- **Ruander Okatatási Kft**

- SZAKDOLGOZAT

Lakatos Gergely

webfejlesztő tanfolyam

Kiskunhalas, 2023

Tartalom

1. Bevezetés

Dolgozatom témáját a saját web shop elkészítése és fejlesztése ihlette, későbbiekben szeretnénk egy családi kereteken belül majd egy baba-mama online áruházat, saját magunk lefejleszteni, jelenleg az oldalon csak fiktív termékek láthatóak és a küllem is változni fog nagyon sok elképzelésem van az alkalmazással folyamatban, de az majd csak későbbiekben valósítjuk meg.

1.1 A dolgozat célkitűzése

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: Esetleges baba vagy kis mama termékek bemutatása vagy avval kapcsolatos segítség nyújtás.

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 weboldalakat leíró jelnyelv, vagy az a kódrendszer, amit a weboldalak elkészítéséhez használnak. A HTML nem programozási nyelv, hanem kódnyelv, aminek segítségével akár a Jegyzettömb 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 Stlye 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 HMTL, 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 Langueage – 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 adatra (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.

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 authorizáció 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 nyilt 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 hibamegelőzési lehetősséggel. 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.

3.1. Laravel sablon kialakítása és db beállítása

A Laravel alapértelmezés szerint a MySQL adatbázist használja. Be kell állítania a kívánt adatbázist a .env fájlban, majd futtatnia kell migrációkat és adatbázisokat kell szeedelnie az alkalmazás inicializálásához.

bash Copy code

```
php artisan migrate
php artisan db:seed
```

A beépített Laravel fejlesztői szerverét a következő parancs segítségével futtathatja:

bash Copy code

```
php artisan serve
```

Ezután az alkalmazás a http://localhost:8000 címen lesz elérhető.

3.2 Laravel breeze

A regisztrációhoz a Laravel Breeze-t használtam, ami egy előre elkészített eszköz a Laravel Breeze segít az alapvető regisztráció- és bejelentkezési funkciók gyors beállításában egy Laravel projektben.

Felhasználói profilok: A Breeze-ben beépített lehetőség van felhasználói profilok kezelésére is. Testre szabható és hozzáadható további mező vagy mezők a felhasználói profilhoz.

3.3 Laratrust

Következkezőnek a Laratrust nevű csomagot telepítettem, ami beállíthatja az alkalmazás jogosultságkezelését a Laratrust segítségével.

Szerepek és jogosultságok definiálása: Most már definiálhatja a szerepeket és jogosultságokat az alkalmazásában a konfigurációs fájlokban, például a config/laratrust/roles.php és config/laratrust/permissions.php fájlokban.

Felhasználó modell beállítása: Nyissa meg a config/laratrust.php konfigurációs fájlt, és állítsa be a user_models értékét a saját User modelljére. Alapértelmezés szerint a User modell lesz a felhasználói modell, de ha más nevet használ itt kell beállítania.

Ugyan itt van mód definiálni az Admin szerepet is. Ezt a config/laratrust/permissions.php konfigurációs fájlban teheti meg.

Például, ha az "admin" szerepnek teljes hozzáférést szeretne adni az összes rendszerfunkcióhoz, akkor hozzárendelheti a megfelelő jogosultságokat ehhez a szerephez.

3.4 View réteg kialakítása:

Ezt követően az Admin Dashboard routing-ot készítettem el.

```
Route::middleware(['auth','role:admin'])->group(function() {
    Route::controller(DashboardController::class)->group(function(){
        Route::get('/admin/dashboard','Index')->name('admin_dashboard');
```

3.5 Admin Template kialakítása

Admin sablon (vagy admin panel) kialakítása során az a cél, hogy egy könnyen használható és funkcionális felhasználói felületet hozzak létre, amely lehetővé teszi a webalkalmazás vagy a rendszer adminisztrációját és kezelését.

Az admin Templete-hez egy előre elkészitett opciót töltöttem le és bontottam szét funkciók és igény szerint.

Az admin panel letöltésének forrása: https://themewagon.com/themes/sneat-vuejs/

3.6 Végpontok

A "végpont" a Laravel keretrendszerben a HTTP-kérések kezelésére és a válaszok küldésére szolgáló elérési útvonalat jelenti.

A végpontok meghatározzák, hogy milyen választ kapjon egy kliens, amikor egy adott URL-t kérdez meg a szerverről. Ezek definiálása a Laravelben általában a routes/web.php vagy routes/api.php fájlban történik, attól függően, hogy a végpontok webalkalmazáshoz vagy webes API-hoz tartoznak-e.

```php
/******Felhasználói endpointok *********/
Route::middleware(['auth', 'role:user'])->group(function()
{
    Route::controller(ClientController::class)->group(function()
    {
        Route::get('/add_to_cart','Add_To_Cart')->name('add_to_cart');
        Route::post('/add_product_to_cart','Add_To_Product_Cart')->name('add_to_product_cart');
        Route::get('/checkout','Checkout')->name('checkout');
        Route::get('/shippinng_address','Shipping_Adress')->name('shipping_address');
        Route::post('/add_shipping_address','Add_Shipping_Address')->name('add_shipping_address');
        Route::post('/place_order','Place_Order')->name('palce_order');
        Route::get('/user_profile','User_Profile')->name('user_profile');
        Route::get('/user_profile/pending_orders','Pending_Orders')->name('pending_orders');
        Route::get('/user_profile/history','History')->name('history');
        Route::get('/new_relese','New_Relese')->name('new_relese');
        Route::get('/todays_deal','Todays_Deal')->name('todays_deal');
        Route::get('/custom_service','Customer_Service')->name('custom_service');
        Route::get('/remove_cart_item{id}','Remove_Cart_Item')->name('remove_cart_item');
    });
});
Route::get('dashboard', function () {
    return view('dashboard');
})->middleware(['auth', 'role:user'])->name('dashboard');
```

Ha „get” kérést intéz akkor az alkalmazás tudni fogja, hogy akkor szeretnék kiolvasni arról a bizonyos dologról, amit ezen az endponton 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.

3.7 Űrlap kialakítása

Kategória oldalbeállítás hozzáadása űrlappal

Próbáltam az egyszerűségre törekedni a kialakítás során, így Input elemeket alakítottam ki az oldalon amik ('hidden/elrejtett')mezők tartalmazzák a termék azonosítóját, árát és

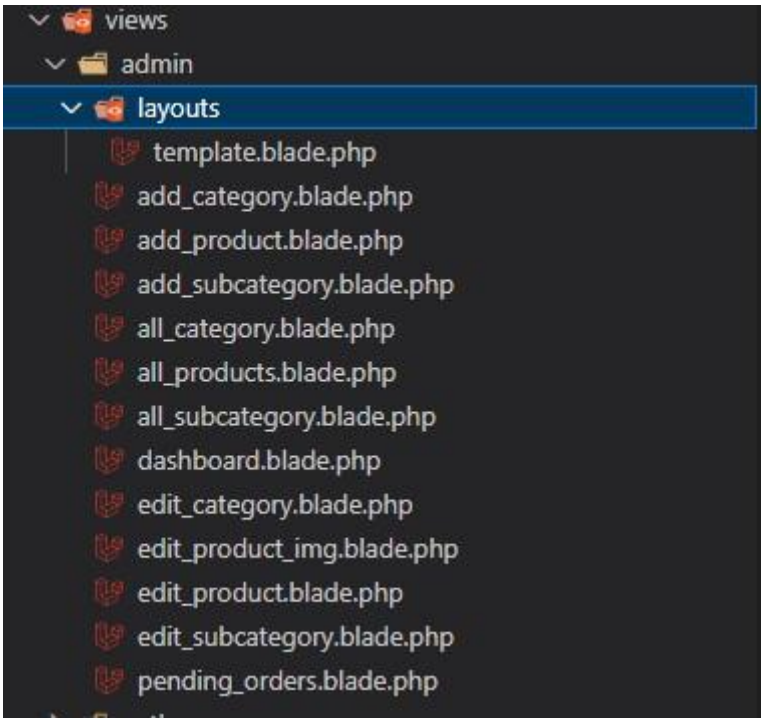
mennyiségét. Ezek az adatok az űrlap elküldésekor kerülnek továbbításra a szerver felé. A $product változókból veszi az adatokat, amelyeknek korábban értéket adtam.

A "Buy Now/Vásárlás" gomb: Ez egy űrlap elküldésére szolgáló gomb, amely a vásárlást indítja. Az űrlap elküldésekor a megadott adatokat a szerver felé küldi, ahol a megfelelő műveletekkel továbbítja a kosárba az adott terméket.

@csrf: Ez a Blade direktíva a Cross-Site Request Forgery (CSRF) elleni védelemhez szükséges token generálására szolgál. Fontos, hogy minden űrlapon alkalmazd ezt a védelmet a biztonság érdekében.

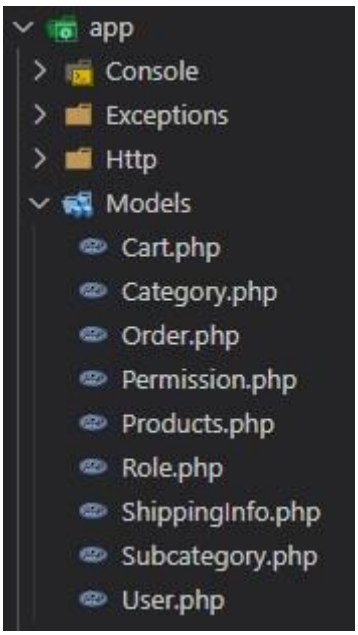
```
<form action="{{ route('add_to_product_cart') }}" method="POST">
    @csrf
    <input type="hidden" value="{{ $product->id }}" name="product_id">
    <input type="hidden" value="{{ $product->price }}" name="price">
    <input type="hidden" value="1" name="quantity">
    <br>
    <input class="btn btn-warning" type="submit" value="Buy Now">
</form>
```

A következőben kialakításra kerül az Add category, all category page, add sub category page, all sub category page ,add product , all product page, pending order kialakítása.



