

<b>AI1</b>	<b>Dokumentacja projektu</b>
<b>Autor</b>	Bartłomiej Florek, 125115
<b>Kierunek, rok</b>	Informatyka, II rok, st. stacjonarne (3,5-l)
<b>Temat projektu</b>	<i>Sklep ze słodyczami na wagę</i>

## Spis treści

Wstęp.....	3
Narzędzia i technologie .....	4
Baza danych .....	5
GUI.....	6
Uruchomienie aplikacji.....	10
Funkcjonalności aplikacji .....	12

# Wstęp

Tematem projektu jest utworzenie strony w tematyce zamawiania słodczy na wagę. Niezalogowany użytkownik może założyć nowe konto oraz przeglądać udostępnione zasoby przez aplikacje. Zalogowany użytkownik, ma przedstawione dostępne produkty oferowane przez sklep, które może zamówić za odpowiednia opłatą. Użytkownik ma dostęp do wielu funkcjonalności oferowanych przez aplikacje takich jak galeria zdjęć, formularz kontaktowy, cennik, odnośniki do social mediów, sprawdzania swoich poprzednich zamówień oraz wiele innych. Admin posiada możliwość dodawanie nowych produktów, ich edycji jak i ich usuwania.

Projekt znajduje się pod linkiem:  
<https://github.com/flor3kk/ApkiInternetowe/tree/main/projekt>

# Narzędzia i technologie

Wersja PHP (w CMD wpisać php -v): PHP 8.2.12 (cli) (built: Oct 24 2023 21:15:15)  
(ZTS Visual C++ 2019 x64)

Wersja Laravela (w terminalu w command prompt wpisać polecenie php artisan --version): Laravel Framework 11.4.0

Wersja XAMPPA: 3.3.0

Wersja Visual Studio Code: 1.89.1 (user setup)

Dokumentacje oraz technologie:

Laravel: <https://laravel.com/docs/11.x>

PHP: <https://www.php.net/manual/en/>

Baza danych: <https://docs.phpmyadmin.net/en/latest/>

Bootstrap: <https://getbootstrap.com/docs/5.3/getting-started/introduction/>

JavaScript: <https://developer.mozilla.org/en-US/docs/Web/JavaScript>

HTML5: <https://developer.mozilla.org/en-US/docs/Glossary/HTML5>

Narzędzia:

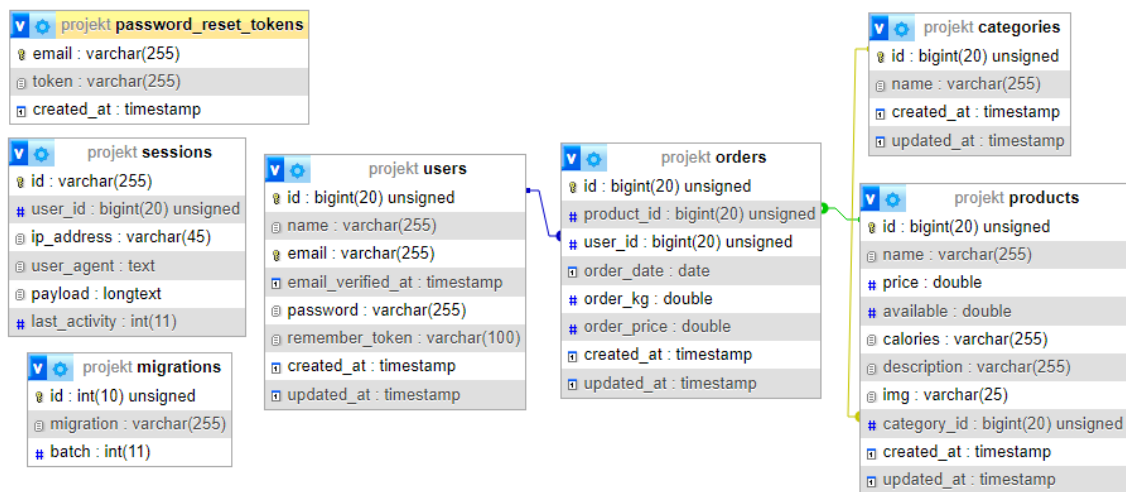
Visual Studio Code: <https://code.visualstudio.com/>

XAMPP: <https://www.apachefriends.org/pl/index.html>

Composer Setup: <https://getcomposer.org/Composer-Setup.exe>

# Baza danych

Diagram ERD wygenerowany przez phpMyAdmin



Rysunek 1 - diagram

Baza danych zawiera takie tabele jak products, która zawiera informacje na temat produktów możliwych do zamówienia, categories zawierających informacje o kategoriach danego produktu,

Orders, która trzyma informacje o zamówieniach poszczególnych użytkowników oraz tabele users zawierającą dane użytkowników. Tabele sessions, password\_reset\_tokens, migrations oraz users są automatycznie tworzone przez framework laravela.

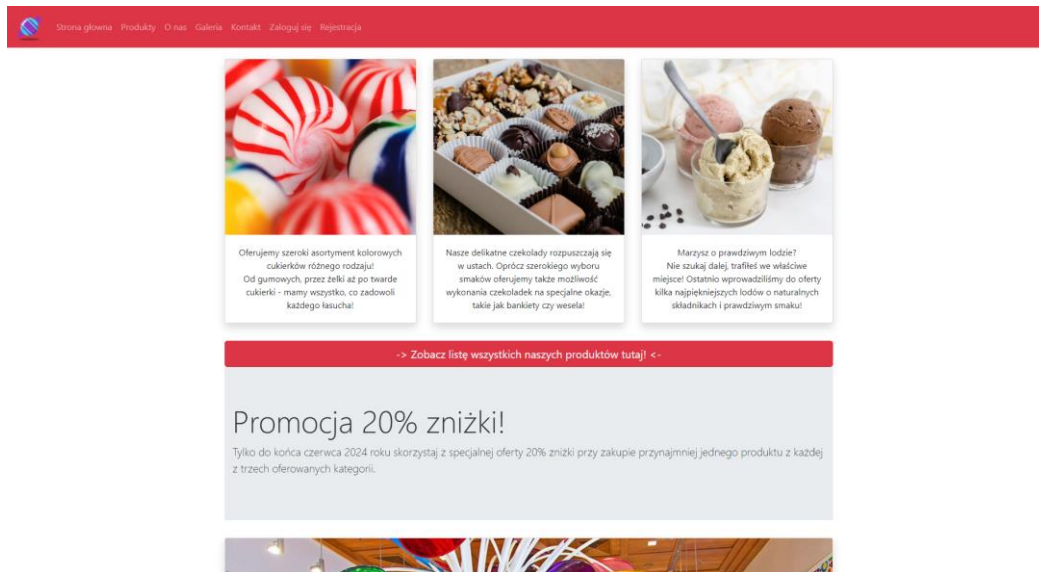
Tabela users do tabeli orders, relacja jeden do wielu,

