Korzystając z świeżo zainstalowanego Laravel:  
1. Utwórz nową trasę „/about”, która używa widoku Blade o nazwie 'about', aby wyświetlić swoje imię i nazwisko jak pokazano poniżej.  
2. Wklej poniżej kod użyty zarówno w pliku routingu, jak i w pliku widoku 'about'.

Aby utworzyć nową trasę /about w Laravel, która wyświetla Twoje imię i nazwisko za pomocą widoku Blade, wykonaj poniższe kroki:

### 1. Utwórz trasę w pliku routingu

Otwórz plik routes/web.php, a następnie dodaj następujący kod, aby zdefiniować trasę /about, która będzie renderować widok o nazwie about:

php

use Illuminate\Support\Facades\Route;

Route::get('/about', function () {

return view('about');

});

### 2. Utwórz widok Blade

Teraz musisz stworzyć plik widoku Blade o nazwie about.blade.php. Utwórz go w katalogu resources/views, tworząc nowy plik o nazwie about.blade.php. W tym pliku wpisz kod HTML, który wyświetli Twoje imię i nazwisko. Na przykład:

blade

<!DOCTYPE html>

<html lang="pl">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>About</title>

</head>

<body>

<h1>Moje imię i nazwisko: Jan Kowalski</h1>

</body>

</html>

### Podsumowanie

* **Plik routingu (**routes/web.php**)**:

php

Skopiuj kod

use Illuminate\Support\Facades\Route;

Route::get('/about', function () {

return view('about');

});

* **Plik widoku Blade (**resources/views/about.blade.php**)**:

blade

<!DOCTYPE html>

<html lang="pl">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>About</title>

</head>

<body>

<h1>Moje imię i nazwisko: Jan Kowalski</h1>

</body>

</html>

Po wykonaniu tych kroków, odwiedzając trasę /about w swojej aplikacji Laravel, zobaczysz stronę wyświetlającą Twoje imię i nazwisko.

1. Skorzystaj z dowolnej usługi AI (np. ChatGPT), aby stworzyć plik Blade wyświetlający przykładowe CV z losowym zdjęciem jak pokazano poniżej.  
   Sformatuj stronę z pomocą Bootstrap.  
     
   2. Użyj NPM, aby zainstalować Bootstrap i zaimportować plik CSS Bootstrap w "resources/css/app.css" używając na samym początku pliku:  
   @import "bootstrap/dist/css/bootstrap.min.css";  
     
   3. W swoim pliku Blade 'about', dołącz skrypty JS i CSS zarządzane przez Vite używając manifestu:  
   @vite(['resources/css/app.css', 'resources/js/app.js'])  
     
   4. Dodaj zdjęcie do folderu "resources/images", a następnie:  
   W swoim pliku Blade odwołaj się do niego używając:  
   src="{{ Vite::asset('resources/images/myimage.jpg') }}"  
   A w resources/js/app.js dodaj następującą linię: import.meta.glob(["../images/\*\*"]);  
     
   5. Użyj npm, aby zbudować wszystkie zasoby Vite.  
     
   6. Wywołaj i wyświetl stronę „about” w swojej przeglądarce.  
     
   7. Wklej poniżej kod użyty w pliku widoku 'about'.

### 1. Tworzenie pliku Blade wyświetlającego przykładowe CV z losowym zdjęciem

W pliku Blade about.blade.php stworzymy przykładowe CV, korzystając z frameworku Bootstrap do stylowania.

#### Plik resources/views/about.blade.php:

blade

<!DOCTYPE html>

<html lang="pl">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Moje CV</title>

@vite(['resources/css/app.css', 'resources/js/app.js'])

</head>

<body>

<div class="container mt-5">

<div class="row">

<div class="col-md-4">

<img src="{{ Vite::asset('resources/images/myimage.jpg') }}" class="img-fluid rounded-circle" alt="Zdjęcie"

style="width: 200px; height: 200px;">

</div>

<div class="col-md-8">

<h1>Jan Kowalski</h1>

<p><strong>Stanowisko:</strong> Full Stack Developer</p>

<p><strong>Telefon:</strong> +48 123 456 789</p>

<p><strong>Email:</strong> jan.kowalski@example.com</p>

<p><strong>LinkedIn:</strong> <a href="https://www.linkedin.com/in/jankowalski" target="\_blank">jan.kowalski</a></p>

<h3>Doświadczenie zawodowe</h3>

<ul>

<li><strong>Senior Developer</strong> - Firma XYZ (2018 - obecnie)</li>

<li><strong>Junior Developer</strong> - Firma ABC (2015 - 2018)</li>

</ul>

<h3>Wykształcenie</h3>

<ul>

<li><strong>Magister Informatyki</strong> - Uniwersytet XYZ (2011 - 2015)</li>

</ul>

<h3>Umiejętności</h3>

<ul>

<li>PHP, Laravel, JavaScript, Vue.js</li>

<li>HTML5, CSS3, Bootstrap</li>

<li>Git, Docker, MySQL</li>

</ul>

</div>

</div>

</div>

</body>

</html>

### 2. Instalacja Bootstrap za pomocą NPM i importowanie pliku CSS

Zainstaluj Bootstrap za pomocą NPM. Otwórz terminal w katalogu głównym projektu Laravel i uruchom:

npm install bootstrap

Następnie, aby zaimportować plik CSS Bootstrap w pliku resources/css/app.css, otwórz go i dodaj następującą linię na początku:

css

Skopiuj kod

@import "bootstrap/dist/css/bootstrap.min.css";

### 3. Dołączenie plików CSS i JS w pliku Blade za pomocą Vite

W pliku Blade about.blade.php już dodałeś dyrektywę @vite, która pozwala na zarządzanie skryptami i stylem za pomocą Vite. Upewnij się, że jest tam:

blade

@vite(['resources/css/app.css', 'resources/js/app.js'])

### 4. Dodanie zdjęcia do folderu resources/images

Prześlij zdjęcie do katalogu resources/images. Załóżmy, że zdjęcie nazywa się myimage.jpg. Aby wyświetlić to zdjęcie w pliku Blade, użyj następującego kodu:

blade

<img src="{{ Vite::asset('resources/images/myimage.jpg') }}" class="img-fluid rounded-circle" alt="Zdjęcie"

style="width: 200px; height: 200px;">

Ponadto, w pliku resources/js/app.js dodaj linię, aby Vite mogło zarządzać tym zdjęciem:

javascript

Skopiuj kod

import.meta.glob(["../images/\*\*"]);

### 5. Budowanie zasobów Vite

Aby zbudować wszystkie zasoby Vite, uruchom następujące polecenie w terminalu:

npm run build

To polecenie spowoduje zbudowanie plików JS i CSS, które będą odpowiednio zoptymalizowane i przygotowane do użycia w produkcji.

### 6. Wywołanie i wyświetlenie strony „about” w przeglądarce

Po zbudowaniu zasobów uruchom serwer lokalny Laravel:

php artisan serve

Następnie odwiedź stronę /about w swojej przeglądarce:

arduino

Skopiuj kod

http://127.0.0.1:8000/about

Strona powinna wyświetlić przykładowe CV z losowym zdjęciem.