3.8 Model és migráció

Az alkalmazás által kezelt információk tartomány-specifikus ábrázolása. A „modell” réteg az app mappán belüli Models mappában található.



Category tábla kialakítása parancssor segítségével:

```
php artisan make:model Category -m
```

A Laravel Migrations rendszerének használatával definiálja a categories táblát, és amikor futtatják a migrációt, létrehozza ezt a táblát az adatbázisban. A Laravel Migrations lehetővé teszi az adatbázisséma könnyű és verziókezelhető kezelését a fejlesztés során.

```php
public function up(): void
{
    Schema::create('categories', function (Blueprint $table) {
        $table->id();
        $table->string('category_name');
        $table->string('slug');
        $table->integer('subcategory_count')->default(0);
        $table->integer('product_count')->default(0);
        $table->timestamps();
    });
}
```

A category nevű tábla a következő paraméterekkel rendelkezik:

id:Ez egy azonosító mező, ami alapértelmezetten az adott rekord egyedi azonosítóját tárolja.

category_name: Ez a kategória nevét tároló mező, ahol a kategória nevek karakterláncokként kerülnek elmentésre.

slug:Ez egy másik karakterlánc mező, amely a kategória nevét URL-barát formában tárolja. Általában szóközök helyett kötőjeleket vagy alsó vonalakat tartalmaz.

subcategory_count: Egy egész szám mező, ami az adott kategória alá tartozó alkategóriák számát tartja nyilván. Alapértelmezett értéke 0.

product_count:Egy egész szám mező, ami az adott kategóriához tartozó termékek számát tárolja. Alapértelmezett értéke 0.

timestamps: Ez a mező az adatok létrehozásának és frissítésének dátumát és időpontját rögzíti.

A következő parancs kiadásával megkapom az adatbázisban a kért értékeket.

```
php artisan migrate
```

#	Név	Típus	Illesztés	Tulajdonságok	Nulla	Alapértelmezett	Megjegyzések	Extra	Művelet
1	id	bigint(20)		UNSIGNED	Nem	*Nincs*		AUTO_INCREMENT	Módosítás Eldobás Több
2	category_name	varchar(255)	utf8mb4_unicode_ci		Nem	*Nincs*			Módosítás Eldobás Több
3	slug	varchar(255)	utf8mb4_unicode_ci		Nem	*Nincs*			Módosítás Eldobás Több
4	subcategory_count	int(11)			Nem	0			Módosítás Eldobás Több
5	product_count	int(11)			Nem	0			Módosítás Eldobás Több
6	created_at	timestamp			Igen	*NULL*			Módosítás Eldobás Több
7	updated_at	timestamp			Igen	*NULL*			Módosítás Eldobás Több

SubCategory "alkategória" kialakítása következő lépésben az adott paraméterekkel, ami majd kapcsolódik egy szülő kategóriához ami"category_id" mező azonosítja, hogy melyik szülő kategóriához tartozik az adott alkategória. Ezt a kapcsolatot gyakran idegen kulcsnak (foreign key) nevezik, és segít az adatbázisban hierarchikus struktúrák létrehozásában.

Ez a kapcsolat lehetővé teszi például azt, hogy könnyen lekérdezhető legyen az adott szülő kategória összes alkategóriája, vagy az alkategóriák számolhatók a szülő kategória alatt tartozó termékek számának meghatározásához.

#	Név	Típus	Illesztés	Tulajdonságok	Nulla	Alapértelmezett	Megjegyzések	Extra	Művelet
1	id	bigint(20)		UNSIGNED	Nem	*Nincs*		AUTO_INCREMENT	Módosítás Eldobás Több
2	subcategory_name	varchar(255)	utf8mb4_unicode_ci		Nem	*Nincs*			Módosítás Eldobás Több
3	category_id	bigint(20)			Nem	*Nincs*			Módosítás Eldobás Több
4	category_name	varchar(255)	utf8mb4_unicode_ci		Nem	*Nincs*			Módosítás Eldobás Több
5	product_count	int(11)			Nem	0			Módosítás Eldobás Több
6	slug	varchar(255)	utf8mb4_unicode_ci		Nem	*Nincs*			Módosítás Eldobás Több
7	created_at	timestamp			Igen	*NULL*			Módosítás Eldobás Több
8	updated_at	timestamp			Igen	*NULL*			Módosítás Eldobás Több

Ezt követően kialakítom a termékekhez kapcsolódó táblát ami már több paraméterrel bír.

id: Ez az egyedi azonosító mező, ami egyedi azonosítót tárol az adott rekordhoz.

product_name: Ez a mező tárolja a termék nevét, ami karakterlánc formában van.

product_short_desc: Ez a mező egy hosszabb karakterláncot tárol, amely rövid leírást ad a termékről.

product_long_desc: Ez a mező egy még hosszabb karakterláncot tárol, amely részletes leírást ad a termékről.

price: Ez egy egész szám mező, ami a termék árát tárolja.

product_category_name: Ez a mező karakterláncot tartalmaz, ami a termék kategória nevét tárolja.

product_subcategory_name: Ez a mező karakterláncot tartalmaz, ami a termék alkategória nevét tárolja.

product_category_id: Ez egy egész szám mező, ami az adott termék kategória azonosítóját tartalmazza. Ez a mező kapcsolatot teremt a kategória táblával, hogy megállapítsa, hogy a termék melyik kategóriához tartozik.

product_subcategory_id: Ez egy egész szám mező, ami az adott termék alkategória azonosítóját tartalmazza. Hasonlóan a product_category_id-hoz, ez a mező kapcsolatot teremt az alkategória táblával.

product_img: Ez a mező karakterláncot tartalmaz, ami a termék képének elérési útját vagy nevét tárolja.

slug: Ez a mező karakterláncot tartalmaz, amely a termék nevét tartalmazza URL-barát formában.

timestamps: Ez a mező az adatok létrehozásának és frissítésének dátumát és időpontját rögzíti.

Ezen a táblán keresztül tárolom a termékek adatait az alkalmazásodban.

1	id	int(10)		UNSIGNED	Nem	*Nincs*	AUTO_INCREMENT	Módosítás	Eldobás	Több
2	product_name	varchar(255)	utf8mb4_general_ci		Nem	*Nincs*		Módosítás	Eldobás	Több
3	product_short_desc	text	utf8mb4_general_ci		Nem	*Nincs*		Módosítás	Eldobás	Több
4	product_long_desc	text	utf8mb4_general_ci		Nem	*Nincs*		Módosítás	Eldobás	Több
5	price	decimal(10,2)			Nem	*Nincs*		Módosítás	Eldobás	Több
6	product_category_name	varchar(255)	utf8mb4_general_ci		Nem	*Nincs*		Módosítás	Eldobás	Több
7	product_subcategory_name	varchar(255)	utf8mb4_general_ci		Nem	*Nincs*		Módosítás	Eldobás	Több
8	product_category_id	int(10)		UNSIGNED	Nem	*Nincs*		Módosítás	Eldobás	Több
9	product_subcategory_id	int(10)		UNSIGNED	Nem	*Nincs*		Módosítás	Eldobás	Több
10	product_img	varchar(255)	utf8mb4_general_ci		Nem	*Nincs*		Módosítás	Eldobás	Több
11	quantity	int(10)		UNSIGNED	Nem	*Nincs*		Módosítás	Eldobás	Több
12	slug	varchar(255)	utf8mb4_general_ci		Nem	*Nincs*		Módosítás	Eldobás	Több
13	created_at	timestamp			Nem	current_timestamp()		Módosítás	Eldobás	Több
14	updated_at	timestamp			Nem	current_timestamp()	ON UPDATE CURRENT_TIMESTAMP()	Módosítás	Eldobás	Több

3.9 Category/ AddCategory és űrlap kilalkítása

Következő lépésben ki alakítom a POST methodust az űrlapon, a HTTP POST kérés egyike az HTTP kérési módszereknek, amelyet a webes alkalmazások használnak a szerverrel történő

kommunikáció során. A POST kérés arra szolgál, hogy adatokat küldjön el a szervernek a kérelem testében.

```
Route::post('/admin/store_category','Store_Category')->name('store_category');
```

Az űrlap elküldéséhez használják a POST kérést: Az űrlap elküldésének folyamata során a böngésző a felhasználó által kitöltött adatokat a POST kérés testében küldi el a szerver felé. Az űrlapban található mezők értékei, például a "Kategória Neve" mezőben megadott érték, a POST kérés testében találhatók. Ezáltal a szerver képes lesz megkapni és feldolgozni ezeket az adatokat.

Az űrlap feldolgozása: A szerver oldali kód (Laravel kontroller) a POST kérés fogadása után megkapja az űrlapból származó adatokat, például a kategória nevét. Ezután a kontroller feldolgozza ezeket az adatokat, például ellenőrzi az adatok érvényességét, majd végrehajtja az adatok mentését az adatbázisban vagy más műveleteket.