Tabela orders do tabeli products, relacja jeden do jeden

Tabela categories do tabeli products, relacja jeden do wielu

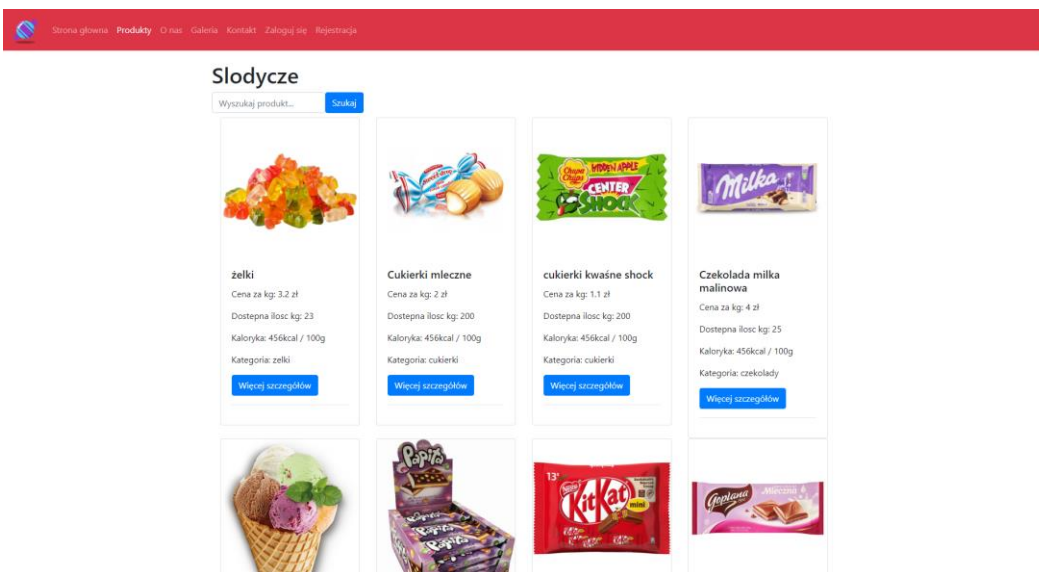
# GUI

Strona główna aplikacji:



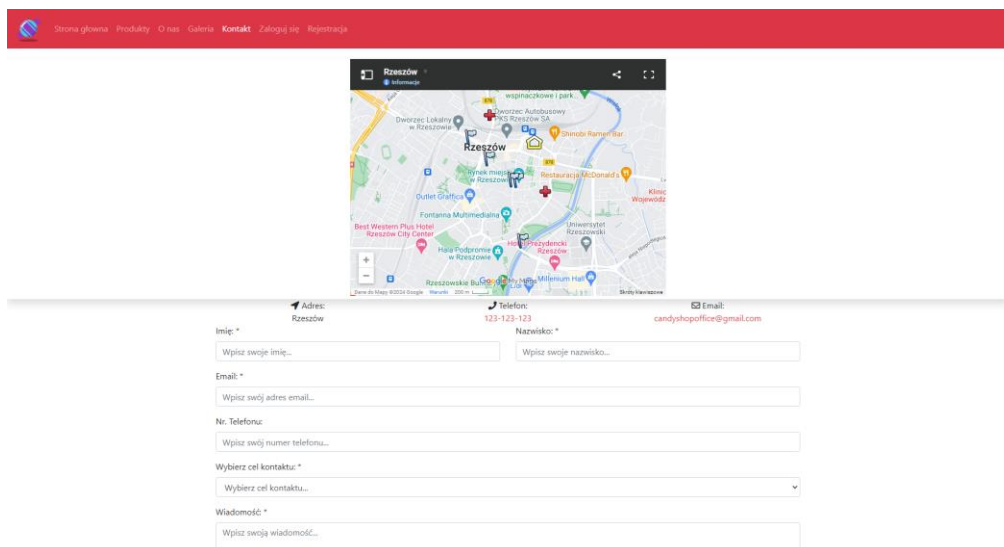
Rysunek 2 - główna strona

Strona przedstawiająca ofertę sklepu:



Rysunek 3 - asortyment

Formularz kontaktowy:



Strona główna Produkty O nas Galeria Kontakt Zaloguj się Rejestracja

Rzeszów

Wpisz swoje imię...

Wpisz swoje nazwisko...

Email: \*

Wpisz swój adres email...

Nr. Telefonu:

Wpisz swój numer telefonu...

Wybierz cel kontaktu: \*

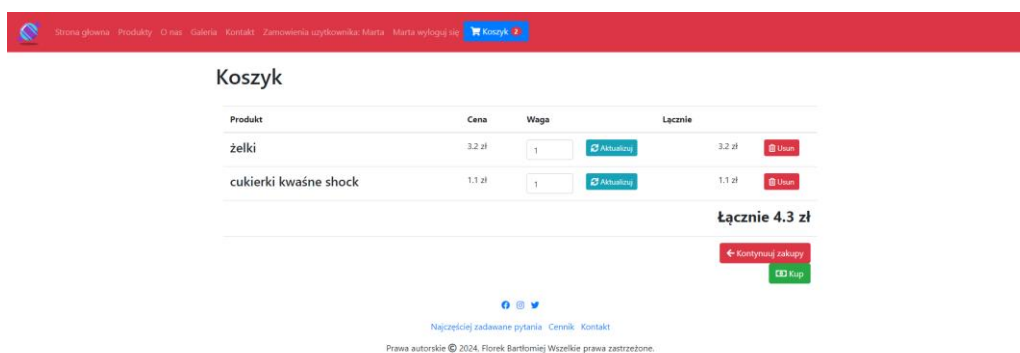
Wybierz cel kontaktu...

Wiadomość \*

Wpisz swoją wiadomość...

Rysunek 4 - formularz kontaktowy

Strona przedstawiająca koszyk i przykładowo dodane do niego produkty:



Strona główna Produkty O nas Galeria Kontakt Zamówienia użytkownika: Marta Marta wyloguj się Koszyk

### Koszyk

Produkt	Cena	Waga	Łącznie
żelki	3,2 zł	1	3,2 zł
cukierki kwaśne shock	1,1 zł	1	1,1 zł

**Łącznie 4.3 zł**

[Kontynuuj zakupy](#) [Kup](#)

[Najczęściej zadawane pytania](#) [Cennik](#) [Kontakt](#)

Prawa autorskie © 2024, Florek Bartomiej Wszelkie prawa zastrzeżone.

