



DOKUMENTACJA TECHNICZNA

Aplikacja do obsługi magazynów



29 GRUDNIA 2022

WSB – INFORMATYKA NIESTACJONARNIE
Grupa pracy inżynierskiej

Spis treści

WPROWADZENIE	2
Informacje ogólne	2
Sposób instalacji na serwerze lokalnym	3
Importowanie plików aplikacji	3
Uruchomienie środowiska aplikacji	4
Pierwsze uruchomienie aplikacji	8
Strona powitalna	8
Strona główna po zalogowaniu	10
Moduł 'Produkty'	11
Moduł 'Magazyny'	13
SPECYFIKACJA TECHNICZNA	15
Dane techniczne	15
Baza danych	16
Schemat bazy danych	16
Struktura poszczególnych tabel	17
Kod źródłowy	19
Migracje	19
Seeding	21
Modele	22
Kontrolery	24
Repozytoria	27
Routes	30

WPROWADZENIE

Informacje ogólne

Aplikacja internetowa umożliwia zarządzanie asortymentem w poszczególnych magazynach. Użytkownik ma możliwość dodania produktu oraz określenia jego miejsca przechowywania. Docelowym odbiorcą naszego systemu mają być niewielkie działy logistyczne w małych firmach, które chcą usystematyzować pracę personelu i mieć przejrzysty wgląd w działanie ów działu.

System składa się z następujących elementów:

1. **Serwera lokalnego** – komputera z zainstalowanym systemem Microsoft Windows w wersji Windows 10 lub 11 (występuje także możliwość zainstalowania aplikacji na zewnętrznym serwerze, dzięki któremu aplikacja stanie się ogólnodostępna). Na serwerze lokalnym pracują aplikacje:
 - XAMPP Control Panel – do zarządzania bazą danych, serwerem Apache oraz jako interpreter języka PHP
 - Composer – do zarządzania pakietami dla języka PHP (aplikacja wiersza poleceń)
 - Laravel – PHP Framework do aplikacji internetowych napisany w języku PHP
 - GitHub Desktop – kontrola wersji Git, klonowanie repozytorium z serwisu GitHub
2. **Aplikacji „Magazyny”** stanowiącej zasadniczy element systemu, napisanej w języku PHP przy użyciu Framework PHP Laravel.

Dostęp do aplikacji możliwy jest poprzez przeglądarkę internetową z poziomu serwera lokalnego lub po wdrożeniu aplikacji na serwer zewnętrzny z poziomu Internetu.

Sposób instalacji na serwerze lokalnym

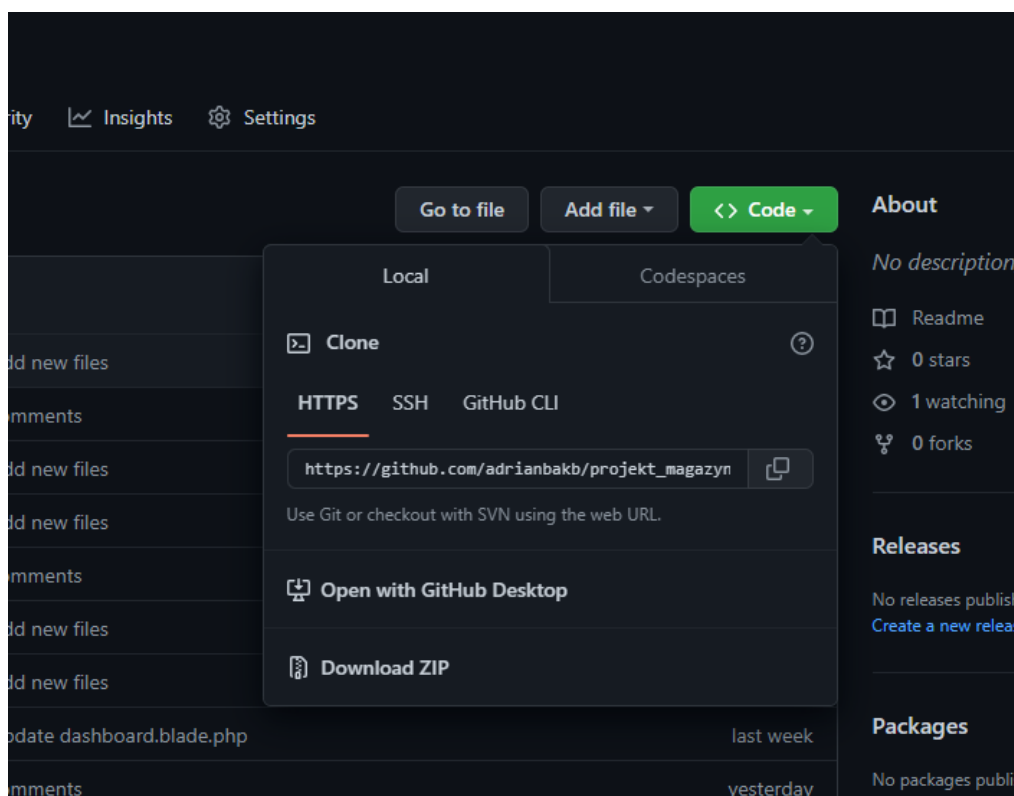
Aplikacja została zaimportowana na hostingowy serwis internetowy przeznaczony do projektów programistycznych wykorzystujący system kontroli wersji Git – Git Hub. Repozytorium aplikacji znajduje się pod wskazanym poniżej linkiem do serwisu Git Hub.

https://github.com/adrianbakb/projekt_magazyn.git

Importowanie plików aplikacji

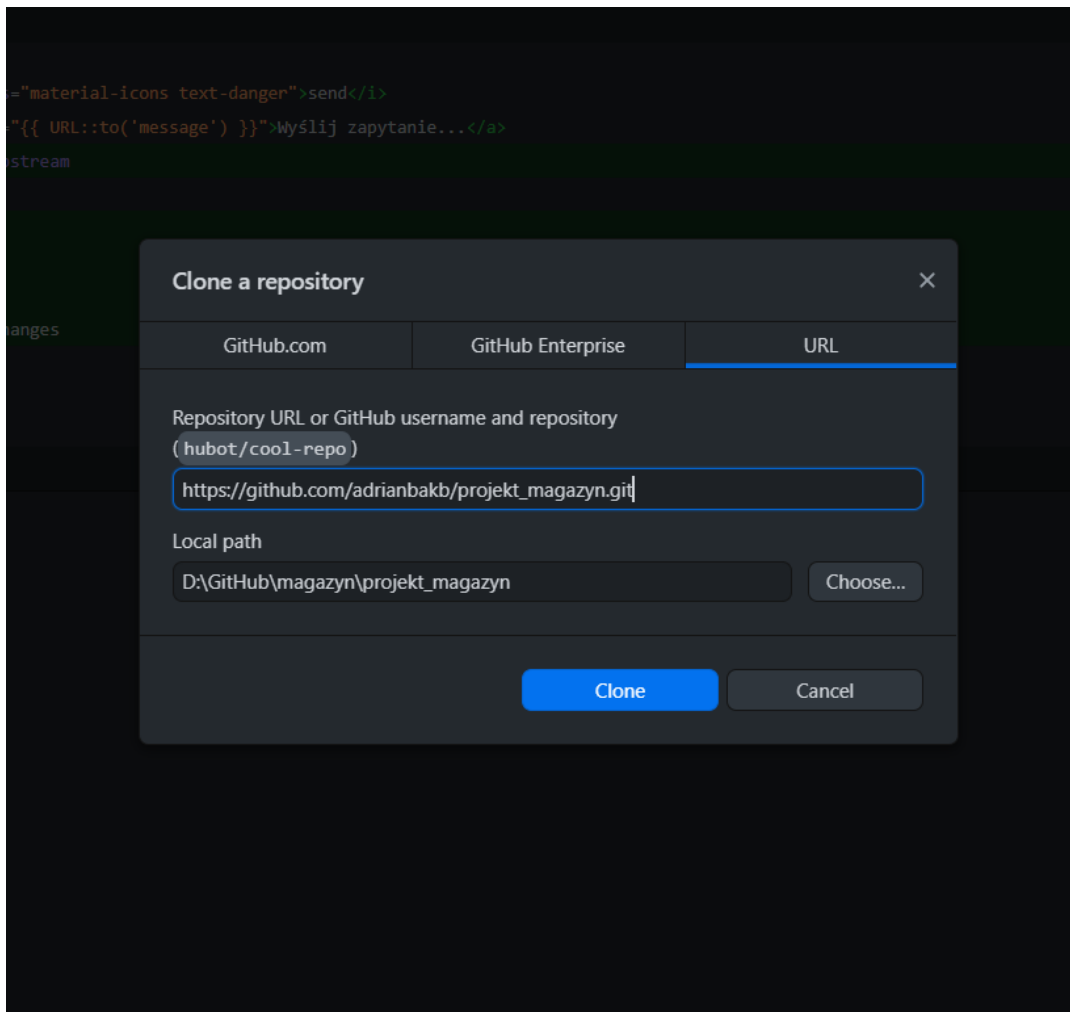
Występują dwa sposoby zaimportowania aplikacji na dysk lokalny:

1. Pobranie aplikacji jako archiwum ZIP oraz rozpakowanie go w docelowym miejscu – **Download ZIP**



Rysunek 1 - pobieranie archiwum ZIP

2. Przy pomocy aplikacji **GitHub Desktop** zaimportowanie (sklonowanie) repozytorium aplikacji na dysk lokalny używając wcześniej skopiowany link do repozytorium

