Fejlesztői Dokumentáció

PiciPiac

Tartalomjegyzék

[1. Bevezetés 3](#_Toc188556713)

[**1.1 Projekt célja és összefoglalója** 3](#_Toc188556714)

[**1.2 Dokumentáció célja** 3](#_Toc188556715)

[**1.3 A PiciPiac főbb funkciói** 3](#_Toc188556716)

[2. Környezeti követelmények 3](#_Toc188556717)

[**2.1 Operációs rendszer** 3](#_Toc188556718)

[**2.2 Szükséges szoftverek** 3](#_Toc188556719)

[**2.3 Hardverkövetelmények** 4](#_Toc188556720)

[**2.4 Laravel-specifikus követelmények** 4](#_Toc188556721)

[3. Telepítési lépések 5](#_Toc188556722)

[**3.1 Projekt klónozása** 5](#_Toc188556723)

[**3.2 Composer függőségek telepítése** 5](#_Toc188556724)

[**3.3 Környezeti változók konfigurálása** 5](#_Toc188556725)

[**3.4 Laravel kulcs generálása** 6](#_Toc188556726)

[**3.5 Adatbázis inicializálása** 6](#_Toc188556727)

[**3.6 Frontend függőségek telepítése** 6](#_Toc188556728)

[**3.7 Alkalmazás indítása** 7](#_Toc188556729)

[**3.8 Hibakeresés** 7](#_Toc188556730)

[4. Használt csomagok és keretrendszerek 7](#_Toc188556731)

[**4.1 Laravel alapcsomagok** 7](#_Toc188556732)

[**4.2 Telepített Composer csomagok** 7](#_Toc188556733)

[**4.3 Frontend keretrendszerek és eszközök** 8](#_Toc188556734)

[**4.4 Fejlesztői eszközök** 8](#_Toc188556735)

[**4.5 Telepítés és verzióellenőrzés** 9](#_Toc188556736)

[5. Adatbázis struktúra 9](#_Toc188556737)

[**5.1 Táblák és oszlopok** 9](#_Toc188556738)

[**1. Felhasználók táblája (users)** 9](#_Toc188556739)

[**2. Termékek táblája (products)** 9](#_Toc188556740)

[**3. Kategóriák táblája (categories)** 9](#_Toc188556741)

[**4. Megrendelések táblája (orders)** 10](#_Toc188556742)

[**5. Megrendelési tételek táblája (order\_items)** 10](#_Toc188556743)

[**5.2 Táblák közötti relációk** 10](#_Toc188556744)

[**1. Felhasználók és megrendelések (users ↔ orders)** 10](#_Toc188556745)

[**2. Megrendelések és tételek (orders ↔ order\_items)** 10](#_Toc188556746)

[**3. Termékek és kategóriák (products ↔ categories)** 11](#_Toc188556747)

[**4. Termékek és eladók (products ↔ users)** 11](#_Toc188556748)

[**5.3 Adatbázis diagram** 11](#_Toc188556749)

[6. MVC Struktúra 11](#_Toc188556750)

[**6.1 Modellek (app/Models/)** 11](#_Toc188556751)

[**Fontosabb modellek:** 11](#_Toc188556752)

[**6.2 Kontrollerek (app/Http/Controllers/)** 12](#_Toc188556753)

[**Fontosabb kontrollerek:** 13](#_Toc188556754)

[**6.3 Nézetek (View-k)** 14](#_Toc188556755)

[**Főbb nézetek:** 14](#_Toc188556756)

[**Blade sablon funkciók:** 15](#_Toc188556757)

[7. CRUD Műveletek és Objektumok 15](#_Toc188556758)

[**7.1 CRUD Műveletek Példák** 15](#_Toc188556759)

[**1. CREATE – Új adat létrehozása** 15](#_Toc188556760)

[**2. READ – Adatok lekérdezése** 16](#_Toc188556761)

[**3. UPDATE – Adatok frissítése** 17](#_Toc188556762)

[**4. DELETE – Adatok törlése** 18](#_Toc188556763)

[**7.2 Objektumok és fontosabb függvények** 18](#_Toc188556764)

[8. RESTful API 19](#_Toc188556765)

[**8.1 Autentikáció és jogosultság** 19](#_Toc188556766)

[**8.2 API végpontok** 19](#_Toc188556767)

[**1. Felhasználói autentikáció** 19](#_Toc188556768)

[**2. Termékek kezelése** 20](#_Toc188556769)

[**3. Megrendelések kezelése** 21](#_Toc188556770)

[**8.3 Tesztelés és dokumentáció** 22](#_Toc188556771)

[**1. API tesztelés** 22](#_Toc188556772)

[**2. Automatikus dokumentáció:** 22](#_Toc188556773)

[9. Fejlesztési lehetőségek 22](#_Toc188556774)

[**9.1 Új funkciók bevezetése** 22](#_Toc188556775)

[**9.2 Tesztelési stratégiák** 23](#_Toc188556776)

[**9.3 Teljesítmény optimalizálás** 24](#_Toc188556777)

[**9.4 Biztonsági fejlesztések** 25](#_Toc188556778)

[10. Következtetések és ajánlások 25](#_Toc188556779)

[**10.1 Összefoglaló** 25](#_Toc188556780)

[**10.2 Javaslatok a fejlesztőknek** 26](#_Toc188556781)

[**10.3 Jövőbeni fejlesztések** 26](#_Toc188556782)

[**10.4 Ajánlások a projekt csapatának** 26](#_Toc188556783)

### 1. Bevezetés

**1.1 Projekt célja és összefoglalója**

A **PiciPiac** egy online shop platform, amely lehetőséget biztosít kézműves termékek, zöldségek és gyümölcsök online vásárlására és értékesítésére. A projekt célja, hogy megkönnyítse a helyi termelők és kézművesek számára termékeik piacra jutását, miközben a vásárlók egyszerűen, otthonukból hozzáférhetnek friss és minőségi termékekhez.