Rysunek 5 - koszyk użytkownika

Na mniejszych ekranach strona przedstawiająca artykuły:



Rysunek 6 - na telefonie iPhone

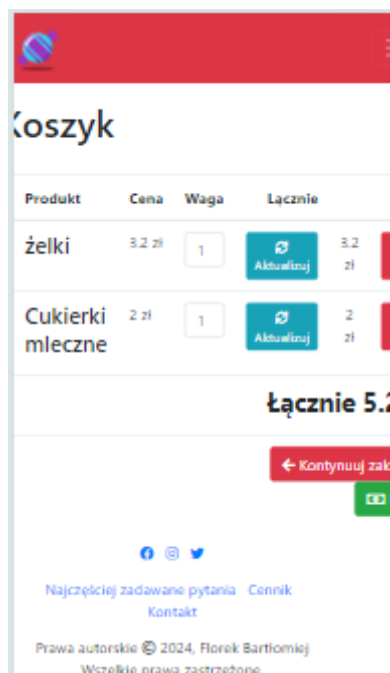
Strona główna:



Rysunek 7 - na telefonie

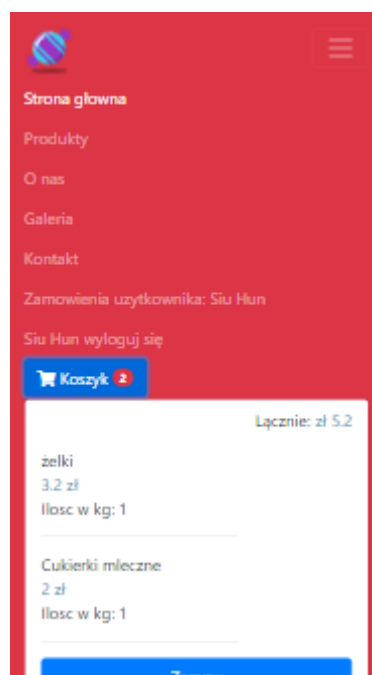


Koszyk:



Rysunek 8 - koszyk na telefonie

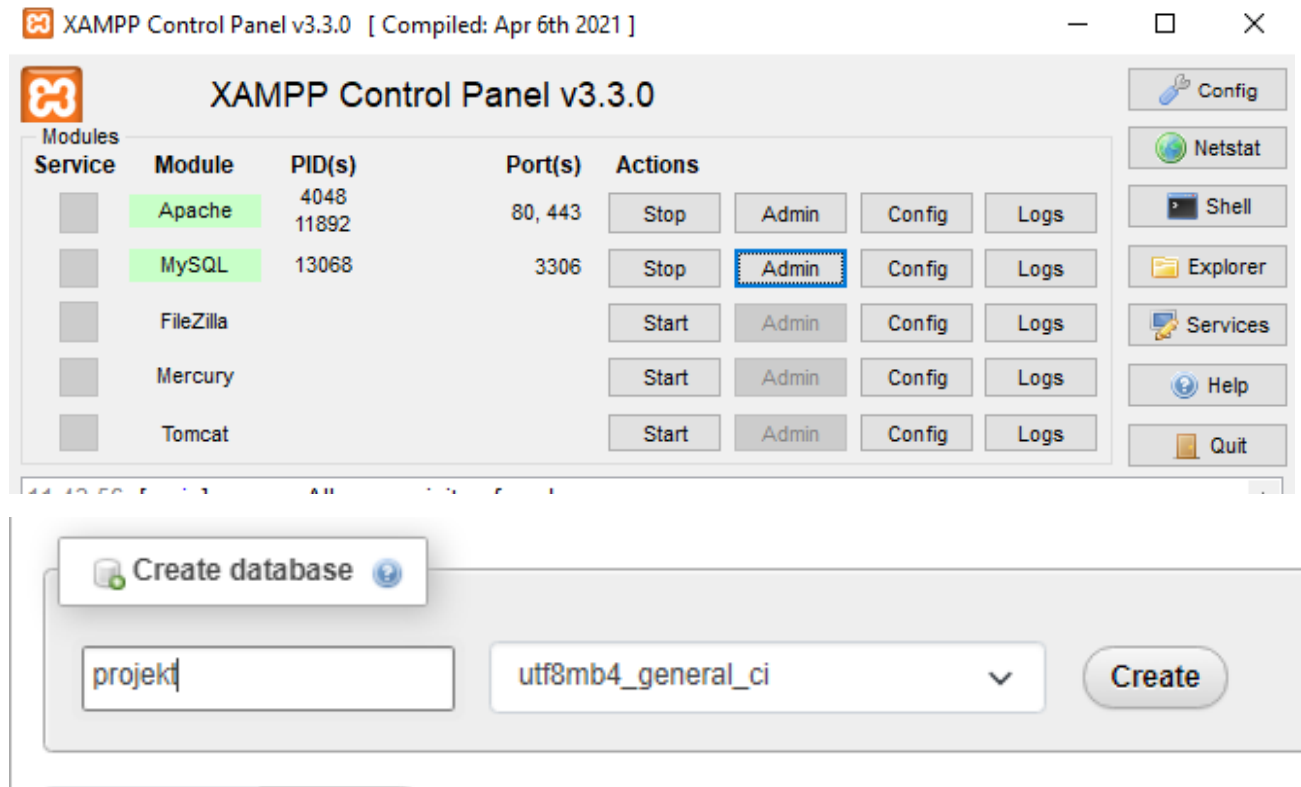
Menu:



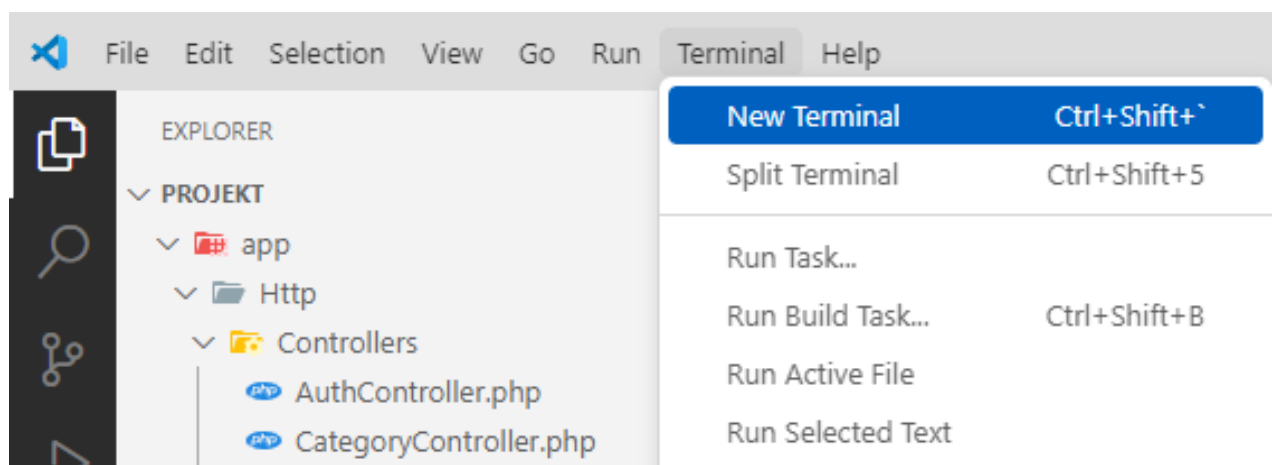
Rysunek 9 - menu na telefonie

## Uruchomienie aplikacji

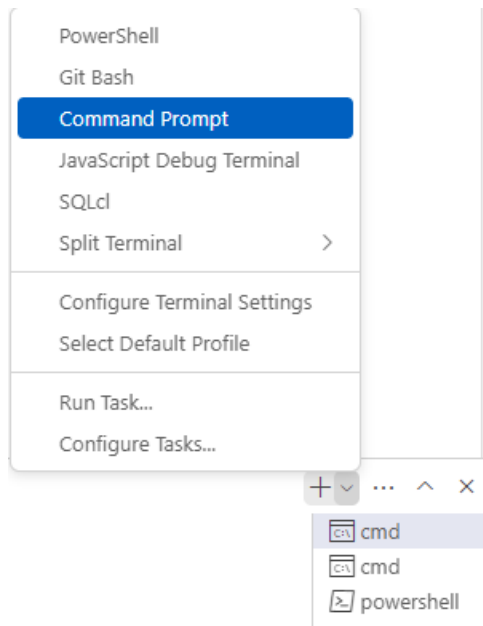
Aby mieć możliwość uruchomienia aplikacji należy mieć pobrane takie aplikacje jak XAMPP, Composer Setup oraz Visual Studio Code. Należy podążać z etapami instalacji. Uruchomić XAMPP Control Panel i wystartować serwer Apache oraz MySQL. Przejść przez przycisk admin przy MySQL do panelu admina. Utworzyć nową bazę danych o nazwie projekt.



Następnie uruchomić aplikację Visual Studio Code, otworzyć folder zawierający pliki projektowe. Przejść do zakładki terminal i otworzyć nową kartę terminala.



Zmienić terminal z powershella na command prompt.



Wpisać komendę 'composer install', to polecenie utworzy folder vendor, teraz trzeba skopiować zawartość pliku .env.example do pliku .env i uzupełnić go tak jak poniżej

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=projekt
DB_USERNAME=root
DB_PASSWORD=
```

*Rysunek 10 - fragment pliku .env*

Po skonfigurowaniu pliku .env należy wpisać w konsoli polecenie 'php artisan key:generate', które wygeneruje klucz do naszego pliku. Bez tego klucza aplikacja nie będzie działała poprawnie. Po wykonaniu tych czynności należy wpisać komendę 'php artisan serve', uruchomi to serwer. Otworzyć drugą kartę terminala i wpisać polecenie 'php artisan migrate', to polecenie utworzy nowe tabele w naszej bazie danych. Aby uzupełnić te tabele przykładowymi danymi musimy wpisać kolejną komendę 'php artisan db:seed'. Teraz po wykonaniu tych poleceń możemy uruchomić przeglądarkę i przejść pod adres <http://127.0.0.1:8000/>.

# Funkcjonalności aplikacji

Gość:

- Utworzenie konta
- Logowanie
- Przeglądanie ogólnodostępnych zasobów
- Wyszukiwanie produktów

Użytkownik:

- Zamawianie produktów
- Dodawanie do koszyka
- Usuwanie swoich zamówień

Administrator:

- Edycja produktów
- Dodawanie nowych produktów
- Usuwanie bieżących produktów

Etap logowania na użytkownika o loginie [jan@email.com](mailto:jan@email.com) o raz hasło 1234, lub na konto admina o loginie [admin@admin.com](mailto:admin@admin.com) o tym samym hasle

## Zaloguj się

Email

jan@email.com

Hasło

....

Wyślij

Email

admin@admin.com

Hasło

....

Wyślij

Tworzenie konta oraz walidacja danych:

- Pole adres e-mail nie jest poprawnym adresem e-mail.
- Pole hasło musi mieć przynajmniej 8 znaków.

Imię

Email

Nieprawidłowy email!

Hasło

Nieprawidłowe hasło!

Powtórz Hasło

Wyślij

Funkcja zajmująca się rejestracją nowego użytkownika

```

51
52 public function registration(){
53     return view('auth.register');
54 }
55
56 public function registrationPost(Request $request)
57 {
58     $request->validate([
59         'name' => 'required',
60         'email' => 'required|email|unique:users',
61         'password' => 'required|min:8',
62         'repass' => 'required|same:password',
63     ], [
64         'repass.same' => 'Potwierdzenie hasła nie zgadza się z hasłem.',
65     ]);
66
67     $data['name'] = $request->name;
68     $data['email'] = $request->email;
69     $data['password'] = Hash::make($request->password);
70
71     $user = User::create($data);
72
73     if(!$user){
74         return redirect(route('registration'))->with("error", "błąd w rejestracji");
75     }
76     return redirect(route('login'))->with("success", "rejestracja przebiegła pomyślnie");
77 }
78

```

Rysunek 11 - fragment kodu do rejestracji

Metoda registration(), jest to metoda GET, która odpowiada za wyświetlanie formularza rejestracji użytkownika, zwracając widok auth.register

Metoda `registrationPost()`, jest to metoda POST, która obsługuje dane przesłane z formularza rejestracji. Przyjmuje obiekt `Request` jako argument, który zawiera dane z formularza. Metoda ta waliduje dane wejściowe, sprawdzając czy wszystkie pola są wymagane, czy pole email jest poprawnym adresem email, niepowtarzającym się w tabeli `users`, czy hasło zawiera co najmniej 8 znaków, i czy pole `repass` jest identyczne.

Jeśli wszystkie dane zostały poprawnie wprowadzone, zostają one przygotowywane do zapisania w bazie danych. Pole `name` oraz `email` pozostaje bez zmian, natomiast hasło jest hashowane za pomocą funkcji `Hash::make()`.


Tworzony jest nowy rekord w tabeli `users`, za pomocą metody `User::create($data)`, w którym przekazujemy podane dane.

Jeśli dodanie przejdzie pomyślnie, użytkownik przekierowywany jest do strony logowania, w przeciwnym wypadku wraca on na stronę rejestracji z odpowiednimi informacjami.

CRUD przeprowadzany przez administratora:

Dodawanie nowych produktów

### Dodaj nowy produkt

Nazwa	<input type="text"/>
Cena za kg	<input type="text"/>
Dostępna ilość kg	<input type="text"/>
Kalorie	<input type="text"/>
Opis	<input type="text"/>
Kategoria	<div>Wybierz kategorię </div>

Wyślij

- Taki nazwa już występuje.
- Pole cena jest wymagane.
- Pole dostępny jest wymagane.
- Pole calories jest wymagane.
- Pole opis jest wymagane.
- Pole category id jest wymagane.

Nazwa

Nieprawidłowa nazwa!

Cena za kg

Nieprawidłowa cena za kg!

Dostępna ilość kg

Nieprawidłowa ilość kg!

Kalorie

Nieprawidłowy kalorie!

Opis

Nieprawidłowy opis!

Kategoria

Wybierz kategorię!

Realizacja walidacji znajduje się w pliku StoreProductRequest, w funkcji rules, która zawiera tablice sprawdzającą wpisane pola