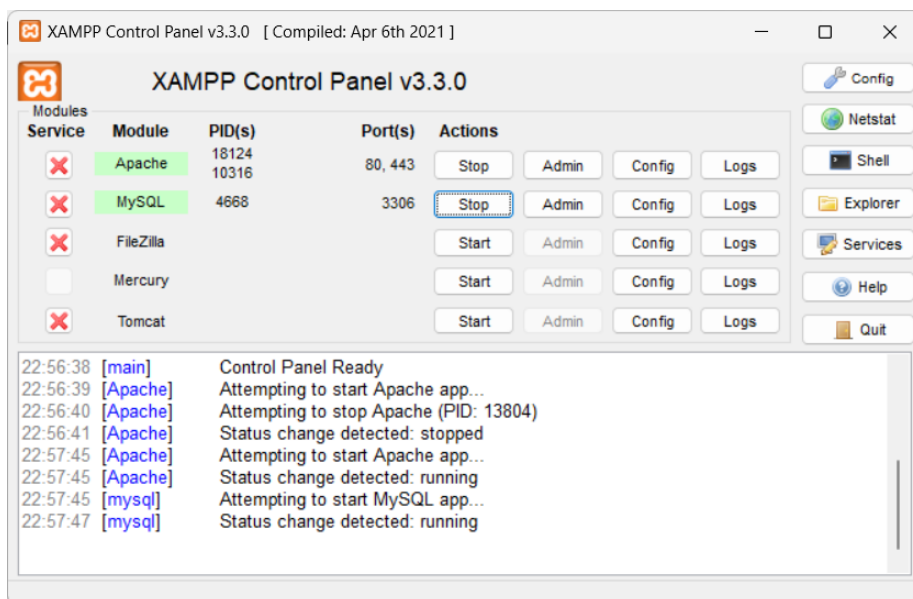


Rysunek 2 - klonowanie repozytorium GitHub Desktop

Uruchomienie środowiska aplikacji

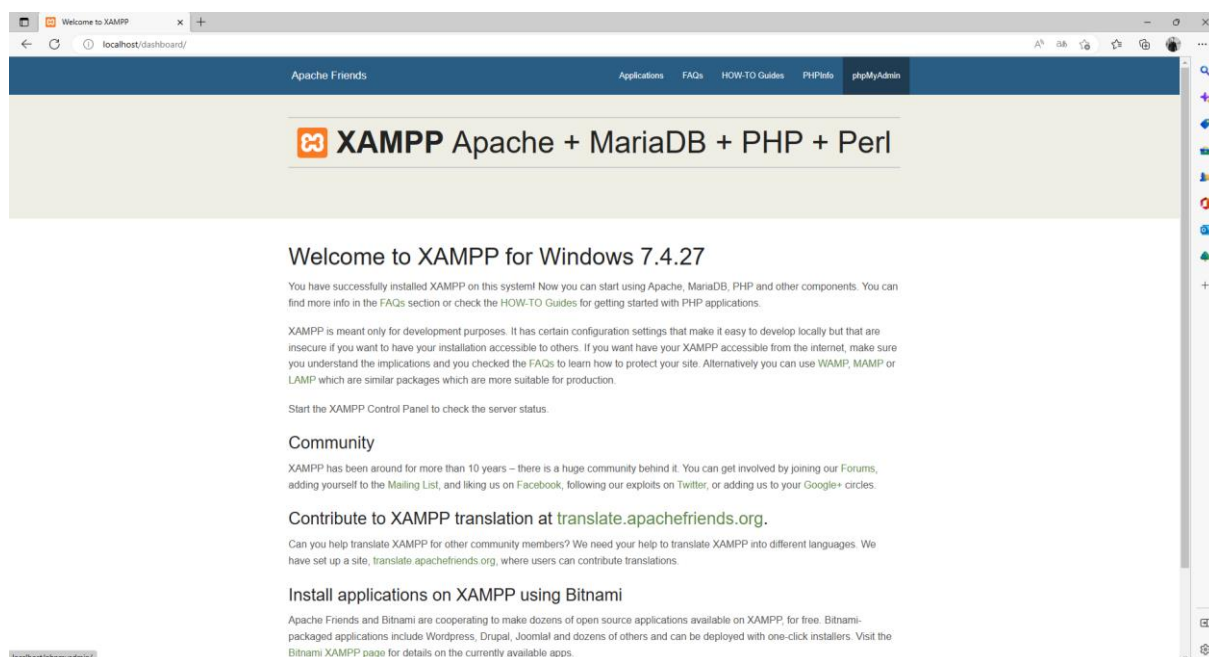
Aby móc rozpocząć korzystanie z aplikacji należy rozpocząć od przygotowania odpowiedniego środowiska dla jej prawidłowego działania.

1. Pierwszym punktem jest uruchomienie aplikacji **XAMPP Control Panel** i wystartowanie serwera **Apache** oraz bazy danych **MySQL**.



Rysunek 3 - Apache i MySQL - Xampp

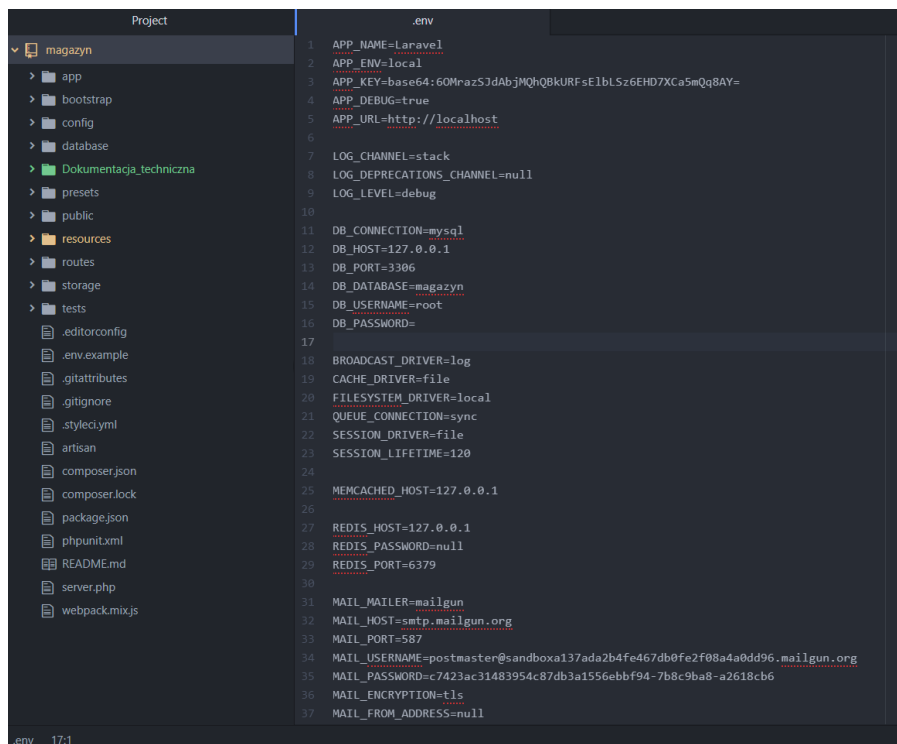
Przy pomocy przeglądarki wchodzimy na lokalną stronę do zarządzania serwerem **XAMPP**. Wpisując 'localhost' pojawi się następująca strona:



Rysunek 4 – localhost – phpMyAdmin

Należy wejść w **phpMyAdmin** oraz utworzyć nową pustą bazę danych.

2. W plikach konfiguracyjnych aplikacji zapisane są parametry związane z konfiguracją serwera bazy danych.

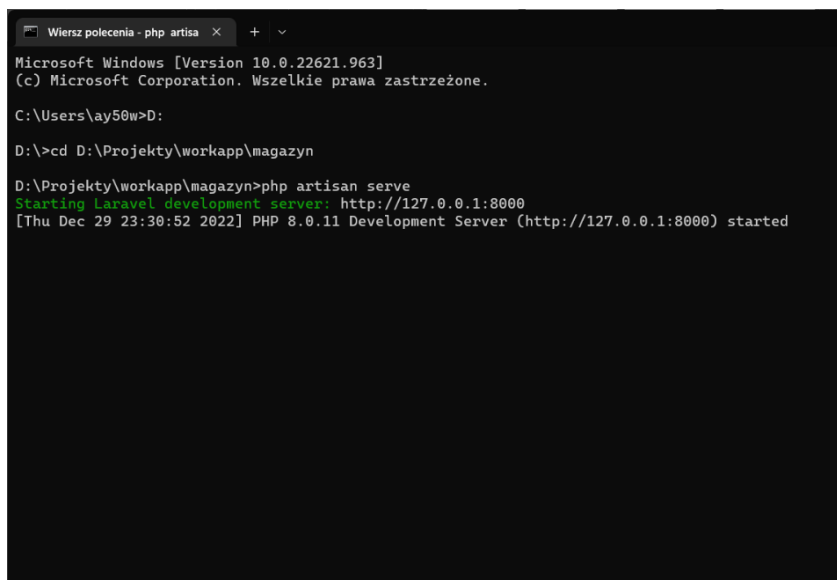


```
1 APP_NAME=Laravel
2 APP_ENV=local
3 APP_KEY=base64:60Mras5JdAbjMQhQ8kURFsElbLSz6EHD7XCas5mQq8AY=
4 APP_DEBUG=true
5 APP_URL=http://localhost
6
7 LOG_CHANNEL=stack
8 LOG_DEPRECATIONS_CHANNEL=null
9 LOG_LEVEL=debug
10
11 DB_CONNECTION=mysql
12 DB_HOST=127.0.0.1
13 DB_PORT=3306
14 DB_DATABASE=magazyn
15 DB_USERNAME=root
16 DB_PASSWORD=
17
18 BROADCAST_DRIVER=log
19 CACHE_DRIVER=file
20 FILESYSTEM_DRIVER=local
21 QUEUE_CONNECTION=sync
22 SESSION_DRIVER=file
23 SESSION_LIFETIME=120
24
25 MEMCACHED_HOST=127.0.0.1
26
27 REDIS_HOST=127.0.0.1
28 REDIS_PASSWORD=null
29 REDIS_PORT=6379
30
31 MAIL_MAILER=mailgun
32 MAIL_HOST=smtp.mailgun.org
33 MAIL_PORT=587
34 MAIL_USERNAME=postmaster@sandboxa137ada2b4fe467db0fe2f08a4a0dd96.mailgun.org
35 MAIL_PASSWORD=c7423ac31483954c87db3a1556ebbf94-7b8c9ba8-a2618cb6
36 MAIL_ENCRYPTION=tls
37 MAIL_FROM_ADDRESS=null
```

Rysunek 5 - konfiguracja pliku .Env

Należy stworzyć plik .env (przekopiować zawartość pliku .env.example) oraz skonfigurować go pod własne wymagania. W tym pliku konfigurujemy między innymi połączenie z utworzoną już bazą danych w **phpMyAdmin**.

3. Następnie należy uruchomić przy pomocy wiersza poleceń serwer **Artisan**. W tym celu uruchamiamy wiersz poleceń, przechodzimy do folderu z aplikacją i wpisujemy polecenie **'php artisan serve'**.