A kategória hozzáadásának kialakítása és a controller kialakítása.

```
class CategoryController extends Controller
{
    public function index()
    {
        $categories = Category::latest()->get();
        return view('admin.all_category', compact('categories'));
    }

    public function Add_Category()
    {
        return view('admin.add_category');
    }

    public function Store_Category(Request $request)
    {
        $request->validate([
            'category_name' => 'required|unique:categories'
        ]);
```

Sikeres átirányítás után térjen vissza evvel az üzenettel.

```php
public function Store_Category(Request $request)
{
    $request->validate([
        'category_name' => 'required|unique:categories'
    ]);

    Category::insert([
        'category_name' => $request->category_name,
        'slug' => strtolower(str_replace('','-',$request->category_name))
    ]);

    return redirect()->route('all_category')->with('message','Category Added Successfully');
}
```

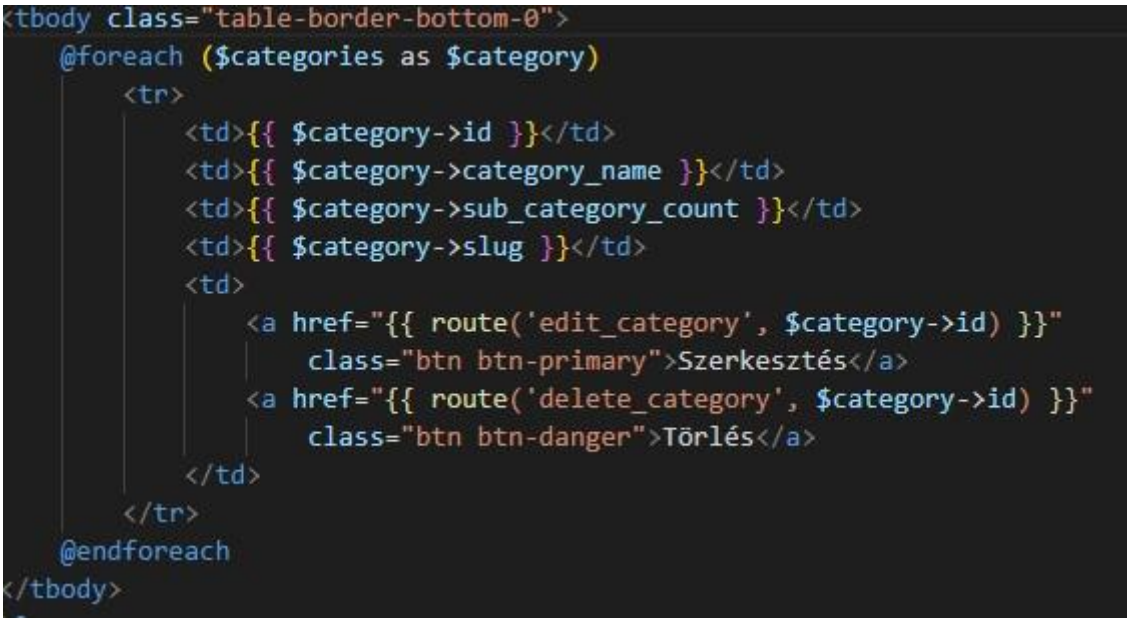
$request->validate([...]): Ebben a részben a Laravel validációt használják a HTTP kérésben érkező adatok ellenőrzésére. A validációs szabályokat egy asszociatív tömbben adták meg. Ebben az esetben a category_name mezőnek kötelezően kitöltöttnek kell lennie ('required'), és egyedinek kell lennie a categories táblában ('unique:categories'). Ha a validáció nem sikerül, a Laravel automatikusan visszairányítja a felhasználót az előző oldalra hibaüzenetekkel.

Category::insert([...]): Ha a validáció sikeres, akkor az új kategória adatait beszúrják az adatbázisba. Ezt a Category modell segítségével teszik meg. Az adatokat egy asszociatív tömbben adják meg, ahol a 'category_name' kulcs az űrlapból érkező category_name adatot tartalmazza, és a 'slug' kulcs egy olyan sztringet tartalmaz, ami a kategória nevét kisbetűssé alakítja és szóközöket kötőjelekre cseréli.

return redirect()->route('all_category')->with('message','Category Added Successfully'): Ha a beszúrás sikeres volt, a kódban a felhasználót átirányítják az összes kategória listázás oldalra ('all_category' útvonalra), és egy sikerüzenetet ('Category Added Successfully') mellékelnek, amelyet a with függvénnyel hozzáadnak az átirányításhoz. Ez az üzenet később megjeleníthető a felhasználónak a weboldalon.

Ez a függvény felelős az új kategória hozzáadásának folyamatáért a Laravel alkalmazásban. Az adatok validálása, az adatbázisba történő beszúrás, majd az átirányítás egy sikerüzenettel mind része az új kategória létrehozásának folyamatának.

Category view megjelenítése az összes nézetben:



Ez a kód részlet felelős a kategóriák dinamikus megjelenítéséért a táblázatban. Minden kategória egy új sorban jelenik meg, és az azonosító, név, alkategória száma és slug megjelenítése a megfelelő cellákban történik. A "Szerkesztés" és "Törlés" gombok lehetővé teszik az adminisztrátoroknak, hogy szerkesszék vagy töröljék a kategóriákat a megfelelő műveletekkel.

@foreach ($categories as $category): Ez a ciklus végigiterál az $categories változón, amely tartalmazza az összes kategória objektumot. Minden kategóriához a ciklus egy-egy alkalommal lép.

<td>{{ $category->id }}</td>: Ez a cella tartalmazza a kategória azonosítóját ($category->id). Az {{ }} zárójelbe tett kifejezés a Blade templating motor által dinamikusan generált tartalmat jelenít meg.

<td>{{ $category->category_name }}</td>: Ebben a cellában a kategória neve jelenik meg ($category->category_name).

<td>{{ $category->sub_category_count }}</td>: Itt a kategória alkategóriáinak számát jeleníti meg ($category->sub_category_count).

<td>{{ $category->slug }}</td>: Ebben a cellában a kategória slug-ja (valószínűleg az URL része) található ($category->slug).

Szerkesztés és Törlés gombok: Minden sorban két gomb található. A "Szerkesztés" gomb egy szerkesztési műveletet indít, a "Törlés" gomb pedig egy törlési műveletet. Az href attribútumok a Laravel route() függvényét használják az útvonalak generálásához. Például: {{ route('edit_category', $category->id) }} és {{ route('delete_category', $category->id) }}.

```php
class CategoryController extends Controller
{
    public function index()
    {
        $categories = Category::latest()->get();
        return view('admin.all_category', compact('categories'));
    }

    public function Add_Category()
    {
        return view('admin.add_category');
    }

    public function Store_Category(Request $request)
    {
        $request->validate([
            'category_name' => 'required|unique:categories'
        ]);

        Category::insert([
            'category_name' => $request->category_name,
            'slug' => strtolower(str_replace(' ','-',$request->category_name))
        ]);

        return redirect()->route('all_category')->with('message','Category Added Successfully');
    }
```

index(): Ez a metódus felelős a kategóriák listázásáért. Lekéri a legújabb kategóriákat a Category modell segítségével a latest() metódussal, majd visszaadja ezeket a nézetnek. A kategóriák listázása általában az "admin.all_category" nézetben történik.

Add_Category(): Ez a metódus egyszerűen visszaadja a "admin.add_category" nézetet, amely az új kategória hozzáadásának űrlapját tartalmazza. Ezt a metódust akkor használják, amikor az adminisztrátorok új kategóriát akarnak hozzáadni.

Store_Category(Request $request): Ez a metódus felelős az új kategória mentéséért az adatbázisba. Először is validálja az érkező HTTP kérésben található adatokat, ellenőrizve, hogy a "category_name" mező kitöltött-e és egyedinek számít-e a "categories" táblában. Ha a validáció sikeres, beszúrja az új kategóriát a "Category" modell segítségével az adatbázisba. A

"slug" mező generálása a kategória nevéből, ahol a szóközöket kötőjelekre cseréli. Végül átirányítja az adminisztrátorokat az összes kategória listázás oldalra a sikerüzenettel.

Ez a kontroller tehát a kategóriákkal kapcsolatos alapvető műveleteket kezeli, például listázás, új kategória hozzáadása és kategória mentése az adatbázisba. Ezzel lehetőséget biztosít az adminisztrátoroknak a kategóriák kezelésére a Laravel alapú webalkalmazásban.

3.10 Edit Category kialakítása:

Az útvonal kialakítással kezdtem ami egy GET-es kérést fogadjon el az /admin/edit_category/{id} címen, és az Edit_Category vezérlőben található metódus kezelje a kérést. Az {id} rész dinamikusan változó, és a kérés során megadott kategória azonosítóját jelképezi. Ezt az azonosítót a vezérlő metódus használja majd az adott kategória betöltéséhez a szerkesztő nézet számára. Az útvonalat "edit_category" néven lehet majd hivatkozni a továbbiakban a az aplikációban.

```
Route ::get('/admin/edit_category/{id}','Edit_Category')->name('edit_category');
```

Ezt követően a Controllerben a hozzá való függvényt is elkészítem ami tartalmazza a paramétert is.

```
public function Edit_Category($id)
{
    $category_info = Category::findOrFail($id);

        return view('admin.edit_category', compact('category_info'));
}
```

public function Edit_Category($id): Ez a függvény egy paramétert vár, amely azonosítja a szerkesztendő kategóriát. A $id változó tartalmazza ezt az azonosítót, és a függvény paraméterlistájában található.

$category_info = Category::findOrFail($id);: Ebben a sorban a $category_info változóban keresed meg a kategóriát az adatbázisból a Category modell segítségével.

A findOrFail metódus a megadott azonosító alapján megkeresi a kategóriát az adatbázisban, és ha nem találja, egy ModelNotFoundException kivételt dob. Ezáltal biztosítottam, hogy csak létező kategóriákat szerkeszthetővé tegyem.

return view('admin.edit_category', compact('category_info'));: Ebben a sorban visszaadod a "admin.edit_category" nézetet, és átadod neki a $category_info változót a compact függvény segítségével. Ez azt jelenti, hogy a nézet elérheti és megjelenítheti a kategória szerkesztő űrlapon használt adatokat.

Tehát ez a függvény a kategória szerkesztő nézetet jeleníti meg az adott kategória adataival, amelyeket az adatbázisból származnak, és az $id paraméter alapján kerülnek kiválasztásra. Ez lehetővé teszi a felhasználók számára, hogy szerkesszék a kategória adatait az adminisztrációs felületen.

