

Ruander Oktatási Kft

# **SZAKDOLGOZAT**

**Kú Bence László**  
**webfejlesztő tanfolyam**

**Kiskunhalas, 2022**

## Tartalom

1.	Bevezetés.....	4
1.1	A dolgozat célkitűzései .....	4
1.2	A szakdolgozat felépítése .....	4
2.	Fejlesztésben használt nyelvek, technológiák.....	5
2.1	Nyelvek .....	5
2.1.1	HTML .....	5
2.1.2	CSS .....	5
2.1.3	JavaScript.....	6
2.1.4	PHP .....	6
2.1.5	SQL.....	6
2.1.6	Adatbázis .....	6
2.2	Technológiák .....	7
2.2.1	Bootstrap.....	7
2.2.2	FontAwesome.....	7
2.2.3	OOP.....	7
2.2.4	MVC .....	7
2.2.5	CRUD.....	8
2.2.6	REST .....	8
2.2.7	Composer .....	9
2.2.8	Laravel .....	9
2.2.9	Laravel Media Library .....	9
2.3	Programok .....	10
2.3.1	PHPStorm.....	10
2.3.2	XAMMP .....	10
3.	Fejlesztési dokumentáció .....	10
3.1	Adatbázis .....	11
3.2	MVC model .....	13
3.1.1	Model réteg .....	13
3.1.2	View réteg .....	14
3.1.3	Controller réteg .....	15
3.3	Végpontok .....	15
3.4	Beléptető rendszer .....	17
3.5	Jogosultságkezelő rendszer .....	19
3.6	Tartalomkezelő rendszer .....	21
3.6.1	Termékek CRUD.....	21

3.6.2	Blog CRUD.....	26
3.6.3	Slider CRUD.....	29
3.6.4	Publikus felületen történő megjelenítés. ....	32
3.6.5	Vezérlőpult felület kialakítása .....	32
4.	Felhasználói dokumentáció .....	33
4.1	Webes felületek bemutatása .....	33
4.1.1	Publikus felület bemutatása .....	34
4.1.2	Adminisztrációs felület bemutatása .....	37
5.	Összefoglalás .....	42
	Források.....	43

## 1. Bevezetés

Dolgozatom témáját párom ihlette, aki hobbiból készít saját kezűleg különböző méretű, színű, alakú horgolt tárgyakat, játékokat. Ezek a „termékek” annyira nagy sikert arattak az ismeretségi körben, hogy úgy gondoltam készítek első körben egy tartalom kezelő oldalt, ahol más is megismerkedhet a már elkészült munkáival az internet segítségével. Az oldal célja az, hogy a már kész termékek elkészítésének folyamatát, akadályait, és annak megoldásait tudassa az oldalra látogatóval, reklámozza ezeket, és természetesen a kapcsolat építés, esetleges segítség nyújtás, szakmai tapasztalat megosztása, majd későbbiekben ezen termékek értékesítése.

### 1.1 A dolgozat célkitűzései

A szakdolgozat célkitűzései 2 fő pontba sorolhatóak:

- **Felhasználói kezelés:** Bárhonnan eltudja érni az oldalt, tudjon hozzáadni új termékeket, felvehessen új tartalmat az oldalra vagy a meglévőt frissíthesse. Tapasztalatait, élményeit, csalódásait megtudja osztani a nagy közönséggel aki ellátogat a weboldalra. Későbbiekben webshop rendeléseit követhesse, kezelhesse.
- **Fogyasztói tájékoztatás:** A horgolás világába történő bevezetése a fogyasztóknak. Kész termékek bemutatása, elkészítéseinek folyamata, hírek, blogok közlése, kapcsolat teremtés. Hírek által horgoláshoz használt új vagy meglévő technológiák, anyagok, eszközök, vagy egyéb kiegészítők széleskörűbb szemléltetése. Blog bejegyzések révén való tapasztalat megosztás, tájékoztatás, tanácsadás.

### 1.2 A szakdolgozat felépítése

A szakdolgozat felépítése a következő:

1. Fejlesztésben használt nyelvek, technológiák, programok bemutatása.
2. Fejlesztési folyamatok sorrendi leírása.
3. Felhasználói leírás és útmutató a tartalom kezelésére, megosztására.

## 2. Fejlesztésben használt nyelvek, technológiák

Ebben a fejezetben írom le, hogy mely nyelveket, technológiákat és programokat használtam, mik ezek pontosan, mit lehet velük csinálni én magam miben használtam fel ezeket a projectem megvalósításában.

### 2.1 Nyelvek

A programozási nyelv a számítástechnikában használt olyan, ember által is értelmezhető utasítások sorozata, amivel közvetlenül, vagy közvetve közölhetjük a számítógéppel egy adott feladat elvégzésének módját. Több különböző programozási nyelv létezik, és mindegyiknek más-más előnyei és hátrányai vannak, illetve más jellegű feladatok elvégzésére használhatóak vagy praktikusak. Ezekből azokat a nyelveket sorolom fel amelyeket a fejlesztés során felhasználtam.

#### 2.1.1 HTML

A HTML angol mozaikszó, jelentése „HyperText Markup Language”, vagyis hiperszöveges jelölőnyelv. A weboldalak leíró jelnyelv, vagy az a kódrendszer, amit a weboldalak elkészítéséhez használnak. A HTML nem programozási nyelv, hanem kódnyelv, aminek segítségével akár a Jegyzetömb segítségével is alkothatóak weboldalak. A hipertext jelenti valójában az interneten található oldalakat (dokumentumokat), amelyek szöveg, kép, videó, hang, animáció, vagy ezek valamilyen kombinációjából áll. A HTML ezeknek a dokumentumoknak az elrendezését, formázását tartalmazza saját jelölőnyelvén. Bár a honlap készítő programok grafikai felületet is biztosítanak a weblap elkészítéséhez, mellette a legtöbb programnál hipertextként is kialakítható egy oldal.

#### 2.1.2 CSS

A CSS (Cascading Style Sheets) stílusleíró nyelv a számítástechnikában, amivel a HTML XHTML típusú dokumentum „stílusát”, megjelenését szabhatjuk testre. Például állíthatunk be egyedi szövegméretet, betűtípust vagy a képek méretét és elhelyezkedését szabhatjuk meg vele akár egyedi animációkat is készíthetünk bizonyos weboldali működéskor.

### 2.1.3 JavaScript

A JavaScript rövidebb nevén JS egy objektumorientált, gyengén típusos kliensoldali programozási nyelv, amely a weboldal futó programokat kezeli. A JavaScript alkalmazások forrás kódját a böngésző teljes mértékben letölti, értelmezi majd lefuttatja. Dinamikus weboldalak készítését teszi lehetővé.

### 2.1.4 PHP

A PHP (PHP: Hypertext Preprocessor) egy szerveroldali programozási nyelv, amely egy a felhasználótól eltérő helyen lévő számítógépen fut le. A működése viszonylag egyszerű, a felhasználó a saját „kliens” gépéről kérést (request) küld az eltérő helyen lévő számítógépnek „szerver” -nek, ahol a program lefut, mire a szerver küld egy választ (responde) . A felhasználó a csak program futásának az eredményét kapja meg, amely a tartalma a megjelenített weboldalnak (a megjelenésért, elhelyezkedésért, stílusért a HTML, CSS felel). A PHP-t helyi számítógépen is lehet használni különböző szerverek futtatásával amikre majd a későbbiekben kitérek.

### 2.1.5 SQL

Az SQL (Structured Query Language – Strukturált Lekérdezési Nyelv) A relációs adatbázisok kezelésére szolgáló legelterjedtebb programozási nyelv.

### 2.1.6 Adatbázis

Az adatbázis azonos minőségű (jellemzőjű), többnyire strukturált adatok összessége, amelyet egy tárolására, lekérdezésére és szerkesztésére alkalmas szoftvereszköz kezel. Az adatbázis (DB: Database) számítógépen (általában háttértárakon) tárolt adatok összessége. Az adatbázist egy adatbázis kezelő rendszer (DBMS: Database Management System) segítségével használhatjuk. Nem minden (számítógépen tárolt) adathalmazt tekintünk adatbázisnak.

## 2.2 Technológiák

### 2.2.1 Bootstrap

A Bootstrap egy nyílt forráskódú keretrendszer (framework), mely HTML, CSS, JavaScript technológiákat használ. Ez a framework a weboldalon lévő elemekhez előre definiált stílusosztályok hozzáadását teszi lehetővé. Alapvetően arra jó, hogy nagyon könnyedén, és minimális energia befektetéssel készítsen a fejlesztő bármely képernyőmérethez azonos (responsive) , igényes, modern, és arányos megjelenésű weboldalt.

### 2.2.2 FontAwesome

A FontAwesome egy CSS alapú betűkészlet és ikonkészlet amely segítségével egyszerűen, gyorsan adhatunk egyedi kinézetet weboldalunk számára.

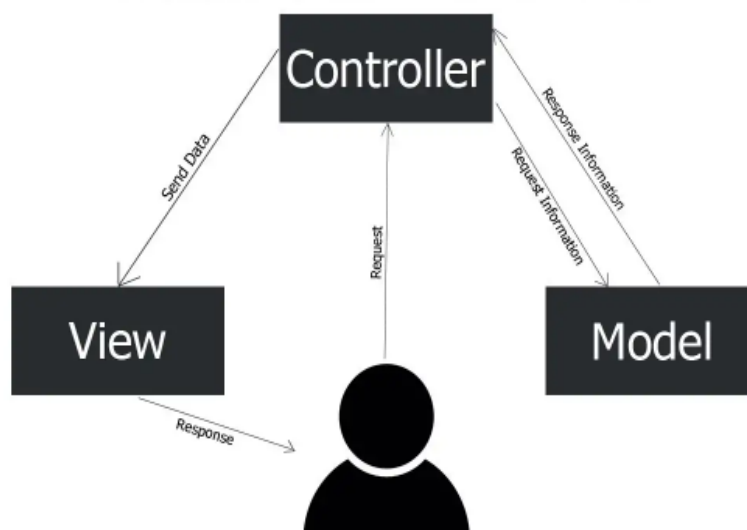
### 2.2.3 OOP

Az OOP (Object-Oriented Programming vagyis Objektum Orientált Programozás) az objektumok fogalmán alapuló programozási paradigma. Az objektumok egységbe foglalják az adatokat és a hozzájuk tartozó műveleteket. Az adatokat ismerik mezők, attribútumok, tulajdonságok néven, a műveleteket metódusokként szokták emlegetni. Az objektum által tartalmazott adatokon általában az objektum metódusai végeznek műveletet. A program egymással kommunikáló objektumok összességéből áll.[1][2] A legtöbb objektumorientált nyelv osztály alapú, azaz az objektumok osztályok példányai, és típusuk az osztály.

### 2.2.4 MVC

Az MVC (Modell-View-Controller vagyis Modell-Nézet-Vezérlő) a szoftver fejlesztésben használt programtervezési minta. Célja az, hogy a felhasználó elé tárt rengetek adatokat szétválassza adataira (modell) és megjelenítésre (view). A vezérlő (controller) szerepe ebben az, hogy ha a megjelenítési rétegből érkezik egy kérés az adatbázis felé akkor a kontroller fogja kiválasztani azt, hogy az adatbázisból milyen adat jelenjen meg a felhasználó előtt.