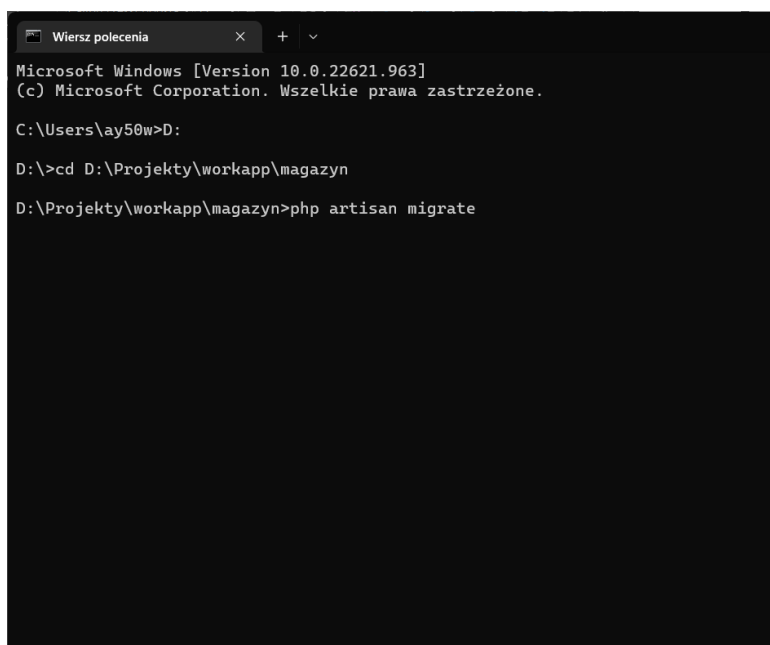


```
Wiersz polecenia - php artisa
Microsoft Windows [Version 10.0.22621.963]
(c) Microsoft Corporation. Wszelkie prawa zastrzeżone.
C:\Users\ay50w>D:
D:\>cd D:\Projekty\workapp\magazyn
D:\Projekty\workapp\magazyn>php artisan serve
Starting Laravel development server: http://127.0.0.1:8000
[Thu Dec 29 23:30:52 2022] PHP 8.0.11 Development Server (http://127.0.0.1:8000) started
```

Rysunek 6 - wiersz poleceń 'php artisan serve'

W tym momencie można już dostać się na stronę aplikacji. Lokalny adres aplikacji to: **127.0.0.1:8000**.

4. Aby aplikacja była w pełni funkcjonalna należy jeszcze dokonać migracji bazy danych poprzez wpisanie polecenia w **nowym oknie** wiersza poleceń **'php artisan migrate'**. Dzięki temu struktura bazy danych zapisana w plikach aplikacji zostanie zaimportowana do utworzonej wcześniej bazy danych.



```
Wiersz polecenia
Microsoft Windows [Version 10.0.22621.963]
(c) Microsoft Corporation. Wszelkie prawa zastrzeżone.
C:\Users\ay50w>D:
D:\>cd D:\Projekty\workapp\magazyn
D:\Projekty\workapp\magazyn>php artisan migrate
```

Rysunek 7 - wiersz poleceń 'php artisan migrate'

Kod aplikacji zawiera również **domyślne dane do zalogowania** dla pierwszego użytkownika:

login: 'admin@admin.pl'

hasło: 'admin@admin.pl'

Aby te dane znalazły się w bazie danych automatycznie, należy po dokonaniu migracji zastosować kolejne polecenie: **'php artisan db:seed' lub 'php artisan migrate: fresh --seed'**. Nie jest to jednak zabieg, który trzeba wykonać. Dodawanie użytkownika może odbyć się poprzez **formularz rejestracji** po uruchomieniu aplikacji.

W tym momencie można już przejść do uruchomienia aplikacji.

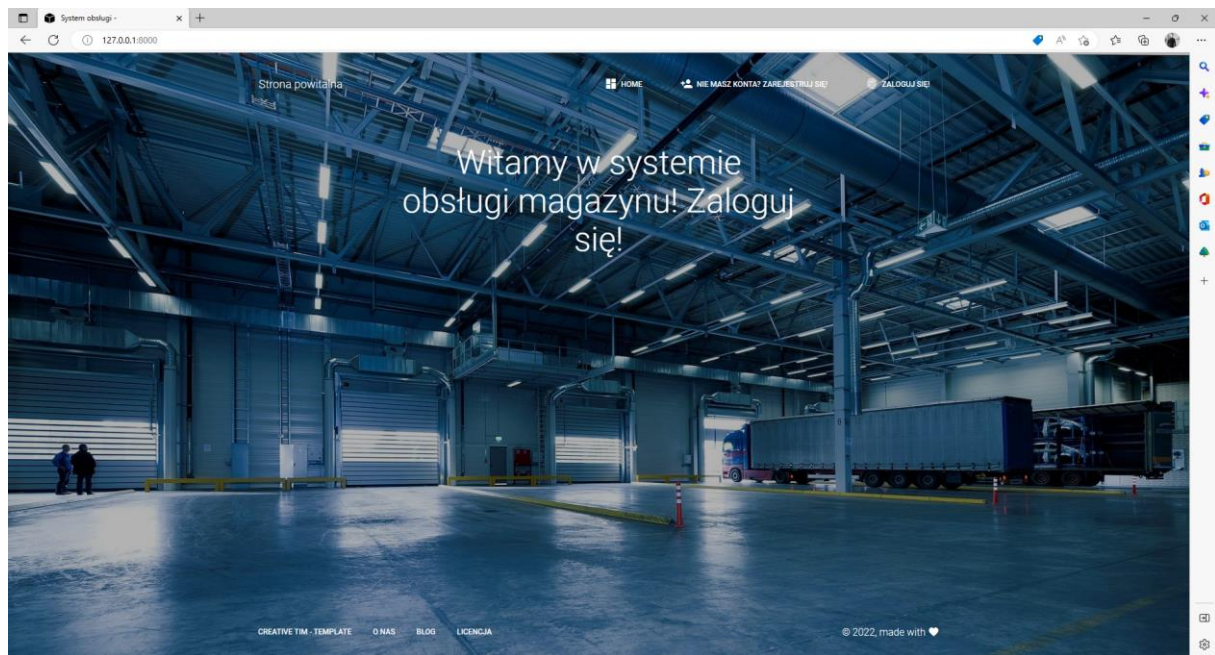
Pierwsze uruchomienie aplikacji

Przy użyciu przeglądarki internetowej i wpisaniu domyślnego adresu hosta: **127.0.0.1:8000** można zacząć korzystać z naszego systemu.

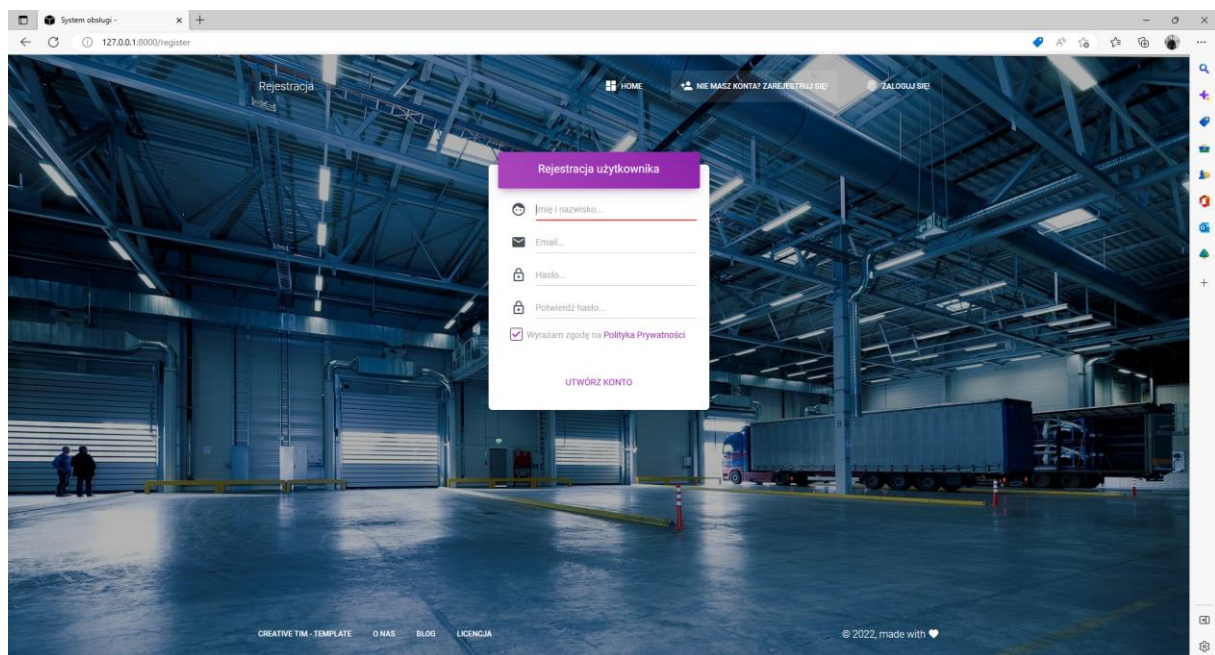
Strona powitalna

Do wyboru na stronie powitalnej są dostępne dwie opcje:

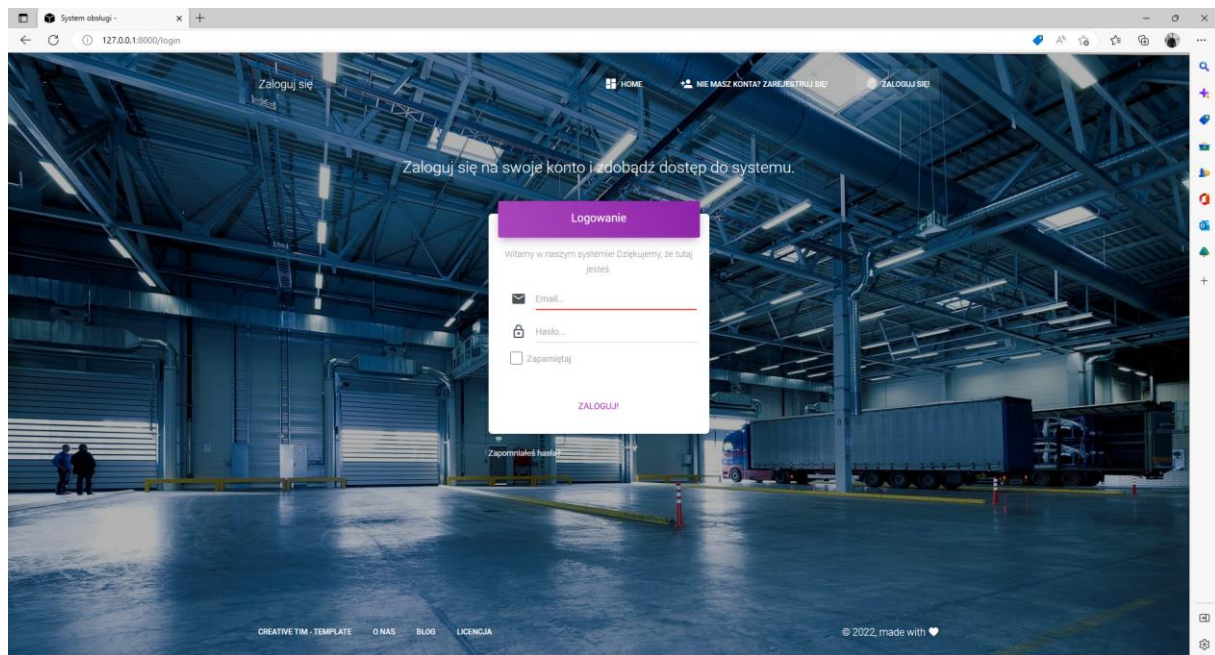
- Zaloguj się
- Zarejestruj się



Rysunek 8 - system obsługi magazynu - strona powitalna



Rysunek 9 - system obsługi magazynu - formularz rejestracji

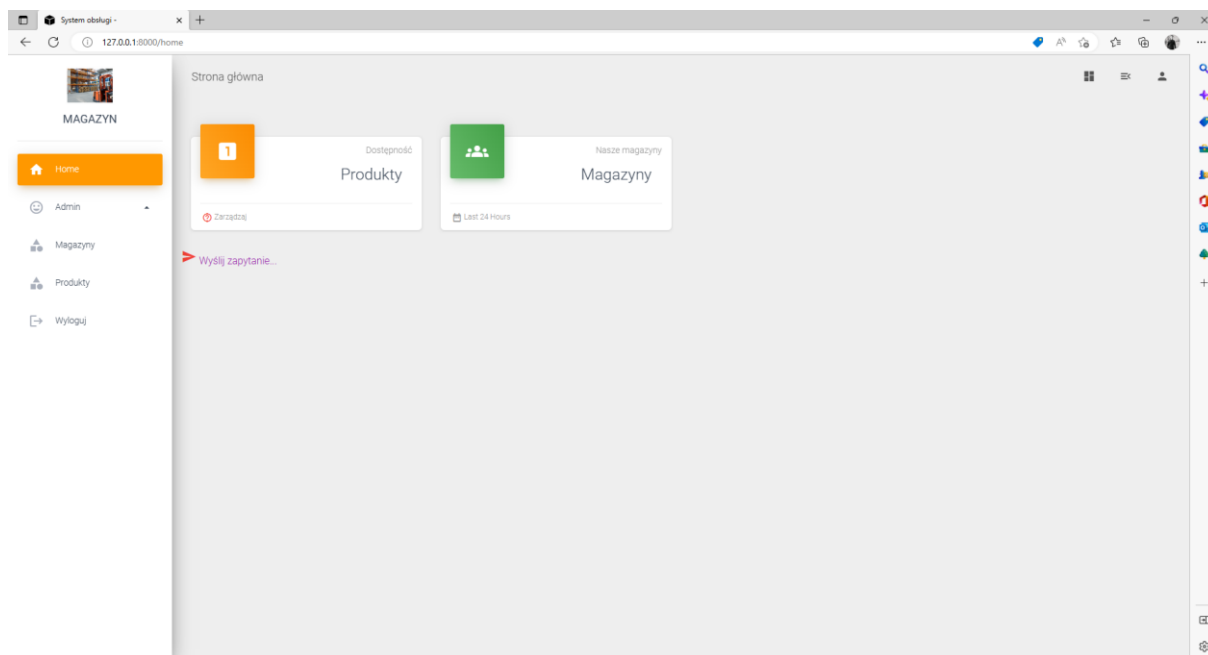


Rysunek 10 - system obsługi magazynu -formularz logowania

Strona główna po zalogowaniu

W tym miejscu użytkownik ma do wyboru dwa następujące moduły:

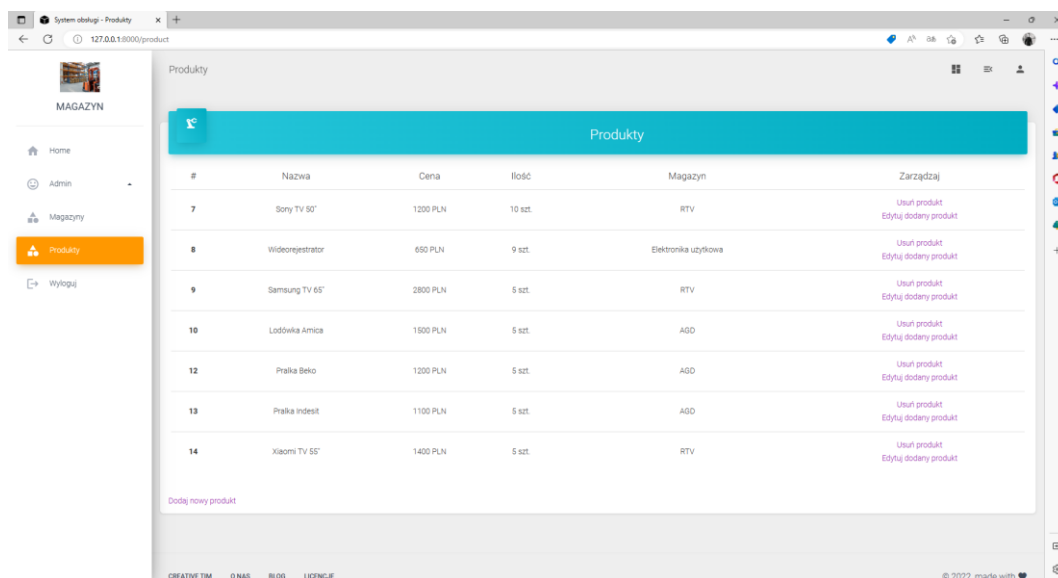
- Moduł 'Produkty'
- Moduł 'Magazyny'



Rysunek 11 - strona główna

Moduł 'Produkty'

Ten moduł odpowiada za dodawanie produktów do bazy danych oraz określenie jego miejsca przechowywania (magazynu). Pierwsza strona modułu zawiera listę wszystkich produktów.

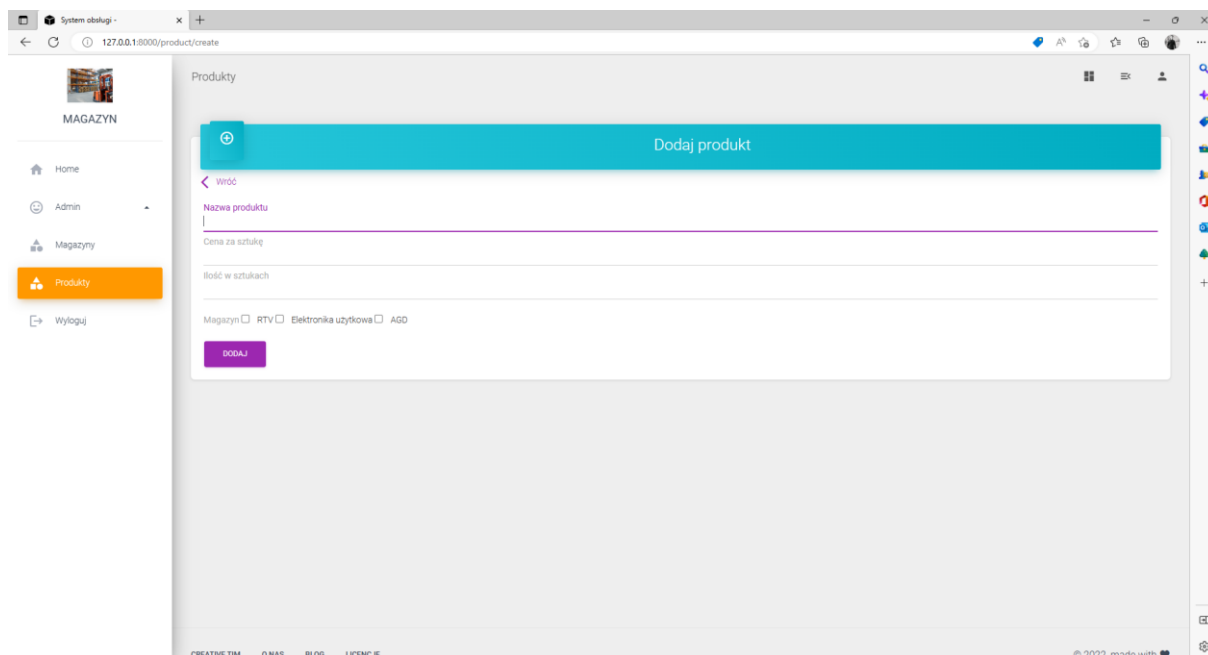


Rysunek 12 - moduł 'produkty' – lista

W tym miejscu użytkownik ma możliwość zarządzania produktami. Opcji jest kilka:

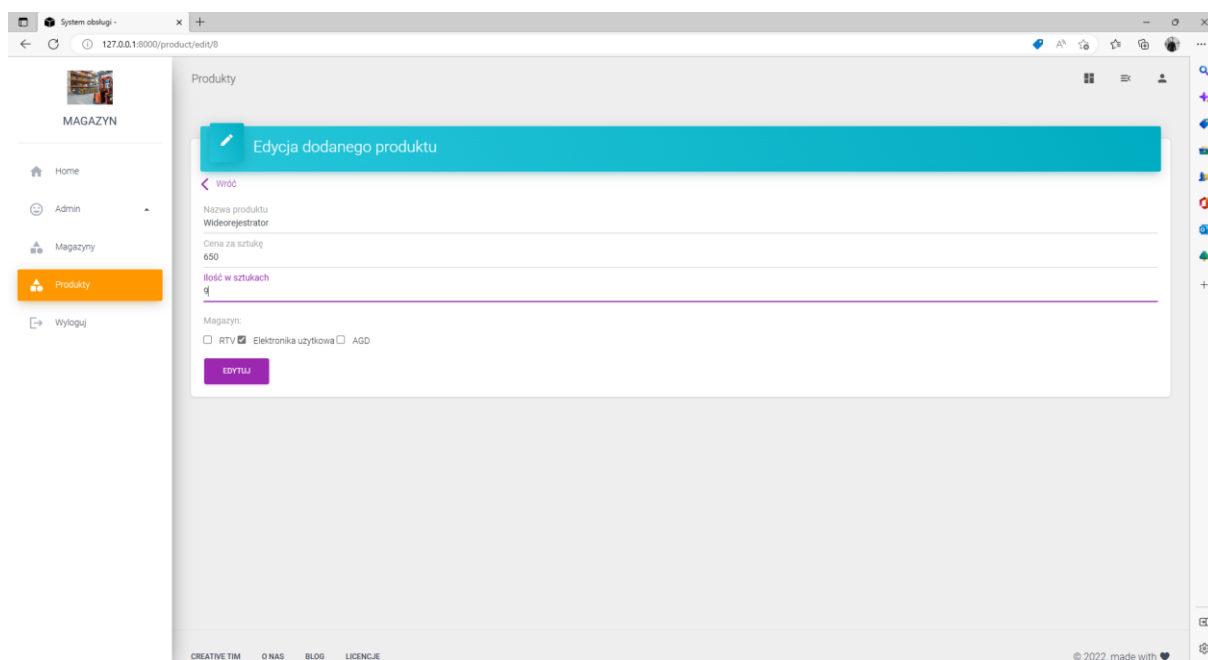
- Dodawanie nowego produktu
- Edytowanie dodanego produktu
- Usuwanie produktu

Po kliknięciu w **Dodaj nowy produkt** pojawi się formularz, który należy wypełnić.