Az űrlap lehetővé teszi a kategória nevének szerkesztését, és az űrlap elküldésekor az adatokat a "update_category" útvonalra küldi a szerkesztési folyamat elvégzéséhez. Az esetleges validációs hibák megjelenítése is kezelve van, és a CSRF védelem is biztosítva van a formon keresztül.

```
<h4 class="fw-bold py-3 mb-4"><span class="text-muted fw-light">Oldal/</span> Kategória szerkesztése</h4>
<div class="container">
    <div class="col-xxl">
        <div class="card mb-4">
            <div class="card-header d-flex align-items-center justify-content-between">
                <h5 class="mb-0">Kategória szerkesztése</h5>
            </div>
            <div class="card-body">
                @if ($errors->any())
                    <div class="alert alert-danger">
                        <ul>
                            @foreach ($errors->all() as $error)
                                <li>{{ $error }}</li>
                            @endforeach
                        </ul>
                    </div>
                @endif
                <form action="{{ route('update_category') }}" method="POST">
                    @csrf
                    <input type="hidden" value="{{ $category_info->id }}" name="category_id">
                    <div class="row mb-3">
                        <label class="col-sm-2 col-form-label" for="basic-default-name">Kategória neve</label>
                        <div class="col-sm-10">
                            <input type="text" class="form-control" id="category_name" name="category_name"
                                value="{{ $category_info->category_name }}" />
                        </div>
                    </div>
                    <div class="row justify-content-end">
                        <div class="col-sm-10">
                            <button type="submit" class="btn btn-primary">Kategória szerkesztése</button>
                        </div>
                    </div>
                </form>
            </div>
        </div>
    </div>
```

A szerkesztési űrlap rendelkezik @if ($errors->any()): Ez egy feltétel, amely ellenőrzi, hogy van-e bármilyen validációs hiba az előző űrlap elküldése után. Ha van, akkor egy hibaüzenetet jelenít meg a Bootstrap "alert" osztály segítségével.

A <form action="{{ route('update_category') }}" method="POST">: Az űrlap kezdete. Az "update_category" útvonalra küldi majd az űrlap adatait.

Az <input type="hidden" value="{{ $category_info->id }}" name="category_id">: Egy rejtett mező, amely az űrlapba beilleszti a szerkesztendő kategória azonosítóját. Ezt az azonosítót a szerkesztési folyamat során szükség lesz azonosítani.

A : Egy űrlap sor kezdete, amelyben a kategória nevét lehet szerkeszteni.

Az <input type="text" class="form-control" id="category_name" name="category_name" value="{{ $category_info->category_name }}" />: Az űrlapmező, amely lehetővé teszi a kategória nevének szerkesztését. Az aktuális kategória nevét a $category_info->category_name változóból tölti be.

A <button type="submit" class="btn btn-primary">Kategória szerkesztése</button>: Az "Kategória szerkesztése" gomb. A gomb stílusát a Bootstrap "btn" osztállyal állítja be.

3.11 Update Category:

Update Category kialakítását ugyan csak a web.php-ban egy POST kéréssel kezdtem.

```
Route::post('/admin/update_category','Update_Category')->name('update_category');
```

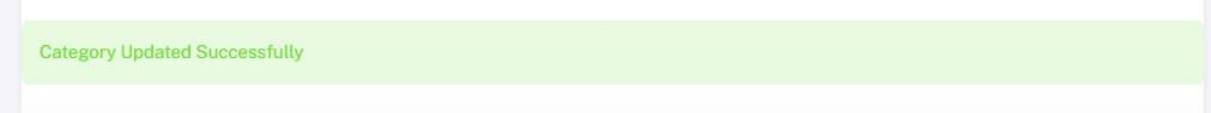
Update_Category függvény,a felelős egy kategória frissítési művelet végrehajtásáért az adminisztrációs felületen.

A $category_id = $request->category_id;: Ebben a sorban kinyertem a kategória azonosítóját a POST kérésből. Azaz megkapjuk azt az azonosítót, amely a kategória azonosítására szolgál.

A $request->validate([...]): Ebben a blokkban az érkező kérés validálását végeztem el. Itt ellenőriztem, hogy a beérkező adatok megfelelnek-e a megadott szabályoknak. A kód jelenleg azt ellenőrzi, hogy a category_name egyedi legyen a categories táblában, és a végén ha minden sikeres volt vissza tér az (all_category) útvonalra egy üzenettel.

Ez az üzenet az értesítésre vagy megerősítésre szolgál, és a "Category Updated Successfully" szöveggel van beállítva.

Ez a függvény tehát a kategória frissítési műveletét végzi el az adminisztrációs felületen. A kódban a validáció segít ellenőrizni, hogy a beérkező adatok megfeleljenek-e a szabályoknak, majd a kategória frissítését hajtja végre az adatbázisban.



3.12 Category Delete kialakítása:

Az útvonal egy GET-es kérésre válaszol ami "/admin/delete_category/{id}" elérési úton lesz elérhető.

```
Route ::get('/admin/delete_category/{id}','Delete_Category')->name('delete_category');
```

A Delete_Category, felelős egy kategória törlési művelet végrehajtásáért az adminisztrációs felületen.

Ugyan csak itt is rendelkezik egy "$id" paraméterrel ez azonosítja a kategóriat amit törölni szeretnénk.

A kódban az {id} rész az útvonalból származik, amikor a kérés érkezik.

A Category::findOrFail($id)->delete();: Ebben a sorban megtalálod a megadott $id-vel rendelkező kategóriát az adatbázisban a Category modell segítségével. Ha a kategória nem található, a findOrFail metódus egy ModelNotFoundException kivételt dob. Ha a kategória megtalálható, a delete metódust hívod meg rajta, ami törli a kategóriát az adatbázisból.

És ismételten ha jól fut le akkor return redirect()->route('all_category')->with('message','Category Deleted Successfully!');: Végül visszairányítom a felhasználót egy másik útvonalra (all_category), és hozzáadom az üzenetet (message) a visszairányításhoz. Az üzenet azt jelzi, hogy a kategória sikeresen törölve lett.

Ez a függvény tehát a kategória törlési műveletét végzi el az adminisztrációs felületen. Először megtalálja a megfelelő kategóriát az adatbázisban, majd törli azt. A felhasználót végül egy másik oldalra irányítja át, és sikeres törlés esetén egy üzenetet jelenít meg.

3.13 A SubCategory kialakítása:

```php
Route::controller(SubCategoryController::class)->group(function(){
    Route::get('/admin/all_subcategory','Index')->name('all_subcategory');
    Route::get('/admin/add_subcategory','Add_SubCategory')->name('add_subcategory');
    Route::post('/admin/store_subcategory','Store_SubCategory')->name('store_subcategory');
    Route::get('/admin/edit_subcategory/{id}', 'Edit_SubCategory')->name('edit_subcategory');
    Route::get('/admin/delete_subcategory/{id}', 'Delete_SubCategory')->name('delete_subcategory');
    Route::post('/admin/update_subcategory', 'Update_SubCategory')->name('update_subcategory');
```

Ez az útvonalcsoport egy egységes helyet biztosít az összes az alalkategóriával kapcsolatos művelet kezeléséhez a SubCategoryController kontrolleren belül. A nevek (name) lehetővé teszik, hogy könnyen hivatkozhass az útvonalakra az alkalmazás különböző részeiben.

A Route::get('/admin/all_subcategory','Index')->name('all_subcategory');: Ez a GET útvonal az "/admin/all_subcategory" címen érhető el, és a Index metódust hívja meg a SubCategoryController kontrollerben. Ezt az útvonalat "all_subcategory" néven lehet majd hivatkozni.

A Route::get('/admin/add_subcategory','Add_SubCategory')->name('add_subcategory');: Ez a GET útvonal az "/admin/add_subcategory" címen érhető el, és a Add_SubCategory metódust hívja meg a SubCategoryController kontrollerben. Ezt az útvonalat "add_subcategory" néven lehet majd hivatkozni.

A Route::post('/admin/store_subcategory','Store_SubCategory')->name('store_subcategory');: Ez a POST útvonal az "/admin/store_subcategory" címen érhető el, és a Store_SubCategory metódust hívja meg a SubCategoryController kontrollerben.

A Route::get('/admin/edit_subcategory/{id}', 'Edit_SubCategory')->name('edit_subcategory');: Ez a GET útvonal dinamikus azonosítóval ({id}) az

"/admin/edit_subcategory/{id}" címen érhető el, és a Edit_SubCategory metódust hívja meg a SubCategoryController kontrollerben. Ezt az útvonalat "edit_subcategory" néven lehet majd hivatkozni.

A Route::get('/admin/delete_subcategory/{id}', 'Delete_SubCategory')->name('delete_subcategory');: Ez a GET útvonal dinamikus azonosítóval ({id}) az "/admin/delete_subcategory/{id}" címen érhető el, és a Delete_SubCategory metódust hívja meg a SubCategoryController kontrollerben.

Route::post('/admin/update_subcategory', 'Update_SubCategory')->name('update_subcategory');: Ez a POST útvonal az "/admin/update_subcategory" címen érhető el, és a Update_SubCategory metódust hívja meg a SubCategoryController kontrollerben.

```php
Route::controller(SubCategoryController::class)->group(function(){
    Route::get('/admin/all_subcategory','Index')->name('all_subcategory');
    Route::get('/admin/add_subcategory','Add_SubCategory')->name('add_subcategory');
    Route::post('/admin/store_subcategory','Store_SubCategory')->name('store_subcategory');
    Route::get('/admin/edit_subcategory/{id}', 'Edit_SubCategory')->name('edit_subcategory');
    Route::get('/admin/delete_subcategory/{id}', 'Delete_SubCategory')->name('delete_subcategory');
    Route::post('/admin/update_subcategory', 'Update_SubCategory')->name('update_subcategory');
```

3.14 SubCategory Controller

A SubCategory Controllert hasonlóan építettem fel mint a Category controllert csak kevés az eltérés de röviden bemutatom.

```php
<?php

namespace App\Http\Controllers\Admin;

use App\Models\Category;
use App\Models\SubCategory;
use App\Http\Controllers\Controller;
use Illuminate\Http\Request;

class SubCategoryController extends Controller
{
    public function Index()
    {
        $all_subcategories = SubCategory::latest()->get();
        return view('admin.all_subcategory', compact('all_subcategories'));
    }

    public function Add_SubCategory()
    {
        $categories = Category::latest()->get();
        return view('admin.add_subcategory', compact('categories'));
    }
```

Műveletek és adatok kezelése: A CategoryController a kategóriák (categories) kezelésére szolgál, míg a SubCategoryController a szubkategóriák (subcategories) kezelésére szolgál. Mindkét kontroller azonos típusú műveleteket tartalmaz (index, hozzáadás, szerkesztés, törlés), de más adatokkal és adatbázistáblákkal dolgozik.