### 7. Kod użyty w pliku Blade about.blade.php

Podsumowując, oto pełny kod widoku about.blade.php:

blade

<!DOCTYPE html>

<html lang="pl">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Moje CV</title>

@vite(['resources/css/app.css', 'resources/js/app.js'])

</head>

<body>

<div class="container mt-5">

<div class="row">

<div class="col-md-4">

<img src="{{ Vite::asset('resources/images/myimage.jpg') }}" class="img-fluid rounded-circle" alt="Zdjęcie"

style="width: 200px; height: 200px;">

</div>

<div class="col-md-8">

<h1>Jan Kowalski</h1>

<p><strong>Stanowisko:</strong> Full Stack Developer</p>

<p><strong>Telefon:</strong> +48 123 456 789</p>

<p><strong>Email:</strong> jan.kowalski@example.com</p>

<p><strong>LinkedIn:</strong> <a href="https://www.linkedin.com/in/jankowalski" target="\_blank">jan.kowalski</a></p>

<h3>Doświadczenie zawodowe</h3>

<ul>

<li><strong>Senior Developer</strong> - Firma XYZ (2018 - obecnie)</li>

<li><strong>Junior Developer</strong> - Firma ABC (2015 - 2018)</li>

</ul>

<h3>Wykształcenie</h3>

<ul>

<li><strong>Magister Informatyki</strong> - Uniwersytet XYZ (2011 - 2015)</li>

</ul>

<h3>Umiejętności</h3>

<ul>

<li>PHP, Laravel, JavaScript, Vue.js</li>

<li>HTML5, CSS3, Bootstrap</li>

<li>Git, Docker, MySQL</li>

</ul>

</div>

</div>

</div>

</body>

</html>

To wszystko! Po wykonaniu tych kroków strona powinna wyświetlać Twoje przykładowe CV z odpowiednim zdjęciem, korzystając z Bootstrap i zarządzania zasobami przez Vite.

Korzystając z wcześniej utworzonego pliku "about":  
  
1. Podziel zawartość pliku na trzy różne widoki układu: nagłówek, treść, stopka.  
2. Utwórz plik "common.blade.php", który zawiera widoki nagłówka i stopki oraz wstawia(yield) dostarczoną sekcję treści.  
3. Zmodyfikuj "about.blade.php", aby rozszerzał plik widoku "common" i dostarczył rzeczywistą treść jako sekcję o nazwie "content" oraz tytuł HTML jako sekcję o nazwie "title".  
4. Dodaj (niefunkcjonalne) menu w nagłówku i stopkę wyświetlającą prostą informację o prawach autorskich jak pokazano poniżej.  
5. Wklej poniżej kod wszystkich nowych lub zmodyfikowanych plików Blade.

### 1. Plik common.blade.php

W tym pliku zdefiniujemy strukturę, która będzie zawierać nagłówek, stopkę oraz miejsce na dynamiczną treść (przekazywaną przez yield).

#### resources/views/common.blade.php:

<!DOCTYPE html>

<html lang="pl">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>@yield('title', 'Moje CV')</title>

@vite(['resources/css/app.css', 'resources/js/app.js'])

</head>

<body>

<!-- Nagłówek -->

<header class="bg-primary text-white p-3">

<div class="container d-flex justify-content-between">

<h1>Moje CV</h1>

<nav>

<ul class="list-unstyled d-flex">

<li class="me-3"><a href="/" class="text-white">Strona główna</a></li>

<li><a href="/about" class="text-white">O mnie</a></li>

</ul>

</nav>

</div>

</header>

<!-- Sekcja treści -->

<div class="container my-5">

@yield('content')

</div>

<!-- Stopka -->

<footer class="bg-dark text-white text-center p-3">

<p>&copy; 2024 Jan Kowalski. Wszystkie prawa zastrzeżone.</p>

</footer>

</body>

</html>

### 2. Zmodyfikowany plik about.blade.php

W tym pliku będziemy rozszerzać widok common.blade.php, dostarczając treść i tytuł jako sekcje o nazwach content i title.

#### resources/views/about.blade.php:

@extends('common')

@section('title', 'O mnie')

@section('content')

<div class="row">

<div class="col-md-4">

<img src="{{ Vite::asset('resources/images/myimage.jpg') }}" class="img-fluid rounded-circle" alt="Zdjęcie"

style="width: 200px; height: 200px;">

</div>

<div class="col-md-8">

<h2>Jan Kowalski</h2>

<p><strong>Stanowisko:</strong> Full Stack Developer</p>

<p><strong>Telefon:</strong> +48 123 456 789</p>

<p><strong>Email:</strong> jan.kowalski@example.com</p>

<p><strong>LinkedIn:</strong> <a href="https://www.linkedin.com/in/jankowalski" target="\_blank">jan.kowalski</a></p>

<h3>Doświadczenie zawodowe</h3>

<ul>

<li><strong>Senior Developer</strong> - Firma XYZ (2018 - obecnie)</li>

<li><strong>Junior Developer</strong> - Firma ABC (2015 - 2018)</li>

</ul>

<h3>Wykształcenie</h3>

<ul>

<li><strong>Magister Informatyki</strong> - Uniwersytet XYZ (2011 - 2015)</li>

</ul>

<h3>Umiejętności</h3>

<ul>

<li>PHP, Laravel, JavaScript, Vue.js</li>

<li>HTML5, CSS3, Bootstrap</li>

<li>Git, Docker, MySQL</li>

</ul>

</div>

</div>

@endsection

Korzystając z wcześniej utworzonego pliku Blade "about" i trasy "about":  
  
1. Zezwól na wszystkie poniższe żądania (i wyświetl podane wartości jak pokazano na zrzucie ekranu poniżej)  
http://localhost/about?reference=123 (wyświetla id 100, referencja 123)  
http://localhost/about/20/?reference=123 (wyświetla id 20, referencja 123)  
http://localhost/about/20/?reference=123 (wyświetla id 20, referencja 123)  
http://localhost/about/2020/?reference=123 (wyświetla id 2020, referencja 123)  
http://localhost/about/2020 (wyświetla id 2020, referencja nie jest wyświetlana)  
http://localhost/about (wyświetla id 100, referencja NIE jest wyświetlana)  
  
2. Zablokuj następujące żądania  
http://localhost/about/abc/?reference=123 (odpowiedz 404)

### 1. Modyfikacja trasy w routes/web.php

Musimy zmodyfikować trasę w pliku routes/web.php, aby odpowiednio obsługiwała te przypadki.

