Hardver Webshop

Vizsgaremek Dokumentáció

2023-2024

Fejlesztők

Bodnár Bence

Frontend

és

Paulusz Zsombor

Backend

https://github.com/egonixaimgod/Vizsgaremek_2024_bbence_pzsombor

Tartalomjegyzék

Felhasználói Dokumentáció

Projekt bemutatása

A projektünk egy számítógépes Webshop-ról szól, ahol lehet rendelni, különböző számítógép hardvereket. Ilyen például egy számítógép processzor, alaplap, memória, videókártya, esetleg egy számítógép tápegység, vagy ház. Jelenleg a webshopunkhoz 6 terméket adtunk hozzá, 2 alaplapot, 2 memória modult, illetve 2 processzort bemutatás céljából, de az adatbázisunkat fel lehet tölteni bármennyi termékkel. Ezen kívül van saját regisztrációs, illetve bejelentkező felületünk, ahol a felhasználók regisztrálni tudnak, és miután hitelesítettük adataikat, be is tudnak jelentkezni, így a megrendelt termékeik össze vannak kapcsolva a bejelentkezett e-mail címükkel, illetve felhasználónevükkel, így látjuk hogy egy adott felhasználó mit rendelt.

Telepítés

Programunk futtatásához Angular-t, Composer-t, Node-js-t, Git-et szükséges telepíteni, ezt megtehetjük az alábbi webolokról:

https://nodejs.org/en/download

https://git-scm.com/downloads

https://getcomposer.org/download/

Ezt követően az oldal github linkjéről egyszerően le lehet clone-ozni a projektünket, ezt 2 paranccsal tehetjük meg:

git clone https://github.com/egonixaimgod/Vizsgaremek_2024_bbence_pzsombor.git
git clone –b master https://github.com/egonixaimgod/Vizsgaremek_2024_bbence_pzsombor.git

Használat

Programunk használatához parancssort kell használni. A parancssoron belül navigáljunk el a program mappájába, 2 külön mappa lesz a 2 klónozott github command után, először az első mappába navigálva a parancssorunkba ki kell adnunk egy parancsot, ami a `npm i -g @angular/cli@16.2.12` ezzel telepítjük az angular CLI-t. Ezután kiadjuk az `npm i` parancsot, utána az `ng serve –o` parancsot. Így már az alapértelmezett böngészőben el is indul a projekt. Ezután belépünk a backend-nek a mappájába, itt a következő parancsokat adjuk ki egymás után:

composer i

copy .env.example .env

php artisan migrate:fresh --seed

php artisan key:generate

php artisan serve

Ezt követően működik a webshop.

Fejlesztői Dokumentáció

Felhasznált technológiák

Keretrendszer

Frontend:

- Angular 16.2.12

Backend:

- Laravel 10.46.0

Programnyelv

Frontend:

- HTML
- CSS
- Typescript

Backend:

- PHP

`Ingyenes/fizetős technológiák`

Frontend:

- Robot-framework: Oldal funkcióinak, működésének tesztelése

Backend:

- Insomnia, Postman: API Végpontok tesztelése

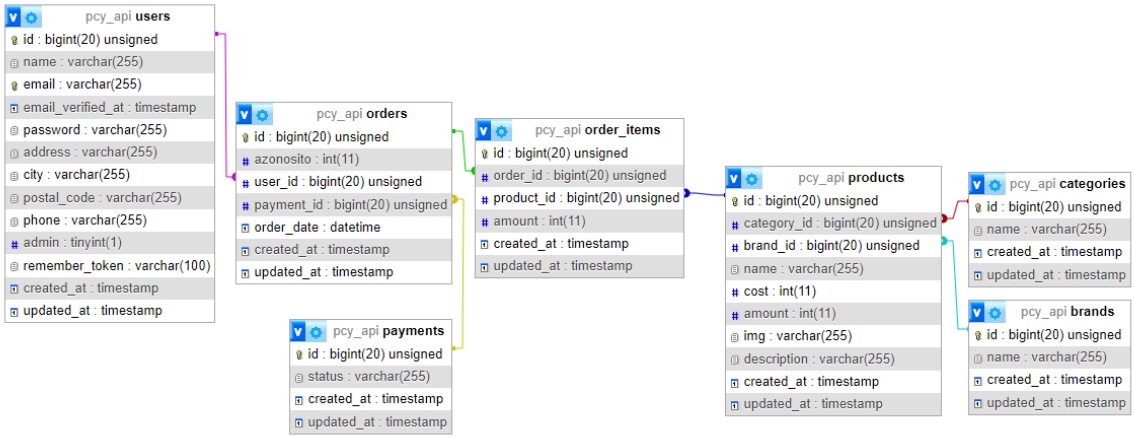
`Ismert hibák:`

A search-bar nem működik megfelelően.

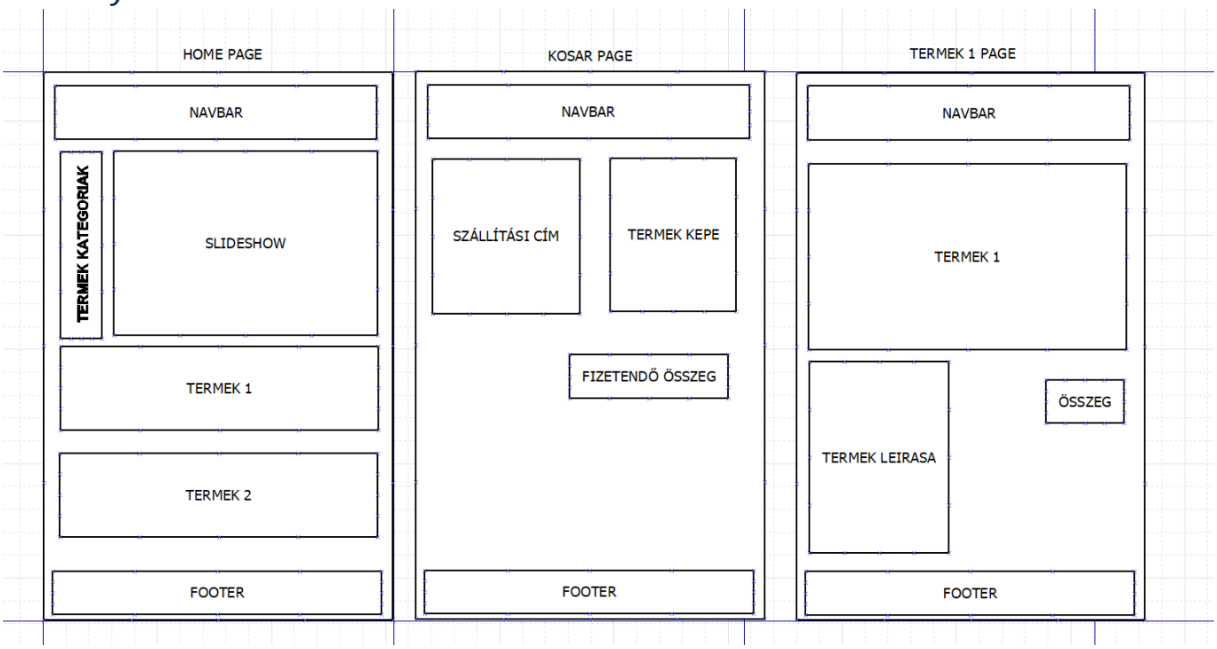
`Fejlesztői lehetőségek:`

Tervek

Adatbázis terv



Látványterv



Program leírása

Backend

Migrációs Fájlok

User Tábla Migráció

Ez a migrációs fájl felelős a `users` tábla létrehozásáért az adatbázisban. A `users` tábla a felhasználók adatait tárolja, és fontos szerepet tölt be a webshop felhasználói rendszerében.

Migráció Fájl

A migrációs fájl neve: `YYYY_MM_DD_HHMMSS_create_users_table.php`

Adatmezők

- **id**: Egyedi azonosító, automatikusan növekszik.
- **name**: A felhasználó neve.
- **email**: A felhasználó e-mail címe, egyedi az adatbázisban.
- **email_verified_at**: A felhasználó e-mail címének megerősítésének ideje, opcionális.
- **password**: A felhasználó jelszava.
- **address**: A felhasználó címe.
- **city**: A felhasználó városa.
- **postal_code**: A felhasználó irányítószáma.
- **phone**: A felhasználó telefonszáma.
- **admin**: Az adminisztrátori jogosultságokat jelzi (true/false).
- **remember_token**: Egy token a felhasználó emlékezetének megőrzésére.
- **created_at**: A rekord létrehozásának időpontja.
- **updated_at**: A rekord frissítésének időpontja.

Létrehozás és Visszavonás

- **Létrehozás**: A `Schema::create('users', function (Blueprint $table)` függvény hozza létre a `users` táblát és definiálja a fenti adatmezőket.
- **Visszavonás**: A `Schema::dropIfExists('users')` függvény törli a `users` táblát az adatbázisból.
- **Létrehozás**: A `Schema::create('users', function (Blueprint $table)` függvény hozza létre a `users` táblát és definiálja a fenti adatmezőket.
- **Visszavonás**: A `Schema::dropIfExists('users')` függvény törli a `users` táblát az adatbázisból.

Brands Tábla Migráció

Ez a migrációs fájl felelős a `brands` tábla létrehozásáért az adatbázisban. A `brands` tábla a termékek márkáit tárolja, és fontos szerepet tölt be a webshop termékkezelő rendszerében.