# Model-View-Controller



1. ábra MVC Model

## 2.2.5 CRUD

A *CRUD* egy meglehetősen elterjedt fogalom. Egy angol mozaikszó, amely a következő szavak kezdőbetűiből áll össze: Create-Read-Update-Delete. Ezek a szavak az adatok kezelésének négy alapvető módját írják le:

- Create: létrehoz
- Read: lekérdez, vagy olvas
- Update: módosít
- Delete (vagy Destroy): Töröl

## 2.2.6 REST

A REST egy HTTP protokollra fejlesztett kommunikációs architektúra típus, mely kliens és szerver közti kommunikáció megvalósítására használható. Kihasználja a HTTP állapotkódokat és metódusokat egyaránt, sőt a specifikáció megköveteli azok használatát, bár az egyedi fejlesztésű REST interfészek tervezése során ezekre gyakran nem ügyelnek a fejlesztők. A kérések URI-k használatával történnek, melyek erőforrásokat azonosítanak, és az URI-k



egységes interfészt biztosítanak a kliens számára. Minden kérésre azonos formátumban reagál a szerver, ez általában JSON, de használható még HTML és XML formátum is. Fontos, hogy a kérések állapotmentesek, tehát nem beszélhetünk „munkafolyamatról”, így az autorizáció folyamatos figyelmet igényel.

### 2.2.7 Composer

A Composer egy alkalmazás szintű függőségkezelő a PHP programozási nyelvhez, amely szabványos formátumot biztosít a PHP szoftverek és a szükséges könyvtárak függőségének kezelésére. Segítségével különböző alkalmazásokat tudunk telepíteni fejleszteni kívánt szoftverünkhöz, ami megkönnyíti annak fejlesztését.

### 2.2.8 Laravel

Nyílt forráskódú Symfony alapú webes keretrendszer, amely MVC architektúrájú webes alkalmazások fejlesztésére használt. Relációs adatbázis elérés és kisegítő alkalmazásokból áll, melyek a karbantartást és verzió kezelést segítik. Több éves tapasztalattal vagy akár kezdő PHP tudással rendelkezőknek is bevált könnyen kezelhető és átlátható rendszer. PHP API fejlesztéshez tökéletesen alkalmas és rengeteg közösségi útmutatóval van ellátva.

### 2.2.9 Laravel Media Library

Spatie fejlesztő csoport által publikált laravel kiegészítő csomag, ami a fileok társítását segíti elő Eloquent modellekkel. Ez egy nyílt forráskóddal rendelkező szoftver, mely mindenféle fájlt képes társítani az Eloquent modellekhez. Egyszerű, gördülékeny API-t biztosít a munkához.

## 2.3 Programok

### 2.3.1 PHPStorm

PhpStorm egy platformokat átívelő IDE (Integrated Development Environment) ami magyarul integrált fejlesztőkörnyezetet jelent. PHP, HTML ÉS JavaScript kódok írására alkalmas kódelemzéssel és hibamegelőzősi lehetőséggel. Java nyelven íródott, ezért rengeteg bővítményt írtak rá ezzel könnyítve a fejlesztők munkáját.

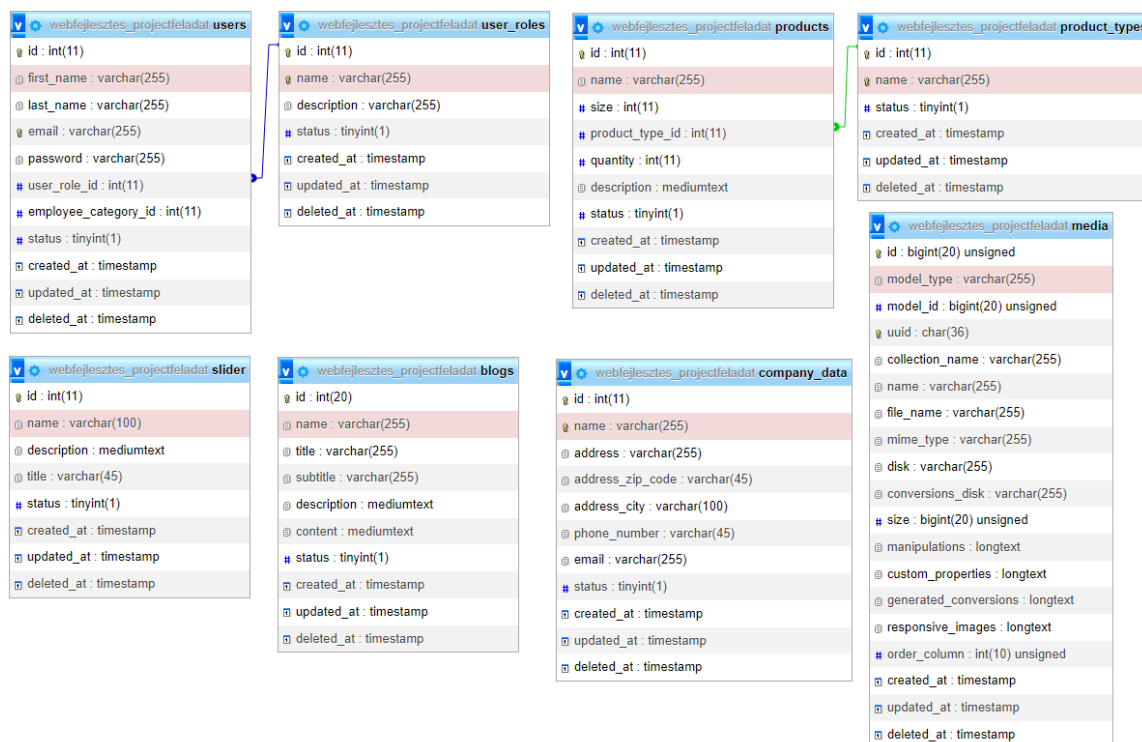
### 2.3.2 XAMMP

A XAMPP egy szabad és nyílt forrású platformfüggetlen webszerver-szoftvercsomag, amelynek legfőbb alkotóelemei az Apache webszerver, a MariaDB adatbázis-kezelő, valamint a PHP és a Perl programozási nyelvek értelmezői (végrehajtó rendszerei). Ez a szoftvercsomag egy integrált rendszert alkot, amely webes alkalmazások készítését, tesztelését és futtatását célozza, és ehhez egy csomagban minden szükséges összetevőt tartalmaz. A rendszer egyik nagy előnye az összehangolt elemek könnyű telepíthetősége. Ennek a programnak a segítségével futtattam lokálisan az adatbázist.

## 3. Fejlesztési dokumentáció

Project feladatomat laravel keret rendszerben valósítottam meg. Úgy gondoltam ennek a framework-nek az elsajátítása, gyakorlása sokkal előnyösebb lesz későbbi munkáim során, mivel folyamatosan frissül, nagy a támogatottsága, egyre népszerűbb, valamint a laravel szerkezete, felépítése megfelel az OOP, és MVC szabályainak. A fejlesztés az adatbázis struktúra elgondolásával , majd megvalósításával kezdődött. Létrehoztam egy webfejlesztés\_projectfeladat nevű adatbázist.

### 3.1 Adatbázis



2. ábra Adatbázis diagramm

Az alkalmazásom adatbázisa 8 darab táblából tevődik össze egyelőre. Minden tábla id mezője egy szám típusú automatikusan növekvő elsődleges kulcs, ez a mező szolgál az azonosításra.

A felhasználónak a users tábla tárolja a szöveges formában kereszt, és vezetéknévét, email címet valamint kódolt formátumban a felhasználó jelszavát. Utóbbi két érték elengedhetetlen ahhoz, hogy a felhasználó igazolni tudja magát, hogy jogosult a weboldal adminisztrációs felületének eléréséhez. A következő érték a user\_role\_id ami egy szám típusú mező, a nevéből is adódik, hogy ez egy idegen kulcs, amivel történik a users tábla összekapcsolása a user\_roles táblával. Az utolsó négy mező mindegyik táblában ugyanúgy megtalálható, ebből az első a státusz oszlop, arra szolgál, hogy ha törölni nem, de inaktívvá szeretnénk tenni bizonyos rekordot ennek a logikai típusú értéknek a változtatásával megtehetjük azt. Az utolsó három dátum típusú mező, amellyel nyomon tudjuk követni, mikor lett létrehozva, módosítva, vagy esetleg törölve az adott rekord.

A user\_roles tábla tárolja a jogosultság kezeléshez szükséges adatokat, id értéke egy automatikusan növekvő szám típusú érték, a tábla elsődleges kulcsa, ezen az értéken kapcsolódik a tábla a users táblához. Tárolja a jogosultság nevét szövegesen, és annak leírását.

A products tábla tartalmaz egy id mezőt, automatikusan növekvő, elsődleges kulcsként , a termék nevét, leírását, méretét, típusát, mely mező tartalmazza a product\_types táblából érkező idegen kulcsot, a termék mennyiségét leírását, és a négy darab állapot mezőt.

A product\_types mező azért jött létre, hogy a jövőbeni webshoppá alakításkor már a termékek külön kategóriájának listázása, vagy kategória szerinti keresés esetleg szűrés ne okozzon problémát. Ebből kifolyólag a product\_types id mezője, ami az elsődleges kulcsa a táblának, ezen a mezőn történik meg a products és product\_types táblák kapcsolása. Továbbá tárolásra kerül még a termék típus neve, és a négy darab állapot mező is.

A slider tábla fogja azokat az értékeket tárolni, amelyek majd megjelennek a publikus felületen a képváltón, mint cím és leírás. Ahogy a fenti tábláknál, itt is meg található a négy darab állapot mező. A név mezőt az adatbázisban való könnyebb keresés miatt vezettem be.

A blog táblán található mezők és annak értékei fogják kiszolgálni, a publikus oldalon lévő blog szekciót adatokkal. Ebben a táblában is a név mező a könnyebb kereshetőség miatt lett megvalósítva. Tárolásra kerül a blog bejegyzés címe, alcíme, rövidebb leírása, és a részletesebb tartalom. Szokásos módon a négy állapot jelző mező itt is megtalálható.

Company\_data tábla adatait a publikus felületen megjelenő lábléc ( footer ) fogja kezelni, mivel ez a tábla a cég adatokat tárolja. Mi a cég neve, hol található, telefonos és email elérhetősége.

Az utolsó tábla (egyelőre) a media, ezt a táblát a Laravel MediaLibrary csomag, hozta létre amely működésére, későbbiekben kitérek. Ez a tábla felel a képek adatainak tárolásáért. Itt a második mező a model\_type fogja azt az adatot tárolni, hogy a feltöltött képet melyik modellhez csoportosítsa. A model\_id azonosítja, hogy az adott modellen belül melyik id-hoz köti azt a képet, amelyik feltöltésre került. A collection\_name egy gyűjteménybe rendezi feltöltött képeinket, a name mező az eredetileg feltöltött kép nevét tárolja, a file\_name ugyan ezt adja vissza, csak fájl kiterjesztéssel együtt. A mime\_type a képfájl típusát tárolja, a disk pedig, hogy a laravel melyik mappából keresse a prezentálni kívánt képet. A generated\_coversations egy fontos mező, mivel itt egy olyan szöveges állományként tárolt adat található, amely a későbbiekben a méretezéshez fontos információ.