#### routes/web.php

use Illuminate\Support\Facades\Route;

Route::get('/about/{id?}', function ($id = 100) {

// Walidacja: Upewnij się, że id to liczba całkowita

if (!is\_numeric($id)) {

abort(404); // Jeśli id nie jest liczbą, zwróć błąd 404

}

// Pobierz parametr 'reference' z query string

$reference = request('reference');

// Zwróć widok z przekazanymi danymi

return view('about', ['id' => $id, 'reference' => $reference]);

});

### Wyjaśnienie:

* {id?}: Parametr id jest opcjonalny. Jeśli nie zostanie podany, domyślnie będzie ustawiony na 100 (tak jak w przypadku http://localhost/about).
* request('reference'): Pobieramy parametr reference z query string.
* abort(404): Używamy tej funkcji, aby zablokować nieprawidłowe zapytania, w tym te, w których id nie jest liczbą (np. abc).

### 2. Modyfikacja widoku about.blade.php

W pliku widoku musimy wyświetlić odpowiednie informacje o id oraz reference, jeśli są dostępne.

#### resources/views/about.blade.php

@extends('common')

@section('title', 'O mnie')

@section('content')

<div class="row">

<div class="col-md-12">

<h2>Informacje</h2>

<p><strong>ID:</strong> {{ $id }}</p>

@if ($reference)

<p><strong>Referencja:</strong> {{ $reference }}</p>

@else

<p><strong>Referencja:</strong> nie została podana.</p>

@endif

</div>

</div>

@endsection

Instalacja Laravel Breeze:  
  
1. Używając Composer, zainstaluj pakiety Laravel Breeze:  
composer require laravel/breeze --dev  
  
2. Używając Artisan, zainstaluj Laravel Breeze na swojej instalacji Laravel:  
php artisan breeze:install  
  
Podczas instalacji, wybierz następujące opcje:  
> Frontend: Blade  
> Brak wsparcia dla trybu ciemnego  
> Pest jako framework testowy  
Uwaga: Jeśli instalujesz Breeze na istniejącej instalacji Laravel, ponieważ Breeze nie dodaje plików, a jedynie je zastępuje, musisz ręcznie połączyć różnice (np. widoki i trasy).  
  
3. Zainstaluj wymagane pakiety NPM:  
npm install  
  
4. Utwórz niezbędne zasoby: npm run dev  
Lub zbuduj zasoby do produkcji: npm run build  
  
5. Zastosuj migracje bazy danych jesli potrzebne.

php artisan migrate  
  
6. Wklej poniżej:  
- Zrzut ekran Profilu zalogowanego użytkownika w przeglądarce.

### Język i lokalizacja: 1. Używając Artisan, stwórz katalog językowy w aplikacji i opublikuj domyślne pliki językowe używane przez Laravel: php artisan lang:publish 2. W katalogu "lang", utwórz nowy folder "pl" i utwórz nowy plik o nazwie "registration.php". 3. W nowym pliku "registration.php", dodaj polskie tłumaczenia dla poniższych kluczy: - username - name - email - password - confirm password - already registered - register 4. W RegisterRouteController (lub odpowiednim kontrolerze) w metodzie wyświetlającej formularz rejestracji, ustaw lokalizację na "pl": 5. Zmodyfikuj widok rejestracji, aby używać tłumaczeń: \_\_("registration.<your-key>") 6. Wklej poniżej: - Zmienioną metodę w RegisteredUserController. - Plik: registration.php z tłumaczeniami. - Zaktualizowany widok formularza rejestracji. - Zrzut ekranu przetłumaczonego formularza rejestracji, w tym URL przeglądarki.

### 1. Stworzenie katalogu językowego i opublikowanie domyślnych plików językowych

W terminalu uruchom polecenie Artisan, aby stworzyć katalog językowy i opublikować domyślne pliki:

php artisan lang:publish

Po tym poleceniu Laravel utworzy wszystkie domyślne pliki językowe w katalogu resources/lang, w tym pliki dla takich języków jak angielski, francuski, itp.

### 2. Utworzenie folderu "pl" i pliku "registration.php"

Następnie, w katalogu resources/lang, utwórz folder pl oraz plik registration.php:

bash

Skopiuj kod

resources/lang/pl/registration.php

### 3. Dodanie tłumaczeń do pliku registration.php

Otwórz plik resources/lang/pl/registration.php i dodaj tłumaczenia dla poniższych kluczy:

#### resources/lang/pl/registration.php

<?php

return [

'username' => 'Nazwa użytkownika',

'name' => 'Imię',

'email' => 'Adres e-mail',

'password' => 'Hasło',

'confirm\_password' => 'Potwierdź hasło',

'already\_registered' => 'Masz już konto?',

'register' => 'Zarejestruj się',

];

### 4. Ustawienie lokalizacji na "pl" w metodzie kontrolera

W kontrolerze, który odpowiada za wyświetlanie formularza rejestracji (np. RegisteredUserController), ustaw lokalizację na polski:

#### app/Http/Controllers/Auth/RegisteredUserController.php

W metodzie, która wyświetla formularz rejestracji (np. create()), dodaj ustawienie lokalizacji:

use Illuminate\Support\Facades\App;

public function create(){

// Ustawienie lokalizacji na polski

App::setLocale('pl');

return view('auth.register');

}

### 5. Zmodyfikowanie widoku formularza rejestracji

W widoku rejestracji, użyj funkcji \_\_() do wyświetlania tłumaczeń dla odpowiednich kluczy. Otwórz plik formularza rejestracji (zwykle znajduje się w resources/views/auth/register.blade.php) i zmodyfikuj go, aby używał tłumaczeń:

#### resources/views/auth/register.blade.php

@extends('common')

@section('content')

<div class="container">

<h2>{{ \_\_('registration.register') }}</h2>

<form method="POST" action="{{ route('register') }}">

@csrf

<div class="mb-3">

<label for="username" class="form-label">{{ \_\_('registration.username') }}</label>

<input type="text" id="username" name="username" class="form-control" value="{{ old('username') }}" required autofocus>

</div>

<div class="mb-3">

<label for="name" class="form-label">{{ \_\_('registration.name') }}</label>

<input type="text" id="name" name="name" class="form-control" value="{{ old('name') }}" required>

</div>

<div class="mb-3">

<label for="email" class="form-label">{{ \_\_('registration.email') }}</label>

<input type="email" id="email" name="email" class="form-control" value="{{ old('email') }}" required>

</div>

<div class="mb-3">

<label for="password" class="form-label">{{ \_\_('registration.password') }}</label>

<input type="password" id="password" name="password" class="form-control" required>

</div>

<div class="mb-3">

<label for="password\_confirmation" class="form-label">{{ \_\_('registration.confirm\_password') }}</label>

<input type="password" id="password\_confirmation" name="password\_confirmation" class="form-control" required>

</div>

<div class="mb-3">

<button type="submit" class="btn btn-primary">{{ \_\_('registration.register') }}</button>

</div>

<div class="mb-3">

<p>{{ \_\_('registration.already\_registered') }} <a href="{{ route('login') }}">{{ \_\_('registration.login') }}</a></p>

</div>

</form>

</div>

@endsection

Dostosowywanie Rejestracji w Laravelu  
  
1. Otwórz widok rejestracji i dodaj nowe pole wejściowe o nazwie 'username'.  
  
2. Zmodyfikuj RegisteredUserController, aby walidował i przechowywał nowe pole 'username'.  
Uwaga: Aby zweryfikować unikalność nazwy użytkownika, użyj reguły walidacji: Rule::unique('users')  
  
3. Używając Artisan, stwórz nowy plik migracji, aby dodać pole username do tabeli użytkowników.  
Uwaga: Aby oznaczyć nowe pole jako unikalne, użyj funkcji unique().  
Zaimplementuj także funkcję down() migracji.  
  