**1.2 Dokumentáció célja**

Ez a dokumentáció a PiciPiac Laravel alapú rendszerének fejlesztési környezetét, technikai részleteit és fejlesztési lehetőségeit írja le. A cél, hogy a fejlesztők könnyen megértsék és bővíthessék a projektet, miközben biztosítjuk a projekt fenntarthatóságát és skálázhatóságát.

**1.3 A PiciPiac főbb funkciói**

1. **Termékek listázása és keresése**:
   * A vásárlók böngészhetik a termékeket kategóriák szerint (pl. kézműves termékek, zöldségek, gyümölcsök).
   * Részletes termékadatlapok elérhetők.
2. **Kosár és rendeléskezelés**:
   * Vásárlók kosárba helyezhetik a termékeket.
   * Megrendelések leadása egyszerű folyamat.
3. **Felhasználói regisztráció és profilok**:
   * Vásárlók és eladók regisztrációja.
   * Profiloldal termékfeltöltéssel az eladók számára.
4. **Adminisztrációs felület**:
   * Termékek, rendelések és felhasználók kezelése.

### 2. Környezeti követelmények

A **PiciPiac** Laravel-alapú alkalmazás megfelelő működéséhez az alábbi szoftver- és hardverkövetelmények teljesítése szükséges.

**2.1 Operációs rendszer**

* **Ajánlott**:
  + Windows 10/11
  + macOS Monterey vagy újabb
  + Linux (Ubuntu 20.04+, CentOS, Fedora)

**2.2 Szükséges szoftverek**

Az alkalmazás fejlesztéséhez és futtatásához az alábbi eszközök és verziók szükségesek:

1. **Laravel**
   * Verzió: 8.1.
   * Szükséges bővítmények:
   * openssl
   * pdo
   * mbstring
   * tokenizer
   * xml
   * ctype
2. **Composer**
   * Verzió: 2.0
   * A függőségek kezeléséhez szükséges.
3. **Webszerver**
   * Apache 2.4 vagy Nginx.
4. **Adatbázis-kezelő**
   * MySQL 5.7 vagy újabb / MariaDB.
   * Alternatív lehetőségek: PostgreSQL.
5. **Node.js és npm**
   * **Node.js**: Verzió 14.1
   * **npm**: Verzió 6.2 vagy újabb (frontend assetekhez, pl. Bootstrap és Sass).
6. **Git**
   * Verziókezeléshez szükséges.

**2.3 Hardverkövetelmények**

Minimális fejlesztői környezet:

* **Processzor**: Intel i3 / AMD Ryzen 3 vagy jobb.
* **Memória**: Legalább 4 GB RAM (8 GB ajánlott).
* **Tárhely**: Legalább 2 GB szabad hely az alkalmazás, függőségek és adatbázis számára.

**2.4 Laravel-specifikus követelmények**

1. **Laravel verzió**: 11.35.0.
2. **Futtatási környezet**:
   * Fejlesztői környezetben: artisan serve segítségével.
   * Éles környezetben: Apache vagy Nginx konfigurációval.
3. **Környezeti változók**: Az env fájl helyes konfigurációja szükséges, amely tartalmazza az adatbázis-kapcsolat, API kulcsok, és más konfigurációk adatait.

**Adatbázis-kapcsolat**:

* + DB\_CONNECTION=mysql
  + DB\_HOST=127.0.0.1
  + DB\_PORT=3306
  + DB\_DATABASE=picipiac
  + DB\_USERNAME=root
  + DB\_PASSWORD=secret

**App kulcsok és egyéb adatok:**

* + APP\_NAME=PiciPiac
  + APP\_=local
  + APP\_KEY=base64:...
  + APP\_DEBUG=true
  + APP\_URL=http://localhost

**Külső API-k kulcsai (pl. fizetési rendszerek, e-mail szolgáltatók):**

* + MAIL\_HOST=smtp.mailtrap.io
  + MAIL\_PORT=2525
  + MAIL\_USERNAME=null
  + MAIL\_PASSWORD=null
  + MAIL\_ENCRYPTION=null

**Cache vagy queue konfigurációk (ha használtok ilyet):**

* + CACHE\_DRIVER=file
  + QUEUE\_CONNECTION=sync

### 3. Telepítési lépések

A következő útmutató bemutatja, hogyan telepítheted a **PiciPiac** Laravel-alapú alkalmazást egy fejlesztői környezetben. Ezek a lépések biztosítják, hogy a projekt helyesen működjön és minden függőség megfelelően települjön.

**3.1 Projekt klónozása**

1. Nyisd meg a terminált vagy a parancssort.
2. Klónozd a projektet a verziókezelőből (GitHub, GitLab, stb.):

git clone https://github.com/felhasznalo/PiciPiac.git

1. Lépj be a projekt könyvtárába:

cd PiciPiac

**3.2 Composer függőségek telepítése**

1. Ellenőrizd, hogy a Composer telepítve van:

composer --version

1. Telepítsd a projekt függőségeit:

composer install

**3.3 Környezeti változók konfigurálása**

1. Másold az .env.example fájlt .env néven:

cp ..example .