Ezzel az adatbázis bemutatása véget ért. A folyamat a laravel keretrendszer telepítésével folytatódott, amit a composer segítségével hajtottam végre. Miután a framework telepítése elkészült, csatlakoztattam az adatbázist a már feltelepített applikációhoz mysql segítségével a .env fileon belül.

```

10
11 DB_CONNECTION=mysql
12 DB_HOST=127.0.0.1
13 DB_PORT=3306
14 DB_DATABASE=webfejlesztés_projectfeladat
15 DB_USERNAME=root
16 DB_PASSWORD=

```

3. ábra .env-ben adatbázis kapcsolódás

Ezt követően létrehoztam az osztályokat a laravel beépített un. artisan parancssori felületével, ezzel egyidőben megadtam további paraméterként, hogy az osztályokhoz hozzon létre migrációs sémát is, valamit készítse el az osztályhoz tartozó controllert is. Erre egy példa:

```

php artisan make:model User -m -c

```

4. ábra Model készítése artsan segítségével

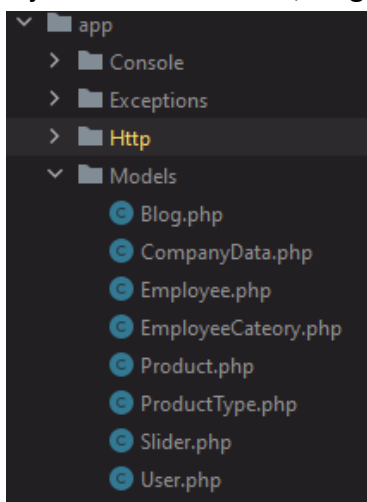
Az „-m” jelzi a kívánt migrációs séma kialakítását, „-c” pedig hogy a controller is alakítsa ki.

## 3.2 MVC model

A project az MVC szabályainak megfelelően készült, így elkülönül a megjelenítő réteg a modell rétegtől.

### 3.1.1 Model réget

Az alkalmazás által kezelt információk tartomány-specifikus ábrázolása. A „modell” réteg az app mappán belüli Models mappában található. Ezen modelleken belül van lehetősége a fejlesztőnek beállítani, hogy melyik modell melyik adat táblát használja, és mi érzékeljen elsődleges kulcsnak az adott táblán belül.



6. ábra Modell mappa struktúrája

```

class User extends Authenticatable
{
    use HasApiTokens, HasFactory, Notifiable, SoftDeletes;

    protected $table = 'users';
}

```

5. ábra Tábla és elsődleges kulcs definiálása

Valamint ha kettő vagy több tábla össze van kapcsolva, akkor a másik osztály segítségével itt meg lehet adni a tábla kapcsoláshoz az oszlop nevét is, ezt egy un. belongsTo beépített laravel függvény kezel, amit egy role() függvényen át hívok meg.

```
public function role(){  
    return $this->belongsTo(related: UserRole::class, foreignKey: 'user_role_id');  
}
```

7. ábra tábla kapcsolat belongsTo segítségével

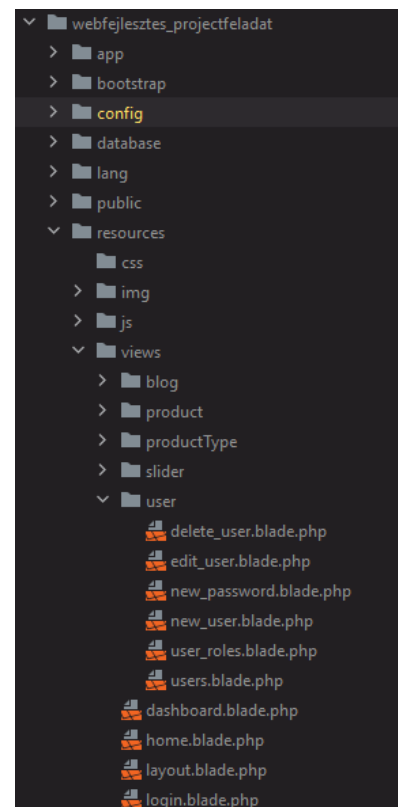
Így a későbbiekben nem kell az sql kérdésben újabb táblakapcsolást létrehozni, role névvel elérjük azt.

### 3.1.2 View réteg

A „view” réteg az resources mappán belüli view mappában található. Igyekeztem itt is tartani a mappa szerkezetet a Classok alapján.

Ezeknél a fájlknál valósul meg a parciális oldal, mivel minden endpoint egy másik oldalra mutat, ahol más tartalom jelenik meg, mindegyiket egy layout.blade.php nevezetű oldal foglal magába. A layout tartalmazza a megjelenítéshez tartozó legfontosabb elemeket, hivatkozásokat. Tartalmazza például a komplett HTML vázat, CSS hivatkozásokat. A Blade támogatja a html felületek öröklését, bizonyos szekciók meghívását, így a felületek elemeit elég egyszer megírni, és a későbbiekben ezeket csak meg kell hívni a különböző beépített Blade tag-ekkel.

1. @section: az örökölt layout file-ban definiált (@yield) szekciót helyettesíti ide, csak az itt megadott tartalmat írja bele
2. @yield: később ezt a layout-ot megöröklő blade file-okban megadott tartalom (@section) ide kerül behelyettesítésre
3. @extends: ezzel a tag-gel örököltethetünk le layoutot az éppen szerkesztett view-hoz

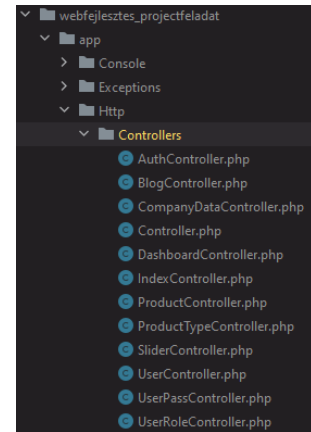


8. ábra View réteg mappaszerkezete

### 3.1.3 Controller réteg

Az utolsó réteg a vezérlő réteg az app mappán belüli Http és azon belül a Controllers mappában találhatóak.

A controllerek feladata az, hogy a végpontok segítségével adatot kérjenek Modell réteg segítségével az adatbázisból, és ezeket az adatokat továbbítsák a megjelenítő rétegbe, hogy az a felhasználó előtt igényesen jelenjen meg.



9. ábra Controller réteg mappaszerkezete

### 3.3 Végpontok

A végpontokra indítanak hívást a kliensek, és a végpont a használt http metódus alapján dönti el, hogy milyen választ adjon. Alapvetően kettő féle hívást használok a weboldalamon.

```

/*****
/*****Felhasználók endpointok*****/
/*****
Route::get( uri: '/users',[UserController::class,'usersList']);
Route::get( uri: '/userAdd',[UserController::class,'UserAdd']);
Route::post( uri: '/userSave',[UserController::class,'userSave']);
Route::get( uri: '/userEditById/{id}', [UserController::class,'userEditById']);
Route::post( uri: '/userUpdate', [UserController::class,'userUpdate']);
Route::get( uri: '/userDeleteById/{id}', [UserController::class,'userDeleteById']);
Route::get( uri: '/userDelete/{user}',[UserController::class,'userDelete']->name( name: 'userDelete');
```

10. ábra Felhasználó végpontok

Ha „get” kérést intéz akkor az alkalmazás tudni fogja, hogy akkor szeretnék kiolvasni arról a bizonyos dologról, amit ezen az endpointon keresztül kiszolgál nekem. Ha „post” kérést hajtok végre, akkor pedig valamilyen műveletet kell majd végre hajtania a controllernek. Ezeket az utasításokat a routes mappán belül a „web.php” nevű fájlban készítettem el. Ezek egy egyszerű route összekötések bizonyos controller php fájlokkal és egy általam írt, abban található CRUD alapú függvényekkel. A CRUD (Create, Read, Update, Delete) a tartós tárolás négy alapvető funkciói. Mint a mellékelt képek mutatják érkezik a kliens oldalról egy: `http://localhost:8000/users` „get”-es kérés, ennél a pontnál a controller réteg tudni fogja hogy a /users végpontnál futtassa le a „usersList()” nevű metódust ami adatbázisból kérje le az összes user-t

```
Route::get( uri: '/users',[UserController::class,'usersList']);
```

11. ábra Felhasználó listázása endpoint

```

class UserController extends Controller {
    public function usersList() {
        $users = User::with( relations: 'role' )->get();
        return view( view: 'user/users', ['users' => $users] );
    }
}

```

12. ábra usersList metódus

és térjen vissza a megfelelő megjelenítési rétegen a megfelelő adathalmazzal.

```

<table class="table table-dark table-striped">
    <thead>
        <tr>
            <th>#</th>
            <th>Név</th>
            <th>Email</th>
            <th>Jogosultság</th>
            <th>Állapot</th>
            <th>Műveletek</th>
        </tr>
    </thead>
    <tbody>
        @php
            $i=1;
        @endphp
        @foreach($users as $user)
            <tr>
                <td>{{ $i++ }}</td>
                <td>{{ $user->last_name }} {{ $user->first_name }}</td>
                <td>{{ $user->email }}</td>
                <td>{{ $user->role->name }}</td>
                <td>{{ $user->status ? 'Aktív' : 'Inaktív' }}</td>

                <td>
                    @if(\Illuminate\Support\Facades\Gate::allows('superadmin'))
                        <a class="btn btn-warning" href="{{ url('userEditById/'.$user->id) }}"> <i
                            class="fa-solid fa-edit"></i>
                            Módosít</a>
                        <a class="btn btn-danger" href="{{ url('userDeleteById/'.$user->id) }}"> <i
                            class="fa-solid fa-trash"></i>
                            Töröl</a>
                    @endif
                </td>
            </tr>
        @endforeach
    </tbody>
</table>

```

13. ábra Listázó megjelenítő réteg

Ugyan ezen az elven működik az új felhasználó hozzáadása is, annyi különbséggel, hogy elsőnek érkezik egy „get”-es kérés ami visszatér egy „új felhasználó felvitele űrlappal, majd a <form> tag attribútumaként megadott „post” metódussal meghívja a userSave metódust, ami viszont már adatot ment az adatbázisba, majd ha a művelet sikeres térjen vissza a listázó oldallal és az oda már betöltött adatokkal. Ugyan ez a logika alapján működik az Update, és a Delete funkció is, annyi különbséggel, hogy amikor egy adatot próbálunk törölni véglegesen az adatbázisból, nem kerül törlésre, csak egy deleted\_at rekordot kap, amely azért hasznos,



mert ha még sincs az adatbázisban, de van hozzá fűzendő más adat, akkor ne fusson hibára az adatbázis.

```
Route::get( uri: '/userAdd',[UserController::class,'UserAdd']);
Route::post( uri: '/userSave',[UserController::class,'userSave']);
```

