aláíráshoz használt algoritmus *HS256*. Itt kiemelném, hogy ez a token nincs titkosítva, csak aláírva.

Amikor a felhasználó kérést küld a webservice-nek, a kérés fejlécében meg kell adnia az *Authorization* mezőt, melynek értéke a séma (*Bearer*), egy szóköz, majd a bejelentkezéskor kapott JWT.

```
1 POST http://www.example.com/api/worker
2
3 Content-Type: application/json
```

```

4 # Hitelesítés:
5 Authorization: Bearer eyJhbG...dQssw5c
6
7 { "name": "Test Worker", "rfid": "abc123456" }

```

2.3. kód. Példa hitelesített kérés

## 2.4. Hozzáférés szabályozás

A jogosultságok kezeléséhez a Bouncer<sup>3</sup> csomagot használtam. Mivel a jogosultságok kiosztásához is szükség van jogosultságra, így a telepítés után automatikusan van egy *Superadmin* felhasználó regisztrálva. Miután ezt megtettük, érdemes deaktiválni ezt a felhasználót. Minden felhasználó csoportokba osztható, és ezekhez a csoportokhoz rendelhetjük a megfelelő jogosultságokat. Gyakorlatilag minden végponthoz tartozik egy megfelelő jog, de hogy ezeket egyszerűbb legyen kezelni, nem csak végpontokhoz, de modellekhez is rendelhetünk szabályokat. Így például, ha szeretnénk egy csoportot, akik tudják kezelni a dolgozókat, de nem tudják törölni őket, akkor adunk egy *engedélyezés* szabályt ennek a csoportnak, a dolgozó modelre nézve, majd adunk egy *tiltó* szabályt ennek a csoportnak, a *destroy* végpontra vagy a modell törlésére. Gyakorlatban ezt meg tehetjük két külön kéréssel, vagy akár egy olyan kéréssel is, ahol a törzsben egy tömbben adjuk meg mind a két szabályt:

```

1 POST /authority HTTP/1.1
2 Host: www.example.com
3 Content-Type: application/json
4 Content-Length: length
5
6
7 [
8   {
9     "auth_group_id": "c4fe980f-0ad9-48c5-b658-
10      ↪ f9fa85973dd9",
11     "on": "Models\Worker",
12     "allow": "*"
13   },
14   {
15     "auth_group_id": "c4fe980f-0ad9-48c5-b658-
16      ↪ f9fa85973dd9",
17     "on": "Models\Worker",
18     "restrict": "destroy"
19   }
20 ]

```

<sup>3</sup>Forráskód: <https://github.com/JosephSilber/bouncer>

---

#### 2.4. kód. Példa a jogosultság kezelésre



## 3. fejezet

# Rest Service

### 3.1. Laravel

A Laravel alapvetően PHP keretrendszer[27], azonban napjainkra egy egész ökoszisztéma épült köré a különböző csomagok által. Ezen csomagok döntő többsége, akár csak maga a keretrendszer, teljesen nyílt forráskódú az MIT szoftverlicenz alapján.

A Laravel MVC architektúrára épül, és egy alapvetően rendkívül expresszív, kényelmes szintaktikát használ. A keretrendszer alapelve, hogy „ne törődjünk az apróságokkal”. Ez a gyakorlatban azt jelenti, hogy minden olyan feladatot belefoglaltak, amit máskülönben a fejlesztők projektenként újra és újra implementálnának. Innen a szlogenjünk: „The PHP Framework for Web Artisans” A projekt 2011 óta folyamatosan aktív, jelenleg a 9.x verziószám a legfrissebb. Ez a verzió *LTS (Long-term support)*, tehát a többi verzióhoz képest jóval tovább lesz támogatva.[27]

A szakdolgozat írásakor messze a Laravel a legelterjedtebb és legnagyobb fejlesztői közösséggel rendelkező PHP keretrendszer.[28]

#### 3.1.1. Eloquent

A Laravel beépített ORM (Object-relational mapping) rendszerrel érkezik, ez az *Eloquent* névre hallgat, és lényegében megfelel az *aktív rekord* programtervezési mintának. Az Eloquent az erőforrások tábláit osztályokként ábrázolja, az elérhető statikus metódusok tehát az egész táblára hatnak, míg egy példánya az osztálynak megfelel egy rekordnak a táblán.[29]

```

1  $user = User::find('test@example.com', 'email');
2
3  /*
4  SELECT * FROM users
5          WHERE email = "test@example.com"
6          LIMIT 1
7  */
8
9  $user->email = 'test2@example.com';
10 $user->save();
11
12 /*
13 UPDATE users SET email = "test2@example.com"
14          WHERE id = "50cb85ae-bdce-4646-a54f-0
15                  ↪ a8d94c551c4"
16 */

```

3.1. kód. Eloquent és az SQL kapcsolata

Továbbá az Eloquent ad egy absztrakciós szintet az adatbázis fölött, így ha cserélődik az adatbázis motor (például MySQL-ről SQL Server-re), akkor elég az csak a konfigurációban megadni, és az Eloquent automatikusan betölti a megfelelő *driver*-t.

Ugyanakkor az eloquent számtalan kritikát kap éppen amiatt, mert az aktív rekord mintára épül. Ez megnehezíti az adatbázis elválasztását a programkódtól a tesztelés alatt. Az eloquent emelett a PHP „varázsmetódusait” használja, ami problémákat okozhat, ha fejlesztés során megváltozik a tábla struktúrája, például a *users* táblán átnevezünk a *created\_at* mezőt *registered\_at*-re, akkor az alkalmazásunkban mindenhol módosítanunk kell a `$user->created_at` kódot, hogy tükrözze ezt: `$user->refistered_at`. Ez egy nagyobb alkalmazásban igen gyorsan komplikálttá tud válni.[30]

A harmadik probléma pedig, hogy az eloquent használata során rendszeresen megszegjük a *legkisebb tudás elvét* például amikor olyan hívásokat írunk, mint ami alább, a 3.2 kódban látható.[30]

```

1  $roles = $user->roles->get(); // Ez megszegi a
    ↪ legkisebb tudás elvét
2  $roles = $user->getRoles();   // Ez nem szegné meg