Adatlekérdezés és validáció: Mindkét kontrollerben van adatlekérdezés a táblákból, például a kategóriák lekérdezése vagy az egyedi ellenőrzés a beérkező adatokra vonatkozóan a validate függvény használatával. Az adatbázis-táblák és a mezőnevek eltérnek a két kontroller között.

Nézetek: A nézetek (view) különböznek a két kontroller között, és az adott funkciókhoz igazodnak. A CategoryController nézetei például a kategóriák listázását, hozzáadását és szerkesztését tartalmazzák, míg a SubCategoryController nézetei a szubkategóriákhoz kapcsolódnak.

Adatok kezelése és átadása a nézetekbe: A CategoryController a kategória nevet és a kategória azonosítóját tárolja és használja az adatbázisban, míg a SubCategoryController a szubkategória nevét, szlugját, a hozzá tartozó kategória nevét és az azonosítóját tárolja.

Útvonalak és elérési útvonalak nevei: A két kontrollernek különböző útvonalkonfigurációi vannak, és eltérő útvonalak nevei vannak a műveletekhez. Például a CategoryController-ben

van egy all_category útvonal, míg a SubCategoryController-ben van egy all_subcategory útvonal.

Ezek a különbségek mutatják, hogy mindkét kontroller különböző típusú adatokat kezel és különböző funkcionalitást szolgál ki az alkalmazásban. Az egyik a kategóriákhoz kapcsolódik, míg a másik a szubkategóriákhoz. A megfelelő kontroller kiválasztása az alkalmazás logikájától és követelményeitől függ.

3.15 Product kialakítása

3.15.1 Route és controller kialakítása

Az általam bemutatott kód egy másik csoportot definiál az útvonalak számára a ProductController kontrollerrel. Hasonlóan az előzőhöz, ez is egyszerűsíti az útvonalak kezelését.

Route::controller(ProductController::class): Ezzel meghatároztam a ProductController-t, amelyet a csoportban lévő útvonalak kezelnek.

->group(function(){ ... }): Ebben a blokkban definiáltam a csoportot az útvonalak számára.

```
Route::controller(ProductController::class)->group(function(){
    Route::get('/admin/all_products','Index')->name('all_products');
    Route::get('/admin/add_product','Add_Product')->name('add_product');
    Route::post('/admin/store_product','Store_Product')->name('store_product');
    Route::get('/admin/edit_product_img/{id}','Edit_Product_Img')->name('edit_product_img');
    Route::post('/admin/update_product_img','Update_Product_Img')->name('update_product_img');
    Route::get('/admin/edit_product/{id}','Edit_Product')->name('edit_product');
    Route::post('/admin/update_product', 'Update_Product')->name('update_product');
    Route::get('/admin/delete_product/{id}', 'Delete_Product')->name('delete_product');
});

Route::controller(OrderController::class)->group(function(){
    Route::get('/admin/pending_order','Index')->name('pending_order');

});
```

Az Útvonalak bemutatása:

Route::get('/admin/all_products','Index')->name('all_products');: Ez a GET útvonal az "/admin/all_products" címen érhető el, és az Index metódust hívja meg a ProductController kontrollerben. Az útvonal neve "all_products".

Route::get('/admin/add_product','Add_Product')->name('add_product');: Ez a GET útvonal az "/admin/add_product" címen érhető el, és az Add_Product metódust hívja meg a ProductController kontrollerben. Az útvonal neve "add_product".

Route::post('/admin/store_product','Store_Product')->name('store_product');: Ez a POST útvonal az "/admin/store_product" címen érhető el, és a Store_Product metódust hívja meg a ProductController kontrollerben. Az útvonal neve "store_product".

Route::get('/admin/edit_product_img/{id}','Edit_Product_Img')->name('edit_product_img');: Ez a GET útvonal dinamikus azonosítóval ({id}) az "/admin/edit_product_img/{id}" címen érhető el, és az Edit_Product_Img metódust hívja meg a ProductController kontrollerben. Az útvonal neve "edit_product_img".

Route::post('/admin/update_product_img','Update_Product_Img')->name('update_product_img');: Ez a POST útvonal az "/admin/update_product_img" címen érhető el, és az Update_Product_Img metódust hívja meg a ProductController kontrollerben. Az útvonal neve "update_product_img".

Route::get('/admin/edit_product/{id}','Edit_Product')->name('edit_product');: Ez a GET útvonal dinamikus azonosítóval ({id}) az "/admin/edit_product/{id}" címen érhető el, és az Edit_Product metódust hívja meg a ProductController kontrollerben. Az útvonal neve "edit_product".

Route::post('/admin/update_product', 'Update_Product')->name('update_product');: Ez a POST útvonal az "/admin/update_product" címen érhető el, és az Update_Product metódust hívja meg a ProductController kontrollerben. Az útvonal neve "update_product".

Route::get('/admin/delete_product/{id}', 'Delete_Product')->name('delete_product');: Ez a GET útvonal dinamikus azonosítóval ({id}) az "/admin/delete_product/{id}" címen érhető el, és a Delete_Product metódust hívja meg a ProductController kontrollerben. Az útvonal neve "delete_product".

Ez az útvonalcsoport létrehoz egy közös helyet az összes azonosítóhoz tartozó termék művelet kezeléséhez a ProductController kontrolleren belül. A névadások (name) lehetővé teszik, hogy könnyen hivatkozni tudjak az útvonalakra a Laravel alkalmazás különböző részeiben.

3.15.2 A Product controller kialakítása:

Az Index: Ez a függvény lekéri a legfrissebb termékeket az adatbázisból és visszaadja őket az "admin.all_products" nézetnek, hogy megjeleníthesse az összes terméket az adminisztrációs felületen.

```
public function Index()
{
    $products = Products::latest()->get();
    return view('admin.all_products', compact('products'));
}
```

Az Add_Product: Ez a függvény előkészíti a "admin.add_product" nézetet a termék hozzáadásához. Lekéri a legfrissebb kategóriákat és alkategóriákat az adatbázisból, majd átadja ezeket a nézetnek, hogy a felhasználó kiválaszthassa, melyik kategóriába és alkategóriába szeretné hozzáadni a terméket

```
public function Add_Product()
{
    $categories = Category::latest()->get();
    $subcategories = Subcategory::latest()->get();

    return view('admin.add_product', compact('categories', 'subcategories'));
}
```

A Store_Product: Ez a függvény felelős a termék hozzáadásának feldolgozásáért. Először validálja a beérkezett adatokat, például a termék nevét, árát, mennyiségét stb. Ezután kezeli

a termék képét (ha van feltöltve), és a termékhez tartozó kategória és alkategória nevét. Végül beszúrja az adatokat az adatbázisba és növeli a kategória és alkategória termékszámát. Az útvonalon keresztül visszairányítja az "admin.all_products" nézetre a sikerüzenettel.

```php
public function Store_Product(Request $request)
{
    $request->validate([
    'product_name' => 'required|unique:products',
    'price' => 'required',
    'quantity' => 'required',
    'product_short_desc' => 'required',
    'product_long_desc' => 'required',
    'product_category_id' => 'required',
    'product_subcategory_id' => 'required',
    'product_img' => 'image|mimes:jpeg,png,jpg,gif,svg|max:2048',

]);

$image = $request->file('product_img');
$img_name = hexdec(uniqid()) . '.' . $image->getClientOriginalExtension();
$request->product_img->move(public_path('upload'), $img_name);
$img_url = 'upload/' . $img_name;

$category_id = $request->product_category_id;
$subcategory_id = $request->product_subcategory_id;

$category_name = Category::where('id', $category_id)->value('category_name');
$subcategory_name = Subcategory::where('id', $subcategory_id)->value('subcategory_name');


Products::insert([
    'product_name' => $request->product_name,
    'product_short_desc' => $request->product_short_desc,
    'product_long_desc' => $request->product_long_desc,
    'price' => $request->price,
    'product_category_name' => $category_name,
    'product_subcategory_name' => $subcategory_name,
    'product_category_id' => $request->product_category_id,
    'product_subcategory_id' => $request->product_subcategory_id,
    'product_img' => $img_url,
    'quantity' => $request->quantity,
    'slug' => strtolower(str_replace(' ','-',$request->product_name)),

]);
Category::where('id', $category_id)->increment('product_count', 1);
Subcategory::where('id', $subcategory_id)->increment('product_count', 1);

return redirect()->route('all_products')->with('message', 'Product Added Successfully!');
}
```

A $request->validate([...]): Ebben a sorban validáltam a beérkezett adatokat a Laravel által nyújtott validációs szabályok segítségével. Ellenőrzi, hogy minden kötelező adatot megadtak-e, és hogy azok a megadott szabályoknak megfelelnek-e.

Az $image = $request->file('product_img');: Ezzel a sorral lekértem a feltöltött termékkép fájlját a kérelemből.

Az $img_name = hexdec(uniqid()) . '.' . $image->getClientOriginalExtension();: Itt generáltam egy egyedi nevet a feltöltött kép számára. A név a hexdec(uniqid()) segítségével egyedi lesz, és hozzáadtam a kép eredeti kiterjesztését (getClientOriginalExtension).

A $category_id és $subcategory_id: Ezekben a változókban eltárolom a termék kategória és alkategória azonosítóját, amit a kérelmetől kapok.

A $category_name és $subcategory_name: Ezekben a sorokban lekérem a kategória és alkategória nevét az azonosítók alapján a Category és Subcategory modellek segítségével.