Migráció Fájl

A migrációs fájl neve: `YYYY_MM_DD_HHMMSS_create_brands_table.php`

Adatmezők

- **id**: Egyedi azonosító, automatikusan növekszik.
- **name**: A márka neve.
- **created_at**: A rekord létrehozásának időpontja.
- **updated_at**: A rekord frissítésének időpontja.

Létrehozás és Visszavonás

- **Létrehozás**: A `Schema::create('brands', function (Blueprint $table)` függvény hozza létre a `brands` táblát és definiálja a fenti adatmezőket.
- **Visszavonás**: A `Schema::dropIfExists('brands')` függvény törli a `brands` táblát az adatbázisból.

Categories Tábla Migráció

Ez a migrációs fájl felelős a `categories` tábla létrehozásáért az adatbázisban. A `categories` tábla a termékek kategóriáit tárolja, és segít rendezni és csoportosítani a webshopban kínált termékeket.

Migráció Fájl

A migrációs fájl neve: `YYYY_MM_DD_HHMMSS_create_categories_table.php`

Adatmezők

- **id**: Egyedi azonosító, automatikusan növekszik.
- **name**: A kategória neve.
- **created_at**: A rekord létrehozásának időpontja.
- **updated_at**: A rekord frissítésének időpontja.

Létrehozás és Visszavonás

- **Létrehozás**: A `Schema::create('categories', function (Blueprint $table)` függvény hozza létre a `categories` táblát és definiálja a fenti adatmezőket.
- **Visszavonás**: A `Schema::dropIfExists('categories')` függvény törli a `categories` táblát az adatbázisból.

Payments Tábla Migráció

Ez a migrációs fájl felelős a `payments` tábla létrehozásáért az adatbázisban. A `payments` tábla a vásárlások fizetési adatait tárolja, és segíti nyomon követni a fizetési tranzakciók állapotát a webshopban.

Migráció Fájl

A migrációs fájl neve: `YYYY_MM_DD_HHMMSS_create_payments_table.php`

Adatmezők

- **id**: Egyedi azonosító, automatikusan növekszik.
- **status**: A fizetési állapot, lehet "Billed" (számlázott) vagy "Paid" (kifizetett).
- **created_at**: A rekord létrehozásának időpontja.
- **updated_at**: A rekord frissítésének időpontja.

Létrehozás és Visszavonás

- **Létrehozás**: A `Schema::create('payments', function (Blueprint $table)` függvény hozza létre a `payments` táblát és definiálja a fenti adatmezőket.
- **Visszavonás**: A `Schema::dropIfExists('payments')` függvény törli a `payments` táblát az adatbázisból.

Orders Tábla Migráció

Ez a migrációs fájl felelős az `orders` tábla létrehozásáért az adatbázisban. Az `orders` tábla rögzíti a felhasználók rendeléseit a webshopban.

Migráció Fájl

A migrációs fájl neve: `YYYY_MM_DD_HHMMSS_create_orders_table.php`

Adatmezők

- **id**: Egyedi azonosító, automatikusan növekszik.
- **azonosito**: A rendelés egyedi azonosítója.
- **user_id**: A rendelést leadó felhasználó azonosítója, idegen kulcs kapcsolat a `users` táblához.
- **payment_id**: A fizetés azonosítója, idegen kulcs kapcsolat a `payments` táblához.
- **order_date**: A rendelés dátuma.
- **created_at**: A rekord létrehozásának időpontja.
- **updated_at**: A rekord frissítésének időpontja.

Idegen Kulcsok

- **user_id**: Az `orders` tábla `user_id` mezője idegen kulcs kapcsolatot tart fenn a `users` táblához. Ha egy felhasználó törlődik, a rendeléshez kapcsolódó rekordok is törlődnek (CASCADE).
- **payment_id**: Az `orders` tábla `payment_id` mezője idegen kulcs kapcsolatot tart fenn a `payments` táblához. Ha egy fizetés törlődik, a rendeléshez kapcsolódó rekordok is törlődnek (CASCADE).

Létrehozás és Visszavonás

- **Létrehozás**: A `Schema::create('orders', function (Blueprint $table)` függvény hozza létre az `orders` táblát és definiálja a fenti adatmezőket és idegen kulcsokat.
- **Visszavonás**: A `Schema::dropIfExists('orders')` függvény törli az `orders` táblát az adatbázisból.

Order_Items Tábla Migráció

Ez a migrációs fájl felelős az `order_items` tábla létrehozásáért az adatbázisban. Az `order_items` tábla rögzíti az egyes rendelésekhez tartozó termékek részleteit, mint például a rendelt termék, annak mennyisége és az adott rendeléshez való kapcsolat.

Migráció Fájl

A migrációs fájl neve: `YYYY_MM_DD_HHMMSS_create_order_items_table.php`

Adatmezők

- **id**: Egyedi azonosító, automatikusan növekszik.
- **order_id**: A rendelés azonosítója, idegen kulcs kapcsolat a `orders` táblához.
- **product_id**: A termék azonosítója, idegen kulcs kapcsolat a `products` táblához.
- **amount**: A rendelt termék mennyisége.

- **`created_at`**: A rekord létrehozásának időpontja.
- **`updated_at`**: A rekord frissítésének időpontja.

Idegen Kulcsok

- **`order_id`**: Az **`order_items`** tábla **`order_id`** mezője idegen kulcs kapcsolatot tart fenn a **`orders`** táblához. Ha egy rendelés törlődik, az ehhez kapcsolódó rekordok is törlődnek (CASCADE).
- **`product_id`**: Az **`order_items`** tábla **`product_id`** mezője idegen kulcs kapcsolatot tart fenn a **`products`** táblához. Ha egy termék törlődik, az ehhez kapcsolódó rekordok is törlődnek (CASCADE).

Létrehozás és Visszavonás

- **Létrehozás**: A **`Schema::create('order_items', function (Blueprint $table)`** függvény hozza létre az **`order_items`** táblát és definiálja a fenti adatmezőket és idegen kulcsokat.
- **Visszavonás**: A **`Schema::dropIfExists('order_items')`** függvény törli az **`order_items`** táblát az adatbázisból.

Products Tábla Migráció

Ez a migrációs fájl felelős a `products` tábla létrehozásáért az adatbázisban. A `products` tábla rögzíti a webshop árucikkeinek adatait, mint például a termék neve, ára, készlete és egyéb tulajdonságok.

Migráció Fájl

A migrációs fájl neve: `YYYY_MM_DD_HHMMSS_create_products_table.php`

Adatmezők

- **id**: Egyedi azonosító, automatikusan növekszik.
- **category_id**: A termék kategória azonosítója, idegen kulcs kapcsolat a `categories` táblához.
- **brand_id**: A termék márka azonosítója, idegen kulcs kapcsolat a `brands` táblához.
- **name**: A termék neve.
- **cost**: A termék ára.
- **amount**: A termék készlete.
- **img**: A termék képének elérési útvonala.
- **description**: A termék leírása (opcionális).
- **created_at**: A rekord létrehozásának időpontja.
- **updated_at**: A rekord frissítésének időpontja.

Idegen Kulcsok

- **category_id**: A `products` tábla `category_id` mezője idegen kulcs kapcsolatot tart fenn a `categories` táblához. Ha egy kategória törlődik, az ehhez kapcsolódó rekordok is törlődnek (CASCADE).
- **brand_id**: A `products` tábla `brand_id` mezője idegen kulcs kapcsolatot tart fenn a `brands` táblához. Ha egy márka törlődik, az ehhez kapcsolódó rekordok is törlődnek (CASCADE).

Létrehozás és Visszavonás

- **Létrehozás**: A `Schema::create('products', function (Blueprint $table)` függvény hozza létre a `products` táblát és definiálja a fenti adatmezőket és idegen kulcsokat.
- **Visszavonás**: A `Schema::dropIfExists('products')` függvény törli a `products` táblát az adatbázisból.

Modelek

User Model

A `User` modell felelős a felhasználói adatok kezeléséért és az azokhoz tartozó műveletek végrehajtásáért.

Relációk

- **orders()**: A felhasználóhoz tartozó rendelések lekérését teszi lehetővé.

Töltött Mezők

- **name**: A felhasználó neve.
- **email**: A felhasználó e-mail címe.
- **password**: A felhasználó jelszava.
- **address**: A felhasználó címe.
- **county**: A felhasználó megyéje (opcionális).
- **city**: A felhasználó városa.
- **postal_code**: A felhasználó irányítószáma.
- **phone**: A felhasználó telefonszáma.
- **admin**: A felhasználó adminisztrátori jogosultságát jelzi (true/false).

Rejtett Mezők

- **password**: A jelszó mező rejtett a szerializáció során.
- **remember_token**: A "remember me" token rejtett a szerializáció során.

Típusátalakítások

- **email_verified_at**: Az e-mail megerősítés dátumára `datetime` típusú átalakítás.
- **password**: A jelszóra `hashed` típusú átalakítás.

Töltött és Rejtett Mezők Beállítása

A `$fillable` és `$hidden` tulajdonságok definiálják, hogy mely mezők töltődhetnek és melyek rejtettek a szerializáció során.

Laravel Sanctum API Tokenek

A `HasApiTokens` trait segítségével lehetőség van API tokenek kezelésére és generálására a felhasználóhoz.

Laravel Eloquent Has Factory

A `HasFactory` trait használatával lehetőség van gyártási modellek definiálására.

Laravel Eloquent Notifiable

A `Notifiable` trait lehetővé teszi az értesítések küldését a felhasználóhoz.