```

3.2. kód. Legkisebb tudás elve, demonstráció

Az első problémára A Database Factory-k (lásd: 3.1.2 fejezet) Seeder-ek (lásd: 3.1.2 fejezet) használata jelent megoldást (tehát mockup adatbázist hozunk létre). Emellett a Laravel alapból számtalan segéd funkcióval rendelkezik, így az adatbázis tesztek elvégzése annyira egyszerű, hogy nem érdemes emiatt elvetni az eloquent-et.[31]

A második probléma jóval komplexebb. Először is néhány IDE (például az *IntelliJ PhpStorm*) képes ilyen módosításokat végezni az adatbázison, majd a megfelelő

bővítményekkel a teljes kódot képes átnézni és elvégezni a szükséges módosításokat. Természetesen teljes mértékben az IDE-re hagyatkozni, és remélni, hogy mindenhol kijavítja a kódunkat, nem egy jó megoldás. Ha tényleg nagyon komplex az alkalmazásunk, akkor alkalmazhatjuk a *Repository* tervezési mintát. Ez egyben a harmadik problémánkat is megoldaná, azonban a téma jelenleg is nagyon vitatott a Laravel fejlesztők között, mivel az Eloquent alapról ad egy absztrakciót az adatbázis felett, így a Repository pattern ténylegesen csak ezt az átnevezési problémát orvosolná nekünk. Ennek ellenére a kódbázisunkat rendkívül megnöveli, illetve jóval kevésbé olvashatóvá válik a kódunk.[32]

Tekintettel arra, hogy a Laravel alapelve az olvasható és „elegáns” szintaxis, így én a szakdolgozatomban a Repository minta elhagyása mellett döntöttem.

### 3.1.2. Database

#### Migration

A Laravel alapról biztosítja nekünk a *Migration*-öket. Ezek lényegében úgy működnek, mint a verzió követő rendszerek, csak az adatbázisra. Minden migration felfogható egy „commit”-nak, amiket `php artisan migrate` paranccsal tudunk „push”-olni az adatbázisunkba. Egy migration bármilyen DDL műveletet tartalmazhat. SQL esetén ezek a *CREATE (TABLE)*, *ALTER*, *DROP* műveletek. Minden migration tartalmaz egy `up()` és `down()` függvényt. Az `up()` arra szolgál, hogy végrehajtsa a módosításokat, a `down()` pedig a visszavonásra.[33]

```
1 class CreateWorkersTable extends Migration {
2
3     public function up(): void {
4         Schema::create('workers', function (Blueprint
5             ↪ $table) {
6             $table->uuid('id')->primary();
7             $table->string('name');
8             $table->string('rfid')->nullable();
9             $table->date('birthdate');
10            $table->string('telephone')->nullable();
11            $table->timestamps();
12        });
13    }
14
15    public function down(): void {
16        Schema::dropIfExists('workers');
17    }
18 }
```

3.3. kód. Példa migration (create\_workers\_table.php)

Egy migration-ben van lehetőségünk több DDL utasítást is kiadni egymás után. Ilyenkor ezek tranzakcióként futnak le (tehát ha egy nem fut le, akkor egyik sem), de fontos megjegyezni, hogy a különböző migration-öket *nem* tranzakcióként kezeli. Itt a rendszernek van egy problémája: Abban az esetben, ha több utasítást adunk ki egy migration-ben, de adatbázisként SQLite-ot szeretnénk használni (ami tesztelésnél elég gyakori), a Laravel így is megpróbálja ezeket tranzakcióként futtatni, azonban az SQLite ezt nem támogatja, így egy ilyen migration nem tud lefutni. Tehát vagy nem használunk SQLite-ot, vagy nem használunk tranzakciókat a migration-öknél. Én ez utóbbit választottam.[34]

## Factory

A Laravel factory osztályaival könnyedén tudunk generált adatokat létrehozni. Ennek leggyakoribb használata a teszt adatok létrehozása, de akkor is alkalmazható, ha egyszerűen egy modell létrehozásánál nem szeretnénk kitölteni valamelyik tulajdonságát, de üresen sem hagyhatjuk.

Itt nagy segítségünkre van a PHP\Faker csomag, ami akár neveket, lakcímeket, és más komplex adatokat is tud generálni nekünk.

Egy factory általában egyetlen függvényt tartalmaz, ami egy „definíciós tömböt” ad vissza. Minden alkalommal, amikor egy model *factory*-jét használjuk, akkor ez a tömb alapján hozza létre a modellt.

```
1 class LockFactory extends Factory
2 {
3     protected $model = Lock::class;
4
5     public function definition(): array {
6         return [
7             'name' => $this->faker->name,
8             'device_key' => $this->faker->boolean(80)
9                             ? Str::random(10)
10                            : null,
11             'status' => $this->faker->numberBetween(0,2),
12         ];
13     }
14 }
```

3.4. kód. Példa Factory (LockFactory.php)

## Seeder

A Seederek módot biztosítanak arra, hogy gyorsan feltöltsük az adatbázisunkat. Ezt használhatjuk akkor is amikor tesztelésre szeretnénk felkészíteni az adatbázisunkat,

illetve abban az alkalmazás első inicializációjakor is. A *seeder*-eket futtathajuk külön-külön, vagy a `php artisan seed` paranccsal lefuttathajuk az összes *seeder*-t, amit a `DatabaseSeeder.php` fájlban megadtunk.

## 3.2. Jogosultságok

A Jogosultságok kezeléséhez számtalan csomag elérhető. Az én választásom a *Bouncer* csomagra esett, első sorban a szintaxisa miatt. Ez a csomag lehetővé teszi, hogy „szerepeket” hozzunk létre (*role*), és ezekhez, vagy akár közvetlenül a felhasználókhöz is rendelhetünk szabályokat:

```
1 $user->assign('admin'); // Ha a role nem létezik,  
    ↪ akkor felveszi újként  
2  
3 Bouncer::allow('admin')->everything();  
4 Bouncer::forbid('admin')->toManage(User::class); //  
    ↪ Megengedi az index(), show(), update(), store  
    ↪ (), destroy() metódusokat  
5  
6 $isUserAllowed = $user->can('edit-users'); //  
    ↪ Igazat ad
```