1. Szerkeszd az .env fájlt a következő környezeti változók hozzáadásával:

APP\_NAME=PiciPiac

APP\_=local

APP\_KEY=

APP\_DEBUG=true

APP\_URL=http://localhost

DB\_CONNECTION=mysql

DB\_HOST=127.0.0.1

DB\_PORT=3306

DB\_DATABASE=picipiac

DB\_USERNAME=root

DB\_PASSWORD=secret

**3.4 Laravel kulcs generálása**

Generáld le az alkalmazás kulcsát, amely az adatok titkosításához szükséges:

artisan key:generate

**3.5 Adatbázis inicializálása**

1. Állítsd be az adatbázist az env. fájlban:

DB\_DATABASE=picipiac

DB\_USERNAME=root

DB\_PASSWORD=secret

1. Hozd létre az adatbázist a MySQL-ben vagy a választott adatbázis-kezelőben:

CREATE DATABASE picipiac;

1. Futtasd a migrációkat a szükséges táblák létrehozásához:

artisan migrate

1. (Opcionális) Ha vannak seederek, töltsd fel az alapadatokat:

artisan db:seed

**3.6 Frontend függőségek telepítése**

1. Győződj meg róla, hogy Node.js telepítve van:

node --version

1. Telepítsd a frontend függőségeket:

npm install

1. Buildeld a frontend asseteket:
   * Fejlesztői mód:

npm run dev

* + Éles környezetre:

npm run build

**3.7 Alkalmazás indítása**

1. Indítsd el a Laravel beépített fejlesztői szerverét:

artisan serve

1. Nyisd meg az alkalmazást a böngészőben:

arduino

http://localhost:8000

**3.8 Hibakeresés**

* Hiba esetén: Ellenőrizd a logokat:

tail -f storage/logs/laravel.log

* Ellenőrizd a környezeti változók helyességét.

### 4. Használt csomagok és keretrendszerek

Ebben a részben részletesen bemutatjuk a **PiciPiac** Laravel-projektben használt csomagokat és keretrendszereket, valamint azok célját és funkcióit. Ezek a csomagok biztosítják az alkalmazás funkcionalitását, biztonságát és bővíthetőségét.

**4.1 Laravel alapcsomagok**

* **Laravel Framework (v11.35.0)**:
  + A php nyelv népszerű webalkalmazás-keretrendszere, amely tiszta kódolást, moduláris felépítést és egyszerű fejlesztést biztosít.
  + Főbb funkciók:
    - Eloquent ORM (adatbázis-relációk kezelése)
    - Blade sablonmotor
    - Beépített migrációk és seeder-ek
    - RESTful API támogatás

**4.2 Telepített Composer csomagok**

A következő csomagok telepítése történt a projektben:

1. **laravel/ui**:
   * Hitelesítési (auth) scaffoldot biztosít egyszerű bejelentkezési és regisztrációs felületekhez.
   * Használat:

artisan ui bootstrap --auth

1. **laravel/sanctum**:
   * RESTful API autentikációt biztosít session tokenekkel és egyszerű stateless API hitelesítéssel.
   * Példa: API végpontok védelme middleware segítségével.
2. **spatie/laravel-permission**:
   * Felhasználók szerepkörök és jogosultságok alapján történő kezelésére szolgál.
   * Példa:

$user->assignRole('admin');

$user->givePermissionTo('edit products');

1. **faker/faker**:
   * Tesztadatok generálására szolgál, például felhasználók vagy termékek létrehozásakor.
2. **guzzlehttp/guzzle**:
   * Külső API-k hívására és integrálására használatos HTTP kliens.
   * Példa:

$response = Http::get('https://api.example.com/products');

1. **barryvdh/laravel-debugbar** (fejlesztési környezethez):
   * Hibakeresést és teljesítményelemzést tesz lehetővé egy felhasználói felületen keresztül.

**4.3 Frontend keretrendszerek és eszközök**

A projekt frontend részét a következő eszközök támogatják:

1. **Bootstrap**:
   * Reszponzív webes felület kialakítására használt CSS keretrendszer.
   * Verzió: 5.x.
   * Telepítés:

npm install bootstrap

1. **Sass**:
   * CSS preprocesszor, amely lehetővé teszi a hatékonyabb stíluskezelést (változók, mixinek, stb.).
   * Telepítés:

npm install sass

1. **Vue.js (opcionális)**:
   * Modern frontend JavaScript keretrendszer a dinamikus komponensekhez.
   * Laravel által támogatott, de a projektben nem kötelező elem.

**4.4 Fejlesztői eszközök**

1. **Tinker**:
   * Laravel konzol eszköze az Eloquent modellek és parancsok gyors tesztelésére.
   * Parancs:

artisan tinker

1. **Laravel Debugbar**:
   * Hasznos a hibák, queryk és teljesítményproblémák nyomon követésére.
   * Csak fejlesztési környezetben aktív.

**4.5 Telepítés és verzióellenőrzés**

* A csomagok telepítése a composer. fájl alapján történik:

composer install

* Verziók ellenőrzése:

composer show

### 5. Adatbázis struktúra

Az adatbázis a **PiciPiac** Laravel-alapú alkalmazás adatkezelésének központi eleme. Ebben a részben bemutatjuk a táblákat, azok oszlopait, valamint a táblák közötti relációkat.