```
/**
 * Get the validation rules that apply to the request.
 *
 * @return array<string, \Illuminate\Contracts\Validation\ValidationRule|array<mixed>|string>
 */
public function rules(): array
{
    return [
        'name' => 'required|string|unique:products,name|max:50',
        'price' => 'required|numeric|max:10|min:1',
        'available' => 'required|numeric|max:30|min:1',
        'calories' => 'required|string|min:10|max:255',
        'description' => 'required|string|min:10|max:255',
        'category_id' => 'required|numeric|exists:categories,id',
    ];
}
```

Rysunek 12 - fragment kodu odpowiedzialny za walidację

Metoda rules(), zwraca tablice reguł walidacji dodawania nowego produktu. Określa ona wymagania dotyczące pól, które muszą zostać spełnione aby dodać nowy produkt do bazy

Reguły walidacji:

- pole 'name' – jest wymagane, typu string, unikalna wartość w tabeli products, oraz max 50 znaków
- pole 'price' – wymagane, musi być liczbą, wartość nie może przekroczyć 10 ale musi być co najmniej 1
- pole 'available' – wymagane, musi być liczbą, max 30, co najmniej 1
- pole 'calories' – wymagane, typu string, min 10 znaków, max 255
- pole 'description' – wymagane, typu string, min 10 znaków, max 255
- pole 'category\_id' – wymagane, musi być liczbą, wartość pola musi istnieć w kolumnie id w tabeli 'categories'

Funkcja w ProductController, która obsługuje dodawanie produktu do bazy

```
public function store(StoreProductRequest $request){  
    $input = $request->all();  
    Product::create($input);  
  
    return redirect()->route('products.home');  
}
```

*Rysunek 13 - funkcja zapisująca nowy produkt*

Funkcja store() przyjmuje request zawierający dane nowego produktu, waliduje je za pomocą klasy StoreProductRequest, a następnie zapisuje te dane w tabeli products.

Pobiera wszystkie dane o nowym produkcie z requesta i zapisuje je w tablicy \$input.

Product::create(\$input) tworzy nowy rekord w tabeli products z danymi zawartymi w tablicy \$input. Teraz następuje przekierowanie do strony głównej produktów.

```
public function create(){  
  
    $categories = Category::all();  
    $user = auth()->user();  
  
    if ($user && ($user->id === 1)) {  
        return view('products.create', compact('categories'));  
    } else {  
        alert("BRAK UPRAWNIEN");  
        return redirect()->route('login.authenticate');  
    }  
  
    // KAZDY MOZE DODAWAC  
    //return view('products.create');
```

*Rysunek 14 - funkcja create*



## Usuwanie wybranego produktu



Usuwanie produktu znajduje się w funkcji `destroy()`, przyjmującej jako argument obiekt `Product`, który jest automatycznie zainicjowany na podstawie ID przekazanego w żądaniu. Następuje sprawdzenie czy aktualnie zalogowany użytkownik jest adminem, a po tym sprawdzenie czy aktualny produkt ma jakieś powiązania z innymi tabelami, np. z zamówieniami, wtedy otrzymamy komunikat, że takie działanie jest niedozwolone.

Jeśli natomiast produkt nie ma żadnych powiązań, tzn. nikt go nie zamówił, zostaje on usunięty, i wracamy do strony głównej z produktami. Jeśli zalogowany użytkownik nie posiada praw do usuwania, aplikacja zabroni mu usunięcia produktu, oraz zwróci odpowiedni błąd i przekieruje go na stronę do logowania.

```
public function destroy(Product $product)
{
    $user = auth()->user();

    if ($user && ($user->id === 1)) {
        if ($product->orders()->count() > 0) {
            return redirect()->route('products.home')->with('error', 'nie mozna usunac produktow ktore zostaly zamowione.');
        }

        $product->delete();
        return redirect()->route('products.home')->with('success', 'produkt usuniety pomyslnie.');
    } else {
        alert("BRAK UPRAWNIEN");
        return redirect()->route('login.authenticate');
    }
}
```

Rysunek 15 - funkcja usuwająca produkt

Edycja wybranego przez admina produktu, zawiera się w funkcji `edit()`, przyjmującej ID klikniętego produktu. Metoda próbuje znaleźć produkt w tabeli `products` oraz pobiera wszystkie rekordy z tabeli `categories` i przypisuje je do zmiennej `$categories`.

Tak samo jak w przypadku usuwania, następuje sprawdzanie aktualnie zalogowanego użytkownika i przekierowanie go bądź nie, do widoku edycji produktu wraz ze zmiennymi.

```
public function edit($ID_produkту)
{
    // KAZDY MOZE EDYTOWAC
    // $product = Product::findOrFail($ID_produkту);
    // return view('products.edit', compact('product'));

    $product = Product::findOrFail($ID_produkту);
    $categories = Category::all();
    $user = auth()->user();

    if ($user && ($user->id === 1)) {
        return view('products.edit', compact('product', 'categories'));
    } else {
        alert("BRAK UPRAWNIEN");
        return redirect()->route('login.authenticate');
    }
}
```

Rysunek 16 - funkcja edytująca produkt

## Edytuj słodycz

Nazwa	<input type="text" value="żelki"/>
Cena za kg	<input type="text" value="3,2"/>
Dostępna ilość kg	<input type="text" value="23"/>
Kaloryka	<input type="text" value="456kcal / 100g"/>
Opis	<input type="text" value="firma xdxdd"/>
Nazwa obrazka	<input type="text" value="zelki.jpg"/>
Kategoria	<input type="text" value="zelki"/>

Wyślij

Po kliknięciu przycisku wyślij, uruchamia się funkcja update(), aktualizującej podany produkt. Funkcja przyjmuje request, zawierający dane do edycji oraz produkt do zaktualizowania. Funkcja realizuje walidacje na takiej samej zasadzie jak w przypadku rejestracji użytkownika. Do tablicy \$input, pobiera wszystkie dane przesłane w requescie. Następnie aktualizuje podany produkt o nowe dane zawarte w tablicy \$input. I wraca do głównego widoku produktów