3.5. kód. Bouncer használata

További előnye a Bouncernek, hogy teljesen integrálva van a Laravel alap *Gate* rendszerével, így használható az beépített *Policy* rendszer, illetve más csomagok is képesek használni a Bouncer által definiált jogosultságokat a *Gate*-eken keresztül.[35]

A megfelelő *endpoint*-ok használatával az adminisztrátori felületen keresztül is oszthatunk ki jogosultságokat, ugyanakkor ehhez meg kell legyen a megfelelő jogunk (nem oszthat bárki, saját kedve szerint jogosultságokat). Éppen ezért, a webservice telepítése után lesz regisztrálva egy *Superadmin* felhasználó (*superadmin@acs.test*), akinek mindenhez van jogosultsága (`Bouncer::allow($superadmin)->everything()`). Miután beállítottuk az admin felületet használó *user*-eket, javasolt törölni ezt a Superadmin felhasználót. Mivel a *User* modell alaphból *Soft Delete*-et használ, így szükség esetén bármikor visszaállíthatjuk, és megmarad a jogosultsági kör.

Amennyiben az alkalmazásunk fejlesztő környezetben fut, néhány további teszt felhasználó is regisztrálásra kerül:

- *superadmin@acs.test*: Minden erőforrás művelet engedélyezett a számára.
- *admin1@acs.test*: A felhasználókat és azok jogosultságait kezelheti, továbbá olvashatja webservice naplóját (Laravel log).

- *hr1@acs.test*: A Dolgozókat és azok csoportjait, valamint a belépési szabályokat kezelheti.
- *supervisor1@acs.test*: A zárat, valamint azok csoportjait kezelheti, továbbá olvashatja a záruk hozzáférési naplóját.
- *security1@acs.test*: Nem módosíthatja, de megnézheti egy dolgozó általános (nem privát jellegű) adatait, de csak akkor, ha ismeri az azonosítóját.

Gyakorlatilag arra a célra szolgál, amikor egy biztonsági őr szeretne meggyőződni arról, hogy kit enged be. Az azonosítót például meg tudhatja onnan, hogy a dolgozó RFID kártyáját leolvassa.

### 3.3. Erőforrások

A projekt erőforrásai lényegében a modellek. Ezek kapcsolatai az alábbi (3.1) ábrán látszik.



3.1. ábra. Modellek kapcsolatai

#### 3.3.1. User

Azon felhasználók adatit tárolja, akik be tudnak jelentkezni az adminisztrációs felületre. A bejelentkeztetésért és jogosultságkezeléssel foglalkozó szolgáltatások ezzel a modellel dolgoznak. Alapbeállítás szerint egyedi UUID-t használ az adatbázis kulcsának, nem sorszámot, valamint a törlés alapértelmezett viselkedése „Soft-Delete”, tehát amikor egy felhasználót törölünk, akkor nem kerül ki az adatbázisból, csupán kitöltésre kerül egy `deleted_at` mező. A Laravel ezt a legtöbb esetben úgy kezeli, hogy az ilyen rekordokat ignorálja, mintha törölve lennének, ugyan akkor bármikor könnyen visszaállíthatjuk ezeket. Így törlés helyett inkább csak deaktiválásról beszélünk.

### 3.3.2. Worker

Azon felhasználók adatait tárolja, akik a zárat tudják használni. Egyedi UUID-t használ kulcsként, nem sorszámot. Általában tartozik hozzájuk valamilyen „ujjlenyomat” (**Fingerprint**), ami alapján azonosíthatóak. Ez lehet RFID kártya száma (ez a legvalószínűbb), felhasználónév, jelszó, de akár bináris adat is lehet. Ez utóbbi akkor szükséges, ha az azonosítást valamilyen komplex rendszer végzi, például ujjlenyomat olvasó, de lehet akár retina szkennel is, stb.

### 3.3.3. Lock

A zárok adatait tárolja, amiket vezérelni szeretnénk. Itt ismételtén található egy ujjlenyomat mező, ezúttal (az ütközések elkerülése végett) `device_fingerprint` néven. Ez azért fontos, mert így felvihetjük például az eszközünk MAC címét, beégetett sorozatszámát, vagy bármilyen egyedi, nem módosítható adatot az eszközről, és a webservice ez alapján tudja azonosítani. A zár nem feltétlenül ajtó zárat jelent. Lehet akár egy szoftver is, ami függetlenül működik, és csak bejelentkezéshez használja ezt a rendszert.

Minden zárhoz tartozik egy állapot. Ezek sorban:

0. Zárva (*Locked*): Ez a zár nem enged át senkit.
1. Működésben (*Operational*): Ez a zár csak azokat engedi át, akik valamilyen szabály szerint átmehetnek.
2. Nyitva (*Open*): Ez a zár mindenkit átenged, aki azonosítható, de a zár használatát naplózza.
3. Ismeretlen (*Unknown*): Ez a zár túl rég óta nem küldött életjelet (*Keep-alive signal*), így a webservice nem tudja ellenőrizni az állapotát.

### 3.3.4. Group

Mind a Worker, mind pedig a Lock modell csoportokhoz rendelhető az egyszerűbb kezelhetőség érdekében. Köszönhetően a Laravel polimorfikus kapcsolatainak (`morphTo()` és `morphToMany()`) nem volt szükség külön-külön modellek létrehozására. A csoportok lényege, hogy amikor egy új zárat üzemelünk be, ne kelljen minden szabályt újra definiálnunk, ha már egyszer egy másik zár esetén megtettük.

### 3.3.5. Bouncer táblák

A Bouncer csomag négy modellt és az azokhoz tartozó adatbázis táblákat hozza létre nekünk. Ezek nagymértékben támaszkodnak a már említett polimorfikus kapcsolatok-

ra, így bármilyen jogosultságot (**ability-t**) hozzá rendelhetünk bármilyen entitáshoz, illetve ezt a jogosultságot adhatjuk akár egy felhasználónak, akár egy jogkörnek (**role**).

## 3.4. AccessRule

A hozzáférési szabályok mondják meg a programnak, hogy átengedhet-e egy dolgozót egy záron. A szabály „egyik oldalán” (**operator**) állhat akár egy vagy több dolgozó, akár egy dolgozó csoport, tehát például nem muszáj két dolgozó miatt egy teljesen új csoportot létrehozni.