4. Zmodyfikuj model użytkownika, aby uwzględniał nowe pole 'username'.  
  
5. Uruchom migrację i użyj formularza rejestracji, aby zarejestrować nowego użytkownika, w tym jego/jej nazwę użytkownika.  
  
6. Wklej poniżej:  
- HTML nowego pola w formularzu rejestracji.  
- Zmodyfikowaną metodę w RegisteredUserController.  
- Plik migracji.  
- Zrzut ekranu nowego rekordu w bazie danych, w tym pole 'username'.

### 1. Dodanie nowego pola wejściowego 'username' w widoku rejestracji

Najpierw otwórz plik formularza rejestracji, który znajduje się w resources/views/auth/register.blade.php, i dodaj pole username:

#### resources/views/auth/register.blade.php

blade

Skopiuj kod

@extends('common')

@section('content')

<div class="container">

<h2>{{ \_\_('registration.register') }}</h2>

<form method="POST" action="{{ route('register') }}">

@csrf

<!-- Nowe pole username -->

<div class="mb-3">

<label for="username" class="form-label">{{ \_\_('registration.username') }}</label>

<input type="text" id="username" name="username" class="form-control" value="{{ old('username') }}" required autofocus>

</div>

<div class="mb-3">

<label for="name" class="form-label">{{ \_\_('registration.name') }}</label>

<input type="text" id="name" name="name" class="form-control" value="{{ old('name') }}" required>

</div>

<div class="mb-3">

<label for="email" class="form-label">{{ \_\_('registration.email') }}</label>

<input type="email" id="email" name="email" class="form-control" value="{{ old('email') }}" required>

</div>

<div class="mb-3">

<label for="password" class="form-label">{{ \_\_('registration.password') }}</label>

<input type="password" id="password" name="password" class="form-control" required>

</div>

<div class="mb-3">

<label for="password\_confirmation" class="form-label">{{ \_\_('registration.confirm\_password') }}</label>

<input type="password" id="password\_confirmation" name="password\_confirmation" class="form-control" required>

</div>

<div class="mb-3">

<button type="submit" class="btn btn-primary">{{ \_\_('registration.register') }}</button>

</div>

<div class="mb-3">

<p>{{ \_\_('registration.already\_registered') }} <a href="{{ route('login') }}">{{ \_\_('registration.login') }}</a></p>

</div>

</form>

</div>

@endsection

### 2. Zmiana RegisteredUserController w celu walidacji i przechowywania 'username'

Teraz musisz zmodyfikować kontroler rejestracji, aby uwzględniał walidację dla nowego pola username. Otwórz plik kontrolera RegisteredUserController (zwykle znajduje się w app/Http/Controllers/Auth/RegisteredUserController.php), a następnie zmień metodę store() na poniższą wersję:

#### app/Http/Controllers/Auth/RegisteredUserController.php

php

Skopiuj kod

use Illuminate\Support\Facades\Hash;use Illuminate\Support\Facades\Validator;use Illuminate\Validation\Rule;use App\Models\User;

public function store(Request $request){

// Walidacja formularza

$request->validate([

'username' => ['required', 'string', 'max:255', Rule::unique('users')],

'name' => ['required', 'string', 'max:255'],

'email' => ['required', 'string', 'email', 'max:255', Rule::unique('users')],

'password' => ['required', 'string', 'min:8', 'confirmed'],

]);

// Tworzenie nowego użytkownika

User::create([

'username' => $request->username,

'name' => $request->name,

'email' => $request->email,

'password' => Hash::make($request->password),

]);

return redirect()->route('login');

}

Wprowadzone zmiany:

* Dodanie walidacji dla pola username z regułą unikalności (Rule::unique('users')).
* Przechowywanie wartości username w nowym użytkowniku.

### 3. Utworzenie migracji, aby dodać pole username do tabeli użytkowników

Teraz, utwórz migrację, aby dodać pole username do tabeli users. W terminalu uruchom poniższe polecenie Artisan:

bash

Skopiuj kod

php artisan make:migration add\_username\_to\_users\_table --table=users

To polecenie stworzy nową migrację. Otwórz nowo utworzony plik migracji (np. database/migrations/xxxx\_xx\_xx\_xxxxxx\_add\_username\_to\_users\_table.php) i zmodyfikuj go w następujący sposób:

#### database/migrations/xxxx\_xx\_xx\_xxxxxx\_add\_username\_to\_users\_table.php

php

Skopiuj kod

<?php

use Illuminate\Database\Migrations\Migration;use Illuminate\Database\Schema\Blueprint;use Illuminate\Support\Facades\Schema;

class AddUsernameToUsersTable extends Migration{

public function up()

{

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

$table->string('username')->unique()->after('id'); // Dodanie unikalnego pola 'username'

});

}

public function down()

{

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

$table->dropColumn('username'); // Usunięcie pola 'username' w przypadku rollbacku migracji

});

}

}

### 4. Zmodyfikowanie modelu użytkownika

Otwórz model User (zwykle znajduje się w app/Models/User.php) i dodaj username do atrybutów, które mogą być masowo przypisane:

#### app/Models/User.php

php

Skopiuj kod

protected $fillable = [

'username', 'name', 'email', 'password',

];

### 5. Uruchomienie migracji

Teraz uruchom migrację, aby zaktualizować tabelę users i dodać pole username:

bash

Skopiuj kod

php artisan migrate

### 6. Zarejestrowanie nowego użytkownika z nazwą użytkownika

Po zaktualizowaniu formularza, kontrolera, migracji i modelu, przejdź do formularza rejestracji w przeglądarce: http://127.0.0.1:8000/register.

Wypełnij formularz, dodając również nazwę użytkownika, a następnie kliknij "Zarejestruj się".

### Podsumowanie

Poniżej zamieszczam zmodyfikowane pliki:

#### HTML nowego pola w formularzu rejestracji

blade

Skopiuj kod

<div class="mb-3">

<label for="username" class="form-label">{{ \_\_('registration.username') }}</label>

<input type="text" id="username" name="username" class="form-control" value="{{ old('username') }}" required autofocus>

</div>

#### Zmodyfikowana metoda store w RegisteredUserController

php

Skopiuj kod

public function store(Request $request){

$request->validate([

'username' => ['required', 'string', 'max:255', Rule::unique('users')],

'name' => ['required', 'string', 'max:255'],

'email' => ['required', 'string', 'email', 'max:255', Rule::unique('users')],

'password' => ['required', 'string', 'min:8', 'confirmed'],

]);

User::create([

'username' => $request->username,

'name' => $request->name,

'email' => $request->email,

'password' => Hash::make($request->password),

]);

return redirect()->route('login');

}

#### Plik migracji

php

Skopiuj kod

class AddUsernameToUsersTable extends Migration{

public function up()

{

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

$table->string('username')->unique()->after('id');

});

}

public function down()

{

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

$table->dropColumn('username');

});

}

}

#### Zrzut ekranu z bazy danych

Aby zrobić zrzut ekranu z nowym rekordem w bazie danych:

1. Po rejestracji użytkownika, otwórz narzędzie do zarządzania bazą danych (np. phpMyAdmin, DBeaver, czy MySQL Workbench).
2. Wybierz tabelę users i sprawdź, czy pojawił się nowy rekord z wartością w kolumnie username.