Brands Model

A `Brands` modell felelős a márkák kezeléséért és az azokhoz tartozó műveletek végrehajtásáért.

Töltött Mezők

- **`name`**: A márka neve.

Típusátalakítások

Nincsenek specifikus típusátalakítások definiálva.

Töltött Mezők Beállítása

A `$fillable` tulajdonság definiálja, hogy mely mezők töltődhetnek fel a modellekben.

Laravel Eloquent Has Factory

A `HasFactory` trait segítségével lehetőség van gyártási modellek definiálására.

Categories Modell

A `Categories` modell felelős a kategóriák kezeléséért és az azokhoz tartozó műveletek végrehajtásáért.

Töltött Mezők

- **`name`**: A kategória neve.

Típusátalakítások

Nincsenek specifikus típusátalakítások definiálva.

Töltött Mezők Beállítása

A `$fillable` tulajdonság definiálja, hogy mely mezők töltődhetnek fel a modellekben.

Laravel Eloquent Has Factory

A `HasFactory` trait segítségével lehetőség van gyártási modellek definiálására.

Payment Modell

A **`Payment`** modell felelős a fizetések kezeléséért és az azokhoz tartozó műveletek végrehajtásáért.

Töltött Mezők

- **`status`**: A fizetés állapota, lehet "Billed" (számlázott) vagy "Paid" (kifizetett).

Típusátalakítások

Nincsenek specifikus típusátalakítások definiálva.

Töltött Mezők Beállítása

A **`$fillable`** tulajdonság definiálja, hogy mely mezők töltődhetnek fel a modellekben.

Laravel Eloquent Has Factory

A **`HasFactory`** trait segítségével lehetőség van gyártási modellek definiálására.

Orders Modell

Az `Orders` modell felelős a rendelések kezeléséért és az azokhoz tartozó műveletek végrehajtásáért.

Relációk

- **`orderItems()`**: A rendeléshez tartozó tételek lekérését teszi lehetővé.

Töltött Mezők

- **`azonosito`**: A rendelés egyedi azonosítója.
- **`user_id`**: A rendelést leadó felhasználó azonosítója.
- **`payment_id`**: A fizetés azonosítója.
- **`order_date`**: A rendelés dátuma.

Típusátalakítások

Nincsenek specifikus típusátalakítások definiálva.

Töltött Mezők Beállítása

A `$fillable` tulajdonság definiálja, hogy mely mezők töltődhetnek fel a modellekben.

Laravel Eloquent Has Factory

A `HasFactory` trait segítségével lehetőség van gyártási modellek definiálására.

Laravel Eloquent HasMany Reláció

A `hasMany` metódus segítségével lehetőség van a `Orders` modellek kapcsolódására más modellekhez, mint például az `OrderItems` modellekhez.

OrderItems Modell

Az `OrderItems` modell felelős a rendelés tételeinek kezeléséért és az azokhoz tartozó műveletek végrehajtásáért.

Relációk

- **`product()`**: A rendelés tételhez tartozó termék lekérését teszi lehetővé.

Töltött Mezők

- **`order_id`**: A rendelés azonosítója.
- **`product_id`**: A termék azonosítója.
- **`amount`**: A termék mennyisége a rendelésben.

Típusátalakítások

Nincsenek specifikus típusátalakítások definiálva.

Töltött Mezők Beállítása

A `$fillable` tulajdonság definiálja, hogy mely mezők töltődhetnek fel a modellekben.

Laravel Eloquent Has Factory

A `HasFactory` trait segítségével lehetőség van gyártási modellek definiálására.

Laravel Eloquent BelongsTo Reláció

A `belongsTo` metódus segítségével lehetőség van az `OrderItems` modellek kapcsolódására más modellekhez, mint például a `Products` modellekhez.

Products Modell

A `Products` modell felelős a termékek kezeléséért és az azokhoz tartozó műveletek végrehajtásáért.

Relációk

- **`category()`**: A termékhez tartozó kategória lekérését teszi lehetővé.
- **`brand()`**: A termékhez tartozó márka lekérését teszi lehetővé.

Töltött Mezők

- **`category_id`**: A termékhez tartozó kategória azonosítója.
- **`brand_id`**: A termékhez tartozó márka azonosítója.
- **`name`**: A termék neve.
- **`cost`**: A termék ára.
- **`amount`**: A termék készlete.
- **`img`**: A termék képének elérési útvonala.
- **`description`**: A termék leírása.

Típusátalakítások

Nincsenek specifikus típusátalakítások definiálva.

Töltött Mezők Beállítása

A `$fillable` tulajdonság definiálja, hogy mely mezők töltődhetnek fel a modellekben.

Laravel Eloquent Has Factory

A `HasFactory` trait segítségével lehetőség van gyártási modellek definiálására.

Laravel Eloquent BelongsTo Relációk

A `belongsTo` metódusok segítségével lehetőség van a `Products` modellek kapcsolódására más modellekhez, mint például a kategóriákhoz és a márkákhoz.

Controllerek

UsersController

A `UsersController` osztály felelős az ügyfélkapcsolati funkciók kezeléséért és az azokhoz tartozó műveletek végrehajtásáért.

Metódusok

index()

- **Leírás**: Az összes felhasználó lekérdezése.
- **Visszatérési érték**: JSON formátumban visszaadja az összes felhasználót.

show($id)

- **Leírás**: Egy adott felhasználó lekérdezése az azonosító alapján.
- **Paraméterek**: `$id` - a felhasználó azonosítója.
- **Visszatérési érték**: JSON formátumban visszaadja a keresett felhasználót, hiba esetén 404-es státuszkóddal.

update(Request $request, $id)

- **Leírás**: Egy adott felhasználó adatainak frissítése.
- **Paraméterek**: `$request` - a HTTP kérés objektuma, `$id` - a felhasználó azonosítója.
- **Bemeneti adatok ellenőrzése**: Az új adatokat a megadott validációs szabályoknak megfelelően ellenőrzi.
- **Visszatérési érték**: JSON formátumban visszaadja a frissített felhasználó adatait, HTTP 200-as státuszkóddal.

destroy($id)

- **Leírás**: Egy adott felhasználó törlése.
- **Paraméterek**: `$id` - a felhasználó azonosítója.
- **Művelet**: Megkeresi és törli a megadott felhasználót az azonosító alapján.
- **Visszatérési érték**: NULL-t ad vissza, HTTP 204-es státuszkóddal.

Validáció

A `update()` metódusban az új adatokat a Laravel beépített validációs rendszerével ellenőrzi a megadott szabályoknak megfelelően.

BrandsController

A `BrandsController` osztály felelős a márkák kezeléséért és az azokhoz tartozó műveletek végrehajtásáért.

Metódusok

index()

- **Leírás**: Az összes márka lekérdezése.
- **Visszatérési érték**: JSON formátumban visszaadja az összes márkát.

create()

- **Leírás**: Új márka létrehozásához tartozó űrlap megjelenítése.
- **Nincs megvalósítva**: Az űrlap megjelenítése nincs implementálva ebben a kontrollerben.

store(Request $request)

- **Leírás**: Új márka létrehozása.
- **Paraméterek**: `$request` - a HTTP kérés objektuma.
- **Visszatérési érték**: JSON formátumban visszaadja az új márkát, HTTP 201-es státuszkóddal.

show($id)

- **Leírás**: Egy adott márka lekérdezése az azonosító alapján.
- **Paraméterek**: `$id` - a márka azonosítója.
- **Visszatérési érték**: JSON formátumban visszaadja a keresett márkát, hiba esetén 404-es státuszkóddal.

edit(Brands $brands)

- **Leírás**: Egy márka szerkesztési űrlapjának megjelenítése.
- **Nincs megvalósítva**: Az űrlap megjelenítése nincs implementálva ebben a kontrollerben.

update(Request $request, $id)

- **Leírás**: Egy adott márka adatainak frissítése.
- **Paraméterek**: `$request` - a HTTP kérés objektuma, `$id` - a márka azonosítója.
- **Bemeneti adatok ellenőrzése**: Az új adatokat a megadott validációs szabályoknak megfelelően ellenőrzi.
- **Visszatérési érték**: JSON formátumban visszaadja a frissített márkát, HTTP 200-as státuszkóddal.

destroy($id)

- **Leírás**: Egy adott márka törlése.
- **Paraméterek**: `$id` - a márka azonosítója.
- **Művelet**: Megkeresi és törli a megadott márkát az azonosító alapján.
- **Visszatérési érték**: NULL-t ad vissza, HTTP 204-es státuszkóddal.

Validáció

A `update()` metódusban az új adatokat a Laravel beépített validációs rendszerével ellenőrzi a megadott szabályoknak megfelelően.

CategoriesController

A `CategoriesController` osztály felelős a kategóriák kezeléséért és az azokhoz tartozó műveletek végrehajtásáért.