A szabály „másik oldalán” (**subject**) pedig állhat egy vagy több zár, vagy egy zár csoport is, tehát itt is érvényes az, hogy két zár miatt nem feltétlenül kell új csoportot létrehozni.

A szabály definíciója JSON formátumban tárolódik. Maga a szabály lehet megengedő, vagy tiltó is. Továbbá definiálhatunk egy számlálót is, ami azt adja, hogy hányszor alkalmazható még ez a szabály. Ennél a számlálónál a 0 érték azt jelenti, hogy a szabály deaktiválva van, még a -1, vagy ha nincs jelen, az azt jelenti, hogy bármennyiszer alkalmazható.

A szabályoknak két alakja van:

1. Speciális (**Specific**)
2. Általános (**Generic**)

### 3.4.1. Speciális hozzáférési szabály

Ez egy olyan szabály, amit egy kifejezett idő intervallumra szeretnénk alkalmazni. Miután az intervallum lejárt a szabály törlődik.

```
1 {
2   "operator_id": "5f1a20f7-368a-453e-b7ad-79
   ↳ b495b086b0",
3   "operator_type": "Worker",
4   "subject_id": "c15ad3b5-9e69-4eb7-9a8b-4
   ↳ bd76316d924",
5   "subject_type": "Group",
6   "count": 1,
7   "definition": {
8     "from": "2021-06-04 10:00:00",
9     "to": "2021-06-04 18:00:00",
10    "allow": true
11  }
12 }
```

3.6. kód. Egy példa speciális szabály létrehozása



### 3.4.2. Általános hozzáférési szabály

Minden olyan szabály, ami nem idő intervallumhoz kötöt, vagy rendszeresen ismétlődik. Az `on_every` tulajdonsággal definiálhatjuk, hogy mely napokon ismétlődik a szabály. Ezt a jelölést többféleképpen is megtehetjük. Felsorolhatjuk tömbben, hogy a hét hanyadik napjain érvényes a szabály, vagy szövegesen is megadhatjuk. A szöveges megadás jelenleg a következő definíciókat támogatja:

- `*`: Minden nap
- `weekday`: Minden hétköznapi
- `weekend`: Minden hétvégén

Hogy a hét mely napjait számolja hétköznapi és hétvégének, az személyre szabható. A `from` és `to` mező ekkor opcionális, alap értéke az adott napon `00:00:00.001` és `23:59:59.999`.

```
1 {  
2   "operator_id": "5f1a20f7-368a-453e-b7ad-79  
   ↳ b495b086b0",  
3   "operator_type": "Worker",  
4   "subject_id": "c15ad3b5-9e69-4eb7-9a8b-4  
   ↳ bd76316d924",  
5   "subject_type": "Group",  
6   "count": 1,  
7   "definition": {  
8       "on_every": "*",  
9       "from": "2021-06-04 10:00:00",  
10      "allow": true  
11  }  
12 }
```

3.7. kód. Egy példa általános szabály létrehozása

Amennyiben a zár nyitási kérelmük mellé egyéb dolgozóval kapcsolatos adatot is csatolunk, úgy ezeket is ellenőrzi a rendszer. Ez például akkor hasznos, ha email cím - jelszó páros véd valamilyen zárat.

## 4. fejezet

# Admin Service

A beléptetőrendszer menedzselését célszerű egy admin felületen megtenni. Az alkalmazás alapból nem rendelkezik semmilyen felülettel, csak az API-on keresztül kommunikálhatunk vele. A szakdolgozatban mellékeltem egy Angular alapú Adminisztrátori felületet, mely teljesen függetlenül, SPA formában működik. Itt azonban kiemelném, hogy a szakdolgozat témája a webservice, és csupán demonstrációs céllal készült hozzá frontend, éppen ezért ezt a részt csak felületesen érintem.

Az Admin Service futhat ugyanazon a szerveren, ahol a webservice, vagy teljesen különbözőn. Utóbbi esetben a `FRONTEND_URL` és a `FRONTEND_PORT` környezeti változók beállítása fontos, ugyanis ezek nélkül a web service CORS beállítása letiltja az üzenet váltást.

Egy valós szituációban az alkalmazó vállalkozás valószínűleg saját felületet írna erre a célra, így a dolgozat ezen részével csak felületesen foglalkozom.

### 4.1. Angular

A frontend réteghez számtalan különböző technológia áll rendelkezésünkre, és ezek alatt nem csak a JavaScript keretrendszereket kell érteni, hiszen a projekt frontendje nem csak böngészőben futhat. Mivel a webservice csupán egy API-t biztosít, így csak az alkalmazón múlik, hogy milyen frontend-et ír hozzá.

A demonstrációhoz én egy Angular-Ionic SPA (**Single Page Application**) mellett döntöttem.

Az Angular (nem összekverendő az AngularJS-el) egy nyílt forráskódú keretrendszer webes alkalmazásokhoz, amit arra terveztek, hogy az egyfejlesztős projektektől, akár a nagy céges szintű alkalmazásokig bármekkora projekt elkészíthető legyen úgy, hogy a kód tiszta, és egyszerűen karbantartható legyen. Az Angular teljes mértékben TypeScript-ben íródott, ami fantasztikus segédeszközök, és statikus típusok használatát teszi lehetővé. Továbbá legnagyobb pozitív tulajdonsága, hogy hatalmas mennyiségű könyvtár áll rendelkezésünkre, hogy megkönnyítse a fejlesztést.[36]

Az alkalmazás legalapvetőbb építőkövei a komponensek. Egy komponens lehet bármi, ami a felhasználónak megjelenik, például egy Beviteli mező, de a teljes oldal, amit megjelenítünk is csak egy komponens, ami más komponenseket ágyaz be. A kódban ezen komponenseket egy TypeScript osztály reprezentál, amit ellátunk az `@Component()` dekorátorral. Ebbe az osztályba felvihetjük a funkcionalitáshoz szükséges kódot (Eseményeket, adatokat, stb.). A komponensekhez tartozik még egy *Template* is, ami lényegében HTML kibővítve, illetve egy *Style*, ami pedig SCSS, de az itt definiált stílusok csak az adott komponensben érvényesek.[37]

Ezek a komponensek modulokba rendeződnek, amik egy fa architektúrában kapcsolódnak egymáshoz. Minden komponens csak a saját, és a „gyermek” modulokban elérhető.