Zrób zrzut ekranu, aby spełnić wymaganie.

Teraz Twoja aplikacja powinna pozwalać na rejestrację użytkowników z nazwą użytkownika!

Jeśli korzystasz z XAMPP, poniżej znajdziesz szczegółowe instrukcje, jak wykonać wszystkie kroki przy użyciu XAMPP jako lokalnego środowiska do pracy z Laravel:

### 1. ****Przygotowanie XAMPP do pracy z Laravel****

Najpierw upewnij się, że masz poprawnie zainstalowany XAMPP oraz Laravel:

1. **Zainstaluj XAMPP** i uruchom Apache oraz MySQL. Możesz to zrobić za pomocą Panelu Sterowania XAMPP.
2. **Ustawienie środowiska Laravel:**
   * Upewnij się, że masz zainstalowane PHP oraz Composer (przyda się do zarządzania zależnościami).
   * Jeśli nie masz zainstalowanego Composer, pobierz Composer.

### 2. ****Uruchomienie serwera Laravel na XAMPP****

Jeśli masz już zainstalowaną aplikację Laravel, uruchom serwer lokalny z poziomu terminala w folderze aplikacji Laravel:

bash

Skopiuj kod

php artisan serve

Aplikacja powinna działać pod adresem: http://127.0.0.1:8000.

### 3. ****Dodanie pola 'username' w formularzu rejestracyjnym****

Oto jak dostosować formularz rejestracji do dodania pola username oraz jak zmodyfikować kontroler i migrację.

#### a) ****Dodanie pola**** username ****w formularzu rejestracji****

Otwórz plik formularza rejestracji resources/views/auth/register.blade.php i dodaj nowe pole username:

blade

Skopiuj kod

<div class="mb-3">

<label for="username" class="form-label">{{ \_\_('registration.username') }}</label>

<input type="text" id="username" name="username" class="form-control" value="{{ old('username') }}" required autofocus>

</div>

#### b) ****Zmodyfikowanie**** RegisteredUserController

Otwórz kontroler RegisteredUserController znajdujący się w app/Http/Controllers/Auth/RegisteredUserController.php i zmodyfikuj metodę store():

php

Skopiuj kod

use Illuminate\Support\Facades\Hash;use Illuminate\Support\Facades\Validator;use Illuminate\Validation\Rule;use App\Models\User;

public function store(Request $request){

$request->validate([

'username' => ['required', 'string', 'max:255', Rule::unique('users')],

'name' => ['required', 'string', 'max:255'],

'email' => ['required', 'string', 'email', 'max:255', Rule::unique('users')],

'password' => ['required', 'string', 'min:8', 'confirmed'],

]);

User::create([

'username' => $request->username,

'name' => $request->name,

'email' => $request->email,

'password' => Hash::make($request->password),

]);

return redirect()->route('login');

}

#### c) ****Tworzenie migracji do dodania pola**** username

Uruchom polecenie Artisan w terminalu, aby stworzyć migrację:

bash

Skopiuj kod

php artisan make:migration add\_username\_to\_users\_table --table=users

Następnie otwórz plik migracji w katalogu database/migrations/xxxx\_xx\_xx\_xxxxxx\_add\_username\_to\_users\_table.php i zmodyfikuj go:

php

Skopiuj kod

public function up(){

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

$table->string('username')->unique()->after('id');

});

}

public function down(){

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

$table->dropColumn('username');

});

}

#### d) ****Zmodyfikowanie modelu**** User

W pliku app/Models/User.php dodaj pole username do atrybutów, które mogą być masowo przypisane:

php

Skopiuj kod

protected $fillable = [

'username', 'name', 'email', 'password',

];

#### e) ****Uruchomienie migracji****

Uruchom migrację, aby dodać pole username do tabeli users:

bash

Skopiuj kod

php artisan migrate

#### f) ****Rejestracja użytkownika****

Uruchom formularz rejestracji w przeglądarce pod adresem: http://127.0.0.1:8000/register. Zarejestruj nowego użytkownika, wypełniając formularz wraz z polem username.

### 4. ****Sprawdzanie bazy danych****

Po pomyślnym zarejestrowaniu użytkownika, sprawdź tabelę users w swojej bazie danych MySQL. Możesz użyć phpMyAdmin, który jest częścią XAMPP, aby sprawdzić zawartość tabeli.

1. Otwórz phpMyAdmin: http://localhost/phpmyadmin
2. Wybierz odpowiednią bazę danych (np. laravel).
3. Przejdź do tabeli users i sprawdź, czy pojawił się nowy rekord z polem username.

### 5. ****Podsumowanie****

Zmiany, które wprowadziłeś:

#### a) HTML nowego pola w formularzu rejestracji

blade

Skopiuj kod

<div class="mb-3">

<label for="username" class="form-label">{{ \_\_('registration.username') }}</label>

<input type="text" id="username" name="username" class="form-control" value="{{ old('username') }}" required autofocus>

</div>

#### b) Zmodyfikowana metoda store w RegisteredUserController

php

Skopiuj kod

public function store(Request $request){

$request->validate([

'username' => ['required', 'string', 'max:255', Rule::unique('users')],

'name' => ['required', 'string', 'max:255'],

'email' => ['required', 'string', 'email', 'max:255', Rule::unique('users')],

'password' => ['required', 'string', 'min:8', 'confirmed'],

]);

User::create([

'username' => $request->username,

'name' => $request->name,

'email' => $request->email,

'password' => Hash::make($request->password),

]);

return redirect()->route('login');

}

#### c) Plik migracji

php

Skopiuj kod

public function up(){

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

$table->string('username')->unique()->after('id');

});

}

public function down(){

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

$table->dropColumn('username');

});

}

#### d) Zrzut ekranu z bazy danych

Otwórz phpMyAdmin, sprawdź tabelę users i upewnij się, że pojawił się nowy rekord z polem username.

Po tych krokach powinieneś mieć działający formularz rejestracji w Laravelu z polem username.