14. ábra Új felhasználó mentése végpontok

```
public function userAdd() {
    if (! Gate::allows( ability: 'superadmin' )) {
        abort( code: 403);
    }
    $userRoles = UserRole::query();
    $employeeCategories = EmployeeCategory::query();
    return view( view: 'user/new_user', [
        'userRoles' => $userRoles->get(),
        'employeeCategories' => $employeeCategories->get(),
    ]);
}

public function userSave(Request $request) {
    if (! Gate::allows( ability: 'superadmin' )) {
        abort( code: 403);
    }
    $request->validate([
        'firstName' => 'required',
        'lastName' => 'required',
        'email' => 'required|email',
        'password' => 'required',
        'userRole' => 'required',
        'employeeCategory' => 'required',
    ]);
    $firstName = $request->firstName;
    $lastName = $request->lastName;
    $email = $request->email;
    $password = Hash::make($request->password);
    $status = $request->status ? 1 : 0;
    $userRole = $request->userRole;
    $employeeCategory = $request->employeeCategory;

    $user = new User();
    $user->first_name = $firstName;
    $user->last_name = $lastName;
    $user->email = $email;
    $user->password = $password;
    $user->status = $status;
    $user->user_role_id = $userRole;
    $user->employee_category_id = $employeeCategory;
    $user->save();
    return redirect( to: '/users');
}
```

15. ábra Új felhasználó mentése függvények

### 3.4 Beléptető rendszer

A weboldalon beléptető rendszert hoztam létre, így aki regisztrálva van egyedi email címmel és hozzátartozó jelszóval az be fog tudni lépni az admin felületre, ezekután tudja kezelni majd az oldalt. Ezt azért tartom fontosnak leírni, mivel a laravel „middleware”

hitelesítési mechanizmusát használtam az illetéktelen behatolások ellen, magyarul azokat a végpontokat amiket az oldalon valaki bejelentkezés nélkül szeretné látni, azt ne tehesse. Ezért ezeket a végpontokat egy csoportba foglaltam.

```
Route::middleware('auth')->group(function(){
```

16. ábra Védett végpontok csoportba foglalása

Ahhoz hogy ez működőképes legyen, létre hoztam egy AuthController nevű vezérlőt, amely megvizsgálja, hogy előzőleg már be van-e jelentkezve az oldalra látogató, ez is kettő végpontból áll, egy „get”-es és egy „post”-os kérésből.

```
Route::get('url: '/login',[AuthController::class,'showLogin'])->name('login');  
Route::post('url: '/login',[AuthController::class,'login']);
```

17. ábra Bejelentkezési felület végpontja

```
class AuthController extends Controller {  
    public function showLogin() {  
        return view('login');  
    }  
  
    public function login(LoginRequest $credentials) {  
        if (Auth::attempt(['email' => $credentials->email, 'password' => $credentials->password])) {  
            return redirect(to: 'dashboard');  
        }  
        return redirect(to: 'login')->with('failed', 'Hibás felhasználónév, vagy jelszó.');
```

18. ábra Bejelentkezési és kiléptetési metódusok

Ha helyes az emailhez tartozó jelszó, megtörtént az azonosítás, átirányítjuk a felhasználót az adminisztrációs felület dashboard-jára (vezérlőpultjára), ellenkező esetben a rendszer tájékoztatja a felhasználót arról, hogy a megadott email cím vagy jelszó helytelen, ebben az esetben a vezérlő újból a bejelentkező felületre irányítja a felhasználót.

A LoginRequest osztály felel a hibakezelésért, ha a felhasználó nem töltött ki bizonyos mezőket, vagy az emailcím nem megfelelő formátummal lett begépelve.

Ha pedig a kijelentkezés végpontra kattint, akkor fusson le a kiléptetése a felhasználónak a weboldáról.

```
Route::get('url: '/logout',[AuthController::class,'logout']);
```

19. ábra Kijelentkezés végpontja

### 3.5 Jogosultságkezelő rendszer

Lényegesnek tartottam, hogy készítsek az alkalmazásomba egy jogosultság kezelő rendszert is, a jövőre nézve. Ezt a funkcionalitást azért tartom kulcsfontosságú résznek, mivel ha ezt a tartalom kezelő rendszert a jövőben webáruházzá szeretnénk tovább fejleszteni akkor lehet, hogy nem csak egy felhasználó fogja majd kezelni. Ennek tudatában, első körben három felhasználói típust képzeltem el.

Az első a „super admin” rangot kapta , mely felhasználónak mindenhez van hozzáférése, esetlegesen új felhasználót tud felvinni (mivel egyelőre regisztrálásra nincs lehetőség az oldalon), tudja azt módosítani, törölni, esetleg más felhasználó jelszavát is módosítani .

A második „admin” ez a felhasználói rang már korlátozott hozzáféréssel rendelkezik, ő nem tud új felhasználót felvinni a rendszerbe, módosítani, törölni azt. Ez a felhasználói szint az oldal kinézetét tudja testreszabni, vagy új tartalmat felvinni, módosítani vagy törölni.

A harmadik az „ügyintéző”. Ez a jogosultság arra elég, hogy új terméket vagy tartalmat töltsön fel a publikus felületre, az adminisztrációs felületen keresztül.

Ezen titulusok kezelésére hoztam létre egy user\_roles nevezetű adat táblát, amelyet tábla kapcsolással kötöttem hozzá a már meglévő users táblához. Első gondolatra megvizsgáltam, hogy aki nem rendelkezik bizonyos user\_role\_id -val az ne lássa a weben a bizonyos menüpontokat.

```
@if(Auth::user()->user_role_id==1)
    <li>
        <a class="dropdown-item" href="/userAdd">
            <i class="fa-solid fa-user-plus"></i> Új felhasználó
        </a>
    </li>
@endif
```

20. ábra Jogosultsággal kezelt menüpont

De ha valaki rendelkezik egy kis jártassággal és kézzel beírja a böngésző URL-jébe a keresendő végpontot akkor bizony mégis megjelenik az a menüpont amit védeni szerettem volna.

Ezt az úgynevezett autorizációt egy beépített laravel Osztállyal valósítom meg, melyet „Gate”-nek neveznek. Ezeket a jogosultságokat App/providers/AuthServiceProvider.php-ban le kell definiálni,

```
/**
 * Register any authentication / authorization services.
 *
 * @return void
 */
public function boot()
{
    $this->registerPolicies();

    Gate::define(ability: 'superadmin', function (User $user) {
        return $user->user_role_id === 1;
    });
    Gate::define(ability: 'admin', function (User $user) {
        return $user->user_role_id === 2;
    });
}
```

21. ábra Jogosultságok definiálása AuthServiceProvider-ben

majd azokon a helyeken ahol szeretném használni egy elágazásba teszem a függvényhívást.

```
public function userAdd() {
    if (! Gate::allows(ability: 'superadmin')) {
        abort(403);
    }
    $userRoles = UserRole::query();
    return view('user/new_user', [
        'userRoles' => $userRoles->get(),
    ]);
}

public function userSave(Request $request) {
    if (! Gate::allows(ability: 'superadmin')) {
        abort(403);
    }
    $request->validate([
```

22. ábra Jogosultsággal védett metódusok

### 3.6 Tartalomkezelő rendszer

Annak érdekében , hogy weboldalunk modern legyen, hatékonyan működjön, és kezelése ne legyen bonyolult, készítettem egy tartalom kezelő rendszert az adminisztrációs felületen. A tartalomkezelő rendszer célja az, hogy a felhasználó bármilyen szakmai tudás nélkül, egyedi tartalmakat tudjon megjeleníteni azon a felületen, amit a fogyasztó fog látni. Ennek működése szintén megfelel az MVC modell szabályainak. Az adatbázis már adott, így már csak a hozzá kapcsolódó modellt, vezérlőt, megjelenítési réteget és adattáblát kellett elkészítenem a felhasználói kezelés alapján.

#### 3.6.1 Termékek CRUD

Elsőnek a termékek (products) kezelését készítettem el, mivel ez a csoport legfontosabb tényezője ennek a weboldalnak. A beépített artisan parancssori felülettel létrehoztam a Product modellt, majd ezzel egyidejűleg elkészítettem a hozzá tartozó ProductControllert is. Ennél a pontnál meg kell jegyezni, hogy a termékekhez kapcsolódó képek feltöltéséhez használom a „Larvel-MediaLibrary”-t. Ez a csomag bármelyféle fájl típusú képet társít az én meglévő laravel projectemhez. Telepítése composerrel valósult meg, majd ennek az Osztálynak is készíteni kellett egy migrációs táblát, amely nem magukat a képeket fogja tárolni, hanem a képek minden olyan szükséges adatát, amely ahhoz szükséges, hogy a képek meghívásakor az alkalmazás tudja melyik mappában keresse a képet, és melyik modellhez kösse azt azonosító szám alapján. Miután a csomag telepítése és konfigurálása megtörtént, a Products nevű modellben meg kellett határozni azt, hogy a termékhez feltöltött képek, milyen gyűjteményben, és méretekből kerüljenek letárolásra. Erre két féle függvényt kellett meghívni, az első regisztrálja a gyűjteménybe a képeket,

```
public function registerMediaCollections(): void {  
    $this->addMediaCollection( name: 'image');  
}
```

23. ábra Kép gyűjteménybe regisztrálása

a másodikkal pedig a különböző méretű kép átalakításokat szabhatjuk meg.

```
public function registerMediaConversions(Media $media = null): void
{
    $this->addMediaConversion( name: 'thumb')
        ->height( height: 75);
    $this->addMediaConversion( name: 'public')
        ->width( width: 200)->height( height: 200);
}
```

24. ábra Kép konverziók megadása

A termékek esetében kettő féle mérettel dolgozok. Az első a listázáson megjelenő kép, ellenőrzés és tájékoztatás céljából „thumb” néven 75x75 pixel méretben kerül lementésre. A második a társadalmi oldalon megjelenő kép 200x200 pixel méretben „public” néven kerül letárolásra.

A ProductControlleren belül létrehoztam a CRUD szabályainak megfelelő függvényeket, a web.php fájlban pedig elkészítettem a már meglévő eljárások végpontjait. Ezeket fogom most részletezni.