Az Angular ezek mellett biztosít nekünk egy automatikus *Dependency Injector*-t is, ami leveszi az osztályok konstruálásának terhét rólunk. Amikor egy osztálynak szüksége van egy másikra, ahelyett, hogy létre kellene hoznunk azt (potenciálisan újabb függőségeket létrehozva), elegendő az adott osztály konstruktorában megadni, hogy mire van szükségünk. Az Angular ezeket automatikusan feloldja. Ezek a injektálható osztályok automatikusan *Singleton*-ként viselkednek, anélkül, hogy nekünk implementálni kellene a tervezési mintát.[38]

## 4.2. Ionic

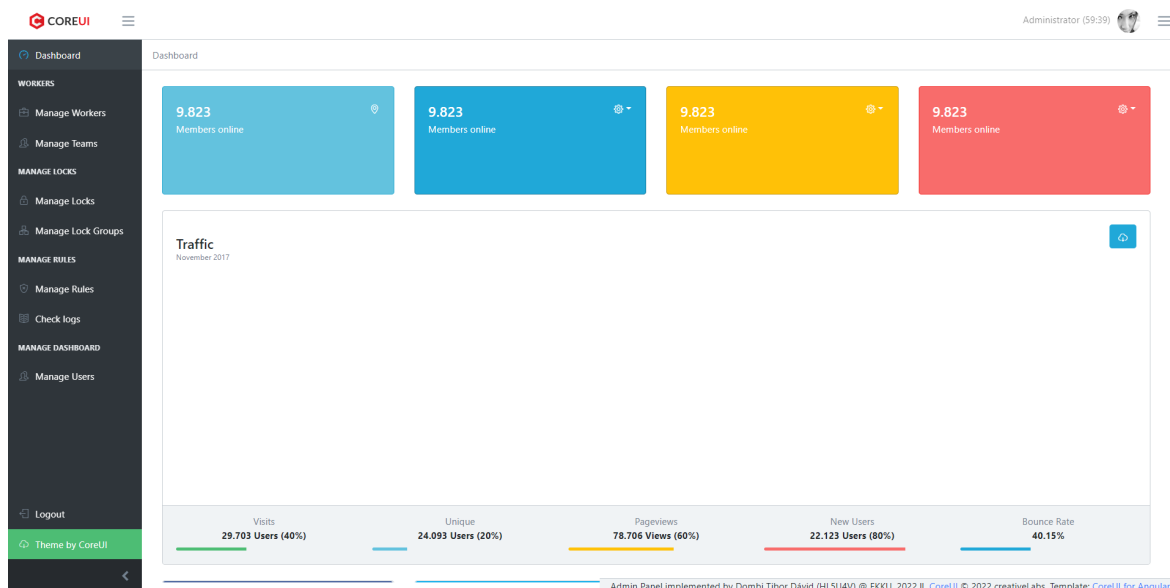
Az Ionic alapvetően egy eszközkészlet cross-platform alkalmazások fejlesztéséhez. Többek közt biztosít egy Angular csomagot, amiben számtalan javítás és kiegészítés van[39]. A három legfontosabb számomra az *Ionic Router* (új *life-cycle* metódusok), a *Storage Service* (intelligens lokális tárhely kezelés), és az *IonicServiceWorker* (PWA integráció).

Az *IonicServiceWorker* miatt, a felület PWA (*Progressive Web Application*)ként is működik. Az *Ionic Capacitor* integrációt is biztosít. Ez azt jelenti, hogy szükség esetén a Frontend natív Android vagy IOS alkalmazásként is lefordítható.[40]

## 4.3. Felület

Mivel a témám csupán a webservice, így nem szerettem volna túl sok időt a design részre fordítani, ugyan akkor azt sem szerettem volna, hogy egy „fehér alapon fekete betűs” felületet adjak át mint szakdolgozat. Szerencsére, számtalan ingyenes és szabadon felhasználható előre kész felület érhető el Angular-ra is. Az én választásom a *CoreUI*-ra esett. Az ez alapján készített nyitó oldalt látni a 4.1 ábrán.

Ez a csomag előre elkészített stíluslapokkal, komponensekkel, *layout*-al érkezik, és gyakorlatilag nulla konfigurációra van szükség a használatához[41]. A csomag alapból Bootstrap4-et tartalmaz, ezt én Bootstrap5-re cseréltem. A csere után azonban



4.1. ábra. Az admin felület

a csomagok telepítése *NPM*-el lehetetlenné vált kompatibilitási okokból, ugyanakkor *Yarn*-al továbbra is probléma (és figyelmeztetések) nélkül működik. A szakdolgozat leadásáig ezt a hibát nem sikerült orvosolni, de mivel saját munkafolyamatomban *Yarn*-t használok ahol csak lehet, illetve azzal a telepítés probléma nélkül lefut, így nem is kezelem prioritásként.

A CoreUI csomagot *Lukasz Holeczek* készítette, és az MIT licenz alatt tette elérhetővé. Ez a projekt szintén MIT licenst használ, és nem kerül eladásra semmilyen formában. Az eredeti licenst én is mellékeltem a forráskódban az **admin-panel** mappában[42]. Továbbá a felületen megtartottam a sablon weboldalára mutató linket (4.1 ábra, oldalsáv utolsó eleme).

### 4.3.1. Adattábla komponens

Mivel az összes erőforrásra, csak egyszerű CRUD műveleteket kell végeznünk, így nem érdemes mindet külön-külön implementálni. Sokkal célravezetőbb, ha készítünk egy univerzális komponenst, amit újra és újra fel tudunk használni az összes erőforráshoz.

Az admin panelben ez a `CrudTableComponent`<sup>1</sup>. Így elég csupán ezt elhelyezni a *template*-ben, és megadni a megfelelő paramétereket. Ezek közül csak az *erőforrás neve*, a *megjelenítendő oszlopok*, illetve az erőforráshoz tartozó *service* megadása kötelező. Opcionálisan letilthatjuk, vagy beállíthatjuk, hogy a keresés mely mezőkben történjen, figyelhetünk a különböző CRUD eseményekre, és még a táblázatuk stílusát is beállíthatjuk csupán a komponens paramétereivel.