Dodawanie ról użytkowników do Laravel z Breeze (3)  
  
1. Używając Artisan, stwórz middleware RoleMiddleware.  
  
2. Dodaj poniższą funkcję handle do middleware RoleMiddleware.  
  
3. Stwórz nowy widok Blade o nazwie "admininfo" i wyświetl tytuł H1 "Admin Information".  
  
4. Stwórz nową trasę "admininfo", która wyświetla widok "admin.info".  
  
5. Dodaj powyższy RoleMiddleware do trasy, aby widok "admin.info" był dostępny tylko dla użytkowników z rolą "admin".  
Wskazówka: Aby przekazać parametr do middleware, dodaj ciąg znaków ":<param>" do nazwy klasy middleware.

### 1. ****Utworzenie middleware**** RoleMiddleware

Aby utworzyć middleware, użyj poniższego polecenia Artisan:

bash

Skopiuj kod

php artisan make:middleware RoleMiddleware

To polecenie utworzy plik RoleMiddleware.php w katalogu app/Http/Middleware.

### 2. ****Dodanie funkcji**** handle ****do middleware****

Otwórz plik app/Http/Middleware/RoleMiddleware.php i zaktualizuj metodę handle, aby sprawdzała, czy użytkownik ma odpowiednią rolę (np. "admin"):

php

Skopiuj kod

namespace App\Http\Middleware;

use Closure;use Illuminate\Http\Request;use Illuminate\Support\Facades\Auth;

class RoleMiddleware{

public function handle(Request $request, Closure $next, $role)

{

// Sprawdzenie, czy użytkownik jest zalogowany i czy ma odpowiednią rolę

if (Auth::check() && Auth::user()->role === $role) {

return $next($request);

}

// Jeśli użytkownik nie ma odpowiedniej roli, przekierowanie do strony głównej

return redirect('/');

}

}

W powyższej metodzie:

* Sprawdzamy, czy użytkownik jest zalogowany (Auth::check()) i czy ma rolę, którą przekazujemy do middleware (np. admin).
* Jeśli użytkownik ma odpowiednią rolę, middleware pozwala przejść do kolejnego kroku w łańcuchu.
* Jeśli użytkownik nie ma odpowiedniej roli, jest przekierowywany na stronę główną (/).

### 3. ****Rejestracja middleware w**** Kernel.php

Aby middleware był dostępny w aplikacji, musisz go zarejestrować w pliku app/Http/Kernel.php. W sekcji $routeMiddleware dodaj nową linię:

php

Skopiuj kod

protected $routeMiddleware = [

// inne middleware...

'role' => \App\Http\Middleware\RoleMiddleware::class,

];

### 4. ****Tworzenie widoku Blade**** admininfo

Teraz utwórz plik Blade dla strony, która będzie dostępna tylko dla administratorów. W katalogu resources/views stwórz folder admin, a w nim plik admininfo.blade.php.

blade

Skopiuj kod

<!-- resources/views/admin/admininfo.blade.php -->

@extends('layouts.app')

@section('content')

<h1>Admin Information</h1>

<p>Welcome, Admin! This page is only accessible to users with the "admin" role.</p>

@endsection

### 5. ****Tworzenie trasy**** admininfo

Otwórz plik routes/web.php i dodaj nową trasę, która będzie wykorzystywać middleware role i będzie dostępna tylko dla użytkowników z rolą admin:

php

Skopiuj kod

use App\Http\Controllers\AdminController;

Route::middleware(['auth', 'role:admin'])->get('/admininfo', function () {

return view('admin.admininfo');

});

W powyższym kodzie:

* Middleware auth sprawdza, czy użytkownik jest zalogowany.
* Middleware role:admin sprawdza, czy użytkownik ma rolę "admin".
* Jeśli oba warunki są spełnione, zostanie wyświetlony widok admininfo.

### 6. ****Dodanie roli do użytkownika****

Aby użytkownicy mieli role, musisz dodać pole role do tabeli users w bazie danych. Możesz to zrobić za pomocą migracji:

1. Utwórz migrację, aby dodać pole role do tabeli users:

bash

Skopiuj kod

php artisan make:migration add\_role\_to\_users\_table --table=users

1. Zaktualizuj migrację, aby dodać pole role:

php

Skopiuj kod

public function up(){

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

$table->string('role')->default('user'); // domyślnie 'user', ale użytkownik może być 'admin'

});

}

public function down(){

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

$table->dropColumn('role');

});

}

1. Uruchom migrację:

bash

Skopiuj kod

php artisan migrate

1. Teraz, kiedy rejestrujesz użytkownika, możesz przypisać rolę "admin" ręcznie w bazie danych lub poprzez aktualizację użytkownika w kontrolerze.

### 7. ****Podsumowanie****

* **Middleware**: Sprawdzanie roli użytkownika przed umożliwieniem dostępu do strony.
* **Widok**: Tworzenie widoku admininfo, który jest dostępny tylko dla administratorów.
* **Trasa**: Utworzenie trasy z middleware role:admin, aby dostęp do strony miał tylko użytkownik z rolą "admin".

Po tych krokach, użytkownicy z rolą admin będą mieli dostęp do widoku admininfo, a pozostali użytkownicy będą przekierowywani na stronę główną lub inną stronę, jeśli nie mają odpowiedniej roli.

### Dodawanie ról użytkowników do Laravel z Breeze (1) 1. Używając Artisan, stwórz model i migrację dla Ról z następującymi polami: - id - name (powinno być unikalne) - timestamps 2. W modelu Ról, używając funkcji users(), zadeklaruj relację, że każda rola: belongsToMany(User::class); 3. Używając Artisan, stwórz seeder Ról, który tworzy 3 role: admin, teacher, student. 4. Zastosuj migrację i zapełnij tabelę Ról.

### 1. Stworzenie modelu i migracji dla ról

Uruchom w terminalu Artisan, aby stworzyć model i migrację:

php artisan make:model Role -m

Edytuj migrację, która została utworzona w folderze database/migrations. Dodaj kolumny zgodnie z wymaganiami:

public function up(){

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

$table->id();

$table->string('name')->unique();

$table->timestamps();

});

}

### 2. Deklaracja relacji w modelach

W modelu Role (app/Models/Role.php) zadeklaruj relację z modelem User:

php

Skopiuj kod

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;use Illuminate\Database\Eloquent\Model;

class Role extends Model{

use HasFactory;

protected $fillable = ['name'];

public function users()

{

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

}

}

W modelu User (app/Models/User.php) dodaj relację w przeciwną stronę:

php

Skopiuj kod

public function roles(){

return $this->belongsToMany(Role::class);

}

### 3. Seeder dla ról

Stwórz seeder:

php artisan make:seeder RoleSeeder

Edytuj plik database/seeders/RoleSeeder.php, aby utworzyć trzy role:

php

Skopiuj kod

namespace Database\Seeders;

use Illuminate\Database\Seeder;use App\Models\Role;

class RoleSeeder extends Seeder{

public function run()

{

Role::create(['name' => 'admin']);

Role::create(['name' => 'teacher']);

Role::create(['name' => 'student']);

}

}

Dodaj seeder do głównego seedera (DatabaseSeeder.php):

php

Skopiuj kod

public function run(){

$this->call(RoleSeeder::class);

}

### 4. Migracja i wypełnienie tabeli

Uruchom migrację:

bash

Skopiuj kod

php artisan migrate

Zapełnij tabelę ról za pomocą seedera:

bash

Skopiuj kod

php artisan db:seed --class=RoleSeeder

Po wykonaniu tych kroków, tabela roles w Twojej bazie danych będzie zawierać trzy role: admin, teacher i student. Możesz teraz wykorzystać tę strukturę do przypisywania ról użytkownikom w aplikacji.

