|  |  |
| --- | --- |
| **AI1** | Dokumentacja projektu |
| **Autor** | Piotr Nowak, 125147 |
| **Kierunek, rok** | Informatyka, II rok, st. Stacjonarne (3,5-I) |
| **Temat projektu** | *Budżet studencki* |

Wstęp

Projekt "Budżet Studencki" ma na celu umożliwienie studentom zarządzanie ich finansami w prosty i przejrzysty sposób. Aplikacja pozwala na rejestrowanie dochodów i wydatków, tworzenie kategorii dla transakcji, zarządzanie oszczędnościami oraz generowanie raportów z aktualnym bilansem. Głównymi użytkownikami są studenci, którzy mogą wprowadzać swoje dane finansowe oraz administratorzy, którzy zarządzają całością aplikacji, w tym użytkownikami i kategoriami transakcji.

Narzędzia i technologie

**Wykorzystane technologie:**

* **PHP 8.3**: Skryptowy język programowania, wykorzystywany na serwerze.

<https://www.php.net/docs.php>

* **Laravel 11.x**: Framework PHP, który wspiera wzorzec MVC i zapewnia szereg narzędzi i bibliotek do szybkiego tworzenia aplikacji webowych.

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

* **MySQL**: Relacyjna baza danych używana do przechowywania danych aplikacji.

<https://dev.mysql.com/doc/>

* **Tailwind CSS**: Narzędzie do szybkiego tworzenia stylów CSS z predefiniowanymi klasami.

<https://tailwindcss.com/docs/installation>

* **Composer**: Menedżer pakietów dla PHP, używany do zarządzania zależnościami.

<https://getcomposer.org/doc/>

Wszystkie powyższe technologie są darmowe i dostępne na licencji open-source.

Baza danych

## Obraz zawierający tekst, zrzut ekranu, Oprogramowanie graficzne, Oprogramowanie multimedialne Opis wygenerowany automatycznieSchemat ERD:

## Opis zawartości bazy danych:

* **Tabela users**: Przechowuje informacje o użytkownikach, w tym ich dane logowania i role.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

Powiązania między tabelami:

1. **users** ↔ **categories**: Każdy użytkownik może mieć wiele kategorii. Powiązanie realizowane jest za pomocą klucza obcego user\_id w tabeli **categories**, który wskazuje na id użytkownika w tabeli **users**.
2. **users** ↔ **transactions**: Każdy użytkownik może mieć wiele transakcji. Powiązanie realizowane jest za pomocą klucza obcego user\_id w tabeli **transactions**, który wskazuje na id użytkownika w tabeli **users**.
3. **users** ↔ **savings**: Każdy użytkownik może mieć wiele oszczędności. Powiązanie realizowane jest za pomocą klucza obcego user\_id w tabeli **savings**, który wskazuje na id użytkownika w tabeli **users**.

* **categories**: Przechowuje kategorie transakcji tworzone przez użytkowników.

Obraz zawierający tekst, zrzut ekranu, Oprogramowanie graficzne, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

Powiązania między tabelami:

1. **categories** ↔ **transactions**: Każda kategoria może być powiązana z wieloma transakcjami. Powiązanie realizowane jest za pomocą klucza obcego category\_id w tabeli **transactions**, który wskazuje na id kategorii w tabeli **categories**.

 **transactions**: Przechowuje informacje o dochodach i wydatkach użytkowników.

Obraz zawierający tekst, zrzut ekranu, Oprogramowanie multimedialne, oprogramowanie

Opis wygenerowany automatycznie

* **savings**: Przechowuje informacje o oszczędnościach użytkowników.

Obraz zawierający zrzut ekranu, tekst, Oprogramowanie graficzne, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

## Tabele i ich pola:

1. **users**

* **id**: (integer, primary key, auto-increment) Unikalny identyfikator użytkownika.
* **name**: (string) Imię i nazwisko użytkownika.
* **email**: (string, unique) Adres email użytkownika.
* **password**: (string) Hasło użytkownika w formie zaszyfrowanej (hash).
* **isAdmin**: (boolean) Flaga oznaczająca, czy użytkownik jest administratorem.
* **profile\_photo**: (string, nullable) Ścieżka do zdjęcia profilowego użytkownika.
* **created\_at**: (timestamp) Data i czas utworzenia rekordu.
* **updated\_at**: (timestamp) Data i czas ostatniej aktualizacji rekordu.