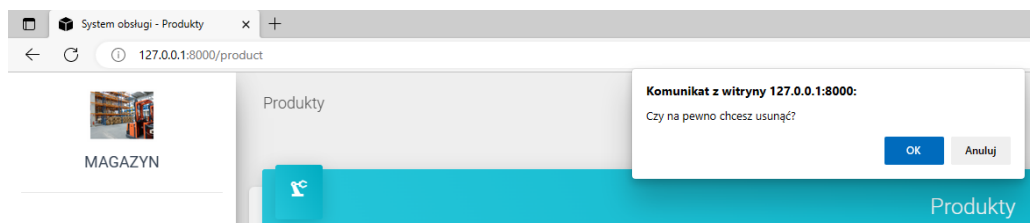
Rysunek 13- 'moduł produkty' - dodaj produkt

Z kolei kliknięcie w **Edytuj dodany produkt** pozwoli użytkownikowi edytować wprowadzone dane.



Rysunek 14- moduł 'produkty' - edytuj produkt

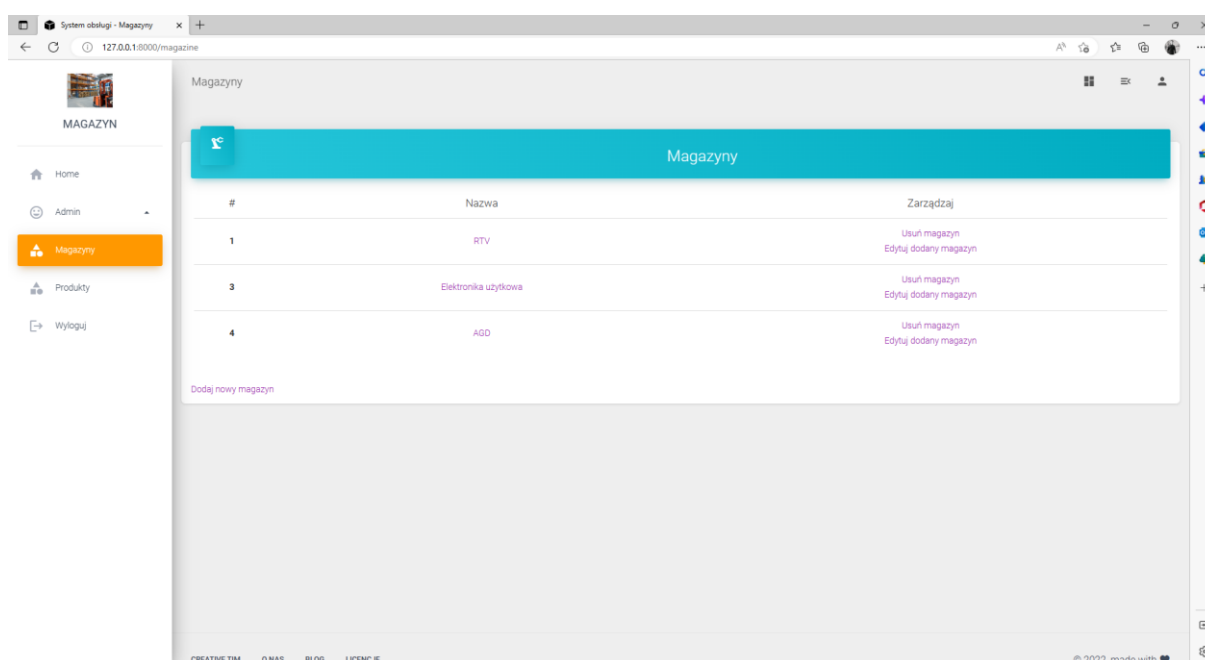
Kliknięcie w **Usuń produkt** spowoduje pojawienie się poniższego komunikatu z witryny:



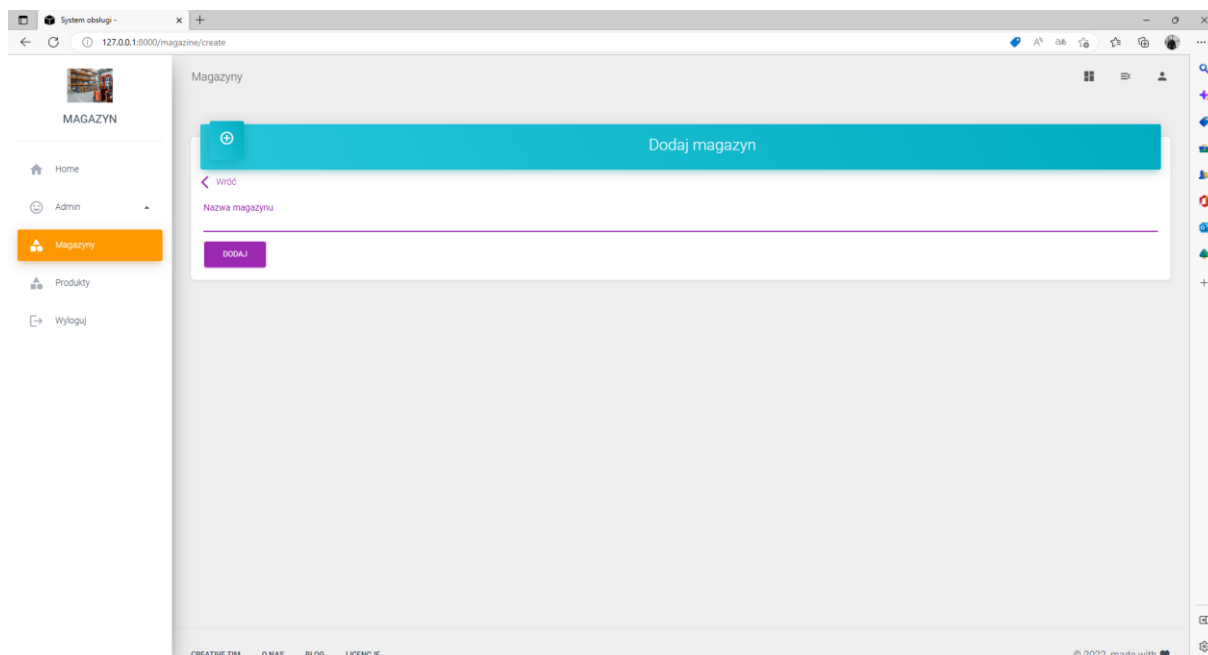
Rysunek 15- moduł 'produkty' - usuń produkt

Moduł 'Magazyny'

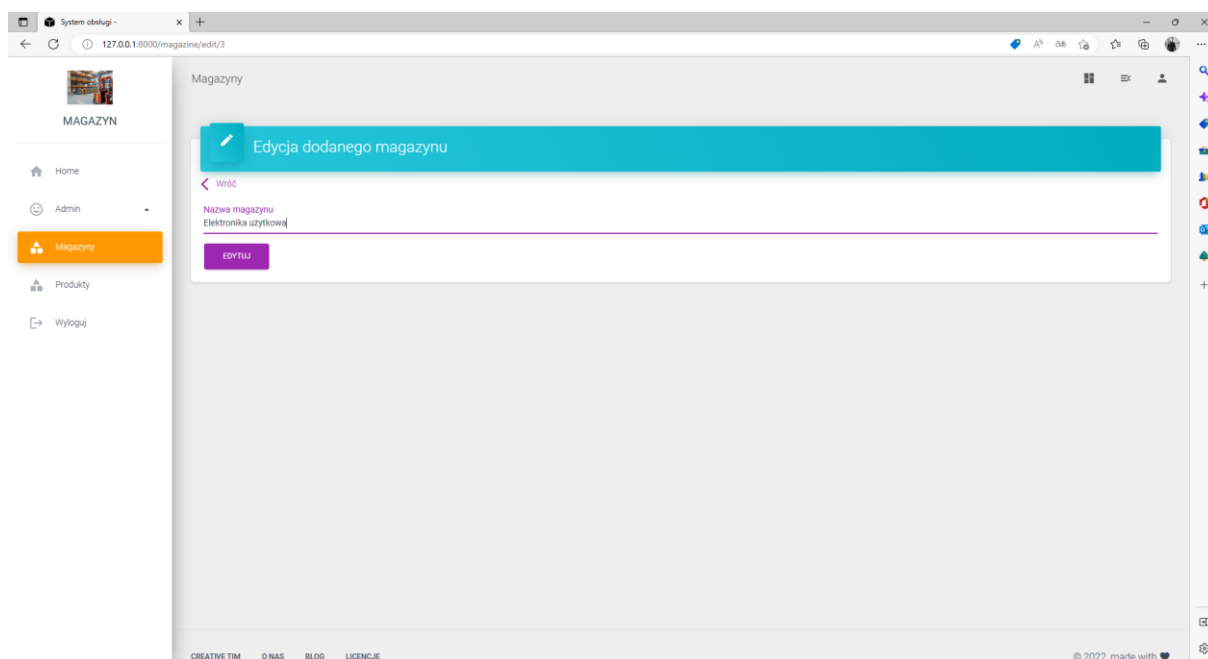
Moduł odpowiada za dodawanie nowych miejsc przechowywania (magazynów, działów itd.). Pierwsza strona modułu zawiera listę wszystkich dostępnych (dodanych) miejsc.



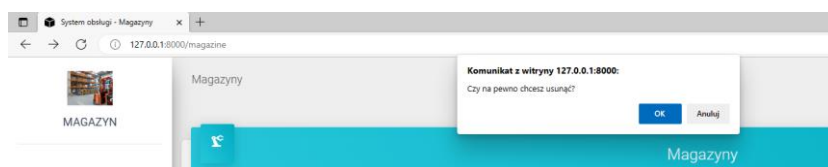
Funkcjonalności dostępne na tej stronie niczym nie odbiegają od modułu '**Produkty**'. Schemat użytkowania jest taki sam.