Dodawanie ról użytkowników do Laravel z Breeze (2)  
  
1. Używając Artisan, stwórz migrację dla tabeli RoleUser z następującym schematem:  
  
Schema::create('role\_user', function (Blueprint $table) {  
$table->id();  
$table->foreignId('user\_id')->constrained()->onDelete('cascade');  
$table->foreignId('role\_id')->constrained()->onDelete('cascade');  
$table->timestamps();  
});  
  
2. W modelu User, używając funkcji roles(), zadeklaruj relację, że każdy użytkownik:  
  
$this->belongsToMany(Role:class)->withTimestamps();  
  
3. W modelu User dodaj funkcję, która sprawdza, czy użytkownik ma daną rolę $role:  
  
public function hasRole($role)  
{  
return $this->roles()->where('name', $role)->exists();  
}  
  
4. Zastosuj migrację.  
  
5. Użyj poniższego RoleController, aby stworzyć 2 tymczasowe trasy w web.php, które wywołują te dwie metody:  
- Przypisz użytkownikowi o id 1 rolę Admin  
- Sprawdź, czy aktualnie zalogowany użytkownik jest administratorem.  
-- Uzupełnij metodę:  
-- -- pobierz aktualnego użytkownika  
-- -- użyj metody hasRole(..) w modelu User  
  
6. Używając przeglądarki, przypisz użytkownikowi o id 1 rolę Admin, a następnie sprawdź, czy aktualnie zalogowany użytkownik jest administratorem czy nie.

1.Utworz migracje

php artisan make:migration create\_role\_user\_table --create=role\_user

1. Edytuj plik migracji:Znajdź plik w katalogu database/migrations (np. 2024\_12\_12\_000001\_create\_role\_user\_table.php) i edytuj jego zawartość, aby wyglądała następująco:

<?php

use Illuminate\Database\Migrations\Migration;

use Illuminate\Database\Schema\Blueprint;

use Illuminate\Support\Facades\Schema;

class CreateRoleUserTable extends Migration

{

/\*\*

\* Run the migrations.

\*

\* @return void

\*/

public function up()

{

Schema::create('role\_user', function (Blueprint $table) {

$table->id();

$table->foreignId('user\_id')->constrained()->onDelete('cascade');

$table->foreignId('role\_id')->constrained()->onDelete('cascade');

$table->timestamps();

});

}

/\*\*

\* Reverse the migrations.

\*

\* @return void

\*/

public function down()

{

Schema::dropIfExists('role\_user');

}

}

## Krok 2: Zdefiniowanie relacji w modelu User

**Edytuj plik** app/Models/User.php**:**

Dodaj relację roles:

public function roles(){

return $this->belongsToMany(Role::class)->withTimestamps();

}

## Krok 3: Funkcja sprawdzająca rolę użytkownika

Dodaj metodę hasRole w modelu User:

ublic function hasRole($role) { return $this->roles()->where('name', $role)->exists(); }

**Krok 4: Zastosowanie migracji**

php artisan migrate

Krok 5: RoleController i trasy w web.php

Stwórz kontroler RoleController:

php artisan make:controller RoleController

**Edytuj plik app/Http/Controllers/RoleController.php:**

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use App\Models\User;

use App\Models\Role;

class RoleController extends Controller

{

// Przypisz rolę użytkownikowi

public function assignAdminRole()

{

$user = User::find(1);

$adminRole = Role::where('name', 'admin')->first();

if ($user && $adminRole) {

$user->roles()->syncWithoutDetaching([$adminRole->id]);

return "Rola admin przypisana użytkownikowi o ID 1.";

}

return "Nie udało się przypisać roli.";

}

// Sprawdź, czy zalogowany użytkownik ma rolę admin

public function isAdmin()

{

$user = auth()->user();

if ($user && $user->hasRole('admin')) {

return "Aktualny użytkownik jest administratorem.";

}

return "Aktualny użytkownik NIE jest administratorem.";

}

}

**Dodaj tymczasowe trasy w routes/web.php:**

use App\Http\Controllers\RoleController;

Route::get('/assign-admin-role', [RoleController::class, 'assignAdminRole'])->name('assign.admin.role');

Route::get('/is-admin', [RoleController::class, 'isAdmin'])->name('is.admin');

**Krok 6: Testowanie w przeglądarce**

Przypisz rolę admin użytkownikowi o ID 1:

Otwórz przeglądarkę i przejdź pod adres:

http://twoja-aplikacja.test/assign-admin-role

Powinieneś zobaczyć komunikat: "Rola admin przypisana użytkownikowi o ID 1."

Sprawdź, czy aktualnie zalogowany użytkownik jest administratorem:

Zaloguj się do aplikacji jako dowolny użytkownik. Następnie przejdź pod adres:

http://twoja-aplikacja.test/is-admin

Wynik będzie zależał od tego, czy zalogowany użytkownik ma przypisaną rolę admin.

Korzystając z nowej instalacji Laravel lub innej:  
  
1. Używając Artisan, stwórz nowy widok Blade o nazwie "secret.page", który wyświetla nagłówek H1 z napisem "Secret Page" i losowy tekst lorem.  
  
2. Stwórz nową trasę, która wyświetla widok "secret.page" gdy użytkownik żąda strony "/secret".  
Nie używaj kontrolera, lecz zwróć widok "secret.page" bezpośrednio z "web.php".  
  
3. Używając Artisan, stwórz nowy kontroler o nazwie "PasswordController".  
  
4. Używając Artisan, stwórz nowy widok Blade o nazwie "password.index". Ten widok powinien:  
- Wyświetlać formularz, który prosi użytkownika o hasło.  
- Formularz powinien POST-ować dane do URL "password.check".  
- Pamiętaj o dodaniu dyrektywy @csrf w formularzu.  
  
5. Stwórz nową nazwane trasę "password.index", która używa PasswordController do wyświetlenia widoku "password.index" gdy użytkownik żąda: "/password".  
  
6. Stwórz nową nazwane trasę "password.check", która wywołuje metodę "check" w PasswordController.  
Ta metoda "check" powinna:  
- Odczytać hasło z żądania użytkownika.  
- Sprawdzić, czy hasło to "abc123" i  
- Ponownie wyświetlić widok "password.index" z dodatkowymi komunikatami "Correct Password" lub "Incorrect password".