Az alábbi részben az összes adatot beszúrom az adatbázisba az insert metódussal, beleértve a termék nevét, leírását, árát, kategória nevét, alkategória nevét, stb. Az elkészített kép URL-je is bekerül az adatbázisba.

```
Products::insert([
    'product_name' => $request->product_name,
    'product_short_desc' => $request->product_short_desc,
    'product_long_desc' => $request->product_long_desc,
    'price' => $request->price,
    'product_category_name' => $category_name,
    'product_subcategory_name' => $subcategory_name,
    'product_category_id' => $request->product_category_id,
    'product_subcategory_id' => $request->product_subcategory_id,
    'product_img' => $img_url,
    'quantity' => $request->quantity,
    'slug' => strtolower(str_replace(' ','-',$request->product_name)),

]);
```

Category::where('id', $category_id)->increment('product_count', 1); és Subcategory::where('id', $subcategory_id)->increment('product_count', 1);: Ezen sorok segítségével növelem a kategória és alkategória termékszámát 1-gyel, mivel hozzáadtam egy új terméket.

Végül a függvény visszairányítja az "admin.all_products" nézetre a sikerüzenettel.

Ez a kód hozzáad egy új terméket az adatbázishoz, a képpel együtt, és frissíti a kategória és alkategória termékszámát is.

3.15.3 A Product view réteg:

All product megjelíntő réteg egy táblázatot jelenít meg az összes termékinformációval az adminisztrációs felületen.

Az első sorban egy táblázatot hoztam létre, és beállítottam a címsort (<h5 class="card-header">Minden elérhető termékinformáció</h5>).

Ezek a címsorok meghatározzák, hogy milyen oszlopokat tartalmaz majd a táblázat. Az "Azonosító" azonosítókat tartalmaz, a "Termék neve" a termék neveket, a "Kép" a termék képeit, az "Ár" az árakat, az "Akció" pedig különböző műveletekhez tartalmaz gombokat.

A táblázat tartalmát egy foreach ciklussal generáltam az alábbi módon:

```
<tbody class="table-border-bottom-0">
    @foreach ($products as $product)
        <tr>
            <td>{{ $product->id }}</td>
            <td>{{ $product->product_name }}</td>
            <td>
                <img style="height:100px;" src="{{ asset($product->product_img) }}" alt="">
                <br>
                <a href="{{ route('edit_product_img', $product->id) }}" class="btn btn-primary">Kép
                    frissítése</a>
            </td>
            <td>{{ $product->price }}</td>
            <td>
                <a href="{{ route('edit_product', $product->id) }}"
                    class="btn btn-primary">Szerkesztés</a>
                <a href="{{ route('delete_product', $product->id) }}" class="btn btn-danger">Törlés</a>
            </td>
        </tr>
    @endforeach
```

A foreach ciklus végigiterál az összes terméken, amelyeket a $products változó tartalmaz.

Minden termékhez egy sor kerül a táblázatban.

Az "Azonosító" oszlopban a termék azonosítóját jelenítettem meg a {{ $product->id }} változó segítségével.

A "Termék neve" oszlopban a termék nevét jelenítettem meg a {{ $product->product_name }} változóval.

A "Kép" oszlopban megjelenítettem a termék képét, valamint egy "Kép frissítése" gombot, amely a termék képének szerkesztéséhez vezet

Az "Ár" oszlopban az árat jelenítettem meg a {{ $product->price }} változóval.

Az "Akció" oszlopban két gomb található: "Szerkesztés" és "Törlés". Ezek a gombok különböző műveleteket hajtanak végre a termékekkel, például szerkesztést vagy törlést, és az útvonalak (route) segítségével vezetnek el a megfelelő helyekre.

3.15.4 Add Product view réteg:

Az űrlapban számos mező található, amelyek lehetővé teszik a felhasználó számára a termék adatainak megadását:

"Termék neve" (product_name): Egy szöveges mező, ahol a felhasználó megadhatja a termék nevét.

"Termék ára" (price): Egy szám mező, ahol a felhasználó megadhatja a termék árát.

"Termék mennyisége" (quantity): Egy szám mező, ahol a felhasználó megadhatja a termék mennyiségét.

"Termék rövid leírása" (product_short_desc): Egy szöveges terület, ahol a felhasználó rövid leírást adhat a termékről.

"Termék hosszú leírása" (product_long_desc): Egy nagyobb szöveges terület, ahol a felhasználó hosszabb leírást adhat a termékről.

"Válasszon kategóriát" (product_category_id): Egy legördülő menü, amely lehetővé teszi a felhasználó számára, hogy kiválassza a termék kategóriáját. Az opciók a $categories változóból kerülnek betöltésre.

"Válasszon alkategoriát" (product_subcategory_id): Egy másik legördülő menü, ahol a felhasználó kiválaszthatja a termék alkategóriáját. Az opciók a $subcategories változóból kerülnek betöltésre.

"Termék kép feltöltése" (product_img): Egy fájl feltöltési mező, ahol a felhasználó feltöltheti a termék képét

Az űrlap alján található egy "Termék hozzáadása" gomb, amely segítségével elküldhető az űrlap adatai a szervernek.

Az űrlap tetején található címsor (<h4>) tartalmazza az űrlap címét, például "Termék hozzáadása".

Az űrlap hibaüzeneteit az @if ($errors->any()) blokkban jelenítettem meg, ha a szerver oldalon valamilyen hiba történik, például egy mező kitöltési hibája.

Ez a kódrészlet lehetővé teszi az adminisztrátorok számára, hogy új termékeket adjanak hozzá a rendszerhez az adminisztrációs felületen keresztül, és minden szükséges információt rögzíthetnek a termékekről, beleértve a nevet, árat, leírásokat, kategóriát, alkategóriát és képet is.

3.15.5 Product IMG feltölése:

Az $id változóban tárololom azon termék azonosítóját, amelynek a képét szerkeszteni szeretném.

A Products::findOrFail($id) segítségével lekérem a kiválasztott termék adatait az adatbázisból.

Végül a view() függvénnyel egy nézetet renderelek, amely tartalmazza az "admin.edit_product_img" nézetet, és átadom neki a kiválasztott termék adatait a compact('product_info') segítségével. Ez a nézet lehetővé teszi az adminisztrátorok számára, hogy megtekintsék és szerkesszék a termék képét az adminisztrációs felületen keresztül.

Ezáltal az "Edit_Product_Img" oldalra jutva az adminisztrátor képes lesz módosítani vagy cserélni a kiválasztott termék képét.

```php
public function Edit_Product_Img($id)
{
    $product_info = Products::findOrFail($id);
    return view('admin.edit_product_img', compact('product_info'));

}
```

3.15.6 Update Product IMG:

Ez a kódrészlet az "Update_Product_Img" elnevezésű függvényt tartalmazza a "ProductController" osztályban. Ennek a függvénynek a célja egy termék képének frissítése.

Az első lépésben validáltam a képfeltöltést, hogy ellenőriztem, hogy a feltöltött fájl valóban kép és megfelelő formátumú (jpeg, png, jpg, gif, svg) és méretű (maximum 2048 KB).

Az $id változóban eltároltam a termék azonosítóját, amit a kép frissítéséhez használok.

Az $image változóban eltárolom a képfájlt, amit a felhasználó feltöltött az űrlapon.

Az $img_name változóban létrehozotam a frissített kép nevét. A nevet úgy generáltam, hogy a jelenlegi időbélyeg (hexadecimális formátumban) hozzáadódik a fájl eredeti kiterjesztéséhez.

A $request->product_img->move() metódussal a feltöltött képet átmásolom a public_path('upload') könyvtárba a létrehozott név alatt.

Az új kép URL-jét az $img_url változóban tárolom.

Megkeresem a terméket az adatbázisban a megadott $id alapján a Products::findOrFail($id) segítségével.

Az update() metódussal frissíted a termék képét az új URL-lel.

Végül visszatérek a "Minden elérhető termékinformáció" oldalra a redirect()->route('all_products') segítségével, és hozzáadok egy sikerüzenetet (->with('message', 'Product Image Updated Successfully!')), amit megjelenítek a felhasználónak a kép frissítése után.

Ez a kódrészlet lehetővé teszi a termék képének frissítését az adminisztrátorok számára az adminisztrációs felületen keresztül.

```php
public function Update_Product_Img(Request $request)
{
    $request->validate([
        'product_img' => 'required|image|mimes:jpeg,png,jpg,gif,svg|max:2048',
    ]);

    $id = $request->id;
    $image = $request->file('product_img');
    $img_name = hexdec(uniqid()) . '.' . $image->getClientOriginalExtension();
    $request->product_img->move(public_path('upload'), $img_name);
    $img_url = 'upload/' . $img_name;

    Products::findOrFail($id)->update([
        'product_img' => $img_url,
    ]);

    return redirect()->route('all_products')->with('message', 'Product Image Updated Successfully!');
}
```

3.15.7 Update Product:

Ennek a függvénynek a célja egy termék adatainak frissítése az adminisztrációs felületen keresztül.

Először is, az $id változóban tárolom azon termék azonosítóját, amelynek az adatait frissíteni szeretném. Az adminisztrátor az "Edit_Product" oldalon választhatja ki a szerkeszteni kívánt terméket, és az azonosító alapján dolgozom.

A validate metódussal ellenőrzöm a beérkezett adatokat. A validációs szabályok meghatározzák, hogy a termék neve (product_name), ára (price), mennyisége (quantity), rövid leírása (product_short_desc) és hosszú leírása (product_long_desc) megfelelő formátumban és érvényes értékekkel rendelkezzenek.

A Products::findOrFail($product_id)->update([]) részben végrehajtom a termék adatainak frissítését az adatbázisban. A update metódus segítségével módosítom a kiválasztott termék adatait az adatbázisban az adminisztrátor által megadott új értékekkel.