**5.1 Táblák és oszlopok**

**1. Felhasználók táblája (users)**

Ez a tábla tárolja az alkalmazás felhasználóit (vásárlók, eladók, adminisztrátorok).

* **Oszlopok**:
  + id: Egyedi azonosító (PRIMARY KEY).
  + name: Felhasználó teljes neve.
  + email: E-mail cím (egyedi).
  + password: Jelszó (titkosítva).
  + role: Felhasználó szerepe (user, seller, admin).
  + created\_at: Rekord létrehozásának időpontja.
  + updated\_at: Rekord utolsó módosításának időpontja.

**2. Termékek táblája (products)**

Ez a tábla tartalmazza a kézműves termékeket, zöldségeket és gyümölcsöket.

* **Oszlopok**:
  + id: Egyedi azonosító (PRIMARY KEY).
  + name: Termék neve.
  + description: Termék leírása.
  + price: Termék ára (decimális érték, pl. 1200.50).
  + stock: Készleten lévő darabszám.
  + category\_id: Kategória azonosítója (FOREIGN KEY).
  + seller\_id: Eladó azonosítója (FOREIGN KEY, users táblára mutat).
  + created\_at, updated\_at.

**3. Kategóriák táblája (categories)**

A termékeket kategóriákba sorolja.

* **Oszlopok**:
  + id: Egyedi azonosító (PRIMARY KEY).
  + name: Kategória neve (pl. "Zöldség", "Gyümölcs").
  + created\_at, updated\_at.

**4. Megrendelések táblája (orders)**

Ez a tábla tárolja a leadott megrendeléseket.

* **Oszlopok**:
  + id: Egyedi azonosító (PRIMARY KEY).
  + user\_id: Vásárló azonosítója (FOREIGN KEY, users táblára mutat).
  + status: Megrendelés státusza (pl. pending, completed, cancelled).
  + total: Megrendelés végösszege.
  + created\_at, updated\_at.

**5. Megrendelési tételek táblája (order\_items)**

Ez a tábla rögzíti a megrendelések termékeit.

* **Oszlopok**:
  + id: Egyedi azonosító (PRIMARY KEY).
  + order\_id: Megrendelés azonosítója (FOREIGN KEY, orders táblára mutat).
  + product\_id: Termék azonosítója (FOREIGN KEY, products táblára mutat).
  + quantity: Rendelési mennyiség.
  + price: Rögzített ár a rendelés leadásakor.

**5.2 Táblák közötti relációk**

**1. Felhasználók és megrendelések (users ↔ orders)**

* Egy felhasználó több megrendelést is leadhat (1:N kapcsolat).
* **Reláció a modellek között**:

// User model

public function orders()

{

return $this->hasMany(Order::class);

}

**2. Megrendelések és tételek (orders ↔ order\_items)**

* Egy megrendelés több terméket tartalmazhat (1:N kapcsolat).
* **Reláció a modellek között**:

// Order model

public function items()

{

return $this->hasMany(OrderItem::class);

}

**3. Termékek és kategóriák (products ↔ categories)**

* Egy termék egy kategóriához tartozik, de egy kategóriának több terméke lehet (N:1 kapcsolat).
* **Reláció a modellek között**:

// Product model

public function category()

{

return $this->belongsTo(Category::class);

}

**4. Termékek és eladók (products ↔ users)**

* Egy eladó több terméket kínálhat (1:N kapcsolat).
* **Reláció a modellek között**:

// Product model

public function seller()

{

return $this->belongsTo(User::class, 'seller\_id');

}

**5.3 Adatbázis diagram**

Egy adatbázisdiagram segítségével vizuálisan ábrázolhatjuk a táblák közötti kapcsolatokat. Példa egy egyszerű EER (Entity-Relationship Diagram) leírásra:

lua

[users]

| 1

|---< orders

| 1

|---< order\_items

| 1

|---< products >--- categories

### 6. MVC Struktúra

A Laravel keretrendszer **MVC (Model-View-Controller)** architektúrája biztosítja a kód strukturált és könnyen kezelhető felépítését. Ez a szekció részletesen bemutatja a **modelleket**, **view-kat** és **kontrollereket**, amelyek a **PiciPiac** projektben használatosak.

**6.1 Modellek (app/Models/)**

A modellek az adatbázistáblák Eloquent ORM-en keresztüli kezelését biztosítják, valamint a relációk és üzleti logika egy részének megvalósítását.

**Fontosabb modellek:**

1. **User.**:
   * A users tábla reprezentációja.
   * **Főbb attribútumok**: id, name, email, password, role.
   * **Relációk**:
     + Egy felhasználónak több megrendelése lehet:

public function orders()

{

return $this->hasMany(Order::class);

}

* + - Ha eladó, több terméke lehet:

public function products()

{

return $this->hasMany(Product::class, 'seller\_id');

}

1. **Product.**:
   * A products tábla reprezentációja.
   * **Főbb attribútumok**: id, name, description, price, stock, category\_id, seller\_id.
   * **Relációk**:
     + Egy termék egy kategóriához tartozik:

public function category()

{

return $this->belongsTo(Category::class);

}

* + - Egy terméknek van egy eladója:

public function seller()

{

return $this->belongsTo(User::class, 'seller\_id');

}

1. **Order.**:
   * A orders tábla reprezentációja.
   * **Főbb attribútumok**: id, user\_id, status, total.
   * **Relációk**:
     + Egy megrendelés több tételből állhat:

public function items()

{

return $this->hasMany(OrderItem::class);

}

* + - Egy megrendelés egy felhasználóhoz tartozik:

public function user()

{

return $this->belongsTo(User::class);

}

**6.2 Kontrollerek (app/Http/Controllers/)**

A kontrollerek felelnek az üzleti logika feldolgozásáért és az adatok átadásáért a view-k számára.