1. **categories**

* **id**: (integer, primary key, auto-increment) Unikalny identyfikator kategorii.
* **name**: (string) Nazwa kategorii.
* **user\_id**: (integer, foreign key) Identyfikator użytkownika, który stworzył kategorię.
* **created\_at**: (timestamp) Data i czas utworzenia rekordu.
* **updated\_at**: (timestamp) Data i czas ostatniej aktualizacji rekordu.

1. **transactions**

* **id**: (integer, primary key, auto-increment) Unikalny identyfikator transakcji.
* **user\_id**: (integer, foreign key) Identyfikator użytkownika, który stworzył transakcję.
* **category\_id**: (integer, foreign key) Identyfikator kategorii, do której należy transakcja.
* **amount**: (decimal) Kwota transakcji.
* **type**: (enum, values: 'Dochody', 'Wydatki') Typ transakcji.
* **description**: (text, nullable) Opis transakcji.
* **date**: (date) Data transakcji.
* **created\_at**: (timestamp) Data i czas utworzenia rekordu.
* **updated\_at**: (timestamp) Data i czas ostatniej aktualizacji rekordu.

1. **savings**

* **id**: (integer, primary key, auto-increment) Unikalny identyfikator oszczędności.
* **user\_id**: (integer, foreign key) Identyfikator użytkownika, który stworzył oszczędność.
* **goal**: (string) Cel oszczędności.
* **amount**: (decimal) Kwota oszczędności.
* **date**: (date) Data, do której użytkownik chce osiągnąć cel oszczędności.
* **created\_at**: (timestamp) Data i czas utworzenia rekordu.
* **updated\_at**: (timestamp) Data i czas ostatniej aktualizacji rekordu.

# GUI:

1. Widok 1: Strona powitalna (widok desktopowy i mobilny)

Obraz zawierający tekst, zrzut ekranu, logo, Czcionka

Opis wygenerowany automatycznie

Obraz zawierający tekst, logo, Czcionka, zrzut ekranu

Opis wygenerowany automatycznie

1. Widok 2: Logowanie (widok desktopowy i mobilny)

Obraz zawierający tekst, zrzut ekranu, design, logo

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, Czcionka, design

Opis wygenerowany automatycznie

1. Widok 3: Rejestracja (widok desktopowy i mobilny)

Obraz zawierający tekst, zrzut ekranu, design

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

1. Widok 4: Dashboard - Demo (widok desktopowy i mobilny)

Obraz zawierający tekst, zrzut ekranu, numer, oprogramowanie

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

1. Widok 5: Dashboard – Admin i User (widok desktopowy i mobilny)

Admin:

Obraz zawierający tekst, zrzut ekranu, numer, oprogramowanie

Opis wygenerowany automatycznie

User:

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Strona internetowa

Opis wygenerowany automatycznie

Admin: User:

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznieObraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

1. Widok 2: Lista Transakcji – User (widok desktopowy i mobilny)

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Ikona komputerowa

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, Czcionka, oprogramowanie

Opis wygenerowany automatycznie

# Uruchomienie aplikacji:

## Wymagania:

* PHP 8.3.7
* Composer v2.7.6
* MySQL 8.4.0
* Node.js 22.0.0 (dla Tailwind CSS)
* Visual Studio Code (najnowsza wersja)
* Przeglądarka
* XAMPP 8.2.12

## Kroki konfiguracyjne:

1. Włączyć aplikacje XAMPP i wystartować moduły Apache oraz MySQL:  
   Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Ikona komputerowa

   Opis wygenerowany automatycznie
2. W zależności od systemu operacyjnego:

Dla systemów MS Windows:

* Kliknąć w plik start.bat (znajduje się w plikach).

Dla systemów Linux i Mac OS:

* Po utworzeniu bazy, migracji oraz dodaniu seederów, użytkownikowi włączy się Visual Studio Code.

1. W lewym górnym rogu kliknąć zakładkę Terminal >> New Terminal (lub Ctrl + Shift + `).
2. Wywołać komendę php artisan serve.
3. Otworzyć kolejny terminal i wywołać komendę npm run dev.
4. Otworzyć przeglądarkę i wpisać adres: <http://127.0.0.1:8000> lub <http://localhost:8000/>.