Végül a redirect metódus segítségével visszairányítom az adminisztrátort az "all_products" nevű oldalra (valószínűleg ez a termékek listázását végző oldal), és egy üzenetet is átadok a with metódus segítségével, hogy értesítsd a felhasználót arról, hogy a termék adatai sikeresen frissítésre kerültek.

Ezzel a függvénnyel az adminisztrátor képes lesz frissíteni egy termék adatait az adminisztrációs felületen keresztül.

```php
public function Update_Product(Request $request)
{
    $product_id = $request->id;

    $request->validate([
        'product_name' => 'required|unique:products',
        'price' => 'required',
        'quantity' => 'required',
        'product_short_desc' => 'required',
        'product_long_desc' => 'required',
    ]);

    Products::findOrFail($product_id)->update([
        'product_name' => $request->product_name,
        'product_short_desc' => $request->product_short_desc,
        'product_long_desc' => $request->product_long_desc,
        'price' => $request->price,
        'quantity' => $request->quantity,
        'slug' => strtolower(str_replace('','-',$request->product_name)),
    ]);

    return redirect()->route('all_products')->with('message', 'Product Information Updated Successfully!');
}
```

3.15.8 Delete Product:

A függvény célja egy termék törlése az adminisztrációs felületről.

Először is, az $id változóban tárolom azon termék azonosítóját, amelyet törölni szeretnélk. Az adminisztrátor az "all_products" oldalon választhatja ki a törölni kívánt terméket, és az azonosító alapján dolgozom.

Ezt követően a $cat_id és $subcat_id változókban lekérem a törlendő termékhez tartozó kategória és alkategória azonosítóit. Ezekre az azonosítókra szükség van, mert a termék törlésekor szükséges csökkenteni a kategória és alkategória "product_count" mezőjét.

A Category::where('id', $cat_id)->decrement('product_count', 1) sorban csökkentem a kategória "product_count" értékét eggyel. Ugyanezt teszem az alkategóriával is a SubCategory::where('id', $subcat_id)->decrement('product_count', 1) sorral. Ez a lépés azért szükséges, hogy a kategóriák és alkategóriák számlálói megfelelően frissüljenek a termék törlése után.

Végül a $id alapján törölni szeretném a terméket a Products::findOrFail($id)->delete() sorral.

Miután a termék sikeresen törölve lett az adatbázisból, a redirect metódus segítségével visszairányítom az adminisztrátort az "all_products" nevű oldalra (valószínűleg ez a termékek listázását végző oldal), és egy üzenetet is átadok a with metódus segítségével, hogy értesítsem a felhasználót arról, hogy a termék sikeresen törölve lett és az információk frissítve lettek.

```php
public function Delete_Product($id)
{
    $cat_id = Products::where('id',$id)->value('product_category_id');
    $subcat_id = Products::where('id',$id)->value('product_subcategory_id');
    Category::where('id',$cat_id)->decrement('product_count',1);
    SubCategory::where('id',$subcat_id)->decrement('product_count',1);
    Products::findOrFail($id)->delete();

    return redirect()->route('all_products')->with('message', 'Product Deleted Information Updated Successfully!');
}
```

3.16 Order kialakítása:

3.16.1 Order route

Az általam megadott útvonal definíció, amely a OrderController-t használja a vezérlőként (controller) az adminisztrációs felületen azonosítatlan feldolgozatlan rendelések (pending orders) listázására.

Route::controller(OrderController::class): Ez a sor meghatározza, hogy a OrderController vezérlő felelős az útvonalak kezeléséért ebben a csoportban.

group(function() { ... }): Ez a függvény csoportba helyezi az útvonalakat, ami azt jelenti, hogy az itt definiált útvonalaknak ugyanazok a közvetlen előtagjai lesznek (a következő kódblokkban minden útvonal az /admin útvonalelemmel kezdődik).

Route::get('/admin/pending_order','Index'): Ez a sor definiálja az adminisztrációs felületen a feldolgozatlan rendeléseket listázó útvonalat. A Route::get metódus azt jelenti, hogy ezt az útvonalat HTTP GET kérésekkel lehet elérni. Az /admin/pending_order a relatív útvonalat határozza meg, és a 'Index' a vezérlőn belüli metódust, ami a OrderController osztályban található Index metódust hívja meg a rendelések listázására.

A ->name('pending_order'): Ez a sor beállítja a nevet (name) az útvonalnak, ami lehetővé teszi a kód más részei számára, hogy könnyen hivatkozzanak erre az útvonalra egy név alapján. Ebben az esetben az útvonal neve 'pending_order'

Evvel meghatároztam egy útvonalat a feldolgozatlan rendelések listázására az adminisztrációs felületen, és ezt az útvonalat a OrderController Index metódusa kezeli. Az útvonal neve pedig 'pending_order'.

```php
Route::controller(OrderController::class)->group(function(){
    Route::get('/admin/pending_order','Index')->name('pending_order');

});
```

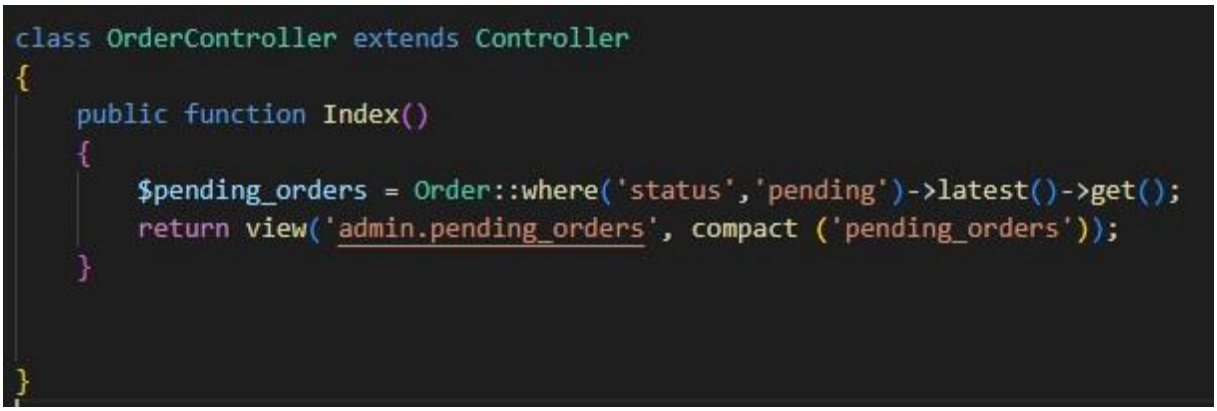
3.16.2 Order controller:

A Index metódus felelős az adminisztrációs oldal megjelenítéséért, ahol a feldolgozatlan rendeléseket jelenítik meg. A $pending_orders változóban lekérdezi az összes olyan rendelést, amelyek státusza "pending" (folyamatban lévő). Ezt a Order::where('status', 'pending')->latest()->get() sorral teszi meg. Az Order modell azonosítja a rendeléseket az adatbázisban.

A compact('pending_orders') rész segítségével átadja a pending_orders változót a nézetnek, így a nézet (HTML sablon) használhatja ezt az adatot a megjelenítéshez.

Végül a metódus visszatér egy nézetet (view) megjelenítő utasítással. A név, amelyet megpróbál megjeleníteni, a 'admin.pending_orders'. Ez a név a resources/views mappában található nézetfájlt azonosítja, amelyet a felhasználó a böngészőjében látni fogja.

Ez a kontroller arra szolgál, hogy a feldolgozatlan rendeléseket lekérdezze és megjelenítse az adminisztrációs felületen. Az elkészült nézet feldolgozza és megjeleníti ezeket az adatokat a weboldalon.



```
class OrderController extends Controller
{
    public function Index()
    {
        $pending_orders = Order::where('status','pending')->latest()->get();
        return view('admin.pending_orders', compact ('pending_orders'));
    }


}
```

3.16.3 Pendig Order view réteg:

: Egy konténer elem, amely tartalmazza a feldolgozatlan rendeléseknek szánt tartalmat. A my-5 osztály az elem margin értékét állítja be a felső és alsó oldalon, hogy megfelelő térköz legyen a tartalom körül.

: Egy kártya elem, amely tartalmazza a rendeléseket. A p-4 osztály a kártya belső térközét állítja be.

: Kártya cím elem, amely tartalmazza a "Pending Orders" (Feldolgozatlan rendelések) címet.

<h2>Pending Orders</h2>: Az aktuális oldal címe.

: A kártya testének eleme, amely tartalmazza a rendeléseket egy táblázatban.

<table class="table">: Egy HTML táblázatot indít, amelyben megjelenítjük a rendeléseket és azok adatait.

<tr>: Egy sor a táblázatban, ami tartalmazza a fejlécet.

A @foreach utasításban végigiterálunk a $pending_orders változón, amely a feldolgozatlan rendeléseket tartalmazza, és minden rendeléshez egy táblázatsort hozunk létre.

Az egyes táblázatsorokban a <td> elemekben jelenítjük meg a rendelés adatait, például a felhasználó azonosítóját, a szállítási információkat, a termék azonosítóját, a mennyiséget és az összes fizetést.

A <a> elem a "Jóváhagyás most" gombot reprezentálja, amelyre kattintva a rendelést jóváhagyhatjuk. Az href attribútumot még be kell állítani a jóváhagyás működéséhez.