**Fontosabb kontrollerek:**

1. **AuthController**:
   * **Funkciók**:
     + Felhasználó regisztráció:

public function register(Request $request)

{

$validated = $request->validate([

'name' => 'required',

'email' => 'required|email|unique:users',

'password' => 'required|confirmed|min:8',

]);

$user = User::create([

'name' => $validated['name'],

'email' => $validated['email'],

'password' => bcrypt($validated['password']),

]);

return response()->(['user' => $user], 201);

}

* + - Bejelentkezés:

public function login(Request $request)

{

if (auth()->attempt($request->only('email', 'password'))) {

return response()->(['message' => 'Login successful']);

}

return response()->(['message' => 'Invalid credentials'], 401);

}

1. **ProductController**:
   * **Funkciók**:
     + Termék lista:

public function index()

{

$products = Product::with('category', 'seller')->get();

return view('products.index', compact('products'));

}

* + - Új termék létrehozása:

public function store(Request $request)

{

$validated = $request->validate([

'name' => 'required',

'price' => 'required|numeric',

'category\_id' => 'required|exists:categories,id',

]);

Product::create($validated + ['seller\_id' => auth()->id()]);

return redirect()->back()->with('success', 'Termék sikeresen hozzáadva!');

}

1. **OrderController**:
   * **Funkciók**:
     + Megrendelések listázása:

public function index()

{

$orders = Order::with('items.product')->where('user\_id', auth()->id())->get();

return view('orders.index', compact('orders'));

}

* + - Új megrendelés létrehozása:

public function store(Request $request)

{

$order = Order::create([

'user\_id' => auth()->id(),

'status' => 'pending',

'total' => $request->total,

]);

foreach ($request->items as $item) {

$order->items()->create($item);

}

return redirect()->route('orders.index')->with('success', 'Megrendelés sikeresen leadva!');

}

**6.3 Nézetek (View-k)**

A nézetek Blade sablonok formájában vannak megvalósítva, és a resources/views/ mappában találhatók.

**Főbb nézetek:**

1. **Autentikáció (auth/)**:
   * **login.blade.**: Bejelentkezési oldal.
   * **register.blade.**: Regisztrációs oldal.
2. **Termékek (products/)**:
   * **index.blade.**: Terméklista megjelenítése.
   * **show.blade.**: Egy adott termék részletei.
3. **Megrendelések (orders/)**:
   * **index.blade.**: A felhasználó összes megrendelése.
   * **details.blade.**: Egy adott megrendelés részletei.

**Blade sablon funkciók:**

* **@foreach** ciklus:

blade

@foreach ($products as $product)

<div>{{ $product->name }} - {{ $product->price }} Ft</div>

@endforeach

* **Komponensek**:
  + Példa: **navbar komponens** (resources/views/components/navbar.blade.):

blade

<nav>

<a href="/">Főoldal</a>

<a href="/products">Termékek</a>

<a href="/cart">Kosár</a>

</nav>

### 7. CRUD Műveletek és Objektumok

Ebben a szekcióban bemutatjuk, hogyan valósulnak meg a **CRUD (Create, Read, Update, Delete)** műveletek a **PiciPiac** Laravel projektben. A CRUD műveletek az alkalmazás alapvető funkciói, amelyek lehetővé teszik az adatok kezelését.

**7.1 CRUD Műveletek Példák**

**1. CREATE – Új adat létrehozása**

Példa: **Új termék hozzáadása**

* **Route (útvonal):**

Route::post('/products', [ProductController::class, 'store'])->name('products.store');

* **Kontroller metódus:**

public function store(Request $request)

{

$validated = $request->validate([

'name' => 'required|string|max:255',

'description' => 'nullable|string',

'price' => 'required|numeric|min:0',

'stock' => 'required|integer|min:0',

'category\_id' => 'required|exists:categories,id',

]);

Product::create(array\_merge($validated, ['seller\_id' => auth()->id()]));

return redirect()->route('products.index')->with('success', 'Új termék sikeresen hozzáadva!');

}

* **Nézet (form):**

blade

<form action="{{ route('products.store') }}" method="POST">

@csrf

<input type="text" name="name" placeholder="Termék neve" required>

<input type="text" name="description" placeholder="Leírás">

<input type="number" name="price" placeholder="Ár" required>

<input type="number" name="stock" placeholder="Készlet" required>

<select name="category\_id" required>

@foreach ($categories as $category)

<option value="{{ $category->id }}">{{ $category->name }}</option>

@endforeach

</select>

<button type="submit">Hozzáadás</button>

</form>

**2. READ – Adatok lekérdezése**

Példa: **Termékek listázása**

* **Route (útvonal):**

Route::get('/products', [ProductController::class, 'index'])->name('products.index');

* **Kontroller metódus:**

public function index()

{

$products = Product::with('category', 'seller')->get();

return view('products.index', compact('products'));

}

* **Nézet:**

blade

<ul>

@foreach ($products as $product)

<li>

<strong>{{ $product->name }}</strong> - {{ $product->price }} Ft

<br>

Kategória: {{ $product->category->name }}

<br>

Eladó: {{ $product->seller->name }}

</li>

@endforeach

</ul>

**3. UPDATE – Adatok frissítése**

Példa: **Termék szerkesztése**

* **Route (útvonal):**

Route::put('/products/{product}', [ProductController::class, 'update'])->name('products.update');