- (create) Ha még nincsenek termékeink legyen lehetőség felvinni újat, ezért készítettem egy végpontot ami a /productsAdd. Ezen a végponton meghívásra kerül a productAdd függvény a controller rétegben, ami egy űrlap betöltését hajtja végre a megjelenítésirétegben.

```
Route::get ( uri: '/productAdd', [ProductController::class, 'productAdd']);
```

25. ábra Új termék létrehozásának végpontja

```
public function productAdd() {
    $productTypes = ProductType::all();
    return view( view: 'product.new_product', ['productTypes' => $productTypes]);
}
```

26. ábra Új termék hozzáadása függvény

Ezen a ponton a vezérlő réteg egy másik adatbázis műveletet is végre hajt annak érdekében, hogy az űrlap felületen az új termék felvételekor kitudjuk választani azt , hogy az adott terméket melyik termék kategóriába soroljuk majd. Ez a lekérdezés azért lényeges, hogy a

felhasználó ne számokat lásson a lenyíló menüben ,amikor kiseretné választani a kívánt kategóriát, hanem az szöveges formában jelenjen meg.

```
<select class="form-select" name="productTypeId" id="productTypeId">
  @foreach($productTypes as $productType)
    <option value="{{ $productType->id }}">{{ $productType->name }}</option>
  @endforeach
</select>
```

27. ábra Dinamikusan megjelenő legördülő menü

Miután a felhasználó kitöltötte az adatokat az úrlapon, és rákattint a mentés gombra, egy „post” kérés fut le. Ezt a post kérést a form elem metódusaként adtam meg, valamint mikor a felhasználó rákattint a küldés gombra, az „action” attribútumaként megadtam azt az útvonalat ami a mentés végpontja lesz.

```
<form action="{{ route('productSave') }}" method="POST">
```

28. ábra Termék mentésének action-je

Ez a productSave egy végpontra mutat ahol a mentési metódus meghívásra kerül.

```
Route::post( uri: '/productSave', [ProductController::class, 'productSave'] )->name( name: 'productSave');
```

29. ábra Termékmentésének végpontja

A productSave függvény lefutásakor minden beviteli mező értéként letárolja egy külön változóba a name attributum értékének alapján , a termék nevét, méretét, kategória azonosítóját, mennyiségét, a hozzátartozó leírást, állapotát, és a termékhez mellékelt kép elérési útvonalát. A Media Library csomag segítségével hozzáadjuk a kép adatait a media táblába, hogy a későbbiekben ennek segítségével a kép megjelenjen a kívánt helyen. A kép lokálisan az alkalmazás gyökérkönyvtárában található „storage/app/public” könyvtárba kerül lementésre. Adatbázisba, csak a kép elérésének útvonala kerül szövegesen.

```
public function productSave(Request $request, Product $product) {
    $name = $request->input( key: 'name');
    $size = $request->input( key: 'size');
    $productTypeId = $request->input( key: 'productTypeId');
    $quantity = $request->input( key: 'quantity');
    $description = $request->input( key: 'description');
    $status = $request->input( key: 'status') ? 1 : 0;
    if ($request->hasFile( key: 'pictureLink') && $request->file( key: 'pictureLink')->isValid()) {
        $product->addMedia($request->file( key: 'pictureLink'))->toMediaCollection( collectionName: 'image');
    }
    $product->name = $name;
    $product->size = $size;
    $product->product_type_id = $productTypeId;
    $product->quantity = $quantity;
    $product->description = $description;
    $product->status = $status;
    $product->save();
}
```

30. ábra Új termék mentésének eljárása

Ha hiba nélkül lefut a függvény akkor az vissza irányít minket a listázó oldalra.

- (read) Mivel a felhasználó szeretné látni, hány darab termék van már feltöltve az adatbázisába, ezért készítettem listázó végpontot, amely a /products-ra mutat.

```
Route::get( uri: '/products', [ProductController::class, 'productsList']);
```

31. ábra Termék listázásának végpontja

Ez a végpont hívja meg a controller rétegben a productList metódust, ami lekéri az adatbázisban szereplő összes terméket, majd azt tovább küldi a products.blade.php megjelenítő rétegnek.

```
public function productList() {  
    $products = Product::all();  
    return view( view: 'product.products', ['products' => $products]);  
}
```

32. ábra Termékek listázásának metódusa

Megvizsgálom azt a megjelenítő rétegben, hogy van e már termék lementve az adatbázisba, ha nincs, erről tájékoztatom a felhasználót, természetesen ha pedig van, akkor listázza ki őket.

```
@if(count($products)==0)  
    <div class="alert alert-danger text-center m-3" role="alert">Sajnálom, egyelőre nincs egy blog  
        bejegyzése sem amit listázni tudnék!  
    </div>
```

33. ábra 0 darab termék esetén a tájékoztatás

- (update) Ha esetleg a felhasználó hibásan töltött fel egy terméket, vagy a jövőben módosítani szeretne valamely értékén, akkor arra is van lehetőség. E művelet a végpontja a /productEdit, ahol már úgy kerül elküldésre a kérés, hogy az tartalmazni fogja a módosítandó termék azonosítóját. Ez a paraméter azért kulcsfontosságú, mert ezzel a számmal fogja lekérni az adott terméket a függvény az adatbázisból.

```
Route::get( uri: '/productEdit/{id}', [ProductController::class, 'productEdit'])->name( name: 'productEditById');
```

34. ábra Termék módosításának get-es végpontja

A függvény tehát vár paraméterül egy \$id értéket, amivel kitudja választani azt az egy terméket, melynek az azonosítója megegyezik a paraméterként adott értékkel. Az adott terméknek lekéri az összes adatát, valamint a szövegesen lenyíló menükhöz szükséges termék kategóriákat, majd továbbítja ezeket a megjelenítési rétegnek ahol az űrlap beviteli mezőinek értékeiként jelenjenek majd meg.



```
public function productEdit($id) {
    $product = Product::where('id', '=', $id)->first();
    $productTypes = ProductType::all();
    return view('product.edit_product', compact('var_name: 'product', ...var_names: 'productTypes'));
}
```

35. ábra Termék módosításának metódusa

Ezt az id értéket le kell tárolni egy rejtett mezőben az űrlapon, mely segítségével az update függvény majd azonosítani tudja, hogy melyik azonosítóval rendelkező termék adatait frissítse.

```
<input type="hidden" name="id" id="id" value="{{ $product->id }}">
```

36. ábra Rejtett id értékkel rendelkező mező

A módosítások elvégzése után elküldjük az űrlapot „post” kéréssel egy /productsUpdate végpontra.

```
Route::post('uri: '/productUpdate', [ProductController::class, 'productUpdate'])->name('name: 'productUpdate');
```

37. ábra Termék módosításának post-os végpontja

Az ott található productsUpdate függvény hívja meg a művelet végrehajtó metódust, miután már a módosított adatokkal rendelkező beviteli mezők értékei le lettek tárolva külön változókba. Ahhoz hogy a függvény tudja, hogy melyik termék adatainak értékeit kell frissítenie, a rejtett beviteli mező értéket veszi alapul. Ha az azonosítás megtörtént le fut az update metódus, majd irányítson vissza a listázó oldalra.

```
public function productUpdate(Request $request) {
    $id = $request->id;
    $name = $request->name;
    $size = $request->size;
    $productTypeId = $request->productTypeId;
    $description = $request->description;
    $status = $request->status ? 1 : 0;
    $product = Product::where('id', '=', $id)->get()->first();
    Product::where('id', '=', $id)
        ->update([
            'name' => $name,
            'size' => $size,
            'product_type_id' => $productTypeId,
            'description' => $description,
            'status' => $status,
        ]);
    if ($request->hasFile('key: 'pictureLink') && $request->file('key: 'pictureLink')->isValid()) {
        $product->clearMediaCollection('image');
        $product->addMedia($request->file('key: 'pictureLink')->toMediaCollection('image');
    }
}
```

38. ábra Módosított termék adatainak metése adatbázisba

- (delete) Ha törölni szeretnénk egy terméket a /productDeleteById végpontra kell kérést küldeni egy paraméterként megadott értékkel, ez az érték alapján beazonosítja, majd törli a terméket azt követően, hogy egy linken keresztül kérést küldtünk a /productDelete végpontra. Ennél a pontnál a termék nem törlődik teljes mértékben, csak a táblában található deleted\_at megkapja az aktuális idő értékét. Ennek következtében ezek a termékek nem lesznek listázva az elkövetkező időben.

### 3.6.2 Blog CRUD

A mai világban már szinte mindenkinek van saját blogja. A blog egy olyan része az oldalnak, ami a szerző saját gondolatairól, vagy életéről szól. Ezek klasszikus webnaplók, ám ha nem is gyakran, de előfordul, hogy egy-egy személyes blogból idővel valódi pénzkereső vállalkozás nő ki. Célom ezzel a funcióval az, hogy ne csak az látogasson el a weboldalra, aki esetlegesen potenciális vevő lenne, hanem az is akit érdekel ez a téma. A program írása tehát a Blog CRUD-al folytatódott. Logikája a termékek kezelésén alapul, az értékek és elnevezések részben eltérőek. Itt is a tervezési fázissal kezdtem, kialakítottam számára a modellt, azzal egyidőben a kontrollert, és a migrációit is.

Beállítottam a modellben a táblát, és az elsődleges kulcsot, valamint a képfeltöltési paramétereket állítottam be. A termékekkel szemben itt három méretben tárolok képet, mivel itt lesz egy nagy felbontású kép, ami a blog saját oldalán nagy méretben fog megjelenni.

```
public function registerMediaConversions(Media $media = null): void
{
    $this->addMediaConversion( name: 'thumb')
        ->width( width: 100);
    $this->addMediaConversion( name: 'blogs')
        ->crop( cropMethod: Manipulations::CROP_CENTER, width: 500, height: 500);
    $this->addMediaConversion( name: 'blog')
        ->crop( cropMethod: Manipulations::CROP_CENTER, width: 1290, height: 500);
}
```

39. ábra Blog modell képeinek konvenciói

Jól kivehető, hogy a thumb mérete megegyezik a product thumb méretével, mivel igyekeztem a felhasználó élményt egységesíteni, viszont azért hogy különböző méretű és képarányú képek esetében a kép minőség ne romoljon ugyan abban a dobozban, ezért átméretezés helyett az

eredeti képből akkora képet vágok ki középtájolással, ami pontosan megegyezik a doboz méretével. Miután ez a konfiguráció befejeződött, neki láttam a CRUD felépítésének. Elkészítettem a végpontokat a termékek alapján, majd elkészítettem hozzá a függvényeket.

- (create) Hogy létrehozassunk egy új rekordot a táblánkon, szükségünk lesz egy űrlapra, amit ki kell töltenünk. Ahhoz hogy ezt véghez vigyük „get” kérést kell intéznünk a /blogAdd végpontra, ahol a blogAdd függvény semmi mást nem tesz, csak visszatér a megjelenítési rétegen azzal az űrlappal, amelynek az action-je a blogSave. Ha minden érték megfelelően van megadva akkor poston keresztül küldjük az adatokat tovább a blogSave végpontra.

```
<form action="{{route('blogSave')}}" method="POST">
```

40. ábra Új blog űrlapjának végpont hívása

A blogSave metódus elmenti változókba a beviteli mezők értékeit, majd ha nincs hiba, a beépített save függvénnyel lementi azokat az adatbázisba.

```
public function blogSave(Request $request, Blog $blog){
    $name = $request->input( key: 'name', default: '');
    $title = $request->input( key: 'subtitle');
    $subtitle = $request->input( key: 'title');
    $description = $request->input( key: 'description');
    $content=$request->input( key: 'content');
    $status = $request->input( key: 'status') ? 1 : 0;
    if ($request->hasFile( key: 'pictureLink') && $request->file( key: 'pictureLink')->isValid()) {
        $blog->addMedia($request->file( key: 'pictureLink'))->toMediaCollection( collectionName: 'image');
    }
    $blog->name = $name;
    $blog->title = $title;
    $blog->subtitle = $subtitle;
    $blog->description = $description;
    $blog->content=$content;
    $blog->status = $status;
    $blog->save();
    return redirect( to: '/blogs');
}
```

41. ábra Új blog mentésének metódusa

- (read) Az eljárás itt is hasonló. /blogs végpontra kérést intézve a blogList metódus kerül meghívásra.

```
Route::get( uri: '/blogs',[BlogController::class,'blogsList']);
```

42. ábra Blogok listázásának végpontja

Ekkor minden blog táblában lévő adat le lesz kérve, majd a BlogController továbbítja ezeket a blogs.blade.php megjelenítő rétegnek egy compact nevű metódussal. Ez a függvény létrehoz egy vagy több tömböt a változókból, és annak értékeiből.