```

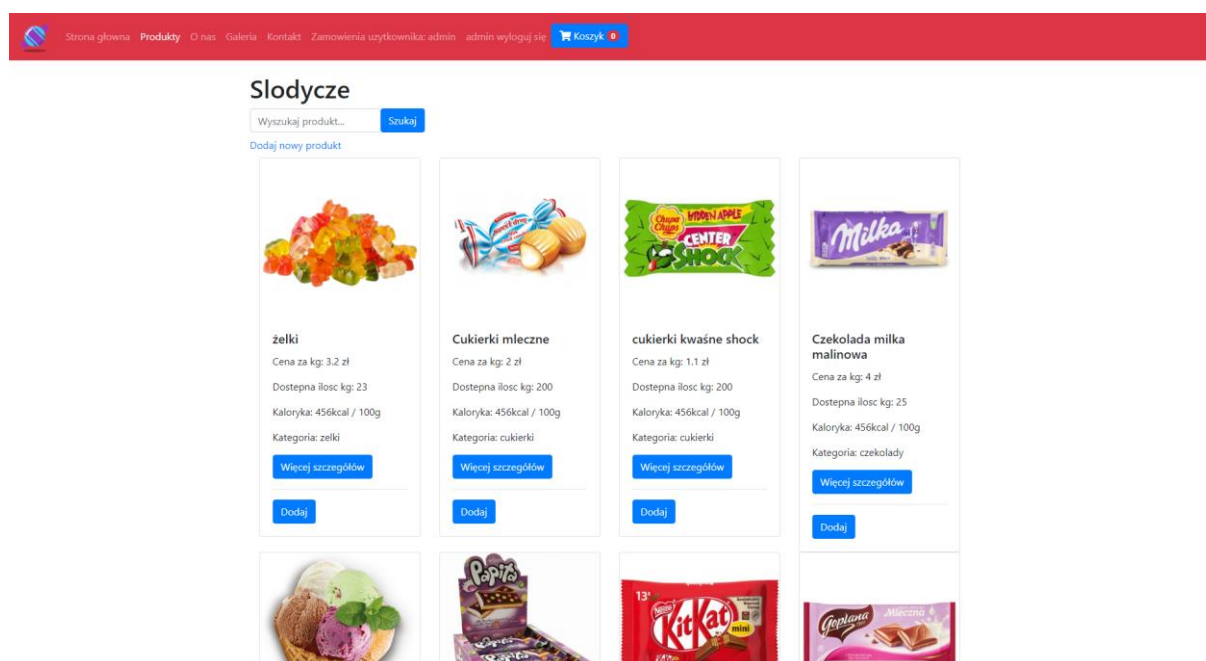
public function update(Request $request, Product $product)
{
    $request->validate([
        'name' => 'required|string|max:50|min:0',
        'price' => 'required|numeric|max:30|min:0',
        'available' => 'required|numeric|max:30|min:0',
        'calories' => 'required|string|max:30|min:0',
        'description' => 'required|string|max:50|min:0',
        'img' => 'required|string|max:50|min:0',
        'category_id' => 'required|numeric|exists:categories,id',
    ]);

    $input = $request->all();
    $product->update($input);
    return redirect()->route('products.home');
}

```

Rysunek 17 - funkcja walidująca edytowane dane

## Przeglądanie ogólnodostępnych zasobów



Funkcja `index()` służy do wyświetlania listy produktów wraz z ich kategoriami. Do zmiennej `$products` przypisywane są produkty wraz z ich kategoriami `with('category')`,

Funkcja zwraca widok `products.home`, przekazując tablice produktów.

```

public function index()
{
    $products = Product::with('category')->get();

    // dd($products);
    return view('products.home', [
        'products' => $products
    ]);
}

```

Rysunek 18 - funkcja zwracająca widok produktów

Poprzez wyszukiwarkę możemy wyszukać interesujące nas produkty, wyszukuje to po nazwie oraz po opisie.

Metoda obsługująca wyszukiwarkę:

```
public function search(Request $request)
{
    $query = $request->input('query');

    // Wyszukaj ogłoszenia zawierające wpisany ciąg znaków w tytule
    $products = Product::where('name', 'like', '%'.$query.'%')
        ->orWhere('description', 'like', '%'.$query.'%')
        ->get();

    // Przekieruj użytkownika do widoku z wynikami wyszukiwania i zmienna query
    return view('search', ['products' => $products, 'query' => $query]);
}
```



Rysunek 19 - funkcja odpowiedzialna za wyszukiwarkę

Do zmiennej \$query przypisujemy wartość wpisaną przez użytkownika.

Metoda wykonuje zapytanie do bazy, szukając produktów, których nazwa 'name' zawiera wpisany przez użytkownika ciąg znaków, albo produktów, których opis 'description' zawiera wpisany przez użytkownika ciąg. Get() pobiera wszystkie produkty spełniające którykolwiek warunek.

Następnie funkcja zwraca widok search i przekazuje do niego tablice danych produktów oraz oryginalne zapytanie użytkownika.

## Wyniki wyszukiwania dla: cukierki

	
<b>Cukierki mleczne</b>	<b>cukierki kwaśne shock</b>
Cena: 2 zł	Cena: 1.1 zł
Kaloryka: 456kcal / 100g	Kaloryka: 456kcal / 100g
Opis: firma ROSHEN	Opis: firma aaaaa
<a href="#">Dodaj</a>	<a href="#">Dodaj</a>

Użytkownik może dodawać produkty do koszyka, które potem w karcie koszyka może edytować w sensie zwiększać ilość, usunąć z koszyka itp. Po złożeniu zamówienia użytkownik może przeglądać swoje zamówienia oraz je usuwać.

Metoda pozwalająca na dodawanie produktów do koszyka:

```

public function addToCart($id)
{
    $product = Product::findOrFail($id);

    $cart = session()->get('cart', []);

    if(isset($cart[$id])) {
        $cart[$id]['quantity']++;
    } else {
        $cart[$id] = [
            "name" => $product->name,
            "price" => $product->price,
            "quantity" => 1
        ];
    }

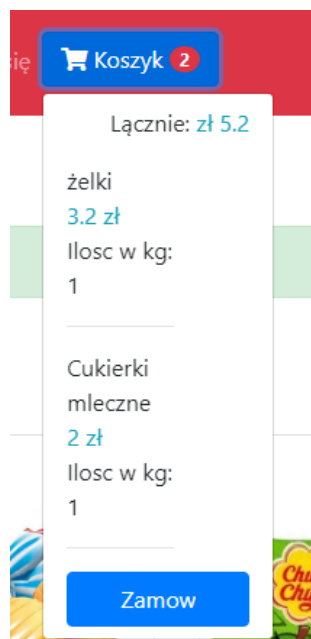
    session()->put('cart', $cart);
    return redirect()->back()->with('success', 'dodano produkt do koszyka!');
}

```

Rysunek 20 - funkcja odpowiedzialna za dodawanie do koszyka

Koszyk przechowywany jest w sesji. Metoda przyjmuje ID jako parametr, następnie próbuje znaleźć produkt o podanym ID w tabeli 'products' i przypisać go do zmiennej \$product.

Do zmiennej \$cart metoda pobiera aktualny stan koszyka, jeśli nie istnieje tworzy pusta tablice. Dodanie lub aktualizowanie produktu w koszyku następuje poprzez sprawdzenie czy produkt o podanym ID znajduje się już w koszyku czy nie. Jeśli tak to zwiększamy jego ilość, jeśli nie to dodajemy go do koszyka z informacjami oraz ustawiamy jego ilość na 1. Na końcu stan koszyka zostaje zapisany w sesji i użytkownik zostaje przekierowany na poprzednią stronę wraz z komunikatem o pomyślnym dodaniu do koszyka.



To jest fragment kodu odpowiedzialny za wyświetlanie produktów w koszyku, na zdjęciu powyżej.