Rysunek 16- moduł 'magazyny' - dodaj magazyn



Rysunek 17 - moduł 'magazyny' - edytuj magazyn



Rysunek 18 - moduł 'magazyny' - usuń magazyn

SPECYFIKACJA TECHNICZNA

Dane techniczne

Serwer WWW:

- Apache/2.4.52 (Win64) OpenSSL/1.1.1m PHP/7.4.27
- Wersja klienta bazy danych: libmysql - mysqlnd 7.4.27
- Rozszerzenie PHP: mysqli curl mbstring
- Wersja PHP: 7.4.27

Framework Laravel:

- Laravel 8.73.2

Szablon aplikacji:

- Material Dashboard Bootstrap

Linki do dokumentacji:

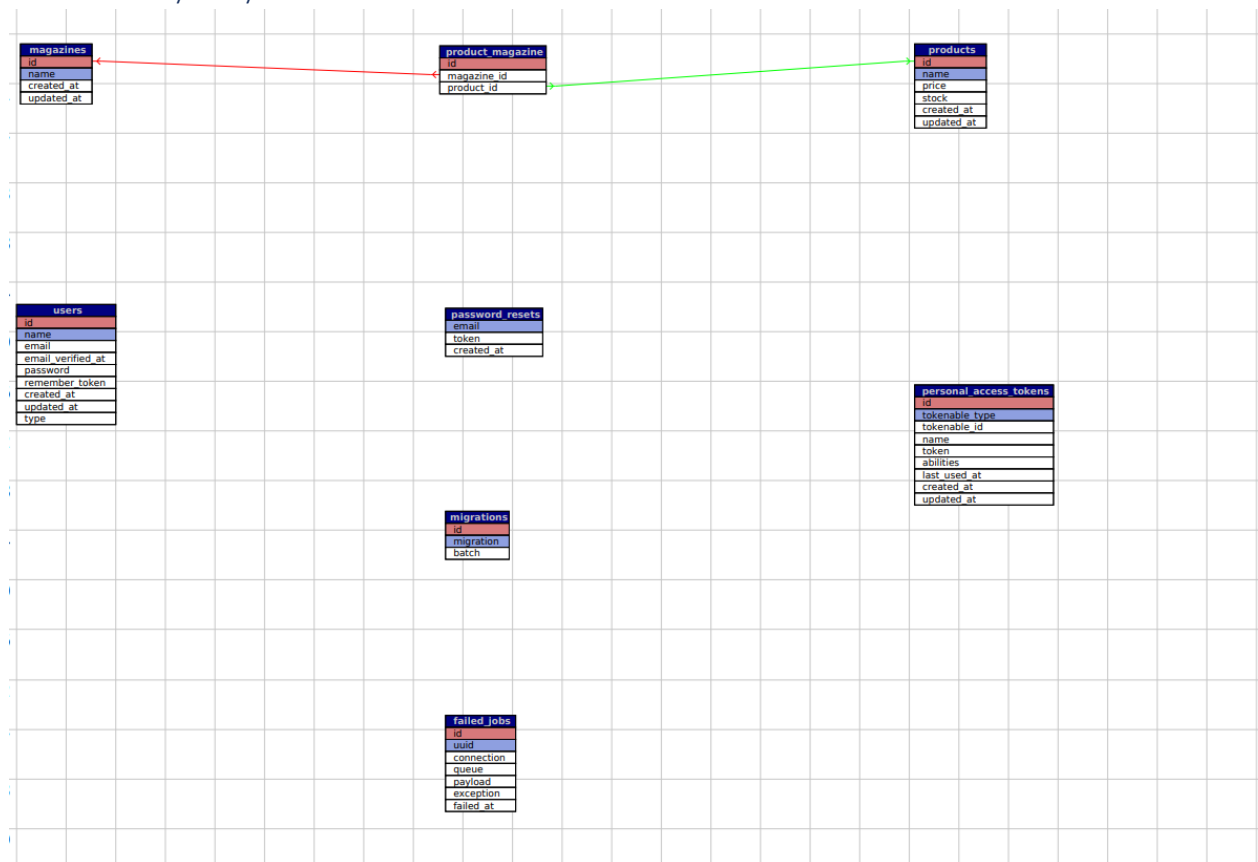
- PHP: <https://www.php.net/manual/en/index.php>
- Laravel 8: <https://laravel.com/docs/8.x/installation>
- Material Dashboard Bootstrap: <https://material-dashboard-laravel.creative-tim.com/documentation/getting-started/overview.html>

Baza danych

Serwer bazy danych:

- Serwer: 127.0.0.1 via TCP/IP
- Typ serwera: MariaDB
- Połączenie z serwerem: SSL nie jest używany
- Wersja serwera: 10.4.22-MariaDB - mariadb.org binary distribution
- Wersja protokołu: 10
- Użytkownik: root@localhost
- Kodowanie znaków serwera: UTF-8 Unicode (utf8mb4)

Schemat bazy danych



Rysunek 19 - schemat bazy danych

Struktura poszczególnych tabel

- Tabela **failed_jobs**

Kolumna	Typ	Atrybuty	Null	Ustawienia domyślne	Dodatkowo	Odsyłacze do	Komentarze
id	bigint(20)	UNSIGNED	Nie		auto_increment		
uuid	varchar(255)		Nie				
connection	text		Nie				
queue	text		Nie				
payload	longtext		Nie				
exception	longtext		Nie				
failed_at	timestamp		Nie	current_timestamp()			

- Tabela **migrations**

Kolumna	Typ	Atrybuty	Null	Ustawienia domyślne	Dodatkowo	Odsyłacze do	Komentarze
id	int(10)	UNSIGNED	Nie		auto_increment		
migration	varchar(255)		Nie				
batch	int(11)		Nie				

- Tabela **password_reset**

Kolumna	Typ	Atrybuty	Null	Ustawienia domyślne	Dodatkowo	Odsyłacze do	Komentarze
email	varchar(255)		Nie				
token	varchar(255)		Nie				
created_at	timestamp		Tak	NULL			

- Tabela **personal_access_tokens**

Kolumna	Typ	Atrybuty	Null	Ustawienia domyślne	Dodatkowo	Odsyłacze do	Komentarze
id	bigint(20)	UNSIGNED	Nie		auto_increment		
tokenable_type	varchar(255)		Nie				
tokenable_id	bigint(20)	UNSIGNED	Nie				
name	varchar(255)		Nie				
token	varchar(64)		Nie				
abilities	text		Tak	NULL			
last_used_at	timestamp		Tak	NULL			
created_at	timestamp		Tak	NULL			
updated_at	timestamp		Tak	NULL			

- Tabela **users**

Kolumna	Typ	Atrybuty	Null	Ustawienia domyślne	Dodatkowo	Odsyłacze do	Komentarze
id	bigint(20)	UNSIGNED	Nie		auto_increment		
name	varchar(255)		Nie				
email	varchar(255)		Nie				
email_verified_at	timestamp		Tak	NULL			
password	varchar(255)		Nie				
remember_token	varchar(100)		Tak	NULL			
created_at	timestamp		Tak	NULL			
updated_at	timestamp		Tak	NULL			
type	varchar(10)		Nie				

- Tabela **magazines**

Kolumna	Typ	Atrybuty	Null	Ustawienia domyślne	Dodatkowo	Odsyłacze do	Komentarze
id	int(10)	UNSIGNED	Nie		auto_increment		
name	varchar(255)		Nie				
created_at	timestamp		Tak	NULL			
updated_at	Timestamp		Tak	NULL			

- Tabela **products**

Kolumna	Typ	Atrybuty	Null	Ustawienia domyślne	Dodatkowo	Odsyłacze do	Komentarze
id	bigint(20)	UNSIGNED	Nie		auto_increment		
name	varchar(255)		Nie				
price	varchar(255)		Nie				
stock	varchar(255)		Nie				
created_at	timestamp		Tak	NULL			
updated_at	timestamp		Tak	NULL			

- Tabela **product_magazine**

Kolumna	Typ	Atrybuty	Null	Ustawienia domyślne	Dodatkowo	Odsyłacze do	Komentarze
id	int(10)	UNSIGNED	Nie		auto_increment		
magazine_id	int(10)	UNSIGNED	Nie			-> magazines.id ON UPDATE RESTRICT ON DELETE CASCADE	
product_id	bigint(20)	UNSIGNED	Nie			-> products.id ON UPDATE RESTRICT ON DELETE CASCADE	

Kod źródłowy

Migracje

Migracje opisują bazę danych i pozwalają na zdefiniowanie bazy danych aplikacji. Framework Laravel posiada wbudowane statyczne fasady, które można opcjonalnie wykorzystać. Jedną z nich jest **Schema** – wykorzystywana na potrzeby definiowania bazy danych.

Każda z migracji składa się z:

- **Klasy**
Nazwy klas pochodzą od poszczególnych funkcjonalności, np. `CreateProductTable`.
- **Metod**
Klasa migracji składa się z dwóch głównych metod. Metody **up** i metody **down**. Metoda **up** jest używana do dodawania nowej tabeli, kolumny lub indeksu w bazie danych. Z kolei metoda **down** jest odwrotnością działania metody **up**.
W ramach fasady **Schema** używane są metody: **create**, **table**, **dropIfExists**. **Create** odpowiada za stworzenie tabeli, **table** za aktualizację tabeli, **dropIfExists** za usuwanie tabeli.
- **Zmiennych**
Zmienna **\$table** przyjmuje nazwy poszczególnych kolumn.

Poniżej przedstawiony został kod źródłowy utworzonych migracji.