Majd az átküldött tömbök adatai megjelennek a listázó felületen.

```
public function blogsList(){
    $blogs=Blog::all();
    return view( view: 'blog.blogs',compact( var_name: 'blogs'));
}
```

43. ábra Összes blog lekérése, majd továbbítása megjelenítőrétegnek

- (update) Blog frissítésekor a /blogEdit vár kérést, hogy le tudja futtatni az ott található blogEdit metódust, ahol id alapján lekéri a táblából a blog bejegyzés adatait ,majd továbbítja ezeket az adatokat az edit\_blog.blade.php megjelenítési rétegnek. Ezek az adatok betöltésre kerülnek a kiíratott űrlap érték mezőibe, ahol tudjuk azokat módosítani, majd a /blogUpdate paranccsal lefut az update metódus és megtörténik az adatok frissítése, ezt követően szintén visszairányít minket a listázó rétegre.

```
public function blogUpdate(Request $request) {
    $id = $request->id;
    $name = $request->input( key: 'name');
    $title = $request->input( key: 'title');
    $subtitle = $request->input( key: 'subtitle');
    $description = $request->input( key: 'description');
    $content = $request->input( key: 'content');
    $status = $request->input( key: 'status') ? 1 : 0;
    $product = Blog::where('id', '=', $id)->get()->first();
    Blog::where('id', '=', $id)
        ->update([
            'name' => $name,
            'title' => $title,
            'subtitle' => $subtitle,
            'description' => $description,
            'content' => $content,
            'status' => $status,
        ]);
    if ($request->hasFile( key: 'pictureLink') && $request->file( key: 'pictureLink')->isValid()) {
        $product->clearMediaCollection('image');
        $product->addMedia($request->file( key: 'pictureLink'))->toMediaCollection('image');
    }
    return redirect( to: '/blogs');
}
```

44. ábra Blog bejegyzés frissítésének metódusa

- (delete) A delete metódus eléréséhez kérést kell intéznünk a /blogDeleteById végpontra id alapján, ami visszatér a blog adataival. Ha biztosak vagyunk a dolgunkban, ezen oldalon állva blogDelete végpont segítségével töröljük az elemet.

```
public function blogDeleteById($id) {
    $blog = Blog::where('id', '=', $id)->first();
    return view('blog.delete_blog', compact('var_name' => 'blog'));
}

public function blogDelete(Blog $blog) {
    $blog->delete();
    return redirect('to: '/blogs');
}
```

45. ábra Blog bejegyzés törlésének metódusai

- Létrehoztam egy extra végpontot, azzal a szándékkal, hogy a publikus felületen bővebb információval szolgálhassunk a bejegyzésről. Mivel az előző végpontok middleware autentikációs csoportjába tartoznak, azért hogy ezeket a funkciókat, megjelenítési rétegeket csak az használhassa, láthassa aki be van jelentkezve, ezzel ellentétben ezt a megjelenítési réteget a nagy közönségnek szánjuk, ezért ezt a végpontot nem védjük. Minden olyan más végpontot is, amit szabadon megtekinthetővé szeretnénk tenni a csoporton kívül helyeztem el. Ezt a végpontot a /showBlog alapján érhetjük el ahol a showBlogByIdOnPublic metódus fut le.

```
Route::get('url: 'showBlog/{blog}', [BlogController::class, 'showBlogByIdOnPublic'])->name('name: 'showBlog');
```

46. ábra Publikus blogbejegyzés végpontja

A függvény futásakor nem történik semmi más csak blog model alapján le van kérve a hozzá tartozó összes adat, és az át van küldve a megjelenítési rétegnek, ahol igényes formában az olvasó elé tárul.

```
public function showBlogByIdOnPublic(Blog $blog) {
    $blog::all();
    return view('blog.show_blog_on_public', compact('var_name' => 'blog'));
}
```

47. ábra Blogbejegyzés lekérése, majd publikus felületre küldése

### 3.6.3 Slider CRUD

A slider egy design elem a weboldal publikus felületén. A sliderek lényege, hogy a honlap egy és ugyanazon felületén megváltozik a szöveg és a kép. Azért került ez a funkció az oldalra, hogy egyetlen helyen egyszerre több információt is el tudjunk helyezni, továbbá izgalmasabbá, változatosabbá tegyük a weboldalt. Hogy ennek a CRUD-nak minden funkcióját használni tudja a felhasználó, minimum admin jogosultsággal kell rendelkeznie, különben a 403-as hibakód jelenik meg.

- (create) Ha a /sliderAdd végpontra bonyolítunk kérést, akkor a SliderController a sliderAdd metódus segítségével kiírat elénk egy újabb űrlapot az új képváltó felvételéhez szükséges beviteli mezőkkel.

```
Route::get('uri: '/sliderAdd',[SliderController::class,'sliderAdd']);
```

48. ábra Új képváltó létrehozásának végpontja

```
public function sliderAdd() {
    if (! Gate::allows(ability: 'superadmin' ) || Gate::allows(ability: 'admin' )) {
        abort( code: 403);
    }
    return view( view: 'slider/new_slider', [
    ]);
}
```

49. ábra Új képváltó létrehozásának metódusa jogosultság kezeléssel

Az űrlap elküldésével a /sliderSave végponton intézünk kérést, amely során a kitöltött űrlap Slider osztályt példányosítva, majd azon egy create függvényt futtatva elkészíti nekünk az új slidert, és lementi adatbázis táblába az értékeit, majd átirányít minket a listázó oldalra.

```
public function sliderSave(Request $request ) {
    if (! Gate::allows(ability: 'superadmin' ) || Gate::allows(ability: 'admin' )) {
        abort( code: 403);
    }
    $slider = Slider::create($request->all());
    if ($request->hasFile( key: 'pictureLink' ) && $request->file( key: 'pictureLink' )->isValid() ) {
        $slider->addMedia($request->file( key: 'pictureLink' ))->withResponsiveImages()->toMediaCollection('image');
    }
    return redirect( to: '/sliders');
}
```

50. ábra Új képváltó mentésének metódusa jogosultság kezeléssel

- (read) A slider listázását is ugyan olyan módon érhetjük el, kérést kell intézni a /sliders végpontra ahol a slidersList függvény lefut, lekéri az összes slider táblában lévő elemet Slider osztály segítségével, majd továbbítja ezeket a megjelenítőrétegnek.

```
public function slidersList() {
    $sliders = Slider::all();
    return view( view: 'slider/sliders', compact( var_name: 'sliders'));
}
```

51. ábra Képváltók listázásának metódusa

- (update) Mint minden végpontnál, az sliderEdit végpontot is összekell kötni egy végponti függvény hívással a web.php-ban. A sliderControllerben mutat a sliderEdit függvényre. A sliderEdit lekéri azonosítószám alapján a slidert adatait, majd ezeket az értékekkel visszatér, az űrlap beviteli mező értékeiként.

```

public function sliderEdit($id) {
    if (! Gate::allows( ability: 'superadmin' ) || Gate::allows( ability: 'admin' )) {
        abort( code: 403);
    }

    $slider = Slider::where('id', '=', $id)->first();
    return view( view: 'slider/edit_slider', compact( var_name: 'slider'));
}

```

52. ábra Képváltó módosításának metódusa jogosultság kezeléssel

Az űrlap módosít gombjára kattintva a /sliderUpdate végponthoz érünk, ahol megtörténi az adatok frissítése , ahol a módosított értékek lesznek tárolva az adatbázisban.

- (delete) Törlés itt is pontosan ugyan úgy működik, mint az előző kettő esetében, /sliderDeleteById végponton lefut a sliderDeleteById függvény, ami id alapján lekérdez adatbázisból, majd /silderDelete végpont meghívja a sliderDelete függvényt a törölni kívánt elemre, amin természetesen egy delete metódus állítja a deleted\_at mezőt az éppen aktuális dátumra.

```

Route::get( uri: '/sliderDeleteById/{id}', [SliderController::class,'sliderDeleteById']->name( name: 'sliderDeleteById'));
Route::get( uri: '/sliderDelete/{slider}',[SliderController::class,'sliderDelete']->name( name: 'sliderDelete'));

```

54. ábra Képváltó törlésének get-es és post-os végpontjai

```

public function sliderDeleteById($id) {
    if (! Gate::allows( ability: 'superadmin' ) || Gate::allows( ability: 'admin' )) {
        abort( code: 403);
    }

    $slider = Slider::where('id', '=', $id)->first();
    return view( view: 'slider/delete_slider', compact( var_name: 'slider'));
}

public function sliderDelete(Slider $slider) {
    if (! Gate::allows( ability: 'superadmin' ) || Gate::allows( ability: 'admin' )) {
        abort( code: 403);
    }

    $slider->delete();
    return redirect( to: '/sliders');
}

```

53. ábra Képváltó törlésének metódusai

### 3.6.4 Publikus felületen történő megjelenítés.

Mivel már minden olyan tartalom, amit a fogyasztói társadalom felé szeretnénk megjeleníteni, elérhető az adatbázisban, és ezeknek kezelése hiba mentes, elkezdtem kialakítani nyilvános oldalt. A nyilvános oldalon fog megjelenni azon adatok halmaza, amellyel a weboldalra látogatót szeretnénk informálni vagy tájékoztatni. Ennek a megvalósítására is az MVC modellt használom. Ennek a működtetéséhez külön osztályra nincsen szükség, mivel meglévő osztályok adatait szolgálom ott ki. Controllerre viszont szükség lesz, valamint végpontra is a controllerben lévő metódusok eléréséhez. Web.php-n található végpont meghívása és összeköttetése szokásos módszerrel megy végbe, annyi különbséggel, hogy ez a végpont az alapértelmezett belépési pontnak minősül a weboldalon.

```
Route::get( uri: '/', [IndexController::class, 'index'] );
```

55. ábra Főoldal végpontja

Ez azt jelenti, hogy az oldal bármilyen végpontra való hivatkozás nélküli megnyitása, ezt az oldalt fogja meghívni, tehát ez lesz az index. Az index végpont végpontos hívása egy index metódust hív meg, amelyet az IndexController kezel. Ebben a függvényben történik meg az adatbázisból való kiolvasása mindazon elemeknek, amelyeket szeretnénk tovább küldeni az index felületére tömb formájában.

```
class IndexController extends Controller {  
    public function index() {  
        $sliders = Slider::all();  
        $products = Product::all();  
        $blogs = Blog::all();  
        $companyData = CompanyData::all();  
        return view( view: 'home', compact( var_name: 'sliders', ...var_names: 'products', 'blogs', 'companyData' ) )  
    }  
}
```

56. ábra Főoldalra történő adat továbbítás

A megjelenítési rétegen ezeket a tömböket foreach ciklussal bejárva íratom ki a látogató számára a home.blade.php parciális oldalon.