* **Kontroller metódus:**

public function update(Request $request, Product $product)

{

$validated = $request->validate([

'name' => 'required|string|max:255',

'description' => 'nullable|string',

'price' => 'required|numeric|min:0',

'stock' => 'required|integer|min:0',

'category\_id' => 'required|exists:categories,id',

]);

$product->update($validated);

return redirect()->route('products.index')->with('success', 'Termék sikeresen frissítve!');

}

* **Nézet (form):**

blade

<form action="{{ route('products.update', $product->id) }}" method="POST">

@csrf

@method('PUT')

<input type="text" name="name" value="{{ $product->name }}" required>

<input type="text" name="description" value="{{ $product->description }}">

<input type="number" name="price" value="{{ $product->price }}" required>

<input type="number" name="stock" value="{{ $product->stock }}" required>

<select name="category\_id" required>

@foreach ($categories as $category)

<option value="{{ $category->id }}" {{ $category->id == $product->category\_id ? 'selected' : '' }}>

{{ $category->name }}

</option>

@endforeach

</select>

<button type="submit">Frissítés</button>

</form>

**4. DELETE – Adatok törlése**

Példa: **Termék törlése**

* **Route (útvonal):**

Route::delete('/products/{product}', [ProductController::class, 'destroy'])->name('products.destroy');

* **Kontroller metódus:**

public function destroy(Product $product)

{

$product->delete();

return redirect()->route('products.index')->with('success', 'Termék sikeresen törölve!');

}

* **Nézet (gomb):**

blade

<form action="{{ route('products.destroy', $product->id) }}" method="POST">

@csrf

@method('DELETE')

<button type="submit" onclick="return confirm('Biztosan törlöd?')">Törlés</button>

</form>

**7.2 Objektumok és fontosabb függvények**

1. **Eloquent modellek használata**:
   * Adatbázis-relációk könnyű kezelése.
   * Példa: Összes termék lekérdezése egy adott kategóriában:

$products = Product::where('category\_id', $categoryId)->get();

1. **Middleware-ek**:
   * Példa: Autentikációs middleware biztosítása az elérési útvonalaknál:

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

Route::get('/orders', [OrderController::class, 'index']);

});

1. **Helper függvények**:
   * Egyedi függvények a globális használatra:

function formatPrice($price)

{

return number\_format($price, 2, ',', '.') . ' Ft';

}

### 8. RESTful API

A **PiciPiac** Laravel-alapú alkalmazás RESTful API támogatással rendelkezik, amely lehetővé teszi az adatok kezelését és a funkciók biztosítását külső kliensek, például mobilalkalmazások vagy frontend rendszerek számára. Ebben a szekcióban részletesen bemutatjuk az API végpontokat, a kérések és válaszok struktúráját, valamint a hitelesítési mechanizmusokat.

**8.1 Autentikáció és jogosultság**

A RESTful API autentikációja a **Laravel Sanctum** csomaggal van implementálva. Ez lehetővé teszi a **session alapú autentikációt** és a **stateless tokenek** használatát.

* **Token alapú autentikáció**:
  + Minden API végpont elérése előtt a felhasználónak be kell jelentkeznie, és egy hozzá tartozó **token** szükséges.
  + **Példa token generálásra**:

public function login(Request $request)

{

$validated = $request->validate([

'email' => 'required|email',

'password' => 'required',

]);

if (!auth()->attempt($validated)) {

return response()->(['message' => 'Invalid credentials'], 401);

}

$token = auth()->user()->createToken('api-token')->plainTextToken;

return response()->(['token' => $token], 200);

}

* **Middleware védelem**:
  + Az API végpontokat az autentikáció middleware védi:

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

Route::get('/products', [ProductController::class, 'index']);

});

**8.2 API végpontok**

**1. Felhasználói autentikáció**

* **POST /api/login**
  + **Leírás**: Felhasználó bejelentkezése és token generálása.
  + **Kérelem minta**:

{

"email": "user@example.com",

"password": "password123"

}

* + **Válasz minta**:

{

"token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9..."

}

* **POST /api/logout**
  + **Leírás**: Felhasználó kijelentkezése és token érvénytelenítése.
  + **Middleware**: auth:sanctum

**2. Termékek kezelése**

* **GET /api/products**
  + **Leírás**: Termékek listázása.
  + **Middleware**: auth:sanctum
  + **Válasz minta**:

[

{

"id": 1,

"name": "Alma",

"description": "Friss, bio alma.",

"price": 1200,

"stock": 50,

"category": "Gyümölcsök",

"seller": "Kiss Péter"

},

{

"id": 2,

"name": "Méz",

"description": "Helyi termelői méz.",

"price": 3500,

"stock": 20,

"category": "Kézműves termékek",

"seller": "Nagy János"

}

]

* **POST /api/products**
  + **Leírás**: Új termék hozzáadása.
  + **Middleware**: auth:sanctum
  + **Kérelem minta**:

{

"name": "Répa",

"description": "Bio répa.",

"price": 800,

"stock": 100,

"category\_id": 1

}

* + **Válasz minta**:

{

"message": "Termék sikeresen hozzáadva.",

"product": {

"id": 3,

"name": "Répa",

"description": "Bio répa.",

"price": 800,

"stock": 100,

"category\_id": 1

}

}

**3. Megrendelések kezelése**

* **GET /api/orders**
  + **Leírás**: A bejelentkezett felhasználó megrendeléseinek listázása.
  + **Middleware**: auth:sanctum
  + **Válasz minta**:

[

{

"id": 1,

"status": "pending",

"total": 4200,

"items": [

{

"product": "Alma",

"quantity": 2,

"price": 1200

},

{

"product": "Méz",

"quantity": 1,

"price": 3500

}

]

}

]

* **POST /api/orders**
  + **Leírás**: Új megrendelés leadása.
  + **Middleware**: auth:sanctum
  + **Kérelem minta**:

{

"items": [

{

"product\_id": 1,

"quantity": 2

},

{

"product\_id": 2,

"quantity": 1

}

],

"total": 4200

}

* + **Válasz minta**:

{

"message": "Megrendelés sikeresen létrehozva.",

"order\_id": 1

}

**8.3 Tesztelés és dokumentáció**

**1. API tesztelés**

* **Postman**:
  + Postman segítségével egyszerűen tesztelhetők az API végpontok.
  + Hozz létre egy **Postman Collection-t**, amely tartalmazza a végpontokat és a kérések példáit.

**2. Automatikus dokumentáció:**

1. **Laravel API Resource-ok**:
   * Az API Resource segítségével szabványos formátumot biztosíthatsz a válaszokhoz:

return new ProductResource($product);

1. **Swagger/OpenAPI**:
   * A **Swagger** generálja az API végpontok dokumentációját.
   * Telepítés:

bash

composer require "darkaonline/l5-swagger"

* + Dokumentáció generálása:

bash

artisan l5-swagger:generate

* + A dokumentáció elérhető: http://localhost/api/documentation.

### 9. Fejlesztési lehetőségek

A **PiciPiac** projekt folyamatosan fejlődhet és bővülhet új funkciókkal. A fejlesztési lehetőségek szekció célja, hogy irányt mutasson azok számára, akik további fejlesztéseket szeretnének végezni, vagy új funkciókat szeretnének hozzáadni az alkalmazáshoz.

**9.1 Új funkciók bevezetése**

1. **Termékek kategorizálása**:
   * Jelenleg a termékek csak alapadatokkal rendelkeznek, de a kategóriák hozzáadása jelentősen javítaná a kereshetőséget és a felhasználói élményt.
   * **Lépések**:
     + Hozz létre egy új categories táblát.
     + Add hozzá a category\_id mezőt a products táblához.
     + Implementálj egy **CategoryController**-t a kategóriák kezeléséhez.
     + Készíts front-end szűrőt a kategóriák szerint történő kereséshez.
   * **Példa migráció**:

public function up()

{

Schema::create('categories', function (Blueprint $table) {

$table->id();

$table->string('name');

$table->timestamps();

});

}

1. **Felhasználói profilok**:
   * Lehetőség a felhasználók számára, hogy bővebb információkat adjanak meg magukról (pl. profilkép, szállítási címek).
   * **Lépések**:
     + Adjon hozzá új mezőket a users táblához: profile\_picture, address.
     + Készíts egy **ProfileController**-t a profilok kezelésére.
     + Készíts egy **profil edit** oldalt, ahol a felhasználók módosíthatják az adataikat.
2. **Hozzászólások és értékelések**:
   * A termékekhez értékeléseket és hozzászólásokat lehetne hozzáadni, hogy a vásárlók véleményt mondjanak a termékekről.
   * **Lépések**:
     + Készíts egy új **reviews** táblát, amely kapcsolódik a products és a users táblához.
     + Adj hozzá egy **ReviewController**-t a hozzászólások kezeléséhez.
     + Az értékeléseket a frontend-en is megjelenítheted.

**9.2 Tesztelési stratégiák**

1. **Egységtesztek**:
   * A Laravel beépített tesztelési eszközeit használhatod az üzleti logika tesztelésére.
   * Példa:
     + Ellenőrizd, hogy egy termék helyesen kerül-e hozzáadásra:

public function testProductCreation()

{

$response = $this->post('/api/products', [

'name' => 'Alma',

'price' => 1200,

'stock' => 50,

'category\_id' => 1

]);

$response->assertStatus(201);

$this->assertDatabaseHas('products', [

'name' => 'Alma'

]);

}

1. **Integrációs tesztek**:
   * Teszteld az alkalmazás különböző részeinek kölcsönhatását, például az autentikáció és a termék hozzáadása közötti folyamatot.
   * Példa:
     + Ellenőrizd, hogy bejelentkezett felhasználó hozzáadhat-e terméket:

public function testAuthenticatedUserCanAddProduct()

{

$user = User::factory()->create();

$this->actingAs($user);

$response = $this->post('/api/products', [

'name' => 'Banán',

'price' => 800,

'stock' => 30,

'category\_id' => 1

]);

$response->assertStatus(201);

}

1. **API tesztelés**:
   * Használj **Postman** vagy **Unit** a RESTful API végpontok tesztelésére.
   * Készíts tesztkéréseket az API végpontok ellenőrzésére, és győződj meg arról, hogy a válaszok megfelelnek az elvárásoknak.