Krok 1: Stworzenie widoku Blade "secret.page"

Utwórz nowy plik widoku:

W terminalu użyj Artisan, aby stworzyć widok:

php artisan make:view secret.page

Jeśli polecenie make:view nie jest dostępne, utwórz plik ręcznie:

touch resources/views/secret/page.blade.php

**Edytuj plik resources/views/secret/page.blade.php:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Secret Page</title>

</head>

<body>

<h1>Secret Page</h1>

<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vivamus lacinia odio vitae vestibulum vestibulum.</p>

</body>

</html>

**Krok 2: Stworzenie trasy do widoku "secret.page"**

Dodaj trasę w routes/web.php:

use Illuminate\Support\Facades\Route;

Route::get('/secret', function () {

return view('secret.page');

});

**Krok 3: Stworzenie kontrolera PasswordController**

Użyj Artisan, aby stworzyć kontroler:

php artisan make:controller PasswordController

Kontroler zostanie utworzony w katalogu app/Http/Controllers/.

**Krok 4: Stworzenie widoku Blade "password.index"**

Utwórz nowy plik widoku:

W terminalu użyj Artisan:

php artisan make:view password.index

Lub stwórz plik ręcznie:

touch resources/views/password/index.blade.php

Edytuj plik resources/views/password/index.blade.php:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Password Check</title>

</head>

<body>

<h1>Enter Password</h1>

@if(session('message'))

<p>{{ session('message') }}</p>

@endif

<form action="{{ route('password.check') }}" method="POST">

@csrf

<label for="password">Password:</label>

<input type="password" id="password" name="password" required>

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

</form>

</body>

</html>

**Krok 5: Stworzenie trasy "password.index"**

Edytuj routes/web.php i dodaj trasę:

use App\Http\Controllers\PasswordController;

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

**Edytuj metodę index w PasswordController:**

Otwórz app/Http/Controllers/PasswordController.php i dodaj:

public function index()

{

return view('password.index');

}

**Krok 6: Stworzenie trasy "password.check"**

Dodaj trasę password.check w routes/web.php:

Route::post('/password/check', [PasswordController::class, 'check'])->name('password.check');

**Edytuj metodę check w PasswordController:**

use Illuminate\Http\Request;

public function check(Request $request)

{

$password = $request->input('password');

$message = $password === 'abc123' ? 'Correct Password' : 'Incorrect Password';

return redirect()->route('password.index')->with('message', $message);

}

Testowanie w przeglądarce

Sprawdzenie strony "Secret Page":

Otwórz przeglądarkę i przejdź pod adres:

http://your-app.test/secret

Powinieneś zobaczyć stronę z nagłówkiem "Secret Page" i tekstem lorem.

Formularz hasła:

Otwórz adres:

http://your-app.test/password

Wprowadź hasło do formularza:

Wpisz abc123 → Zobaczysz komunikat: "Correct Password".

Wpisz inne hasło → Zobaczysz komunikat: "Incorrect Password".

1. Zmodyfikuj metodę check(...) w PasswordController i:  
   - Ustaw zmienną sesyjną o nazwie "okpassword" z wartością true lub false w zależności od podanego hasła.  
     
   2. Zmodyfikuj widok "secret.page" i używając warunku @if wyświetl treść "secret.page" tylko wtedy, gdy zmienna sesyjna "okpassword" została ustawiona i zawiera wartość true.  
   - Jeśli zmienna sesyjna "okpassword" nie jest ustawiona lub jest fałszywa, wyświetl wiadomość "No access".

**Zmodyfikowana metoda check(...) w PasswordController**

public function check(Request $request)

{

$password = $request->input('password');

// Ustawienie zmiennej sesyjnej "okpassword"

if ($password === 'abc123') {

session(['okpassword' => true]);

$message = 'Correct Password';

} else {

session(['okpassword' => false]);

$message = 'Incorrect Password';

}

return redirect()->route('password.index')->with('message', $message);

}

**Zmodyfikowany widok secret.page**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Secret Page</title>

</head>

<body>

@if(session('okpassword') === true)

<h1>Secret Page</h1>

<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vivamus lacinia odio vitae vestibulum vestibulum.</p>

@else

<h1>No access</h1>

<p>You do not have permission to view this page.</p>

@endif

</body>

</html>

**Wyjaśnienia**

Zmiana w metodzie check(...):

Zmienna sesyjna okpassword jest ustawiana na true, jeśli hasło to abc123, w przeciwnym wypadku na false.

Komunikat "Correct Password" lub "Incorrect Password" jest nadal przekazywany do widoku formularza.

Zmiana w widoku secret.page:

Warunek @if(session('okpassword') === true) sprawdza, czy użytkownik ma dostęp.

Jeśli zmienna sesyjna okpassword nie jest ustawiona lub jest fałszywa, wyświetla się komunikat "No access".

Testowanie:

Po wpisaniu poprawnego hasła abc123 w formularzu i przejściu do /secret widok "Secret Page" zostanie wyświetlony.

W przeciwnym wypadku użytkownik zobaczy komunikat "No access".

**Krok 1: Usunięcie warunku z widoku secret.page**

Edytuj plik resources/views/secret/page.blade.php:

Usuń warunek @if, aby widok wyglądał tak:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Secret Page</title>

</head>

<body>

<h1>Secret Page</h1>

<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vivamus lacinia odio vitae vestibulum vestibulum.</p>

</body>

</html>

**Krok 2: Stworzenie middleware CheckPassword**

Użyj Artisan, aby stworzyć middleware:

php artisan make:middleware CheckPassword

Middleware zostanie utworzony w katalogu app/Http/Middleware.

**Krok 3: Zmodyfikowanie metody handle(...) w middleware CheckPassword**

Otwórz plik app/Http/Middleware/CheckPassword.php i edytuj metodę handle:

<?php

namespace App\Http\Middleware;

use Closure;

use Illuminate\Http\Request;

class CheckPassword

{

/\*\*

\* Handle an incoming request.

\*

\* @param \Illuminate\Http\Request $request

\* @param \Closure $next

\* @return mixed

\*/

public function handle(Request $request, Closure $next)

{

// Sprawdzenie sesji "okpassword"

if (session('okpassword') === true) {

return $next($request); // Kontynuuj żądanie

}

// Jeśli sesja "okpassword" nie istnieje lub jest fałszywa, przekieruj

return redirect()->route('password.index');

}

}

**Krok 4: Dodanie middleware do trasy secret.page**

Edytuj plik routes/web.php:

Dodaj middleware do trasy /secret:

use Illuminate\Support\Facades\Route;

use App\Http\Middleware\CheckPassword;

Route::get('/secret', function () {

return view('secret.page');

})->middleware(CheckPassword::class);

Działanie

Gdy użytkownik otwiera /secret:

Middleware sprawdza zmienną sesyjną okpassword.

Jeśli okpassword jest ustawione i ma wartość true, użytkownik widzi stronę "Secret Page".

Jeśli okpassword jest nieustawione lub ma wartość false, middleware przekierowuje użytkownika na /password (widok "password.index"), aby ponownie wprowadził hasło.

Testowanie:

Spróbuj otworzyć /secret bez wcześniejszego ustawienia hasła → zostaniesz przekierowany na /password.

Wprowadź poprawne hasło (abc123), a następnie przejdź na /secret → strona tajna będzie widoczna.