<sup>1</sup> Forráskód: <https://github.com/dombidav/hl5u4v-thesis/blob/main/admin-panel/src/app/shared-components/crud-table/crud-table.component.ts>

A 4.1 kód a legegyszerűbb implementálást mutatja, kiegészítve a `refreshTable` paraméterrel, ami egy `BehaviorSubject` (speciális `Observable`) példány. A változásán keresztül bármikor frissíthetjük a táblánkat a szülő komponensből. Erre azért van szükség, mert az `<app-crud-table>` komponens nem tud arról, hogy megváltozott az adat, ha ezt a változtatást egy másik komponens teszi meg. Ilyen eset amikor a szerkesztéshez átnavigáljuk a felhasználót egy másik oldalra, majd vissza ide.

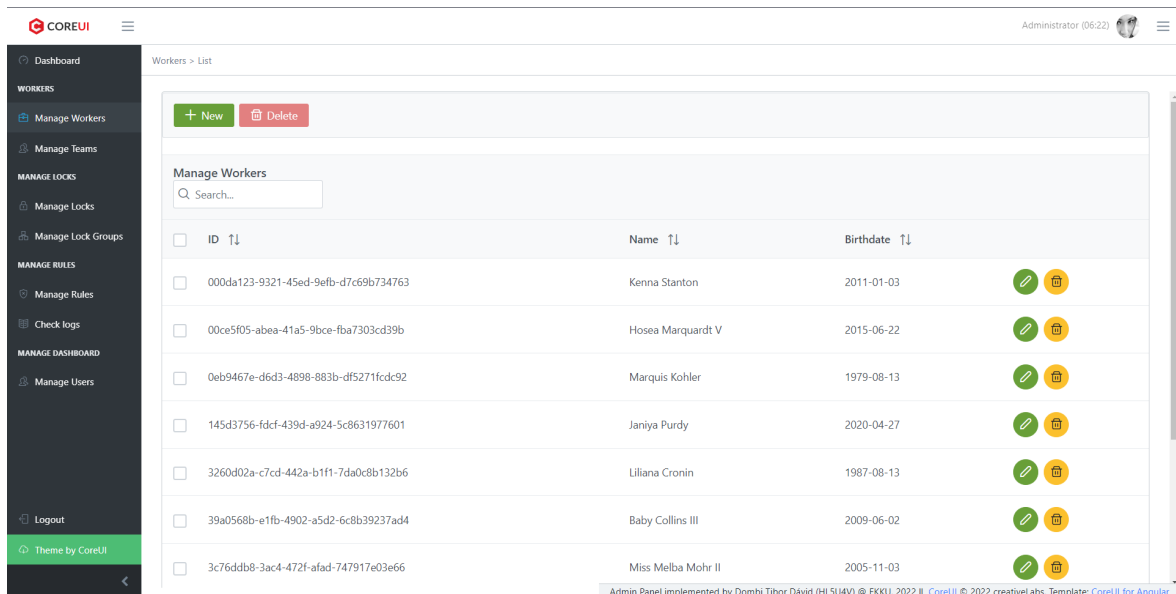
```

1 <app-layout>
2   <app-crud-table
3     resourceName='Workers'
4     [refresh]='refreshTable'
5     [service]='workersService'
6     [columns]="columns">
7   </app-crud-table>
8 </app-layout>

```

4.1. kód. `workers.component.html`

Ez az egyszerű kód (4.1) a 4.2 ábrán látható eredmény generálja nekünk. A táblázat oszlopai rendezhető növekvő vagy csökkenő sorrendben, szerkeszthetünk, létrehozhatunk és szerkeszthetjük az adatokat, illetve a kijelölés segítségével csoportosan is törölhetjük azokat. Mivel a komponens paraméterei között nem korlátoztuk a keresést, így az összes *megjelenített* mezőben keres.



4.2. ábra. A `CrudTableComponent` megjelenése

## 5. fejezet

# Hardveres megvalósítási javaslat

Bár a szakdolgozat csak a webservice résszel foglalkozik, mint projekt úgy gondolom fontos javaslatot tenni a hardveres megvalósításra, ha más miatt nem is, legalább azért hogy egy alapot adjon annak, aki ténylegesen szeretné beüzemelni az alkalmazást. Természetesen számtalan módon megoldható az eszközök összekapcsolása, valamint a webservice-al való kommunikáció, ez amit ide írok egy példa csupán.

### 5.1. Elosztott rendszerű megoldás

Ebben a példa szituációban minden zárt valamilyen Wi-Fi képes mikrokontroller vezérel. Az ESP8266 egy viszonylag olcsó ilyen eszköz. Kössük hozzá a zármechanika motorját, illetve a leolvasónkat, ami legyen például egy RFID leolvasó. Ezután az ESP-nkel kapcsolódjunk Wi-Fi-n keresztül az internetre. Az ESP rendelkezik egy beégetett gyártási számmal (Device ID), ez lesz az eszközünk ujjlenyomata (Például: f84020329c0e).

Amikor egy dolgozó a leolvasóhoz érinti a saját RFID kártyáját, akkor az ESP a kártya azonosítóját olvassa, és nem a dolgozóét, ezért az adatfelvitelkor a kártya azonosítóját kell megadjuk mint a dolgozó ujjlenyomata (például: f3949935-5425-4879-b713-bd9f3bd77241). A leolvasás után az ESP HTTP POST kérést indít a webservice felé, ami így néz ki (API-KEY hitelesítéssel):

```
1 POST /api/acs/confirm/f84020329c0e
2 Host: www.acs.example.com
3 Content-Type: application/json; charset=utf-8
4 Content-Length: {LENGTH}
5 Accept: application/json
6 X-API-KEY: kLhIlq0vCRcawFlD-tdMBPCHDUGhtuCu0
7
8 {
9     "worker_rfid": "f3949935-5425-4879-b713-
    ↪ bd9f3bd77241"
```

### 5.1. kód. Elosztott rendszer HTTP kérése

Figyeljük meg, hogy az eszköz ujjenyomata az URI-ben kapott helyet. Ezt az ujjenyomatot nyugodtan kicserélhetjük az eszköz tényleges, adatbázisban használt kulcsára, a hitelesítés akkor is működni fog. Ezután a szerver vagy engedélyezi, vagy valamilyen hiba üzenettel válaszol.