## Funkcjonalności aplikacji: Logowanie:

Przykładowe konta:  
Admin:

Login: [admin@example.com](mailto:admin@example.com)

Hasło: password

User:

User1  
 Login: [user1@example.com](mailto:user1@example.com)

Hasło: password

User2

Login: [user2@example.com](mailto:user2@example.com)

Hasło: password

User3

Login: [user3@example.com](mailto:user3@example.com)

Hasło: password

Obraz zawierający tekst, zrzut ekranu, design

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, design

Opis wygenerowany automatycznie

## CRUD (admin):

Dodawanie transakcji przez administratora:

Obraz zawierający tekst, oprogramowanie, Ikona komputerowa, Strona internetowa

Opis wygenerowany automatycznieObraz zawierający tekst, zrzut ekranu, oprogramowanie, Ikona komputerowa

Opis wygenerowany automatycznie

## Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Ikona komputerowa Opis wygenerowany automatycznie

Użytkownik admin od razu zobaczy swoje transakcje w widoku Dashboard.

Przed dodaniem transakcji:  
Obraz zawierający tekst, zrzut ekranu, oprogramowanie, numer

Opis wygenerowany automatycznie

Po dodaniu transakcji:

Obraz zawierający tekst, zrzut ekranu, numer, oprogramowanie

Opis wygenerowany automatycznie

**Dodawanie Transakcji w Aplikacji Budżet Studencki**

Aplikacja Budżet Studencki pozwala użytkownikom na dodawanie nowych transakcji do ich budżetu. Proces ten obejmuje kilka kroków zarówno na froncie (w widoku), jak i na backendzie (w kontrolerze i modelu).

**Frontend**

**Widok dodawania transakcji (transactions/create.blade.php)**

Widok dodawania transakcji udostępnia użytkownikowi formularz, w którym można wprowadzić szczegóły nowej transakcji.

<?php

@**extends**('layouts.app')

@section('content')

<div **class**="flex justify-center items-center min-h-screen bg-gray-100">

<div **class**="bg-white p-8 shadow-md rounded-lg w-full max-w-md">

<h1 **class**="text-2xl font-bold mb-6 text-center">Dodaj Transakcję</h1>

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

@csrf

<div **class**="mb-4">

<label **for**="type" **class**="block text-gray-700 font-bold mb-2">Typ</label>

<select **name**="type" id="type" **class**="block appearance-none w-full bg-gray-200 border border-gray-200 text-gray-700 py-3 px-4 pr-8 rounded leading-tight focus:outline-none focus:bg-white focus:border-gray-500">

<option **value**="Dochody" {{ old('type') == 'Dochody' ? 'selected' : '' }}>Dochody</option>

<option **value**="Wydatki" {{ old('type') == 'Wydatki' ? 'selected' : '' }}>Wydatki</option>

</select>

@error('type')

<p **class**="text-red-500 text-xs mt-2">{{ **$message** }}</p>

@enderror

</div>

<div **class**="mb-4">

<label **for**="category\_id" **class**="block text-gray-700 font-bold mb-2">Kategoria</label>

<select **name**="category\_id" id="category\_id" **class**="block appearance-none w-full bg-gray-200 border border-gray-200 text-gray-700 py-3 px-4 pr-8 rounded leading-tight focus:outline-none focus:bg-white focus:border-gray-500">

@**foreach**(**$categories** **as** **$category**)

<option **value**="{{ **$category**->id }}" {{ old('category\_id') == **$category**->**id** ? 'selected' : '' }}>{{ **$category**->**name** }}</option>

@**endforeach**

</select>

@error('category\_id')

<p **class**="text-red-500 text-xs mt-2">{{ **$message** }}</p>

@enderror

</div>

<div **class**="mb-4">

<label **for**="amount" **class**="block text-gray-700 font-bold mb-2">Kwota</label>

<input **type**="number" **name**="amount" id="amount" step="0.01" **min**="0.01" **max**="99999999.99" **value**="{{ old('amount') }}" **class**="block appearance-none w-full bg-gray-200 border border-gray-200 text-gray-700 py-3 px-4 rounded leading-tight focus:outline-none focus:bg-white focus:border-gray-500">

@error('amount')

<p **class**="text-red-500 text-xs mt-2">{{ **$message** }}</p>

@enderror

</div>

<div **class**="mb-4">

<label **for**="date" **class**="block text-gray-700 font-bold mb-2">**Data**</label>

<input **type**="date" **name**="date" id="date" **value**="{{ old('date') }}" **class**="block appearance-none w-full bg-gray-200 border border-gray-200 text-gray-700 py-3 px-4 rounded leading-tight focus:outline-none focus:bg-white focus:border-gray-500">

@error('date')

<p **class**="text-red-500 text-xs mt-2">{{ **$message** }}</p>

@enderror

</div>

<div **class**="mb-4">

<label **for**="description" **class**="block text-gray-700 font-bold mb-2">Opis</label>