Metódusok

index()

- **Leírás**: Az összes kategória lekérdezése.
- **Visszatérési érték**: JSON formátumban visszaadja az összes kategóriát.

store(Request $request)

- **Leírás**: Új kategória létrehozása.
- **Paraméterek**: `$request` - a HTTP kérés objektuma.
- **Visszatérési érték**: JSON formátumban visszaadja az új kategóriát, HTTP 201-es státuszkóddal.

show($id)

- **Leírás**: Egy adott kategória lekérdezése az azonosító alapján.
- **Paraméterek**: `$id` - a kategória azonosítója.
- **Visszatérési érték**: JSON formátumban visszaadja a keresett kategóriát, hiba esetén 404-es státuszkóddal.

update(Request $request, $id)

- **Leírás**: Egy adott kategória adatainak frissítése.
- **Paraméterek**: `$request` - a HTTP kérés objektuma, `$id` - a kategória azonosítója.
- **Bemeneti adatok ellenőrzése**: Az új adatokat a megadott validációs szabályoknak megfelelően ellenőrzi.
- **Visszatérési érték**: JSON formátumban visszaadja a frissített kategóriát, HTTP 200-as státuszkóddal.

destroy($id)

- **Leírás**: Egy adott kategória törlése.
- **Paraméterek**: `$id` - a kategória azonosítója.
- **Művelet**: Megkeresi és törli a megadott kategóriát az azonosító alapján.
- **Visszatérési érték**: NULL-t ad vissza, HTTP 204-es státuszkóddal.

Validáció

A `update()` metódusban az új adatokat a Laravel beépített validációs rendszerével ellenőrzi a megadott szabályoknak megfelelően.

PaymentController

A `PaymentController` osztály felelős a fizetési módok kezeléséért és az azokhoz tartozó műveletek végrehajtásáért.

Metódusok

index()

- **Leírás**: Az összes fizetési mód lekérdezése.
- **Visszatérési érték**: JSON formátumban visszaadja az összes fizetési módot.

show($id)

- **Leírás**: Egy adott fizetési mód lekérdezése az azonosító alapján.
- **Paraméterek**: `$id` - a fizetési mód azonosítója.
- **Visszatérési érték**: JSON formátumban visszaadja a keresett fizetési módot, hiba esetén 404-es státuszkóddal.

Validáció

A `PaymentController` osztályban nincs validáció, mivel csak lekérdezéseket hajt végre.

OrdersController

A `OrdersController` osztály felelős a rendelések kezeléséért és az azokhoz tartozó műveletek végrehajtásáért.

Metódusok

index()

- **Leírás**: Az összes rendelés lekérdezése.
- **Visszatérési érték**: JSON formátumban visszaadja az összes rendelést.

placeOrder(Request $request)

- **Leírás**: Új rendelés felvétele.
- **Paraméterek**: `$request` - a HTTP kérés objektuma.
- **Bemeneti adatok ellenőrzése**: Ellenőrzi a beérkező adatokat, hogy megfelelnek-e a validációs szabályoknak.
- **Lépések**:

1. Validálja a beérkező adatokat.
2. Létrehozza az új rendelést.
3. Létrehozza a rendeléshez tartozó tétel(ek)et.

- **Visszatérési érték**: JSON formátumban visszaadja az új rendelést, HTTP 201-es státuszkóddal.

showOrders($perPage = 10)

- **Leírás**: A bejelentkezett felhasználó rendeléseinek lekérdezése.
- **Paraméterek**: `$perPage` - oldalankénti elemek száma (opcionális, alapértelmezett érték: 10).
- **Visszatérési érték**: JSON formátumban visszaadja a bejelentkezett felhasználó rendeléseit.

updateOrder(Request $request, $order_id)

- **Leírás**: Egy rendelés frissítése.
- **Paraméterek**: `$request` - a HTTP kérés objektuma, `$order_id` - a frissítendő rendelés azonosítója.
- **Bemeneti adatok ellenőrzése**: Ellenőrzi a beérkező adatokat, hogy megfelelnek-e a validációs szabályoknak.
- **Lépések**:

1. Validálja a beérkező adatokat.
2. Frissíti a rendelés adatait, ha szükséges.
3. Frissíti a rendeléshez tartozó tételek(ek)et, ha szükséges.

- **Visszatérési érték**: JSON formátumban visszaadja a frissített rendelést, HTTP 201-es státuszkóddal.

deleteOrder($id)

- **Leírás**: Egy rendelés törlése.
- **Paraméterek**: `$id` - a törlendő rendelés azonosítója.
- **Művelet**: Megkeresi és törli a megadott rendelést az azonosító alapján.
- **Visszatérési érték**: NULL-t ad vissza, HTTP 204-es státuszkóddal.

Validáció

- A `placeOrder()` és `updateOrder()` metódusokban az új adatokat a Laravel beépített validációs rendszerével ellenőrzi a megadott szabályoknak megfelelően.

OrderItemsController

Az `OrderItemsController` osztály felelős a rendelés tétel(ek) kezeléséért és az azokhoz tartozó műveletek végrehajtásáért.

Metódusok

index()

- **Leírás**: Az összes rendelés tétel(ek) lekérdezése.
- **Visszatérési érték**: JSON formátumban visszaadja az összes rendelés tétel(ek)et.

store(Request $request)

- **Leírás**: Új rendelés tétel(ek) felvétele.
- **Paraméterek**: `$request` - a HTTP kérés objektuma.
- **Bemeneti adatok ellenőrzése**: Ellenőrzi a beérkező adatokat, hogy megfelelnek-e a validációs szabályoknak.
- **Lépések**:

1. Validálja a beérkező adatokat.
2. Létrehozza az új rendelés tétel(ek)et.

- **Visszatérési érték**: JSON formátumban visszaadja az új rendelés tétel(ek)et, HTTP 201-es státuszkóddal.

show($id)

- **Leírás**: Egy adott rendelés tétel(ek) lekérdezése az azonosító alapján.
- **Paraméterek**: `$id` - a rendelés tétel(ek) azonosítója.
- **Visszatérési érték**: JSON formátumban visszaadja a keresett rendelés tétel(ek)et, hiba esetén 404-es státuszkóddal.

showOrderItems($order_id)

- **Leírás**: Egy adott rendeléshez tartozó tétel(ek) lekérdezése az order_id alapján.
- **Paraméterek**: `$order_id` - a rendelés azonosítója.
- **Visszatérési érték**: JSON formátumban visszaadja a keresett rendeléshez tartozó tétel(ek)et, hiba esetén 404-es státuszkóddal.

update(Request $request, $id)

- **Leírás**: Egy rendelés tétel(ek) frissítése.
- **Paraméterek**: `$request` - a HTTP kérés objektuma, `$id` - a frissítendő rendelés tétel(ek) azonosítója.
- **Bemeneti adatok ellenőrzése**: Ellenőrzi a beérkező adatokat, hogy megfelelnek-e a validációs szabályoknak.
- **Lépések**:

1. Validálja a beérkező adatokat.
2. Frissíti a rendelés tétel(ek)et.

- **Visszatérési érték**: JSON formátumban visszaadja a frissített rendelés tétel(ek)et, HTTP 200-as státuszkóddal.

destroy($id)

- **Leírás**: Egy rendelés tétel(ek) törlése.
- **Paraméterek**: `$id` - a törlendő rendelés tétel(ek) azonosítója.
- **Művelet**: Megkeresi és törli a megadott rendelés tétel(ek)et az azonosító alapján.
- **Visszatérési érték**: NULL-t ad vissza, HTTP 204-es státuszkóddal.

Validáció

- A `store()` és `update()` metódusokban az új adatokat a Laravel beépített validációs rendszerével ellenőrzi a megadott szabályoknak megfelelően.

ProductsController

Az **`ProductsController`** osztály felelős a termékek kezeléséért és az azokhoz tartozó műveletek végrehajtásáért.

Metódusok

index()

- **Leírás**: Az összes termék lekérdezése.
- **Visszatérési érték**: JSON formátumban visszaadja az összes terméket.

store(Request $request)

- **Leírás**: Új termék felvétele.
- **Paraméterek**: **`$request`** - a HTTP kérés objektuma.
- **Bemeneti adatok ellenőrzése**: Nincs külön validáció, minden bemenet elfogadásra kerül.
- **Lépések**:

1. Létrehozza az új terméket.

- **Visszatérési érték**: JSON formátumban visszaadja az új terméket, HTTP 201-es státuszkóddal.

show($id)

- **Leírás**: Egy adott termék lekérdezése az azonosító alapján.
- **Paraméterek**: **`$id`** - a termék azonosítója.
- **Visszatérési érték**: JSON formátumban visszaadja a keresett terméket, hiba esetén 404-es státuszkóddal.

update(Request $request, $id)

- **Leírás**: Egy termék frissítése.
- **Paraméterek**: **`$request`** - a HTTP kérés objektuma, **`$id`** - a frissítendő termék azonosítója.
- **Bemeneti adatok ellenőrzése**: Ellenőrzi a beérkező adatokat, hogy megfelelnek-e a validációs szabályoknak.
- **Lépések**:

1. Validálja a beérkező adatokat.
2. Frissíti a terméket.

- **Visszatérési érték**: JSON formátumban visszaadja a frissített terméket, HTTP 200-as státuszkóddal.

destroy($id)

- **Leírás**: Egy termék törlése.
- **Paraméterek**: **`$id`** - a törlendő termék azonosítója.
- **Művelet**: Megkeresi és törli a megadott terméket az azonosító alapján.
- **Visszatérési érték**: NULL-t ad vissza, HTTP 204-es státuszkóddal.

Validáció

- A **`update()`** metódusban az új adatokat a Laravel beépített validációs rendszerével ellenőrzi a megadott szabályoknak megfelelően.

AuthController

Az `AuthController` osztály felelős az autentikációs funkciók és a felhasználói profilkezelés végrehajtásáért.