- Tworzenie tabeli **products**

```
2022_12_17_232838_create_product_table.php
1 <?php
2
3 use Illuminate\Database\Migrations\Migration;
4 use Illuminate\Database\Schema\Blueprint;
5 use Illuminate\Support\Facades\Schema;
6
7 class CreateProductTable extends Migration
8 {
9     /**
10      * Run the migrations.
11      *
12      * @return void
13      */
14     public function up()
15     {
16         Schema::create('products',function(Blueprint $table){
17             $table->id();
18             $table->string('name');
19             $table->string('price');
20             $table->string('stock');
21             $table->timestamps();
22         });
23     }
24
25     /**
26      * Reverse the migrations.
27      *
28      * @return void
29      */
30     public function down()
31     {
32         Schema::dropIfExists('products');
33     }
34 }
35
```

Rysunek 20 - migracje, tworzenie tabeli 'products'

- Tworzenie tabeli **magazines**

```
2022_12_18_004327_create_magazines_table...
1  <?php
2
3  use Illuminate\Database\Migrations\Migration;
4  use Illuminate\Database\Schema\Blueprint;
5  use Illuminate\Support\Facades\Schema;
6
7  class CreateMagazinesTable extends Migration
8  {
9      /**
10       * Run the migrations.
11       *
12       * @return void
13       */
14     public function up()
15     {
16         Schema::create('magazines', function (Blueprint $table) { //stworzenie tabeli
17             $table->increments('id');
18             $table->string('name');
19             $table->timestamps();
20         });
21     }
22
23     /**
24      * Reverse the migrations.
25      *
26      * @return void
27      */
28     public function down()
29     {
30         Schema::dropIfExists('magazines');
31     }
32 }
33
```

Rysunek 21- migracje, tworzenie tabeli 'magazines'

- Tworzenie tabeli **product_magazine**

W ramach fasady **Schema**, metoda **table** aktualizuje tabelę **product_magazine** o klucze obce.

```

2022_12_18_010935_create_product_magazin...
1  <?php
2
3  use Illuminate\Database\Migrations\Migration;
4  use Illuminate\Database\Schema\Blueprint;
5  use Illuminate\Support\Facades\Schema;
6
7  class CreateProductMagazineTable extends Migration
8  {
9      /**
10       * Run the migrations.
11       *
12       * @return void
13       */
14      public function up()
15      {
16          Schema::dropIfExists('product_magazine');
17          Schema::create('product_magazine', function (Blueprint $table) { //stworzenie tabeli pivot
18              $table->increments('id');
19              $table->unsignedInteger('magazine_id')->unsigned();
20              $table->unsignedBigInteger('product_id')->unsigned();
21          });
22          Schema::table('product_magazine', function(Blueprint $table) { //nadanie kluczy obcych
23              $table->foreign('magazine_id')->references('id')->on('magazines')->onDelete('cascade');
24              $table->foreign('product_id')->references('id')->on('products')->onDelete('cascade');
25          });
26      }
27
28      /**
29       * Reverse the migrations.
30       *
31       * @return void
32       */
33      public function down()
34      {
35          Schema::dropIfExists('product_magazine');
36      }
37  }
38
39

```

Rysunek 22- migracje, tworzenie tabeli 'product_magazine'

Seeding

Seeder składa się z:

- **Klasy** - UserTableSeeder
- **Metod:**
 - run** – wywoływana podczas wykonywania komendy **php artisan db:seed**
 - table** – w ramach **fasady DB**, aktualizuje stan tabeli dodając do niej przypisane wartości

Poniżej znajduje się kod źródłowy wykorzystywanego seedera.

- **UserTableSeeder.php**

Statycznie zapisane parametry poszczególnych pól tabeli users, takich jak:

- name
- email
- email_verified_at
- password
- created_at
- update_at
- type

```

UsersTableSeeder.php
1 <?php
2 namespace Database\Seeders;
3
4 use Illuminate\Support\Facades\DB;
5 use Illuminate\Database\Seeder;
6 use Illuminate\Support\Facades\Hash;
7
8 class UsersTableSeeder extends Seeder
9 {
10     /**
11      * Run the database seeds.
12      *
13      * @return void
14      */
15     public function run()
16     {
17         DB::table('users')->insert([
18             'name' => 'Admin',
19             'email' => 'admin@admin.pl',
20             'email_verified_at' => now(),
21             'password' => Hash::make('admin@admin.pl'),
22             'created_at' => now(),
23             'updated_at' => now(),
24             'type' => 'admin'
25         ]);
26     }
27 }
28

```

Rysunek 23- seeders

Modele

Modele generowane są poprzez wykonanie polecenia w wierszu poleceń:

php artisan make:model nazwamodelu

Struktura modelu:

- **Klasy** - o ściśle określonej nazwie nawiązującej do poszczególnej tabeli bazy danych, są rozszerzeniem podstawowej klasy **Model**
- **Właściwości klasy** – zmienna tablicowa **\$fillable** zawierająca wartości, które należało wypełnić w bazie danych metodą **create**, domyślne zabezpieczenie wymagane przez **Eloquent**

Poniżej znajduje się kod źródłowy wygenerowanych modeli.

- Model **Product.php**

Model zawiera dodatkowo metodę **magazine**, która określa relację wiele do wielu pomiędzy tabelą **products** a **magazines**.

```

Product.php
1  <?php
2
3  namespace App\Models;
4
5  use Illuminate\Database\Eloquent\Factories\HasFactory;
6  use Illuminate\Database\Eloquent\Model;
7
8  class Product extends Model
9  {
10     use HasFactory;
11
12     protected $fillable = [
13         'name',
14         'price',
15         'stock'
16     ];
17
18     public function magazine(){ //metoda zawierająca określenie relacji pomiędzy tabelami
19         return $this->belongsToMany(Magazine::class, 'product_magazine');
20     }
21 }
22
23

```

Rysunek 24 - modele, Product.php

- Model **Magazine.php**

```

Magazine.php
1  <?php
2
3  namespace App\Models;
4
5  use Illuminate\Database\Eloquent\Factories\HasFactory;
6  use Illuminate\Database\Eloquent\Model;
7
8  class Magazine extends Model
9  {
10     use HasFactory;
11
12     protected $fillable = [
13         'name'
14     ];
15
16 }
17

```

Rysunek 25 - modele, Magazine.php

- Model **User.php**

Klasa **class User** rozszerza podstawową klasę **Authenticatable**. Model zawiera dodatkowe właściwości klasy: **\$hidden** (ukrycie atrybutu) oraz **\$casts** (rzutowanie atrybutu).


```

User.php
1  <?php
2
3  namespace App\Models;
4
5  use Illuminate\Contracts\Auth\MustVerifyEmail;
6  use Illuminate\Database\Eloquent\Factories\HasFactory;
7  use Illuminate\Foundation\Auth\User as Authenticatable;
8  use Illuminate\Notifications\Notifiable;
9  use Laravel\Sanctum\HasApiTokens;
10 use Illuminate\Database\Eloquent\Model;
11
12 class User extends Authenticatable
13 {
14     use HasApiTokens, HasFactory, Notifiable;
15
16     /**
17      * The attributes that are mass assignable.
18      *
19      * @var string[]
20      */
21     protected $fillable = [
22         'name',
23         'email',
24         'password',
25         'type'
26     ];
27
28     /**
29      * The attributes that should be hidden for serialization.
30      *
31      * @var array
32      */
33     protected $hidden = [
34         'password',
35         'remember_token',
36     ];
37
38     /**
39      * The attributes that should be cast.
40      *
41      * @var array
42      */
43     protected $casts = [
44         'email_verified_at' => 'datetime',
45     ];
46
47 }

```

Rysunek 26 - modele, User.php

Kontrolery

Kontrolery grupują powiązaną logikę obsługi żądań w jedną klasę. Na przykład klasa **class UserController** obsługuje wszystkie przychodzące żądania związane z użytkownikami.

Kontrolery obsługujące moduły Produktu oraz Magazynu korzystają z osobnego, dodatkowego repozytorium opisanego w punkcie Repozytoria.

Poniżej kod źródłowy kontrolerów.

- **ProductController.php**

Struktura kontrolera produktów:

- Klasa – **ProductController** rozszerzająca podstawową klasę **Controller**
- Metody:
 - **index**: obsługa głównej listy produktów
 - **listByMagazine**: wyświetla nazwę magazynu przypisanego do produktu
 - **show**: wyświetla szczegóły produktu

- **create:** obsługa formularza dodawania produktu
- **store:** obsługa walidacji wprowadzonych danych oraz zapisu ich do bazy danych
- **edit:** obsługa formularza edycji
- **delete:** obsługa żądania usunięcia produktu
- **editStore:** obsługa walidacji edytowanych danych oraz zapisu ich do bazy danych

```

ProductController.php
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6 use App\Models\Magazine;
7 use App\Models\User;
8 use App\Models\Product;
9 use App\Repositories\ProductRepository;
10 use Illuminate\Support\Facades\Auth;
11
12 class ProductController extends Controller
13 {
14
15     public function __construct(){
16         $this->middleware('auth');
17     }
18
19     public function index(ProductRepository $prodRepo){ //metoda zawierająca funkcję wyświetlania listy produktów
20
21         $product = $prodRepo->getAllProduct();
22
23         return view('product.list',["productlist"=>$product,
24                                     "footerYear">date("Y"),
25                                     "title">"Produkty",
26                                     ]);
27     }
28
29     public function listByMagazine(ProductRepository $prodRepo,$id){
30
31
32         $product = $prodRepo->getProductByMagazine($id);
33
34         return view('product.list',["productlist"=>$product]);
35     }
36
37
38     public function show(ProductRepository $prodRepo,$id){ //metoda zawierająca funkcję wyświetlania szczegółów danego produktu
39
40         $product = $prodRepo->find($id);
41
42         return view('product.show');
43     }
44
45     public function create(){ //metoda zawierająca funkcję wyświetlania formularza dodawania produktu
46
47         /**if(Auth::user()->type != 'admin')
48         {
49             return redirect()->route('login');
50         }
51     }

```

Rysunek 27- ProductController.php

```

ProductController.php
47: /*(Auth::user()->type != 'admin')
48: {
49:     return redirect()->route('login');
50: }*/
51:
52: $magazine = Magazine::All(); // zwraca $magazine pobierając dane z tabeli "Magazine"
53:
54: return view("product.create", ["magazine" => $magazine]);
55: }
56:
57: public function store(Request $request){ //zapisywanie danych do tabeli
58:
59:     $request->validate([
60:         'name' => 'required|max:255', //pole wymagane max 255 znaków //validacja danych wprowadzonych do formularza
61:         'price' => 'required', //pole wymagane
62:         'stock' => 'required'
63:     ]);
64:
65:     $product = new Product; //dodawanie lekarza do bazy poprzez przeliczenie danych z formularza
66:     $product->name = $request->input('name');
67:     $product->price = $request->input('price');
68:     $product->stock = $request->input('stock');
69:     $product->save();
70:     $product->magazine()->sync($request->input('magazine.id'));
71:
72:     return redirect()->action('App\Http\Controllers\ProductController@index');
73: }
74:
75: public function edit(ProductRepository $prodRepo,$id){ //edytowanie istniejących danych, wyświetlenie formularza edycji
76:
77:     $magazine = Magazine::All();
78:     $product = $prodRepo->find($id);
79:
80:     return view("product.edit",["magazine" => $magazine, "product" => $product]);
81: }
82:
83: public function delete(ProductRepository $prodRepo,$id){ //usunięcie danego magazynu z tabeli, usunięcie po "id"
84:
85:     $product = $prodRepo->delete($id);
86:     return redirect('product');
87: }
88:
89: public function editStore(Request $request){ //zapisywanie edytowanych danych
90:
91:     $product = Product::find($request->input('id'));
92:     $product->name = $request->input('name');
93:     $product->price = $request->input('price');
94:     $product->stock = $request->input('stock');
95:     $product->save();
96:     $product->magazine()->sync($request->input('magazine'));
97:
98:     return redirect()->action('App\Http\Controllers\ProductController@index');
99: }
100: }
101:

```

Rysunek 28 - ProductController.php cd.