## 5.2. Csillagpontos megoldás

Amennyiben nem szeretnénk minden zárhoz külön-külön mikrokontrollert elhelyezni, elképzelhető az is, hogy egy nagyobb, több csatlakozási lehetőséggel rendelkező vezérlő egységet használunk (például egy Raspberry Pi-t), és ehhez, csillagpontosan hozzákötünk minden zárat és minden leolvasót.

A leolvasásra hasonlóan reagál, mint az elosztott rendszer, de itt a vezérlőnek tudnia kell, hogy melyik leolvasó melyik zárhoz tartozik. Az üzenet URI-jében az adott zár ujjenyomatát kell használni. Arra nincs szükség, hogy a vezérlő az API kulcsát is ismerje minden egyes zárnak, használhatja minden lekéréshez a saját API kulcsát.

## 6. fejezet

# Továbbfejlesztési lehetőségek

Természetesen, mint minden szoftver, ez az alkalmazás is rendelkezik hibákkal, hiányosságokkal. Ezeket a jövőben érdemes lenne kijavítani, vagy néhol újra gondolni.

### Bejelentkezés és Regisztráció

A bejelentkezés és a regisztráció során a felhasználó adatai (kiemelten a jelszó) titkosítatlanul kerülnek átadásra, és csak a szerveren kezeltem le. Ezt jó lenne már a kliens oldalon titkossá tenni. Erre megoldást jelenthet a JWT.

### Képek feltöltése

Jelenleg a webservice nem támogatja a képek feltöltését, pedig mind az adminisztrátori felület használóit, mind a dolgozókat könnyebb lenne beazonosítani.

### Modellek

Fentebb már részleteztem az Eloquent problémáit (lásd: 3.1.1 - Eloquent fejezet), ezt orvosolandó implementálni lehetne a `repository pattern` irányelveit.

### Rest API helyett GraphQL

Az alkalmazás méreteiből adódóan messze túl sok végponttal rendelkezik. Ez orvosolható lenne azzal, ha hagyományos Restful API helyett GraphQL-t használna az alkalmazás.



## Adminisztrátori tevékenységek naplózása

Jelenleg csak a zárák használata, valamint a Laravel-el kapcsolatos események kerülnek naplózásra. Érdeemes lenne az olyan eseményeket is feljegyezni, mint például, ha valaki új rekordot vitt fel, vagy épp törölt egyet.

## Szabály definíció tisztázása

A mostani struktúrája a hozzáférési szabály definícióknak elég körülményes, és nem evidens. Érdeemes lenne új struktúrában gondolkodni.

## Bouncer csomag elvetése

Bár szintaxisa miatt igen kedvelt csomag, sajnos mind teljesítményben, mind korlátoaltsága miatt alulmarad a Laravel `Spatie`-vel szemben.

## MongoDB támogatás

Habár sem a Laravel, sem a PHP nem támogatja alapból a MongoDB adatbázisokat, a PHP a megfelelő kiegészítővel, a Laravel pedig a megfelelő csomag telepítésével, viszonylag egyszerűen képessé tehető rá. A MongoDB egyre nagyobb népszerűségnek örvend az elmúlt években, így egyáltalán nem elképzelhetetlen, hogy valaki ezt szeretné használni az alkalmazáshoz. Azonban ekkor két probléma merül fel:

Az első, hogy a Bouncer egészen egyszerűen nem képes a Mongo környezetben működni.

A második, hogy a `MongoDB-Eloquent` csomag használatához egy teljesen másik absztrakt Model osztálytól kell örököltetnünk a modelljeinket, és a két absztrakt osztálynak nincs közös őse, így nem lehet egyszerűen „kicserélni” egyiket a másikkal. A legtöbb beépített függvény a Laravelben pedig az eredeti Model típust várja paramétereiben.

# Összegzés

A projekt célja az volt, hogy egy könnyen integrálható és tovább fejleszthető beléptetőrendszer szoftveres megoldása legyen. Rendkívül fontos volt számomra, hogy minél inkább nyitottá tegyem a projektet a közösség számára, méghozzá úgy, hogy ne csak egyszerűen elérhetővé tegyem a forráskódot, de minél több eszközt és dokumentációt mellékeljek. Továbbá, az alkalmazás természetesen nyitott a közösségi fejlesztés felé is.

A szakdolgozat leadása után a projekt fejlesztése nem áll meg. Mindenképpen implementálni szeretném a továbbfejlesztési lehetőségek között leírtakat, felhasználóbarátabbá tenni az admin felületet, illetve valós, fizikai megvalósítás is tervben van. Remélem, hogy ezt már a közösséggel együtt tehetem majd meg.

A rendszer második verzióját úgy tervezem megvalósítani, hogy programozói tudás nélkül is beüzemelhető és használható legyen. Ez a verzió akár szolgáltatásként is nyújtható lesz, folyamatos támogatás mellett. Tanulmányaimat követően elképzelhetőnek tartom, hogy majd saját vállalkozásomban foglalkozom ezzel a továbbfejlesztett verzióval.

Úgy gondolom, hogy hiányosságai ellenére a projekt jelen állapotában is alkalmazható egy valós probléma megoldására.

# Köszönetnyilvánítás

Szeretnék köszönetet mondani egyetemi oktatóimnak a tőlük kapott tudásért és segítségért.

Külön köszönöm Dr. Tajti Tibor segítségét ebben a projektben és egyéb dolgokban, valamint azért, hogy szakmai gyakorlatom alatt bevezetett az IoT világába.

Köszönöm hallgatótársaimnak, barátaimnak, hogy mindig voltak köztük olyanok, akik segíteni tudtak és figyeltek rám.

Végül de nem utolsó sorban szeretnék köszönetet mondani szüleimnek, családomnak, hogy mellettem álltak, és tanulmányaim alatt végig támogattak.