Metódusok

register(Request $request)

- **Leírás**: Új felhasználó regisztrációja.
- **Paraméterek**: `$request` - a HTTP kérés objektuma.
- **Bemeneti adatok ellenőrzése**: A beérkező adatokat validálja a megadott szabályoknak megfelelően.
- **Lépések**:

1. Létrehozza az új felhasználót a beérkező adatok alapján.

- **Visszatérési érték**: JSON formátumban visszaadja az új felhasználót és egy üzenetet, hogy a regisztráció sikeres volt.

login(Request $request)

- **Leírás**: Felhasználó bejelentkeztetése.
- **Paraméterek**: `$request` - a HTTP kérés objektuma.
- **Bemeneti adatok ellenőrzése**: A beérkező adatokat validálja a megadott szabályoknak megfelelően.
- **Lépések**:

1. Ellenőrzi a felhasználónév és a jelszó párosítását.
2. Ha a bejelentkezés sikeres, JWT tokent hoz létre a felhasználó számára.

- **Visszatérési érték**: JSON formátumban visszaadja a JWT tokent és a felhasználót, ha a bejelentkezés sikeres volt.

profile(Request $request)

- **Leírás**: Az aktuális felhasználó profiljának lekérése.
- **Paraméterek**: `$request` - a HTTP kérés objektuma.
- **Működés**: Ellenőrzi, hogy a felhasználó be van-e jelentkezve. Ha igen, visszaadja az aktuális felhasználó adatait.
- **Visszatérési érték**: JSON formátumban visszaadja az aktuális felhasználó adatait.

logout(Request $request)

- **Leírás**: Felhasználó kijelentkeztetése.
- **Paraméterek**: `$request` - a HTTP kérés objektuma.
- **Működés**: Ellenőrzi, hogy a felhasználó be van-e jelentkezve. Ha igen, törli a felhasználó összes aktív tokent.
- **Visszatérési érték**: JSON formátumban visszaadja a kijelentkezés sikerét.

updateProfile(Request $request)

- **Leírás**: Felhasználó profiljának frissítése.
- **Paraméterek**: `$request` - a HTTP kérés objektuma.
- **Működés**: Ellenőrzi, hogy a felhasználó be van-e jelentkezve. Ha igen, validálja a beérkező adatokat, majd frissíti a felhasználó adatait.
- **Visszatérési érték**: JSON formátumban visszaadja a frissített felhasználó adatait.

deleteProfile(Request $request)

- **Leírás**: Felhasználó profiljának törlése.

- **Paraméterek**: `$request` - a HTTP kérés objektuma.
- **Működés**: Ellenőrzi, hogy a felhasználó be van-e jelentkezve. Ha igen, törli a felhasználó adatait.
- **Visszatérési érték**: JSON formátumban visszaadja a törlés sikerét.

Validáció

- Az `updateProfile()` metódusban a beérkező adatokat a Laravel beépített validációs rendszerével ellenőrzi a megadott szabályoknak megfelelően.

Végpontok

Admin végpontok

Az itt látható kódrészlet az adminisztrátor jogosultságokkal rendelkező felhasználók számára elérhető API végpontokat definiálja. Itt csak az adminisztrátorok hajthatnak végre műveleteket, például márkák, kategóriák, termékek, rendelési tételek és felhasználók kezelése terén.

Ezen végpontok csak akkor lesznek elérhetők, ha a felhasználó autentikálva van Sanctum segítségével és rendelkezik adminisztrátori jogosultságokkal. A `auth:sanctum` middleware biztosítja az autentikációt, míg az `admin` middleware az adminisztrátori jogosultságok ellenőrzését végzi.

- **Brands Controller:**
- `/addBrand`: Új márka hozzáadása.
- `/updateBrand/{id}`: Márka frissítése az azonosító alapján.
- `/deleteBrand/{id}`: Márka törlése az azonosító alapján.
- **Categories Controller:**
- `/addCategory`: Új kategória hozzáadása.
- `/updateCategory/{id}`: Kategória frissítése az azonosító alapján.
- `/deleteCategory/{id}`: Kategória törlése az azonosító alapján.
- **Products Controller:**
- `/products`: Új termék létrehozása.
- `/products/{id}`: Termék frissítése az azonosító alapján.
- `/products/{id}`: Termék törlése az azonosító alapján.
- **OrderItems Controller:**
- `/order_items`: Összes rendelési tétel lekérése.
- **Orders Controller:**
- `/orders`: Összes rendelés lekérése.
- **Users Controller:**
- `/users`: Összes felhasználó lekérése.
- `/users/{id}`: Egy felhasználó részletes adatainak lekérése az azonosító alapján.
- `/updateUser/{id}`: Felhasználó frissítése az azonosító alapján.
- `/deleteUser/{id}`: Felhasználó törlése az azonosító alapján.

User végpontok

Az itt látható kód egy olyan végpontcsoportot definiál, amely az autentikált felhasználók számára elérhető. Ezek a végpontok a felhasználói profil kezelését és a rendelésekkel kapcsolatos műveleteket teszik lehetővé.

Ezen végpontok csak akkor lesznek elérhetők, ha a felhasználó autentikálva van Sanctum segítségével. Az autentikációt a `auth:sanctum` middleware biztosítja.

- **Felhasználói profil kezelése:**
- `/auth/logout`: Kijelentkezés.
- `/auth/profile`: Felhasználói profil lekérése.
- `/auth/deleteProfile`: Felhasználói profil törlése.
- `/auth/updateProfile`: Felhasználói profil frissítése.
- **Rendelések kezelése:**
- `/placeOrder`: Új rendelés leadása.
- `/showOrders`: Felhasználó összes rendelésének lekérése.
- `/showOrderItems/{id}`: Egy rendeléshez tartozó összes tétel lekérése.
- `/updateOrder/{id}`: Rendelés frissítése.
- `/deleteOrder/{id}`: Rendelés törlése.
- **OrderItems Controller:**
- `/order_items`: Összes rendelési tétel lekérése.
- `/order_items/{id}`: Egy rendelési tétel részleteinek lekérése az azonosító alapján.
- `/order_items/{id}`: Rendelési tétel frissítése az azonosító alapján.
- `/order_items/{id}`: Rendelési tétel törlése az azonosító alapján.
- **Orders Controller:**
- `/orders`: Összes rendelés lekérése.
- `/orders/{id}`: Egy rendelés részleteinek lekérése az azonosító alapján.
- `/orders/{id}`: Rendelés frissítése az azonosító alapján.
- `/orders/{id}`: Rendelés törlése az azonosító alapján.
- **Users Controller:**
- `/users/{id}`: Felhasználó frissítése az azonosító alapján.
- `/users/{id}`: Felhasználó törlése az azonosító alapján.

Autentikáció nélküli végpontok

Ezek a végpontok az autentikáció nélkül elérhető funkciókat teszik elérhetővé a felhasználók számára.

Ezeket a végpontokat az autentikáció nélkül elérhető szolgáltatásokhoz, például a termékek, kategóriák vagy márkák megtekintéséhez használhatják a felhasználók. Az autentikációs middleware-t nem alkalmazzák ezeken a végpontokon, így a felhasználók szabadon hozzáférhetnek hozzájuk az autentikáció nélkül is.

- **Authentikáció:**
- `/auth/register`: Regisztráció.
- `/auth/login`: Bejelentkezés.
- **Brands Controller:**
- `/brands`: Összes márka lekérése.
- `/brands/{id}`: Egy márka részleteinek lekérése az azonosító alapján.
- **Categories Controller:**
- `/categories`: Összes kategória lekérése.
- `/categories/{id}`: Egy kategória részleteinek lekérése az azonosító alapján.
- **Payment Controller:**
- `/payment`: Összes fizetési mód lekérése.
- `/payment/{id}`: Egy fizetési mód részleteinek lekérése az azonosító alapján.
- **Products Controller:**
- `/products`: Összes termék lekérése.
- `/products/{id}`: Egy termék részleteinek lekérése az azonosító alapján.

Middleware-k

IsAdmin Middleware

Az `IsAdmin` middleware felelős az adminisztrátori jogosultságok ellenőrzéséért egy bejövő kérés esetén. Ennek a middleware-nek a használata során ellenőrizzük, hogy a felhasználó rendelkezik-e adminisztrátori jogosultsággal. Ha igen, engedélyezzük a kérés feldolgozását, különben pedig visszautasítjuk a hozzáférést.

Működés

Az `IsAdmin` middleware a következőképpen működik:

- Ha a bejelentkezett felhasználó rendelkezik adminisztrátori jogosultsággal, akkor a kérés folytatódik és a hozzáférés engedélyezett.
- Ha a felhasználó nem rendelkezik adminisztrátori jogosultsággal, akkor egy 403-as státuszkóddal ellátott választ kap, ami azt jelzi, hogy a felhasználó nem rendelkezik megfelelő jogosultságokkal.

Ez a middleware hasznos biztonsági réteget képezhet egy alkalmazásban, amely korlátozza a kritikus műveletek végrehajtását csak az engedélyezett felhasználók számára.

Használat

Az `IsAdmin` middleware-t úgy használhatjuk, hogy regisztráljuk az alkalmazásunk middleware csoportjába, majd hozzárendeljük a megfelelő útvonalakhoz vagy kontrollerekhez.

Példa a middleware regisztrálására a `app/Http/Kernel.php` fájlban:

```
protected $routeMiddleware = [
        // …
        ’admin’ => \App\Http\Middleware\IsAdmin::class,
];
```

Ezután az adminisztrátori jogosultságot ellenőrizhetjük az útvonalaknál vagy a kontrollereknél a `admin` kulcsszó használatával:

```
Route::middleware(’admin’)->get(’admin/dashboard’, function () {
        // ...
});
```

Ez biztosítja, hogy csak az adminisztrátorok férjenek hozzá az adott útvonalakhoz vagy kontrollerekhez.