### 3.6.5 Vezérlőpult felület kialakítása

A vezérlőpult kialakítását hagytam a legvégére. Mivel idő szűkében voltam, ezért csak egy egyszerű tájékoztató jellegű oldalt hoztam létre. Létrehoztam számára is egy vezérlőt DashboardController néven, ahol a már adatbázisban lévő adatok értékeit lekérem,



belehelyezem azokat tömbökbe, és átküldöm a megjelenítési rétegnek.

```
public function index(){
    $sliders =Slider::all();
    $products=Product::all();
    $blogs=Blog::all();
    $companyDatas=CompanyData::all();
    return view( view: 'dashboard',compact( var_name: 'sliders', ...var_names: 'products','blogs','companyDatas'));
}
```

57. ábra Vezérlőpulton megjelenő adatok metódusa

A /dashboard végponttal kötöttem össze, amely kérésénél az index függvény fut le.

```
Route::get( uri: '/dashboard',[DashboardController::class,'index']);
```

58. ábra Vezérlőpult végpontja

Ezzel a tartalomkezelő rendszer kialakításának végére értem. Továbbiakban szeretném ezeket plusz funkciókkal bővíteni, például dolgozói nyilvántartással, valamint annak kategorizálásával, jelenléti kezelő rendszerrel stb. A dashboard kialakítását is szeretném bővíteni, hogy a felhasználó lássa, hogy melyik a legnézettebb blog bejegyzése, vagy a webshop megvalósítását követően egyből jelenjenek meg az ezzel kapcsolatos információk. A meglévő listázó oldalakra szeretnék szűrést, rendezést kialakítani, sajnos ezek idő hiányában nem valósultak egyelőre meg.

## 4. Felhasználói dokumentáció

Ebben a fejezetben bemutatom a felhasználói útmutatóját a webes applikációnak. Az alkalmazás webes felületen elérhető, ezért előzetesen semmilyen program telepítését nem igényli.

### 4.1 Webes felületek bemutatása

A weboldal neve Didus handmade. Ez a weboldal tartalom megosztás céljából készült. A weboldalt kettő nagyobb részre lehetne felosztani. Az oldal teljes mértékben kijelző barát (reszponzív) minden kijelző méreten jól olvasható. Az első része a publikus felület lenne, amelyre a felhasználó, vagy látogató ellátogat, ott találja azt a tartalmat, amelyet majd az adminisztrációs felületen fogunk kezelni. Ez a tartalom dinamikusan változik, a felhasználó az adminisztrációs felületen tudja megszabni, hogy mi jelenjen meg a publikus felületen.

#### 4.1.1 Publikus felület bemutatása

Ha a felhasználó megnyitja a weboldalt, a publikus oldal fog megjelenni. A publikus felület 5 nagyobb részre bontható.

1. Navigációs menü
2. Képváltó
3. Termékek
4. Blog bejegyzések
5. Cég adatok

1) Navigációs menü: A felső fekete háttérű navigációs mező kettő elemet tartalmaz. Az egyik a főoldali linket tartalmazza, a másik linkre kattintva, a felhasználót a bejelentkező felületre navigálja, ahol hitelesítés után be tud lépni az adminisztrációs felületre. Ha sikeres az azonosítás ebben a sorban fognak megjelenni az adminisztrációs menüpontok is. A felső navigációs mező minden aloldalról elérhető, és bármely pontján halad a felhasználó az mindvégig látható marad.



59. ábra Vendégként megjelenő menü sáv

2) Képváltó: A második elem a weboldalon a „slider” magyarul képváltó. Célja, hogy egyetlen helyen egyszerre több információt is el tudjunk helyezni, továbbá izgalmasabbá, változatosabbá tegyük a weboldalt. A képváltónak 3 része van, a nagyobb vastagon szedett szöveg az első része a cím, második része a cím alatt megjelenő hosszabb szöveg a leírás, és a harmadik része pedig az egyedi háttérkép. A képek és leírásai automatikusan váltják egymást, valamint a kép jobb és baloldalán vízszintesen középen elhelyezkedik kettő gomb amivel manuálisan is lehet a képeket váltani.



60. ábra Slider megjelenése adatokkal

3) Termékek: A termékek rész, ahol az elkészült termékek képpel, névvel, lenyíló leírással jelennek meg, valamint a mérete, kategóriája és darabszáma is fel van tüntetve az egyes kártyákon, amelyek 3 oszlopban helyezkednek el egymás mellett. A termék neve alatt található egy link, melyre kattintva az adott termék leírása jelenik meg.



61. ábra Termékek megjelenése adatokkal

4) Blogok, bejegyzések: Ebben a részben két oszlopos elrendezésben a blogbejegyzések jelennek meg a baloldali oszlopban egy vastagabb cím, és egy halványabb alcím, valamint egy rövid szöveges leírás, valamint a link a bejegyzés saját oldalára. A jobb oldalán a bejegyzéshez tartozik egy kép is. A bejegyzés saját oldalán ezek az elemek egymás alatt egy sorban helyezkednek el, és a bejegyzés bal alsó sarkában egy főoldalra mutató link van.

## ~ Blog bejegyzéseim ~

Lorem ipsum dolor sit amet.  
Lorem ipsum.

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Aliquam amet asperiores aut dolores eligendi eos maiores nihil soluta ullam vero!

[Kattints a bővebb leírásért!](#)



62. ábra Blog bejegyzés főoldali megjelenése

Lorem ipsum dolor sit amet.



Lorem ipsum.

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Animi autem dignissimos distinctio ea, error ex facilis fugiat hic iure iusto labore maiores maxime, minima necessitatibus nesciunt numquam optio perspiciatis placeat provident quis repellendus repudiandae sapiente sit tempora temporibus! Consequatur cumque eius necessitatibus neque, odit officiis placeat praesentium quos ratione tenetur.

[Vissza a főoldalra](#)

63. ábra Blog megjelenése saját oldalán

- 5) Cég adatok: Ezt az egységet az oldal láblécének is nevezhetnénk ( footer ), ahol a céggel kapcsolatos adatok jelennek meg, kapcsolat teremtés céljából. Tartalmazza a cég nevét, cég székhelyét, elérhetőségeit, valamint az oldal tetejére navigáló gombot.

64. ábra Footer megjelenése



#### 4.1.2 Adminisztrációs felület bemutatása

Az adminisztrációs felületet, csak hitelesítéssel bejelentkezett felhasználó éri el. Regisztrációra nincs lehetőség egyenlőre, kizárólag az oldal üzemeltetője képes új felhasználót felvinni. A bejelentkező felületet a publikus felületről érjük el a felső navigációs menü jobb oldali gombja az.

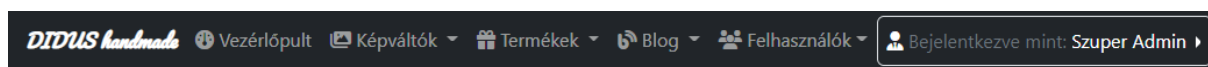


65. ábra Bejelentkezés gomb elhelyezkedése menüsávban

A bejelentkezés gombra kattintva az oldal a bejelentkezési felületre navigál minket, ahol a felhasználó a saját egyedi email címének és jelszavának megadásával, képes belépni az adminisztrációs felületre. Ha a bejelentkező fél hibásan adja meg email címét vagy nem tölti ki valamelyik mezőt, akkor az oldal jelezni fogja ezt neki. Ha helytelen az email cím vagy jelszó, a bejelentkezési felület erről is informál minket.

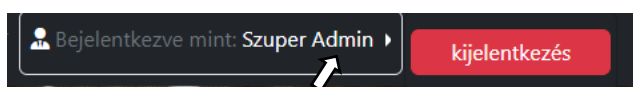
66. ábra Bejelentkezési űrlap

Amennyiben sikeres a beléptetés, a beléptető rendszer a vezérlőpult ( dashboard ) oldalára navigál minket, ahol találhatóak az általános információk. Ezzel egyidőben a felső navigációs sávban megjelentek új menüpontok.



67. ábra Adminisztrációs oldal menü elemeinek elhelyezkedése

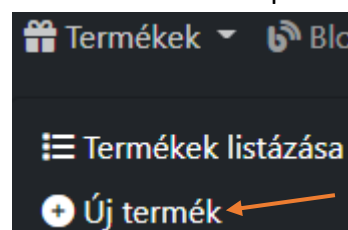
Ezekkel a menüpontokkal különböző funkciókat kezelhetünk. A weboldal nevére kattintva eljutunk a főoldalra, a bejelentkezés megszűnése nélkül, így könnyebb ellenőrizni milyen tartalmat adtunk hozzá, módosítottunk, vagy töröltünk a publikus felülethez. A második a vezérlőpult, ahová a bejelentkezést követően jutottunk el. A harmadik egy lenyíló gomb, ahol a képváltókat tudja a felhasználó kezelni. Annak felső menü pontja a listázás, az alsó menü pontja az új képváltó felvételének oldalára navigál, ahhoz hogy ezt a menüpontot elérjük, minimum admin ranggal kell rendelkezni a felhasználónak. Negyedik menü eleme a termékek kezelése, a lenyíló gombok közül a listázást vagy új terméket felvitelét választhatjuk. Ötödik lenyíló menü elem a blog bejegyzések listázását, vagy új blogbejegyzés készítését teszi nekünk elérhetővé. A felhasználók menüpont következik ahol 3 lenyíló elem kapott helyet. Az első a felhasználók listázásának navigációs gombja, a második menüpontot csak superadmin jogosultsággal rendelkező felhasználó látja, és éri el, a harmadik egy tájékoztató jellegű oldalra navigál, ahol a jogosultságok bővebb kifejtését találja a használó. A navigációs sáv legutolsó eleme tájékoztatja a felhasználót milyen néven van bejelentkezve, a nevére kattintva kijelentkezhetsz a rendszerből, de a rendszer 30 perc után automatikusan kilépteti, ha inaktív a felhasználó.



68. ábra Bejelentkezett felhasználó nevének megjelenítése menüsávban

Következőnek a tartalom kezelését szeretném bemutatni. Mivel minden tartalom feltöltését a publikus oldalra ugyan azon az elven kell elvégezni, ezért példának okáért egy termék feltöltésének menetét, annak módosítását majd törlését mutatom be.

Első lépésként meg kell keresnünk a fenti navigációs sávban a termékek menüpontot, majd rákattintva az új termék felvitelére menüpontra, a rendszer átnavigálja a felhasználót az új termék felvitelének üres űrlapjára.