**9.3 Teljesítmény optimalizálás**

1. **Adatbázis indexelés**:
   * Az adatbázis lekérdezések gyorsítása érdekében indexeket adhatsz a leggyakrabban keresett mezőkhöz, mint például a price, category\_id, stock és created\_at.
   * **Példa migráció**:

public function up()

{

Schema::table('products', function (Blueprint $table) {

$table->index('category\_id');

$table->index('price');

});

}

1. **Cache használata**:
   * A Laravel beépített **cache** mechanizmusát használhatod a gyakran lekért adatok gyorsításához (például termékek listázása, kategóriák).
   * **Példa cache használatára**:

$products = Cache::remember('products', 60, function () {

return Product::all();

});

1. **Adatbázis optimalizálás**:
   * Az adatbázis lekérdezések finomhangolása érdekében alkalmazhatsz **Eager Loading**-ot, hogy elkerüld a "N+1" problémát.
   * **Példa Eager Loading**:

$products = Product::with('category', 'seller')->get();

**9.4 Biztonsági fejlesztések**

1. **Bevitel ellenőrzése és tisztítása**:
   * Mindig ellenőrizd és tisztítsd meg a felhasználói beviteleket az űrlapokból és API kérésekből.
   * **Laravel Validation** használata az adatbevitel védelme érdekében:

$validated = $request->validate([

'name' => 'required|string|max:255',

'price' => 'required|numeric|min:0',

]);

1. **XSS és CSRF védelem**:
   * A Laravel automatikusan védi a cross-site scripting (XSS) és cross-site request forgery (CSRF) támadások ellen, de érdemes mindig frissíteni és figyelemmel kísérni a biztonsági best practice-eket.
   * A **Sanctum** biztosítja a CSRF támadások ellen védelmet, ha megfelelően konfigurálod az API-kat.

### 10. Következtetések és ajánlások

Ebben a záró szekcióban összefoglaljuk a **PiciPiac** Laravel-alapú alkalmazás jelenlegi állapotát, a dokumentáció főbb pontjait, és javaslatokat fogalmazunk meg a projekt jövőbeni fejlesztésére és karbantartására.

**10.1 Összefoglaló**

A **PiciPiac** egy modern, Laravel-alapú online piac, amely lehetőséget biztosít kézműves termékek, zöldségek és gyümölcsök vásárlására és értékesítésére. A rendszer főbb funkciói:

* Termékek böngészése, vásárlása és értékesítése.
* Kosár és megrendelés-kezelés.
* RESTful API támogatás a külső kliensek számára.
* Felhasználók autentikációja és jogosultságkezelése.
* Adminisztrációs felület az adatok kezelésére.

A dokumentáció célja, hogy megkönnyítse a fejlesztők munkáját a projekt megértésében, használatában és továbbfejlesztésében. Az MVC architektúra, az adatbázis struktúra, a RESTful API, valamint a CRUD műveletek bemutatása hozzájárul ahhoz, hogy az alkalmazás bővítése és karbantartása hatékonyan történhessen.

**10.2 Javaslatok a fejlesztőknek**

1. **Kód karbantartás**:
   * Rendszeresen ellenőrizd a kódbázist a nem használt függvények, csomagok és modellek eltávolítására.
   * Használj statikus kódelemző eszközöket, például **Stan** vagy **Laravel Pint**, hogy biztosítsd a kód minőségét.
2. **Dokumentáció frissítése**:
   * A projekt bővítésekor vagy új funkciók hozzáadásakor gondoskodj arról, hogy a dokumentáció is naprakész legyen (pl. új RESTful API végpontok, adatbázis-módosítások).
   * A Swagger vagy Postman dokumentációkat rendszeresen regeneráld.
3. **Tesztelési folyamatok bővítése**:
   * A jelenlegi egység- és integrációs teszteket egészítsd ki több edge case teszttel.
   * Automatizáld a tesztelési folyamatot CI/CD pipeline segítségével (pl. GitHub Actions, GitLab CI).
4. **Teljesítmény és skálázhatóság**:
   * Implementáld az adatbázis lekérdezések gyorsítását, például indexekkel és cache-eléssel.
   * Használj horizontális skálázási megoldásokat (pl. külső API vagy microservices bevezetése az igények növekedése esetén).
5. **Felhasználói élmény javítása**:
   * Hozz létre fejlettebb keresési funkciókat (pl. kategóriák szerinti szűrés, ajánlások).
   * Integrálj visszajelzéseket (értékelések, kommentek), hogy a vásárlók jobban tájékozódhassanak.

**10.3 Jövőbeni fejlesztések**

1. **Mobilalkalmazás fejlesztése**:
   * Az API kiterjesztése lehetővé teszi mobilalkalmazások (iOS/Android) létrehozását a vásárlók és eladók számára.
2. **Fizetési opciók integrálása**:
   * A népszerű fizetési megoldások, például PayPal, Stripe vagy Barion integrálása bővíthetné a vásárlók lehetőségeit.
3. **Analitika és riportok**:
   * Hozz létre részletes adminisztrátori riportokat a forgalom, bevételek és trendek nyomon követésére.
4. **Nemzetközi piacra lépés**:
   * Többnyelvűség implementálása a Laravel **localization** támogatásával.
   * Valutaváltó funkció bevezetése a nemzetközi vásárlók számára.

**10.4 Ajánlások a projekt csapatának**

* **Fejlesztői csapat**: Biztosíts folyamatos képzéseket a Laravel új verzióival és best practice-eivel kapcsolatban.
* **Projektmenedzsment**: Alakíts ki hatékony kommunikációs csatornákat a fejlesztői és a tervezési csapat között.
* **Felhasználói visszajelzések gyűjtése**: Folyamatosan figyelj a felhasználók igényeire, és priorizáld a legfontosabb funkciókat.