Seeder-ek

BrandsSeeder

A `BrandsSeeder` osztály felelős a márkák létrehozásáért az adatbázisban. Ezek a márkák segítenek az alkalmazásban található termékek azonosításában és csoportosításában.

Létrehozott márkák

1. **Intel**: Az Intel egy világszerte elismert vállalat, amely processzorokat és egyéb számítógépes hardvereket gyárt.
2. **Amd**: Az AMD egy másik nagyobb szereplő a számítógépes hardverpiacon, amely szintén processzorokat és más alkatrészeket gyárt.
3. **G.Skill**: A G.Skill egy olyan márka, amely a számítógép-memóriamodulokra (RAM) specializálódott, köztük különféle sebességekkel és kapacitásokkal rendelkező memóriamodulokat kínál.
4. **Gigabyte**: A Gigabyte egy olyan vállalat, amely széleskörű számítógépes alkatrészeket kínál, ideértve az alaplapokat, videokártyákat és egyéb perifériákat.
5. **MSI**: Az MSI egy másik olyan vállalat, amely számítógépes alkatrészeket gyárt, beleértve az alaplapokat, videokártyákat és egyéb hardvereket.

Használat

A `BrandsSeeder` osztály a Laravel `db:seed` parancsának használatával futtatható, vagy más inicializálási folyamatok részeként. A `run` metódusában létrehozott márkák az alkalmazás számára biztosítják az alapvető márkák kezdeti adatokkal való feltöltését az adatbázisban.

CategoriesSeeder

A `CategoriesSeeder` osztály felelős a kategóriák létrehozásáért az adatbázisban. Ezek a kategóriák segítenek csoportosítani a termékeket az alkalmazásban.

Létrehozott kategóriák

1. **Processor**: Ez a kategória a processzorokat tartalmazza, amelyek az informatikai eszközök egyik alapvető elemei.
2. **Memory**: A "Memory" kategória a memóriamodulokat vagy RAM (Random Access Memory) eszközöket foglalja magában, amelyek az adatok ideiglenes tárolására szolgálnak a számítógépben.
3. **Motherboard**: Ez a kategória az alaplapokat vagy az alaplaphoz kapcsolódó kiegészítő eszközöket tartalmazza. Az alaplap az egyik legfontosabb hardveralkatrész a számítógépben, mivel összeköti a különböző alkatrészeket és biztosítja az adatok továbbítását.

Használat

A `CategoriesSeeder` osztály a Laravel `db:seed` parancsának használatával futtatható, vagy más inicializálási folyamatok részeként. A `run` metódusában létrehozott kategóriák az alkalmazás számára biztosítják az alapvető kategóriák kezdeti adatokkal való feltöltését az adatbázisban.

PaymentSeeder

A `PaymentSeeder` osztály felelős a fizetési módok létrehozásáért az adatbázisban. Ezek a fizetési módok segítenek az alkalmazásban a rendelések fizetésének kezelésében és kategorizálásában.

Létrehozott fizetési módok

1. **Boltban**: Ez a fizetési mód azt jelenti, hogy a vásárlók személyesen fizetnek a boltban, amikor átveszik a megrendelt termékeket.
2. **Háznál**: Ez a fizetési mód azt jelenti, hogy a vásárlók a szállítási szolgáltatás által szállított termékek kiszállításakor fizetnek.

Használat

A `PaymentSeeder` osztály a Laravel `db:seed` parancsának használatával futtatható, vagy más inicializálási folyamatok részeként. A `run` metódusában létrehozott fizetési módok az alkalmazás számára biztosítják az alapvető fizetési módok kezdeti adatokkal való feltöltését az adatbázisban.

ProductsSeeder

A `ProductsSeeder` osztály felelős különböző termékek létrehozásáért és feltöltéséért az adatbázisba. Ezek a termékek reprezentálják a boltban elérhető különböző hardverkomponenseket, például processzorokat, memóriákat és alaplapokat.

Létrehozott termékek

1. **i7-11700k**
    - Kategória: Processzor
    - Márka: Intel
    - Ár: 117000 Ft
    - Készleten lévő mennyiség: 0
    - Kép: .\assets\images\proci1.jpg
    - Leírás: I7-11700k
2. **Ryzen 5 5600x**
    - Kategória: Processzor
    - Márka: Amd
    - Ár: 60000 Ft
    - Készleten lévő mennyiség: 0
    - Kép: .\assets\images\proci2.jpg
    - Leírás: Ryzen 5 5600x
3. **16GB DDR4 3200MHz CL16 Trident Z RGB**
    - Kategória: Memória
    - Márka: G.Skill
    - Ár: 25000 Ft
    - Készleten lévő mennyiség: 0
    - Kép: .\assets\images\memoria1.jpg
    - Leírás: 16GB DDR4 3200MHz CL16 Trident Z RGB
4. **16GB DDR4 3733MHz Kit Aorus RGB**
    - Kategória: Memória
    - Márka: Gigabyte
    - Ár: 26000 Ft
    - Készleten lévő mennyiség: 0
    - Kép: .\assets\images\memoria2.jpg
    - Leírás: 16GB DDR4 3733MHz Kit Aorus RGB
5. **B550 Gaming X V2**
    - Kategória: Alaplap
    - Márka: Gigabyte
    - Ár: 45000 Ft
    - Készleten lévő mennyiség: 0
    - Kép: .\assets\images\alaplap1.jpg
    - Leírás: B550 Gaming X V2
6. **MSI MPG Z590 GAMING PLUS**
    - Kategória: Alaplap
    - Márka: MSI
    - Ár: 90000 Ft
    - Készleten lévő mennyiség: 0

- Kép: .\assets\images\alaplap2.jpg
- Leírás: MSI MPG Z590 GAMING PLUS

Használat

A `ProductsSeeder` osztály a Laravel `db:seed` parancsának használatával futtatható, vagy más inicializálási folyamatok részeként. A `run` metódusában létrehozott termékek az alkalmazás számára biztosítják az alapvető termékek kezdeti adatokkal való feltöltését az adatbázisban.

DatabaseSeeder Documentation

1. **Create Admin User:**
    - Creates an admin user with the following credentials:
        - Name: admin
        - Email: admin@example.com
        - Password: admin
        - Address: ''
        - City: ''
        - Postal Code: 0
        - Phone: ''
        - Admin: true
2. **Seed Other Tables:**
    - Calls the seeder classes responsible for seeding other tables:
        - `BrandsSeeder`: Seeds brands data.
        - `CategoriesSeeder`: Seeds categories data.
        - `PaymentSeeder`: Seeds payment methods data.
        - `ProductsSeeder`: Seeds products data.

Usage

The `DatabaseSeeder` class is executed when running the `db:seed` Artisan command. It initializes the database with essential initial data, including an admin user and various seed data for other tables.

Frontend

Komponensek

About Komponens

about.component.html

- Az **`about.component.html`** fájl tartalmazza a "Rólunk" oldal HTML kódját, amely a PC Webshop bemutatását és köszönetét tartalmazza a felhasználóknak.

about.component.ts

- Az **`about.component.ts`** fájl egy üres TypeScript osztályt tartalmaz, amelynek nincs saját logikája vagy adattagja, mivel a komponens statikus tartalmat jelenít meg.

Cart Komponens

cart.component.html

- Az **`cart.component.html`** fájl tartalmazza a kosár komponens HTML kódját, amely megjeleníti a felhasználó kosarában lévő termékeket és az összesített árat.

cart.component.ts

- Az **`cart.component.ts`** fájlban található osztály felelős a kosár komponens logikájáért. Ez a logika magában foglalja a kosár elemeinek betöltését, a képfájl útvonalak generálását, a termékek eltávolítását a kosárból, az összesített ár kiszámítását és a továbblépés műveletét.
- A kosár elemek dinamikusan generált képeinek eléréséhez használja a **`generateImagePath`** metódust, amely az adott termék nevéből állítja elő az elérési útvonalat.

Choose Komponens

choose.component.html

- A **`choose.component.html`** fájl tartalmazza a választás komponens HTML kódját, amely lehetővé teszi a felhasználó számára, hogy kiválassza a szállítási módot és folytassa a vásárlást.

choose.component.ts

- A **`choose.component.ts`** fájl tartalmazza a ChooseComponent osztályt, amely felelős a választás komponens logikájáért. Ez a logika magában foglalja a szállítási módozat változását, a fizetési azonosító frissítését és a rendelés leadását.

- Az choose komponens lehetővé teszi a felhasználó számára, hogy válasszon a különböző szállítási módok közül, és folytassa a rendelési folyamatot. A felhasználó választása alapján frissül a fizetési azonosító, és a rendelés gomb megnyomására a rendelés elküldésre kerül, amennyiben a felhasználó be van jelentkezve.

Footer Komponens

footer.component.html

- A **`footer.component.html`** fájl tartalmazza az alkalmazás láblécének HTML kódját, amely a weboldal alján jelenik meg. Itt található a szerzői jogi szöveg, amely a PCY Webshop 2024 évi jogait védik.

footer.component.ts

- A **`footer.component.ts`** fájlban található FooterComponent osztály felelős az alkalmazás láblécének logikájáért. Jelenleg nincs szükség külön logikára a lábléc megjelenítéséhez vagy működéséhez.

- Az alkalmazás lábléce a weboldal alján jelenik meg, és a szerzői jogi szöveg tartalmazza a PCY Webshop 2024 évi jogait.