```

@if (Auth::check())
<div class="dropdown">
<button type="button" class="btn btn-primary" data-toggle="dropdown">
<i class="fa fa-shopping-cart" aria-hidden="true"></i> Koszyk <span class="badge badge-pill badge-danger">{{ count((array) session('c
</button>

<div class="dropdown-menu pl-3 pr-3">
<div class="row total-header-section">
<div class="col-lg-12 col-sm-12 col-12 total-section text-right">
<p>Łącznie: <span class="text-info">zł {{ $total }}</span></p>
</div>
</div>
</div>
@if(session('cart'))
@foreach(session('cart') as $id => $details)
<div class="row cart-detail">
<div class="col-lg-8 col-sm-8 col-8 cart-detail-product">
{{ $details['name'] }} <br>
<span class="price text-info">{{ $details['price'] }} zł</span> <br>
<span class="count">Ilość w kg: {{ $details['quantity'] }}</span>
<hr>
</div>
</div>
@endforeach
<div class="row">
<div class="col-lg-12 col-sm-12 col-12 text-center checkout">
<a href="{{ route('cart') }}" class="btn btn-primary btn-block">Zamów</a>
</div>
</div>
</div>

```

Rysunek 21 - fragment kodu odpowiedzialny za wyświetlanie produktów w koszyku

Po kliknięciu w przycisk 'zamów', aplikacja przekieruje nas do okna koszyka. Ta akcja realizowana jest za pomocą funkcji `cart()`, która zwraca widok 'cart'

```

public function cart()
{
    return view('cart');
}

```

Rysunek 22 - funkcja zwracająca widok koszyka

Widok koszyka zawiera aktualne produkty w koszyku, ich cenę, wagę/ilość, którą możemy aktualizować, ich łączną cenę oraz przycisk usunąć:

## Koszyk

Produkt	Cena	Waga		Łącznie	
żelki	3.2 zł	<input type="text" value="1"/>	<button>Aktualizuj</button>	3.2 zł	<button>Usun</button>
Cukierki mleczne	2 zł	<input type="text" value="1"/>	<button>Aktualizuj</button>	2 zł	<button>Usun</button>
<b>Łącznie 5.2 zł</b>					
					<button>Kontynuuj zakupy</button>
					<button>Kup</button>

Po kliknięciu w przycisk kontynuuj zakupy, aplikacja wróci nas do strony głównej z produktami, a przycisk kup zresetuje koszyk i wyśle do bazy danych, do tabeli 'orders', obecne zamówienie.

Fragment kodu odpowiedzialny za usunięcie z koszyka:

```

public function remove(Request $request)
{
    if($request->id) {
        $cart = session()->get('cart');
        if(isset($cart[$request->id])) {
            unset($cart[$request->id]);
            session()->put('cart', $cart);
        }
        session()->flash('success', 'Produkt usunięty z koszyka!');
    }
    return redirect()->route('cart');
}

```

*Rysunek 23 - funkcja odpowiedzialna za usuwanie produktu z koszyka*

Metoda przyjmuje obiekt request, przechowujący dane przesłane w formularzu. W metodzie następuje sprawdzenie czy w żądaniu znajduje się ID produktu, jeśli nie metoda zakończy się i przekieruje nas do strony koszyka. Jeśli jednak ID znajduje się w żądaniu, wtedy następuje pobranie do zmiennej \$cart aktualnego stanu koszyka. Teraz metoda sprawdza czy produkt znajduje się w koszyku, jeśli tak to unset(\$cart[\$request->id]) usunie go z koszyka i zapisze stan koszyka w sesji. Zostaje dodany komunikat o pomyślnym usunięciu z koszyka i przekierowanie do koszyka.

Fragment kodu odpowiedzialny za aktualizowanie ilości konkretnego produktu:

```

public function updatee(Request $request)
{
    if($request->id && $request->quantity){
        $cart = session()->get('cart');
        $cart[$request->id]["quantity"] = $request->quantity;
        session()->put('cart', $cart);
        session()->flash('success', 'Koszyk zaktualizowany!');
    }
    return redirect()->route('cart');
}

```

*Rysunek 24 - funkcja aktualizująca ilość produktów w koszyku*

Tak samo jak w przypadku usunięcia z koszyka sprawdzane jest czy w żądaniu znajduje się ID produktu jak i jego ilość. Jeśli któryś z tych warunków nie jest spełniony następuje zakończenie metody i zwrócenie widoku koszyka. Jeśli natomiast oba warunki są spełnione pobieramy aktualny stan koszyka z sesji, i następuje aktualizacja ilości produktu o podanym ID na nową wartość podaną w requeście. Potem zostaje zapisany nowy stan koszyka w sesji i poinformowanie użytkownika o pomyślnej aktualizacji koszyka.

Fragment kodu odpowiedzialny zapisanie zamówienia do tabeli po kliknięciu w przycisk 'kup', znajduje się w pliku OrderController:

```

public function store(Request $request)
{
    $cart = session('cart');

    foreach ($cart as $productId => $details) {
        $order = new Order();
        $order->product_id = $productId;
        $order->user_id = Auth::id();
        $order->order_date = now();
        $order->order_kg = $details['quantity'];
        $order->order_price = $details['price'] * $details['quantity'];
        $order->save();
    }

    session()->forget('cart');

    return redirect()->route('products.home')->with('success', 'Zamówienie zostało złożone pomyślnie!');
}

```

Rysunek 25 - funkcja wysyłająca zamówienie do bazy

Metoda przyjmuje jako parametr obiekt request, zawierające dane o nowym zamówieniu. Do zmiennej \$cart zostaje przypisany aktualny stan koszyka z sesji. Teraz następuje iteracja w pętli po każdym produkcie zawartym w koszyku z aktualnej sesji. W pętli dla każdego produktu tworzone jest nowe zamówienie z jego obecnymi informacjami takimi jak: data zamówienia, ilość sztuk/kg, cena. Po zapisaniu danego zamówienia, koszyk w sesji zostaje opróżniony i użytkownik jest przekierowywany do głównej strony z produktami z informacją, że jego zamówienie zostało pomyślnie złożone.

Strona przedstawiająca aktualne zamówienia dla obecnie zalogowanego użytkownika:

## Lista zamówień dla użytkownika: admin

#	Produkt	Data	Waga	Cena	
4	żelki	2024-05-17	5 kg	27.12 zł	Usuń
5	cukierki kwaśne shock	2024-05-17	3 kg	20.5 zł	Usuń
10	Baton Mars	2024-05-17	2 kg	2.4 zł	Usuń
11	Ciastka Milka Choco Cookies	2024-05-17	5 kg	12.5 zł	Usuń
12	żelki	2024-05-20	3 kg	9.6 zł	Usuń
13	Cukierki mleczne	2024-05-20	1 kg	2 zł	Usuń

Fragment kodu odpowiedzialny za wyświetlanie zamówień, zawarty w pliku OrderController:



```

public function showOrders()
{
    if (Auth::check()) {
        // id usera
        $userId = Auth::id();

        // pobieranie zamówienia
        $orders = Order::where('user_id', $userId)->get();

        return view('orders.show', ['orders' => $orders]);
    } else {
        // przekierowanie do loginu
        return redirect()->route('login');
    }
}

```

Rysunek 26 - funkcja wyświetlająca zamówienia użytkownika

Metoda sprawdza czy użytkownik jest zalogowany, jeśli nie to przekierowuje go do strony logowania, a jeśli jest zalogowany, to przypisuje do zmiennej \$userId, ID aktualnie zalogowanego użytkownika. Do zmiennej \$orders zostają przypisane wszystkie zamówienia z tabeli 'orders' dla użytkownika o podanym ID (w naszym przypadku aktualnie zalogowanego). Na koniec zostaje zwrócony widok 'orders.show' wraz z tablicą orders, zawierającą wszystkie zamówienia.

Strona z zamówieniami umożliwia użytkownikowi usuwanie zamówień. To fragment kodu za to odpowiedzialny:

```

public function destroy(Order $order)
{
    $order->delete();
    return redirect()->route('orders.show');
}

```

Rysunek 27 - funkcja usuwająca zamówienia

Metoda destroy(), przyjmuje jako parametr obiekt typu Order. Następnie pobrany obiekt \$order zostaje usunięte z bazy i aplikacja przekieruje nas na stronę zamówień.

## Lista zamówień dla użytkownika: admin

#	Produkt	Data	Waga	Cena	
4	zёлki	2024-05-17	5 kg	27,12 zł	<button>Usuń</button>
10	Baton Mars	2024-05-17	2 kg	2,4 zł	<button>Usuń</button>

Po kliknięciu 'więcej szczegółów' w karcie produktu, aplikacja otworzy szczegółową kartę z danymi o produkcie. Realizuje to metoda show() w ProductController:

```

public function show(Product $product)
{
    return view('products.show', compact('product'));
}

```

Rysunek 28 - funkcja zwracająca szczegółowy widok produktu

Metoda ta przyjmuje jako parametr obiekt typu Product, i zwraca widok 'products.show' z przekazaniem do niego parametru.

Aby móc zapisywać własne logi do pliku, musimy wykonać polecenie: 'php artisan make:middleware LogActivity', utworzy to nowy middleware w aplikacji o podanej nazwie. Middleware jest warstwą pośrednią, która może być wykonywana przed lub po przetworzeniu żądania http przez aplikację. W miejscu 'Log::channel(...)' następuje podmiana nazwy kanału na który chcemy wysyłać logi na poziomie 'notice', które informuje, że ktoś wykonał metodę http na określonym URI.

```

public function handle(Request $request, Closure $next): Response
{
    Log::channel('projekt_errors')->notice(
        "Someone did {$request->method()} on '{$request->route()->uri()}'"
    );
    $email = $request->user() ? $request->user()->email : 'anonymous';
    Log::channel('single')->notice(
        "'{$email}' did {$request->method()} on '{$request->getRequestUri()}'"
    );
    return $next($request);
}

```

Rysunek 29 - funkcja ustawiająca nowy kanał dla logów

Zapisywanie logów do pliku o nazwie 'projekt\_errors.log'. W pliku 'config/logging.php' utworzony jest nowy kanał o nazwie 'projekt\_errors', który będzie rejestrował tylko błędy o poziomie 'error'. Określenie sterownika jako 'single', oznacza, że kanał będzie używał pojedynczego sterownika logowania. Path określa ścieżkę do pliku, w którym będą rejestrowane błędy.

```

'channels' => [
    'projekt_errors' => [
        'driver' => 'single',
        'path' => storage_path('logs/projekt_errors.log'),
        'level' => 'error',
    ],
],

```

Rysunek 30 - dodawanie nowego kanału

W pliku 'bootstrap/app.php' aby poprawnie działało zapisywanie logów do pliku 'projekt\_errors', trzeba ustawić obsługę raportowania wyjątków.

```

return Application::configure(basePath: dirname(__DIR__))
    ->withRouting(
        web: __DIR__.'/../routes/web.php',
        commands: __DIR__.'/../routes/console.php',
        health: '/up',
    )
    ->withMiddleware(function (Middleware $middleware) {
        $middleware->web(LogActivity::class);
    })
    ->withExceptions(function (Exceptions $exceptions) {

        $exceptions->report(function (Exception $e) {
            Log::channel('projekt_errors')->error($e->getMessage());
        });

    })->create();

```

Rysunek 31 - przekierowanie błędów do kanału

`$exceptions->report...` raportuje każdy wyjątek przechwycony w aplikacji i wysyła go do kanału 'projekt\_error', który ustawiliśmy wcześniej.

Aby móc zmienić język wyświetlanych komunikatów o błędach walidacji musimy wykonać następujące komendy: 'composer require laravel-lang/common - - dev', 'php artisan lang:add pl' i na koniec 'php artisan lang:update'.

Te polecenia wykonają po kolei: dodanie pakietu laravel-lang/common jako zależność. Pakiet ten zawiera zestaw plików tłumaczeń dla Laravela w wielu językach. Kolejna komenda dodaje nowy język tłumaczeń do aplikacji, w tym przypadku polski. Ostatnie polecenie aktualizuje tłumaczenia aplikacji na podstawie plików znajdujących się w pakiecie zainstalowanym pierwszą komendą. Oznacza to, że zostaną dodane brakujące tłumaczenia, a także zaktualizowane istniejące tłumaczenia.

W 'config/app.php' została wprowadzona zmiana aby błędy walidacji były wyświetlane w języku polskim. Dzięki poprzednio wykonanym poleceniom.

```

79      */
80
81      'locale' => 'pl',
82

```

Rysunek 32 - zmiana języka komunikatów