- **MagazineController.php**

Struktura kontrolera magazynów:

- Klasa – **MagazineController** rozszerzająca podstawową klasę **Controller**
- Metody:
 - **index**: obsługa głównej listy magazynów
 - **show**: wyświetla szczegóły magazynu
 - **create**: obsługa formularza dodawania magazynu
 - **store**: obsługa walidacji wprowadzonych danych oraz zapisu ich do bazy danych
 - **edit**: obsługa formularza edycji
 - **delete**: obsługa żądania usunięcia produktu
 - **editStore**: obsługa walidacji edytowanych danych oraz zapisu ich do bazy danych

```

MagazineController.php
1 <?php
2 namespace App\Http\Controllers;
3
4 use Illuminate\Http\Request;
5 use App\Models\Magazine;
6 use App\Repositories\MagazineRepository;
7 use Illuminate\Support\Facades\Auth;
8
9 class MagazineController extends Controller
10 {
11     public function __construct(){
12         $this->middleware('auth');
13     }
14
15     public function index(MagazineRepository $magRepo){ //metoda zawierająca funkcję wyświetlania listy magazynów
16         $magazine = $magRepo->getAllMagazine();
17
18         return view('magazine.list', ['magazinelist'=>$magazine,
19                                     'footerYear'=>date("Y"),
20                                     'title'=>"Magazyny",
21                                     ]);
22     }
23
24     public function show(MagazineRepository $magRepo,$id){ //metoda zawierająca funkcję wyświetlania szczegółów danego magazynu
25         $magazine = $magRepo->find($id);
26
27         return view('magazine.show', ['magazine'=>$magazine, 'title'=>"Magazyny"]);
28     }
29
30     public function create(){ //metoda zawierająca funkcję wyświetlania formularza dodawania produktu
31         return view('magazine.create');
32     }
33
34     public function store(Request $request){ //zapisywanie danych do tabeli
35
36         $magazine = new Magazine;
37         $magazine->name = $request->input("name");
38         $magazine->save();
39     }
40 }

```

Rysunek 29 - MagazineController.php

```

41 public function store(Request $request){ //zapisywanie danych do tabeli
42     $magazine = new Magazine;
43     $magazine->name = $request->input("name");
44     $magazine->save();
45
46     return redirect()->action('App\Http\Controllers\MagazineController@index');
47 }
48
49 public function edit(MagazineRepository $magRepo,$id){ //edytowanie istniejących danych, wyświetlanie formularza edycji
50     $magazine = $magRepo->find($id);
51
52     return view('magazine.edit', ['magazine' => $magazine]);
53 }
54
55 public function delete(MagazineRepository $magRepo,$id){ //usuwanie danego magazynu z tabeli, usuwanie po "id"
56     $magazine = $magRepo->delete($id);
57     return redirect('magazine');
58 }
59
60 public function editStore(Request $request){ //zapisywanie edytowanych danych
61
62     $magazine = Magazine::find($request->input("id"));
63     $magazine->name = $request->input("name");
64     $magazine->save();
65
66     return redirect()->action('App\Http\Controllers\MagazineController@index');
67 }
68 }

```

Rysunek 30 - MagazineController.php cd.

Repozytoria

Repozytoria zawierają podstawowe odwołania do bazy danych. Takie jak : zwracanie wszystkich danych z tabeli, **dodawanie** (create), **edytowanie** (update) oraz **usuwanie** (delete) rekordu z tabeli, **wyszukiwanie** rekordu po id (find).

- **BaseRepository.php**

To repozytorium definiuje podstawowe funkcjonalności związane z obsługą bazy danych.

Struktura BaseRepository:

- Klasa – klasa abstrakcyjna BaseRepository zapewniająca szablon dla korzystających z niej podklas
- Metody:
 - **getAll**: zwraca wszystkie dostępne kolumny tabeli
 - **create**: dodaje datę dodania rekordu
 - **update**: dodaje datę aktualizacji rekordu
 - **delete**: usuwa rekord
 - **find**: przeszukiwanie tabeli po numerze id
- Zmienna - **\$model** z modyfikatorem dostępu - protected

Poniżej znajduje się kod źródłowy.

```

BaseRepository.php
1  <?php
2
3  namespace App\Repositories;
4
5  use Illuminate\Database\Eloquent\Model;
6
7  abstract class BaseRepository{
8
9      protected $model;
10
11     public function getAll($columns = array('*')){ //zwracanie danych z tabeli
12         return $this->model->get($columns);
13     }
14
15     public function create($date){ //dodawanie rekordu do tabeli
16         return $this->model->create($date);
17     }
18
19     public function update($date,$id){ //edytowanie istniejącego rekordu
20         return $this->model->where("id","=", $id)->update($date);
21     }
22
23     public function delete($id){ //usuwanie rekordu
24         return $this->model->destroy($id);
25     }
26
27     public function find($id){ //przeszukiwanie tabeli po "id"
28         return $this->model->find($id);
29     }
30
31 }
32

```

Rysunek 31 - BaseRepository.php

- **ProductRepository.php**

Jest to rozszerzenie podstawowej klasy BaseRepository.

Struktura ProductRepository:

- Klasa: ProductRepository rozszerzająca klasę BaseRepository, opowiada za obsługę zapytań do bazy danych
- Metody:
 - **__construct(Product \$model)**: tworzenie obiektu zawierającego jako parametr zmienną \$model z klasy abstrakcyjnej **BaseRepository**
 - **getAllProduct**: zwraca wszystkie rekordy z tabeli products
 - **getAllMagazine**: zwraca wszystkie rekordy z tabeli magazines
 - **getProductByMagazine**: zwraca id magazynu z tabeli magazines

Poniżej znajduje się kod źródłowy.

```
ProductRepository.php
1 <?php
2
3 namespace App\Repositories;
4
5 use App\Models\Product;
6 use DB;
7
8
9
10 class ProductRepository extends BaseRepository{
11
12     public function __construct(Product $model){
13
14         $this->model = $model;
15     }
16     public function getAllProduct(){ //funkcja pobierająca wszystkie rekordy tabeli
17         return $this->model->get();
18     }
19
20     public function getAllMagazine(){ //funkcja pobierająca wszystkie rekordy tabeli
21         return DB::table('magazines')->get();
22     }
23
24     public function getProductByMagazine($id){ //funkcja pobierająca "id" magazynu, w celu przypisania produktu do odpowiedniego magazynu
25
26         return $this->model->where('magazine',function($q) use ($id){
27
28             $q->where('magazine.id',$id);
29         })->orderBy('name','asc')->get();
30     }
31
32 }
33 }
```

Rysunek 32 - ProductRepository.php

- **MagazineRepository.php**

Struktura MagazineRepository:

- Klasa: MagazineRepository rozszerzająca klasę BaseRepository, opowiada za obsługę zapytań do bazy danych
- Metody:
 - **__construct(Product \$model)**: tworzenie obiektu zawierającego jako parametr zmienną \$model z klasy abstrakcyjnej **BaseRepository**
 - **getAllMagazine**: zwraca wszystkie rekordy z tabeli magazines

Poniżej znajduje się kod źródłowy.

```

MagazineRepository.php
1  <?php
2
3  namespace App\Repositories;
4
5  use App\Models\Magazine;
6  use DB;
7
8
9  class MagazineRepository extends BaseRepository{
10
11     public function __construct(Magazine $model){
12
13         $this->model = $model;
14     }
15     public function getAllMagazine(){ //funkcja pobierająca wszystkie rekordy tabeli
16         return DB::table('magazines')->get();
17     }
18
19 }
20

```

Rysunek 33 - MagazineRepository.php

Routes

Trasy (routes) są zdefiniowane w pliku web.php. Zostały one podzielone na grupy poszczególnych funkcjonalności.

- **web.php**

```

web.php
1  /
2  */
3
4  Route::get('/', function () {
5      return view('welcome');
6  });
7
8  Auth::routes();
9
10 //routes dla funkcji Produkty
11 Route::group(['middleware' => 'auth'], function () {
12     Route::get('/product/edit/{id}', 'App\Http\Controllers\ProductController@edit'); //formularz edycji rekordu
13     Route::post('/product/edit/{id}', 'App\Http\Controllers\ProductController@editStore'); //zapisywanie edytowanych danych
14     Route::get('/product/create', 'App\Http\Controllers\ProductController@create'); //formularz dodawania rekordu
15     Route::post('/product', 'App\Http\Controllers\ProductController@store'); //zapisywanie danego rekordu
16     Route::get('/product', 'App\Http\Controllers\ProductController@index'); //dane z tabeli w formie listy
17     Route::get('/product/{id}', 'App\Http\Controllers\ProductController@show'); //szczegóły wybranego rekordu
18     Route::get('/product/magazine/{id}', 'App\Http\Controllers\ProductController@listByMagazine'); //id magazynu dla danego produktu
19     Route::get('/product/delete/{id}', 'App\Http\Controllers\ProductController@delete'); //usuwanie rekordu
20 });
21
22 //routes dla funkcji Magazyny
23 Route::group(['middleware' => 'auth'], function () {
24     Route::get('/magazine/edit/{id}', 'App\Http\Controllers\MagazineController@edit');
25     Route::post('/magazine/edit/{id}', 'App\Http\Controllers\MagazineController@editStore');
26     Route::get('/magazine/create', 'App\Http\Controllers\MagazineController@create');
27     Route::post('/magazine', 'App\Http\Controllers\MagazineController@store');
28     Route::get('/magazine/', 'App\Http\Controllers\MagazineController@index');
29     Route::get('/magazine/{id}', 'App\Http\Controllers\MagazineController@show');
30     Route::get('/magazine/delete/{id}', 'App\Http\Controllers\MagazineController@delete');
31 });
32
33 //routes Login
34 Route::post('/login', 'App\Http\Controllers\Auth\LoginController@login')->name('login');
35
36 Route::group(['middleware' => 'auth'], function () {
37     Route::get('/home', 'App\Http\Controllers\HomeController@index')->name('home');
38 });
39
40 //routes panel użytkownika
41 Route::group(['middleware' => 'auth'], function () {
42     Route::resource('user', 'App\Http\Controllers\UserController', ['except' => ['show']]);
43     Route::get('/profile/edit', 'App\Http\Controllers\ProfileController@edit');
44     Route::put('/profile', ['as' => 'profile.update', 'uses' => 'App\Http\Controllers\ProfileController@update']);
45     Route::put('/profile/password', ['as' => 'profile.password', 'uses' => 'App\Http\Controllers\ProfileController@password']);
46 });

```

Rysunek 34- web.php