Ez a nézet felelős a feldolgozatlan rendelések megjelenítéséért az adminisztrációs felületen, és lehetőséget nyújt azok jóváhagyására.

```
@section('page_title')
    Pending Orders - Eshop
@endsection
@section('content')
    <div class="container my-5">
        <div class="card p-4">
            <div class="card-title">
                <h2>Pending Orders</h2>
            </div>
            <div class="card-body">
                <table class="table">
                    <tr>
                        <th>Felhasználó azonosítója</th>
                        <th>Szállítási információk</th>
                        <th>Termék azonosítója</th>
                        <th>Mennyiség</th>
                        <th>Összes fizetés</th>
                        <th>Akció</th>
                    </tr>
                    @foreach ($pending_orders as $order)
                        <tr>
                            <td>{{ $order->user_id }}</td>
                            <td>
                                <ul>
                                    <li>Telefonszám - {{ $order->shipping_phone_number }}</li>
                                    <li>Város - {{ $order->shipping_city }}</li>
                                    <li>Irányítószám - {{ $order->shipping_postal_code }}</li>
                                </ul>
                            </td>
                            <td>{{ $order->product_id }}</td>
                            <td>{{ $order->quantity }}</td>
                            <td>{{ $order->total_price }}</td>
                            <td><a href="" class="btn btn-success">Jóváhagyás most</a></td>
                        </tr>
                    @endforeach
            </div>
            </table>
        </div>
```

4. Publikus felület bemutatása:

Ha a felhasználó megnyitja a weboldalt, a publikus oldal fog megjelenni. A publikus felület 3 nagyobb részre bontható.

1. Fejléc.
2. Összes termék.
3. Beúszó menü a baloldalon.
4. Cégadatok.

1. Fejléc:
   Navigációs menü: A felső fekete hátterű navigációs mező hat elemet tartalmaz. A fejléc elemeit csak bejelentkezés után lehet használni.
   A 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.

2. Összes termék:
   A fő oldal tartalma az összes termék megjelenítésére szolgál nem kategorizált sorrendben.
   Lehetőségünk van a termék azonnali vásárlására vagy megtekintésére is.

3. Bal oldali beúszó menü:
   Tartalmazza az oldalon látható termékek kategóriáit és egy főoldal gombot, ez beúszó menü végig aktívan a képernyő bal oldalán marad.

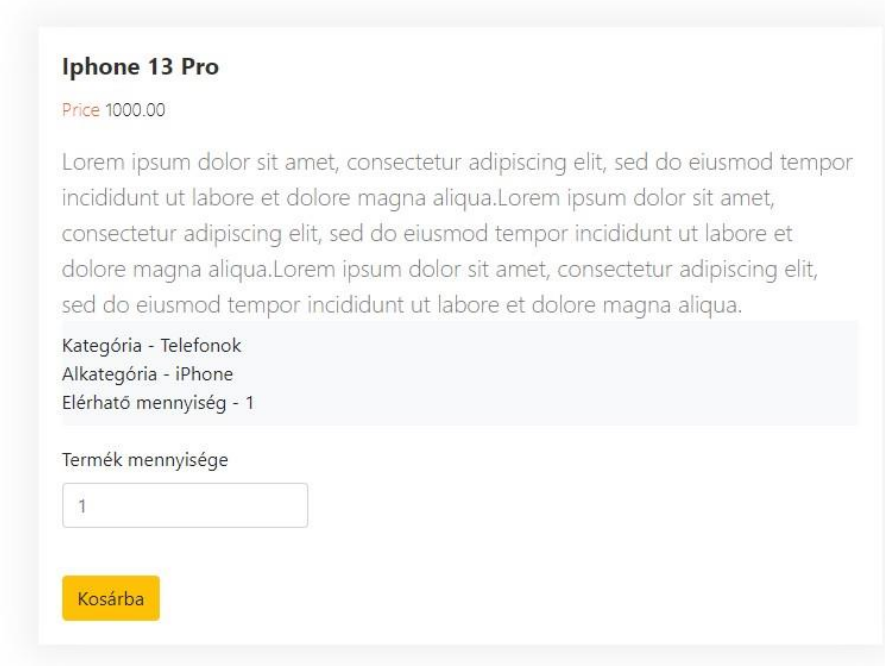
4. Cégadatok:
   A lábjegyzék az aktuális cég nevéből adódik majd minden adatával egyben.

Adminisztrációs felület bemutatása:

Az adminisztrációs felületet, csak hitelesítéssel bejelentkezett felhasználó éri el. Regisztrálni a fejléc jobb oldalán lehet, ha a felhasználó még nem rendelkezik profillal az oldalon.

Amennyiben sikeres a beléptetés, a beléptető rendszer a vezérlőpult (dashboard) oldalára navigál minket, ahonnan a regisztrált felhasználó képes lesz a termék azonnali vásárlására vagy megtekintésére.

Ha a megtekintés gombot használjuk, ami a termék mező jobb alsó sarkában található kapunk egy rövid és egy hosszú leírást a termékről, a termék áráról, kategóriájáról információt a termék elérhető mennyiségéről.



Továbbá rendelkezik egy számlálóval, ahol a kívánt termék mennyiségét meg tudjuk határozni, és ezt követően a kosárba helyezni.

A kosár bemutatása:

Ha sikeres volt a választás egy üzenet fog megjelenni hogy „a terméket hozzá adtuk a korarához” .

Továbbiakban látható lesz majd a választott termék képe, neve, mennyisége, illetve értéke.

Ez mellett található még kettő gomb, amely egy „vissza” és „kosár” néven vannak elhelyezve a termékkel egy sávban és a lap alján. Itt lehetőségünk van a kosárban lévő termék törlésére is.

Adja meg szállítási adatait.

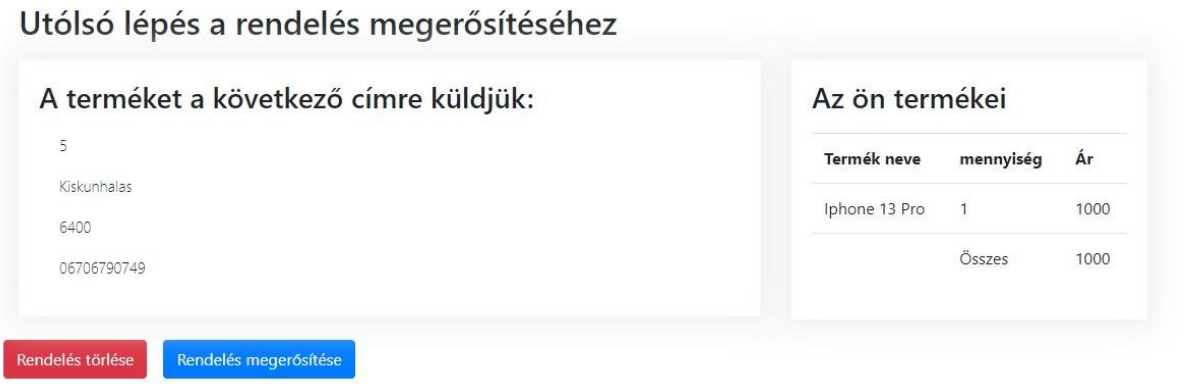
Ezen az oldalon kell kitölteni egy űrlapot, amin a telefonszám, város és irányítószám található ez a rendeléshez szükséges alap adatokkal szolgál.

Rendelkezik még egy „Következő” gombbal, ami tovább irányít az utolsó lépéshez a rendelés megerősítéséhez.

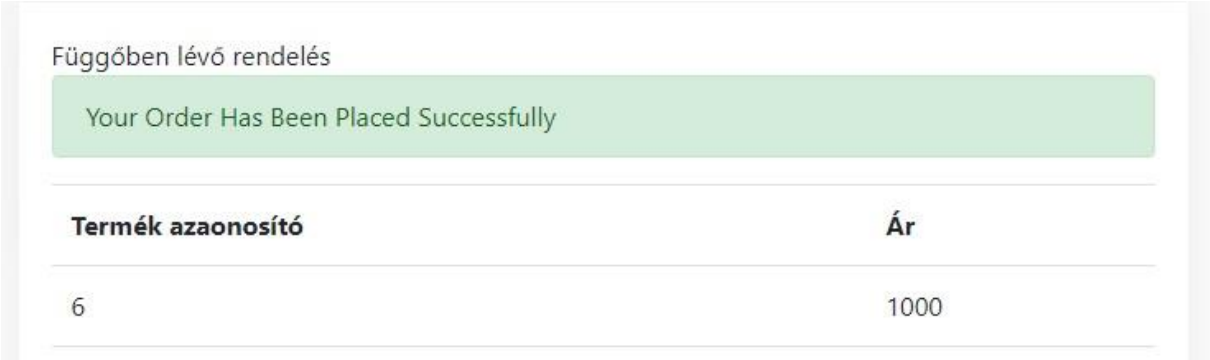
Az utolsó lépés a rendeléshez.

Itt található a rendelt tárgyak mennyisége és értéke részletesen, valamint a termék szállítási címére vonatkozó adatok.

Ez az oldal is rendelkezik kettő darab gombbal, ami lehetőséget ad a rendelés törlésére vagy a megerősítésére.



Ha mindennel elégedettek voltunk és a rendelés megerősítése mellett vagyunk, akkor ezt követően a user profil/ függőben lévő rendelések oldalra irányt minket, ahol látható lesz az általunk választott termék azonosítója valamint értéke.



5.Összefoglalás

Projektem elsődleges célja az volt, hogy rendelkezzünk a családi vállalkozásban lévő baba-mama bolthoz tartozó saját fejlesztésű webes felülettel és adatbázissal, amit a későbbiekben szeretnénk majd publikálni, de addig még számtalan dolgot szeretnék tovább fejleszteni, mind az alkalmazásban, mind a küllemében. Ezen project fejlesztése során rengeteg új ismerettel és tapasztalattal gazdagodtam, ezt a munkát szeretném kamatoztatni. Célom ismereteim még magasabb szintre emelése, hogy bonyolultabb rendszerek megírására is képes legyek. A laravel felépítése elsőre bonyolultnak tűnt, de egyre jobban elmélyedve benne megértettem a működésének logikáját, ennek eredményeképpen sikerült megvalósítani ezt a projectet.

Valamint köszönöm a Ruander oktatási központnak, hogy lehetővé tették, hogy részt tudjak venni ezen az oktatáson és megszerezzem ezt a tudást, valamint szeretnék köszönetet mondani oktatóimnak is Nagy Ferencnek és Blahut Lórántnak, hogy átadták szakértelmüket. Továbbá köszönöm családomnak és barátaimnak, hogy támogattak és türelmesek voltak velem a képzés ideje alatt.