# Irodalomjegyzék

- [1] Github Octoverse Statisztika 2021,  
<https://octoverse.github.com/>
- [2] PHP Market Share és statisztikák  
<https://w3techs.com/technologies/details/pl-php>
- [3] Projekt GitHub repository-ja  
<https://github.com/dombidav/hl5u4v-thesis>
- [4] PHP Szerverek  
<https://serverguy.com/servers/php-servers/>
- [5] Laravel által támogatott DBMS-ek  
<https://laravel.com/docs/9.x/database>
- [6] Laravel Sail  
<https://laravel.com/docs/9.x/sail>
- [7] Telepítő varázsló forráskódja  
[https://github.com/dombidav/hl5u4v-thesis/blob/main/scripts/  
install-wizard.js](https://github.com/dombidav/hl5u4v-thesis/blob/main/scripts/install-wizard.js)
- [8] Postinstall dokumentáció  
<https://yarnpkg.com/package/postinstall>
- [9] Yarn  
<https://classic.yarnpkg.com/en/docs/cli/install>
- [10] Redis mint Cache  
<https://www.honeybadger.io/blog/laravel-caching-redis>
- [11] Laravel Mail  
<https://laravel.com/docs/9.x/mail>
- [12] Github Actions  
<https://github.com/features/actions>

- [13] Heroku  
<https://devcenter.heroku.com/articles/how-heroku-works>
- [14] PHP Unit  
<https://phpunit.readthedocs.io/en/9.5/writing-tests-for-phpunit.html>
- [15] Artisan Teszt a Laravel dokumentációjában  
<https://laravel.com/docs/9.x/console-tests#input-output-expectations>
- [16] Mi a CI/CD, bevezető  
<https://www.redhat.com/en/topics/devops/what-is-ci-cd>
- [17] Mi a RestAPI  
<https://www.redhat.com/en/topics/api/what-is-a-rest-api>
- [18] OpenAPI definíció  
<https://github.com/dombidav/hl5u4v-thesis/blob/main/docs/openapi.json>
- [19] OpenAPI definíció a stoplight.io-n  
<https://hl5u4v-thesis.stoplight.io/docs/hl5u4v-thesis/branches/main/>
- [20] Bread vs. CRUD  
[https://www.reddit.com/r/webdev/comments/3l2vio/buttonaction\\_language\\_crud\\_vs\\_bread/](https://www.reddit.com/r/webdev/comments/3l2vio/buttonaction_language_crud_vs_bread/)
- [21] HTTP metódusok  
<https://www.restapitutorial.com/lessons/httpmethods.html>
- [22] URI, erőforrás azonosítás  
[https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics\\_of\\_HTTP/Identifying\\_resources\\_on\\_the\\_Web](https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/Identifying_resources_on_the_Web)
- [23] Laravel Sanctum és a policy-k használata együtt  
<https://laracasts.com/discuss/channels/laravel/how-to-use-policies-with-laravel-sanctum>
- [24] JWT bevezető  
<https://jwt.io/introduction>
- [25] JWT Handbook  
Sebastián E. Peyrott, Auth0 Inc (Version 0.14.1, 2016-2018)  
[https://assets.ctfassets.net/2ntc334xpx65/o5J4X472PQUI4ai6cAcqg/13a2611de03b2c8edbd09c3ca14ae86b/jwt-handbook-v0\\_14\\_1.pdf](https://assets.ctfassets.net/2ntc334xpx65/o5J4X472PQUI4ai6cAcqg/13a2611de03b2c8edbd09c3ca14ae86b/jwt-handbook-v0_14_1.pdf)

- [26] JWT Usecase Workflow  
[https://documentation.softwareag.com/webmethods/api\\_gateway/yai10-5/10-5\\_API\\_Gateway\\_webhelp/index.html#page/api-gateway-integrated-webhelp/co-jwt\\_usecase\\_workflow.html](https://documentation.softwareag.com/webmethods/api_gateway/yai10-5/10-5_API_Gateway_webhelp/index.html#page/api-gateway-integrated-webhelp/co-jwt_usecase_workflow.html)
- [27] Laravel, Wikipedia the free Encyclopedia  
<https://en.wikipedia.org/wiki/Laravel>
- [28] Devecosystem 2020, PHP  
<https://www.jetbrains.com/lp/devecosystem-2020/php/>
- [29] Eloquent, Laravel Docs  
<https://laravel.com/docs/9.x/eloquent>
- [30] The biggest problem with eloquent accessors magic  
<https://laraveldaily.com/the-biggest-problem-with-eloquent-accessors-magic/>
- [31] Database testing, Laravel Docs  
<https://laravel.com/docs/9.x/database-testing/>
- [32] Repository Pattern in Laravel Application  
<https://www.twilio.com/blog/repository-pattern-in-laravel-application>
- [33] Database: Migrations, Laravel Docs  
<https://laravel.com/docs/9.x/migrations>
- [34] Migration probléma SQLite adatbázis alatt.  
(Megjegyzés: A szakdolgozati projekt létrehozásakor a probléma még fennállt)  
<https://github.com/laravel/framework/issues/35162>
- [35] Bouncer, Authorizing Users  
<https://github.com/JosephSilber/bouncer#authorizing-users>
- [36] Angular.io, What is Angular?  
<https://angular.io/guide/what-is-angular>
- [37] Angular.io, Components  
<https://angular.io/guide/what-is-angular#components>
- [38] Angular.io, Dependency Injection  
<https://angular.io/guide/what-is-angular#dependency-injection>
- [39] Ionic Dokumentáció  
<https://ionicframework.com/docs>

- [40] Ionic Dokumentáció, Capacitor integráció  
[https://ionicframework.com/docs/developing/android#  
running-with-capacitor](https://ionicframework.com/docs/developing/android#running-with-capacitor)
- [41] CoreUI Dokumentáció  
<https://coreui.io/angular/docs/4.0/getting-started/introduction>
- [42] Mellékelt CoreUI Licenz  
[https://github.com/dombidav/hl5u4v-thesis/blob/main/admin-panel/  
LICENSE](https://github.com/dombidav/hl5u4v-thesis/blob/main/admin-panel/LICENSE)

## NYILATKOZAT

Alulírott DOMBI TIBOR DÁVID, büntetőjogi felelősségem tudatában kijelentem, hogy az általam benyújtott, WEBSERVICE MENEDZSEL-  
HETŐ BELEPTETŐ RENDSZERHEZ 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 Katolikus Egyetem könyvtárába kerül elhelyezésre, ahol a könyvtár olvasói hozzájuthatnak.

Kelt: EGER, 2022 év APRILIS hó 17 nap.

Dombi Tibor

aláírás