<textarea **name**="description" id="description" rows="4" **class**="block appearance-none w-full bg-gray-200 border border-gray-200 text-gray-700 py-3 px-4 rounded leading-tight focus:outline-none focus:bg-white focus:border-gray-500">{{ old('description') }}</textarea>

@error('description')

<p **class**="text-red-500 text-xs mt-2">{{ **$message** }}</p>

@enderror

</div>

<div **class**="flex items-center justify-center">

<button **type**="submit" **class**="bg-blue-500 hover:bg-blue-700 text-white font-bold py-2 px-4 rounded focus:outline-none focus:shadow-outline">Dodaj</button>

</div>

</form>

</div>

</div>

@endsection

?>

**Backend**

**Kontroler AdminController**

Kontroler zarządza logiką dodawania nowej transakcji. Poniżej znajduje się fragment kodu odpowiedzialny za dodawanie transakcji.

<?php

**public** **function** storeTransaction(Request **$request**)

{

**$request**->**validate**([

'type' => 'required|in:Dochody,Wydatki',

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

'amount' => 'required|numeric|min:0.01|max:99999999.99',

'date' => 'required|date',

'description' => 'nullable|string',

]);

**$data** = **$request**->all();

**$data**['user\_id'] = auth()->**user**()->**id**;

Transaction::create(**$data**);

**return** redirect()->route('transactions.index')->with('success', 'Transakcja została dodana.');

}

?>

**Model Transaction**

Model Transaction definiuje właściwości transakcji oraz relacje z innymi modelami.

<?php

**namespace** App\Models;

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

**use** Illuminate\Database\Eloquent\Model;

**use** Carbon\Carbon;

**class** Transaction **extends** Model

{

**use** HasFactory;

**protected** **$fillable** = [

'user\_id', 'category\_id', 'amount', 'type', 'description', 'date',

];

**protected** **$dates** = [

'date',

];

**public** **function** category()

{

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

}

**public** **function** getDateAttribute(**$value**)

{

**return** Carbon::parse(**$value**);

}

}

**Proces Dodawania Transakcji**

1. **Wyświetlenie Formularza**: Użytkownik klika przycisk dodawania nowej transakcji, co powoduje wyświetlenie formularza dodawania transakcji. Formularz zawiera pola takie jak typ transakcji (Dochody/Wydatki), kategoria, kwota, data i opis.
2. **Wprowadzenie Danych**: Użytkownik wprowadza szczegóły nowej transakcji do formularza.
3. **Przesłanie Formularza**: Po wprowadzeniu danych użytkownik przesyła formularz, który wysyła żądanie POST do serwera z danymi nowej transakcji.
4. **Walidacja Danych**: Kontroler AdminController odbiera dane i sprawdza, czy wszystkie pola są poprawnie wypełnione zgodnie z regułami walidacji.
5. **Zapisanie Transakcji w Bazie Danych**: Po pomyślnej walidacji, dane transakcji są zapisywane w bazie danych przy użyciu modelu Transaction.
6. **Przekierowanie i Komunikat**: Po zapisaniu transakcji, użytkownik jest przekierowany z powrotem do listy transakcji z komunikatem o sukcesie.

Użytkownik admin może zaktualizować transakcje (lub kategorie, oszczędności):

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Ikona komputerowa

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Ikona komputerowa

Opis wygenerowany automatycznie

Zmieniamy przykładowo dane: kwota 50,00 -> 150,00 i Data 01.05.2023 -> 03.06.2024.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Ikona komputerowa

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Ikona komputerowa

Opis wygenerowany automatycznie

**Aktualizowanie Transakcji w Aplikacji Budżet Studencki**

Aplikacja Budżet Studencki pozwala użytkownikom na aktualizowanie szczegółów istniejących transakcji. Proces aktualizacji transakcji obejmuje kilka kroków zarówno na froncie (w widoku), jak i na backendzie (w kontrolerze i modelu).

**Frontend**

**Widok edycji transakcji (transactions/edit.blade.php)**

Widok edycji transakcji umożliwia użytkownikowi wprowadzenie zmian w istniejącej transakcji. Formularz jest wypełniany danymi bieżącej transakcji, które użytkownik może modyfikować.