69. ábra Új termék menüpontja

Az űrlap egyszerűen értelmezhető utasításokat ad a felhasználónak. Az első mező a termék nevét határozza meg, a második a termék méretét számmal kell megadni, harmadik elem az űrlapban egy dinamikusan megjelenő, lenyíló menü, mely elemei közül a felhasználónak lehetősége van meghatározni, hogy a terméket melyik termék kategóriába szeretné sorolni. Továbbá megadhatja, hogy az adott termékből hány darab elérhető és készíthet hozzá leírást. Az utolsó mezőben választhatja ki melyik képet szeretné feltölteni a termékhez. Az aktív pipa alapértelmezetten be van pipálva, így a termék státusza a feltöltést követően aktív lesz, tehát a publikus felületre ki fog kerülni. Ha a felhasználó elkészült az adatok megadásával, a jobb alsó sarokban található MENT gombra kattintva véglegesítheti a termék feltöltésének folyamatát.



 **Új termék felvitele** 

Név\*

Adja meg a termék nevét

Méret

Adja meg a termék méretét

Típus\*

Baba csörgő

Mennyiség\*

Adja meg a termék darabszámát

Leírás

Írjon hozzá leírást

Adja meg a fájlt:

Fájl kiválasztása Nincs fájl kiválasztva

☒ Aktív

← Vissza

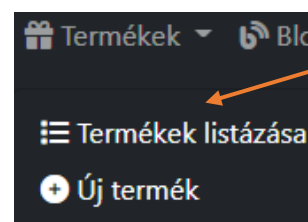
MENT

70. ábra Új termék felvitelének űrlapja

A mentést követően a felhasználót a felület automatikusan a listázó oldalra navigálja, ahol leellenőrizheti a feltöltése eredményét.

A listázó oldalt egyébként a fenti navigációs menü Termékek listázása menüpont alatt bármelyik aloldalról könnyedén eléri.

A listázó oldalon, táblázat formájában jelennek meg azok az adatok, amelyeket a felhasználó előzőleg felvitt a termékekhez. Ha nincs egy listázandó termék sem az adatbázisban akkor a rendszer tájékoztatja erről a felhasználót. A már feltöltött termékek adatai egymás mellett egy sorban helyezkednek el. Az első sorszámmal látja el az egyes sorokat (ezek a számok nem a tényleges azonosítósámok), a második oszlop tartalmazza a termék nevét, harmadik a méretét, negyedik a mennyiségét, ötödik a státusztát (megjelenik e a publikus felületen) hatodik oszlopban a „thumb” méretű, a termékhez feltöltött kép jelenik meg. Utolsó oszlop tartalmazza azokat a műveleteket, amikkel kezelni tudjuk egy termékünket. A sárga színű gombbal a kiválasztott termék adatait tudjuk módosítani, felülírni, a piros gombbal törölhetjük a listából a terméket egy jóváhagyás után.



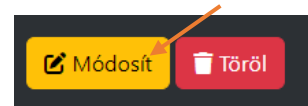
72. ábra Termékek listázásának menüpontja

#	Név	Kategória	Méret	mennyiség	Státusz	Kép	Műveletek
7	Teszt termék	Egyéb	1	5	Inaktív	TESTZ KÉP	<a href="#">Módosít</a> <a href="#">Töröl</a>

71. ábra Termékek listázása adminisztrációs felületen

A teszt termék státusza inaktív, így ez a termék nem jelenik meg majd a publikus felületen.

Ha módosítani szeretne valami adatot a termékén, akkor a sárga módosít gombra kell kattintania a felhasználónak.



73. ábra Műveletek gombjai

A rendszer automatikusan átirányítja arra a felületre, ahol a termék aktuális adatait tudja módosítani egy űrlap segítségével. Az űrlap beviteli mezőiben már a termék adatainak tárolt értékei jelennek meg, amiket bármikor módosíthat.

A zöld MÓDOSÍT gombra kattintva véglegesítheti a változtatásokat. Ha megtörténik az űrlap elküldése, azt követően ismét a listázó oldalon találja magát a felhasználó, ahol leellenőrizheti módosításainak végeredményét.



## Teszt termék nevű termék módosítása

Név\*

Teszt termék

Méret

1

Típus\*

Egyéb ▾

Mennyiség\*

5

Leírás



Teszt leírás

Adja meg a fájlt:

Fájl kiválasztása Nincs fájl kiválasztva

☒ Aktív

← Vissza

 MÓDOSÍT 

74. ábra Termék adatainak módosítása



A felhasználó törölni is képes terméket a listából, ha a műveletek oszlopban a piros Töröl gombra kattint. Ebben a pillanatban még nem törölte a terméket, a rendszer a törlés előtt egy jóváhagyást kér a felhasználótól. Ha ezen a ponton is a piros TÖRÖL gombra kattint, akkor véglegesíti a termék törlését. A törlés elvégzése után a rendszer a felhasználót a listázó oldalra továbbítja.

## Teszt termék törlése?

TESZT KÉP

Valóban törli #19 ID-val rendelkező terméket?

← VISSZA

 TÖRÖL 

75. ábra Termék törlésének megerősítése

## 5. Összefoglalás

Ezen project fejlesztése során rengeteg új ismerettel és tapasztalattal gazdagodtam, ezt a munkát szeretném kamatoztatni. Szeretném ismereteimet még magasabb szintre emelni, hogy bonyolultabb rendszerek megírására is képes legyek. A laravel felépítése elsőre bonyolultnak tűnt, de egyre jobban belemélyedve megértettem a működésének logikáját, ennek eredményeképpen sikerült megvalósítani ezt a projectet. A jövőben szeretném bővíteni új tartalmakkal, kiegészítőkkel ezt a tartalom megosztó rendszert, majd webshoppá bővíteni, ha az érdekeltségi kör ezt engedi. Köszönöm a Ruander oktatási központnak, hogy lehetővé tették hogy részt tudjak venni ezen az oktatáson, és megszerezsem ezt a tudást, valamint szeretnék köszönetet mondani oktatóimnak is Nagy Ferencnek, és Blahut Lórántnak, hogy átadták ezt a tudást. Továbbá szeretném megköszönni családomnak és barátaimnak, hogy támogattak, és türelmesek voltak velem a képzés ideje alatt.

## Ábrajegyzék

1. ábra MVC Model.....	8
2. ábra Adatbázis diagramm.....	11
3. ábra .evn-ben adatbázis kapcsolódás.....	13
4. ábra Model készítése artsan segítségével.....	13
5. ábra Tábla és elsődleges kulcs definiálása .....	13
6. ábra Modell mappa struktúrája.....	13
7. ábra tábla kapcsolat belongsTo segítségével .....	14
8. ábra View réteg mappaszerkezete .....	14
9. ábra Controller réteg mappaszerkezete.....	15
10. ábra Felhasználó végpontok.....	15
11. ábra Felhasználó listázása endpoint.....	15
12. ábra userList metódus .....	16
13. ábra Listázó megjelenítő réteg.....	16
14. ábra Új felhasználó mentése végpontok .....	17
15. ábra Új felhasználó mentése függvények.....	17
16. ábra Védett végpontok csoportba foglalása .....	18
17. ábra Bejelentkezési felület végpontja .....	18
18. ábra Bejelentkezési és kiléptetési metódusok .....	18
19. ábra Kijelentkezés végpontja.....	19
20. ábra Jogosultsággal kezelt menüpont .....	19
21. ábra Jogosultságok definiálása AuthserviceProvider-ben.....	20
22. ábra Jogosultsággal védett metódusok.....	20
23. ábra Kép gyűjteménybe regisztrálása .....	21
24. ábra Kép konverziók megadása .....	22
25. ábra Új termék létrehozásának végpontja .....	22
26. ábra Új termék hozzáadása függvény .....	22
27. ábra Dinamikusan megjelenő legördülő menü .....	23
28. ábra Termék mentésének action-je .....	23
29. ábra Termékmentésének végpontja .....	23
30. ábra Új termék mentésének eljárása .....	23
31. ábra Termék listázásának végpontja .....	24
32. ábra Termékek listázásának metódusa .....	24
33. ábra 0 darab termék esetén a tájékoztatás.....	24
34. ábra Termék módosításának get-es végpontja .....	24
35. ábra Termék módosításának metódusa .....	25
36. ábra Rejtett id értékkel rendelkező mező .....	25
37. ábra Termék módosításának post-os végpontja .....	25
38. ábra Módosított termék adatainak betétele adatbázisba .....	25
39. ábra Blog modell képeinek konvenciói.....	26
40. ábra Új blog űrlapjának végpont hívása .....	27
41. ábra Új blog mentésének metódusa .....	27
42. ábra Blogok listázásának végpontja .....	27
43. ábra Összes blog lekérése, majd továbbítása megjelenítőrétegnek.....	28
44. ábra Blog bejegyzés frissítésének metódusa.....	28
45. ábra Blog bejegyzés törlésének metódusai.....	29
46. ábra Publikus blogbejegyzés végpontja .....	29

47. ábra Blogbejegyzés lekérése, majd publikus felületre küldése.....	29
48. ábra Új képváltó létrehozásának végpontja .....	30
49. ábra Új képváltó létrehozásának metódusa jogosultság kezeléssel .....	30
50. ábra Új képváltó mentésének metódusa jogosultság kezeléssel.....	30
51. ábra Képváltók listázásának metódusa .....	30
52. ábra Képváltó módosításának metódusa jogosultság kezeléssel.....	31
53. ábra Képváltó törlésének metódusai .....	31
54. ábra Képváltó törlésének get-es és post-os végpontjai .....	31
55. ábra Főoldal végpontja.....	32
56. ábra Főoldalra törénő adat továbbítás.....	32
57. ábra Vezérlőpulton megjelenő adatok metódusa.....	33
58. ábra Vezérlőpult végpontja .....	33
59. ábra Vendégként megjelenő menü sáv.....	34
60. ábra Slider megjelenése adatokkal .....	35
61. ábra Termékek megjelenése adatokkal.....	35
62. ábra Blog bejegyzés főoldali megjelenése .....	36
63. ábra Blog megjelenése saját oldalán.....	36
64. ábra Footer megjelenése.....	36
65. ábra Bejelentkezés gomb elhelyezkedése menüsávban .....	37
66. ábra Bejelentkezési űrlap .....	37
67. ábra Adminisztrációs oldal menü elemeinek elhelyezkedése .....	38
68. ábra Bejelentkezett felhasználó nevének megjelenítése menüsávban .....	38
69. ábra Új termék menüpontja.....	38
70. ábra Új termék felvitelének űrlapja.....	39
71. ábra Termékek listázása adminisztrációs felületen.....	40
72. ábra Termékek listázásának menüpontja.....	40
73. ábra Műveletek gombjai .....	40
74. ábra Termék adatainak módosítása .....	41
75. ábra Termék törlésének megerősítése.....	41

## Források

<https://matebalazs.hu/html.html>

<https://lexiq.hu/programozasi-nyelv>

<https://psprog.hu/article/rest-alapu-kommunikacio>

<https://composite-solutions.eu/blog/az-mvc-minta-laravel>

<https://hu.wikipedia.org/wiki/JavaScript>

<https://laravel.com/docs/9.x>

<https://spatie.be/docs/laravel-medialibrary/v8/introduction>

<https://getbootstrap.com/docs/5.2/getting-started/introduction/>

<https://fontawesome.com/>

<https://api.jquery.com/>