Header Komponens

header.component.html

- A **`header.component.html`** fájl tartalmazza az alkalmazás fejlécének HTML kódját, amely a weboldal tetején jelenik meg. Ez a fejléc tartalmazza a PCY Webshop nevet, a navigációs menüt, valamint a kereső funkciót és a bejelentkezés/ regisztráció vagy profil menüt, attól függően, hogy a felhasználó be van-e jelentkezve.

header.component.ts

- A **`header.component.ts`** fájlban található HeaderComponent osztály felelős az alkalmazás fejlécének logikájáért. Ez a logika magában foglalja a kijelentkezés folyamatát, valamint ellenőrzi, hogy a felhasználó be van-e jelentkezve vagy admin jogosultsága van-e.

- Az alkalmazás fejléce a weboldal tetején jelenik meg, és tartalmazza a PCY Webshop nevet, a navigációs menüt, a kereső funkciót, valamint a bejelentkezés/ regisztráció vagy profil menüt a felhasználó bejelentkezési állapotától és jogosultságától függően. A kijelentkezés folyamata a felhasználó kattintása után történik, és átirányítja a felhasználót a kezdőoldalra.

Home Komponens

home.component.html

- A **`home.component.html`** fájl tartalmazza az alkalmazás kezdőoldalának HTML kódját, amely a felhasználót köszönti, bemutatja a webshopot és néhány kiemelt terméket. Emellett tartalmaz egy képsorozatot (carousel-t), amely bemutatja a webshop néhány fontos részletét.

home.component.ts

- A **`home.component.ts`** fájlban található HomeComponent osztály felelős az alkalmazás kezdőoldalának logikájáért. Ez a logika magában foglalja a termékek letöltését az API-ból, valamint a kosárba való termék hozzáadását.

- Az alkalmazás kezdőoldala köszönti a felhasználót, bemutatja a webshopot és néhány kiemelt terméket, amelyeket a felhasználó megrendelhet. A termékek letöltése az API-ból történik, és azokat dinamikusan jelenítjük meg a képernyőn. A kosárba való termék hozzáadása a "Rendeld meg!" gombra kattintva lehetséges.

Login Komponens

login.component.html

- A **`login.component.html`** fájl tartalmazza a bejelentkezési űrlap HTML kódját, amely lehetővé teszi a felhasználó számára, hogy bejelentkezzen az alkalmazásba. Az űrlap tartalmazza az e-mail cím és jelszó mezőket, valamint egy "Tovább" gombot a bejelentkezés folyamatának indításához.

login.component.ts

- A **`login.component.ts`** fájlban található LoginComponent osztály felelős a bejelentkezési űrlap logikájáért. Ez a logika magában foglalja a felhasználó bejelentkezését az AuthService segítségével.

- A bejelentkezési űrlap lehetővé teszi a felhasználó számára, hogy megadja az e-mail címét és jelszavát, majd a "Tovább" gombra kattintva bejelentkezhet az alkalmazásba. A bejelentkezési adatokat az AuthService segítségével ellenőrzik és kezelik.

Myorders Komponens

myorders.component.html

- A **`myorders.component.html`** fájl tartalmazza a felhasználó rendeléseinek listázására szolgáló HTML kódot. A rendelések listázása kártyákban történik, ahol minden kártya egy rendelést képvisel. A kártya tartalmazza a rendelés azonosítóját, fizetési módját, rendelés dátumát, valamint a rendeléshez tartozó termékek listáját.

myorders.component.ts

- A **`myorders.component.ts`** fájlban található MyordersComponent osztály felelős a felhasználó rendeléseinek megjelenítéséért és kezeléséért. Ez a komponens lekéri a rendeléseket az API-tól, valamint a rendelésekhez tartozó termékeket, majd megjeleníti azokat a felhasználónak. Emellett a felhasználó törölheti a rendeléseket a rendelésből.

- A myorders komponens lehetővé teszi a felhasználó számára, hogy megtekinthesse a rendeléseit, valamint törölje azokat a rendelésből, ha szükséges. Az API-tól történő adatlekérés és -kezelés segítségével dinamikusan jelenítjük meg a rendeléseket és azokhoz tartozó termékeket.

Product-edit Komponens

product-edit.component.html

- A **`product-edit.component.html`** fájl tartalmazza a termék szerkesztésére szolgáló űrlapot, amely lehetővé teszi a felhasználó számára, hogy módosítsa a termék adatait, például a nevet, árat, leírást, kategória- és márkaid-t. Az űrlap tartalmaz egy "Mentés" gombot, amelynek megnyomásával a módosítások mentésre kerülnek.

product-edit.component.ts

- A **`product-edit.component.ts`** fájlban található ProductEditComponent osztály felelős a termék szerkesztésének logikájáért. Ez a komponens inicializálja az űrlapon megjelenített termék adatait, majd lehetővé teszi a felhasználó számára, hogy módosítsa ezeket az adatokat. A módosításokat a "Mentés" gombra kattintva küldi el az API-nak a frissítéshez.

- A product-edit komponens lehetővé teszi a felhasználó számára, hogy módosítsa egy termék adatait, és ezeket a módosításokat mentse az alkalmazásban. A frissítés a "Mentés" gomb megnyomásával történik, és az adatokat az AuthService segítségével az API-nak küldik el. Ha a frissítés sikeres, a komponens újratölti az aktuális oldalt, hogy az új adatok megjelenjenek.

Products komponens

products.component.html:

- Az **`products.html`** fájl tartalmazza a termékek listázásához szükséges DOM elemeket. A Bootstrap segítségével kialakított kártyákban jelennek meg a termékek adatai, mint például a név, ár és kép. Adminisztrátori jogosultsággal rendelkező felhasználók számára további információk (pl. termékazonosító, kategória- és márkaid) jelennek meg. Emellett az adminisztrátoroknak lehetőségük van a termékek szerkesztésére és törlésére.
- Az adminisztrátoroknak külön űrlap jelenik meg, amely lehetővé teszi új termékek hozzáadását az alkalmazáshoz.

products.component.ts:

- A **procuts.ts** fájl tartalmazza a ProductsComponent osztályt, amely felelős a termékek listázásának, hozzáadásának, szerkesztésének és törlésének logikájáért.
- Az ngOnInit metódusban inicializáljuk a komponensben használt adatokat, mint például a termékek listáját és a termék mennyiségeket.
- A getProducts metódus segítségével lekérjük a termékeket az adatszolgáltató szolgáltatástól.
- Az onAddToCart metódus felelős a termék kosárba helyezéséért. Ha a felhasználó be van jelentkezve, akkor hozzáadjuk a terméket a kosárhoz, ellenkező esetben figyelmeztetést jelenítünk meg a bejelentkezésre.
- Az addProduct metódus felelős az új termék hozzáadásáért az adatszolgáltató szolgáltatáshoz.
- A onDeleteProduct metódus felelős a termék törléséért az adatszolgáltató szolgáltatásból.
- Az editProduct metódus a kiválasztott termék szerkesztésére szolgál.

- Összességében a products komponens lehetővé teszi a felhasználók számára a termékek böngészését, a kosárba helyezését, az új termékek hozzáadását, valamint az adminisztrátorok számára a termékek szerkesztését és törlését az alkalmazásból.

Profile komponens

profile.component.html:

- Az **profile.html** fájl tartalmazza a felhasználó profiljának megjelenítéséhez és szerkesztéséhez szükséges DOM elemeket. A kártya stílusú megjelenítésben láthatók a felhasználó neve, e-mail címe, címe, városa, irányítószáma és telefonszáma. Ha a felhasználó szerkeszteni szeretné a profilját, akkor az input mezők jelennek meg a szöveges adatok helyett.
- A felhasználóknak lehetőségük van a profiljuk szerkesztésére, illetve a profil törlésére. A profil törlése előtt megerősítést kérünk a felhasználótól.

profile.component.ts:

- A **profile.ts** fájl tartalmazza a ProfileComponent osztályt, amely felelős a profil megjelenítésének és kezelésének logikájáért.
- A constructorban inicializáljuk az AuthService-t és a Router-t, amelyekre szükségünk van a felhasználói műveletek elvégzéséhez.
- Az updateProfile metódus felelős a felhasználó profiljának frissítéséért. Az AuthService segítségével meghívjuk a megfelelő szolgáltatást a felhasználó adatainak frissítésére.
- A deleteProfile metódus felelős a felhasználó profiljának törléséért. Mielőtt törölnénk a profilt, megerősítést kérünk a felhasználótól, majd a megfelelő szolgáltatást hívjuk meg az AuthService segítségével a profil törléséhez.

- A reloadCurrentRoute metódus segítségével újratöltjük az aktuális útvonalat a profil frissítése után.

- Összességében a profile komponens lehetővé teszi a felhasználók számára, hogy megtekintsék, szerkesszék és töröljék a profiljukat az alkalmazásban.

Profile-edit komponens

profile-edit.component.html:

- A **`profile-edit.html`** fájl tartalmazza a szerkesztő űrlap elemeket, amelyek lehetővé teszik a felhasználói profil adatainak szerkesztését. A név, e-mail cím, cím, város, irányítószám és telefonszám mezőket jelenítjük meg, amelyeket az adott felhasználó adataival előtöltünk. A felhasználó módosíthatja ezeket az adatokat, majd a "Mentés" gombra kattintva frissítheti a profilját.

profile-edit.component.ts:

- A **`profile-edit.ts`** fájl tartalmazza a `ProfileEditComponent` osztályt, amely felelős a profil szerkesztésének logikájáért.
- A `ngOnInit` metódusban az `AuthService` segítségével lekérjük a felhasználó profilját a `getProfile` metódussal, és az adatokat beállítjuk az `editedProfile` változóba.
- Az `updateProfile` metódus felelős a profil frissítéséért. Az `HttpClient` segítségével PUT kérést hajtunk végre a megfelelő végpont felé, ahol a frissítés megtörténik. Ha a frissítés sikeres, értesítjük a felhasználót, hogy jelentkezzen be újra, mivel a profiladatai módosultak. Ha hiba történik a frissítés közben, megjelenítünk egy hibaüzenetet.
- A `reloadCurrentRoute` metódus segítségével újratöltjük az aktuális útvonalat a profil frissítése után.
- Ezáltal a `profile-edit` komponens lehetővé teszi a felhasználók számára, hogy módosítsák a profiljuk adatait, és frissítsék azokat az alkalmazásban.

Signup componens

signup.component.html:

- Az **`signup.html`** tartalmazza a regisztrációs mezőket, mint például a név, e-mail cím, jelszó, cím, város, irányítószám és telefonszám. Mindegyik mezőnek van néhány validációs szabálya, mint például a kötelező kitöltés, minimum és maximum hossz.
- A "Tovább" gombra kattintva a felhasználó regisztrációs adatai elküldődnek a szerverre.

signup.component.ts:

- A **`signup.ts`** fájl tartalmazza a `SignupComponent` osztályt, amely felelős a regisztráció logikájáért.
- Az `register` metódus validálja az űrlapot a `signupForm.invalid` tulajdonság alapján. Ha az űrlap érvénytelen, megjelenik egy figyelmeztető üzenet, és a metódus visszatér. Ha az űrlap érvényes, akkor a regisztrációs adatokat elküldi a szerverre a `HttpClient` segítségével POST kéréssel.
- Ha a regisztráció sikeres, megjelenik egy sikerüzenet, és a felhasználót átirányítjuk a bejelentkezési oldalra. Ha a regisztráció sikertelen, megjelenik egy hibaüzenet, és a felhasználó visszatérhet a regisztrációs űrlaphoz.

- Ezáltal a `signup` komponens lehetővé teszi a felhasználók számára, hogy regisztráljanak az alkalmazásba, és létrehozzák a fiókjukat.

Service.ts fájlok

Auth.service.ts

- **isLoggedIn**: Egy logikai érték, ami azt jelzi, hogy a felhasználó be van-e jelentkezve vagy sem.
- **isAdmin**: Egy logikai érték, ami azt jelzi, hogy a felhasználó adminisztrátor-e vagy sem.
- **userData**: Egy objektum, amely tárolja a bejelentkezett felhasználó adatait, például a nevet, e-mail címet, stb.
- **token**: Egy token, amely az autentikációhoz használt.

Az osztály a következő metódusokat tartalmazza:

- **checkLoggedInStatus()**: Ellenőrzi, hogy a felhasználó be van-e jelentkezve. Ha igen, betölti a felhasználó adatait a `localStorage`-ból.
- **login(userData: any)**: Meghívja a szerver `login` végpontját, hogy bejelentkezzen a felhasználó. Ha a bejelentkezés sikeres, beállítja a `isLoggedIn` értéket, menti a felhasználó adatait a `localStorage`-ba és beállítja az `isAdmin` értéket.
- **updateProfile(userData: any)**: Elküldi a frissített profiladatokat a szerverre a `PUT` kéréssel.
- **deleteProfile()**: Elküldi a profil törlésének kérését a szervernek a `DELETE` kéréssel.
- **getProfile()**: Lekéri a felhasználó profiladatait a szerverről.
- **logout()**: Kijelentkezteti a felhasználót, törli a felhasználó adatait a `localStorage`-ból.
- **reloadAndNavigate()**: Újratölti az oldalt, majd átirányítja a felhasználót a kezdőoldalra.

- Ezek az osztály funkciói lehetővé teszik a felhasználók számára a bejelentkezést, profiljuk frissítését, törlését és lekérését, valamint a kijelentkezést.

Base.service.ts

- **host**: Az API végpont címe, ahol a termékekkel kapcsolatos adatok elérhetők.
- **selectedProduct**: Egy privát változó, amely tárolja a kiválasztott termék adatait.

Az osztály a következő metódusokat tartalmazza:

- **getProducts()**: Ez a metódus egy HTTP GET kérést küld a szervernek az összes termék lekérésére a megadott API végpontra. A visszatérési érték egy `Observable`, amely a termékek adatait tartalmazza.
- **setSelectedProduct(product: Product)**: Beállítja a kiválasztott termék adatait a megadott `Product` objektum alapján.
- **getSelectedProduct()**: Visszaadja a kiválasztott termék adatait.

- Ez az osztály lehetővé teszi a többi komponens számára, hogy kommunikáljanak a szerverrel és kezeljék a termékekkel kapcsolatos műveleteket, például a termékek lekérdezését, kiválasztását és megjelenítését.

Cart.service.ts

- **cartItems**: Egy privát tömb, amely tárolja a kosárban lévő elemeket.

Az osztály a következő metódusokat tartalmazza:

- **addToCart(item: any, quantity: number = 1)**: Ez a metódus hozzáad egy elemet a kosárhoz vagy növeli annak mennyiségét, ha az már benne van. Ha az elem már létezik a kosárban, akkor csak a mennyiségét növeli, különben hozzáadja az elemet a kosárhoz.
- **getCartItems()**: Visszaadja a kosárban lévő elemek tömbjét.
- **removeFromCart(index: number)**: Törli az adott indexű elemet a kosárból.
- **clearCart()**: Törli az összes elemet a kosárból, azaz kiüríti azt.

- Ez az osztály lehetővé teszi a többi komponens számára, hogy hozzáférjenek és manipulálják a kosár tartalmát. Például hozzáadhatnak elemeket, eltávolíthatnak elemeket, vagy törölhetik az összes elemet a kosárból.

Myorders.service.ts

- **apiUrl**: Az API végpont alapértelmezett URL-címe, ahol a rendeléseket le lehet kérni.

Az osztály a következő metódust tartalmazza:

- **getOrders()**: Ez a metódus aszinkron módon lekéri a felhasználó rendeléseit a szerverről. A kérést JSON típusú adatként küldi el, és hozzáadja az Authorization fejlécet a felhasználóhoz tartozó tokennel. Ha a kérés sikeres, a válaszban kapott rendeléseket visszaadja. Ha hiba történik a kérés során, a metódus egy üres tömböt ad vissza, és a hibát naplózza a konzolon.

- Ez az osztály lehetővé teszi a többi komponens számára, hogy hozzáférjenek a felhasználó rendeléseinek adataihoz és megjelenítsék azokat a felhasználói felületen. A `MyordersService` felelős az adatok lekérdezéséért és kezeléséért, miközben a többi komponens a lekért adatokat megjeleníti a felhasználói felületen.

Myordersitems.service.ts

- **apiUrl**: Az API végpont alapértelmezett URL-címe, ahol a rendelés tételeit le lehet kérni és törölni.

Az osztály a következő metódusokat tartalmazza:

- **getOrders(orderId: number)**: Ez a metódus aszinkron módon lekéri a megadott orderId-hoz tartozó rendelés tételeit a szerverről. A kérést JSON típusú adatként küldi el, és hozzáadja az Authorization fejlécet a felhasználóhoz tartozó tokennel. Ha a kérés sikeres, a válaszban kapott rendelés tételeit visszaadja. Ha hiba történik a kérés során, a metódus egy üres tömböt ad vissza, és a hibát naplózza a konzolon.
- **deleteOrderItem(orderId: number)**: Ez a metódus aszinkron módon törli a megadott orderId-hoz tartozó rendelés tételeket a szerverről. A kérést HTTP DELETE kérés típusúként küldi el, és hozzáadja az Authorization fejlécet a felhasználóhoz tartozó tokennel. Ha a törlés sikeres, a választ visszaadja. Ha hiba történik a törlés során, a metódus egy hibajelzést generál, és a hibát naplózza a konzolon.

- Ez az osztály lehetővé teszi a többi komponens számára, hogy hozzáférjenek a felhasználó rendeléseinek tételeihez és megjelenítsék azokat a felhasználói felületen. A `MyordersitemsService` felelős az adatok lekérdezéséért és kezeléséért, miközben a többi komponens a lekért adatokat megjeleníti vagy kezeli a felhasználói felületen.

Order.service.ts

- **apiUrl**: Az API végpont alapértelmezett URL-címe, ahol a rendelési műveletek elvégezhetők.

Az osztály a következő metódust tartalmazza:

- **order(userData: any): Observable<boolean>**: Ez a metódus aszinkron módon küldi el a felhasználó rendelési adatait a szervernek. A kérést POST kérés típusúként küldi el, JSON típusú adatként, és hozzáadja az Authorization fejlécet a felhasználóhoz tartozó tokennel. Ha a rendelés sikeres, a válaszban kapott adatokat a `userData` változóban tárolja és `true` értékkel tér vissza. Ha hiba történik a kérés során, a metódus egy hibajelzést generál, a hibát naplózza a konzolon, törli a `userData` változót, és `false` értékkel tér vissza.

- Ez az osztály lehetővé teszi a többi komponens számára, hogy rendeléseket hozzanak létre és küldjenek a szervernek. A `OrderService` felelős a rendelési adatok kezeléséért és a szerverrel való kommunikációért, miközben a többi komponens a felhasználói felületen megjeleníti vagy kezeli a rendelési adatokat.

Köszönet

Köszönjük tanárainknak a sok segítséget.