<?php

@**extends**('layouts.app')

@section('content')

<div **class**="container mx-auto mt-8 px-4">

<h1 **class**="text-2xl font-bold mb-6 text-center">Edytuj Transakcję</h1>

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

@csrf

@method('PUT')

<div **class**="mb-4">

<label **for**="type" **class**="block text-gray-700 font-bold mb-2">Typ</label>

<select **name**="type" id="type" **class**="block appearance-none w-full bg-gray-200 border border-gray-200 text-gray-700 py-3 px-4 pr-8 rounded leading-tight focus:outline-none focus:bg-white focus:border-gray-500">

<option **value**="Dochody" {{ **$transaction**->**type** == 'Dochody' ? 'selected' : '' }}>Dochody</option>

<option **value**="Wydatki" {{ **$transaction**->**type** == 'Wydatki' ? 'selected' : '' }}>Wydatki</option>

</select>

@error('type')

<p **class**="text-red-500 text-xs mt-2">{{ **$message** }}</p>

@enderror

</div>

<div **class**="mb-4">

<label **for**="category\_id" **class**="block text-gray-700 font-bold mb-2">Kategoria</label>

<select **name**="category\_id" id="category\_id" **class**="block appearance-none w-full bg-gray-200 border border-gray-200 text-gray-700 py-3 px-4 pr-8 rounded leading-tight focus:outline-none focus:bg-white focus:border-gray-500">

@**foreach**(**$categories** **as** **$category**)

<option **value**="{{ **$category**->id }}" {{ **$transaction**->**category\_id** == **$category**->**id** ? 'selected' : '' }}>{{ **$category**->**name** }}</option>

@**endforeach**

</select>

@error('category\_id')

<p **class**="text-red-500 text-xs mt-2">{{ **$message** }}</p>

@enderror

</div>

<div **class**="mb-4">

<label **for**="amount" **class**="block text-gray-700 font-bold mb-2">Kwota</label>

<input **type**="number" **name**="amount" id="amount" step="0.01" **min**="0.01" **value**="{{ **$transaction**->amount }}" **class**="block appearance-none w-full bg-gray-200 border border-gray-200 text-gray-700 py-3 px-4 rounded leading-tight focus:outline-none focus:bg-white focus:border-gray-500">

@error('amount')

<p **class**="text-red-500 text-xs mt-2">{{ **$message** }}</p>

@enderror

</div>

<div **class**="mb-4">

<label **for**="date" **class**="block text-gray-700 font-bold mb-2">**Data**</label>

<input **type**="date" **name**="date" id="date" **value**="{{ **$transaction**->date->format('Y-m-d') }}" **class**="block appearance-none w-full bg-gray-200 border border-gray-200 text-gray-700 py-3 px-4 rounded leading-tight focus:outline-none focus:bg-white focus:border-gray-500">

@error('date')

<p **class**="text-red-500 text-xs mt-2">{{ **$message** }}</p>

@enderror

</div>

<div **class**="mb-4">

<label **for**="description" **class**="block text-gray-700 font-bold mb-2">Opis</label>

<textarea **name**="description" id="description" rows="4" **class**="block appearance-none w-full bg-gray-200 border border-gray-200 text-gray-700 py-3 px-4 rounded leading-tight focus:outline-none focus:bg-white focus:border-gray-500">{{ **$transaction**->**description** }}</textarea>

@error('description')

<p **class**="text-red-500 text-xs mt-2">{{ **$message** }}</p>

@enderror

</div>

<div **class**="flex items-center justify-center">

<button **type**="submit" **class**="bg-blue-500 hover:bg-blue-700 text-white font-bold py-2 px-4 rounded focus:outline-none focus:shadow-outline">Zaktualizuj</button>

</div>

</form>

</div>

@endsection

?>

**Backend**

**Kontroler AdminController**

Kontroler zarządza logiką aktualizacji transakcji. Poniżej znajduje się fragment kodu odpowiedzialny za aktualizację transakcji.

<?php

**public** **function** updateTransaction(Request **$request**, **$id**)

{

**$categories** = Category::all();

**if** (**$categories**->**isEmpty**()) {

Category::create(['name' => 'Jedzenie', 'user\_id' => Auth::id(), 'created\_at' => now(), 'updated\_at' => now()]);

Category::create(['name' => 'Transport', 'user\_id' => Auth::id(), 'created\_at' => now(), 'updated\_at' => now()]);

Category::create(['name' => 'Rozrywka', 'user\_id' => Auth::id(), 'created\_at' => now(), 'updated\_at' => now()]);

}

**$request**->**validate**([

'type' => 'required|in:Dochody,Wydatki',

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

'amount' => 'required|numeric|min:0.01|max:99999999.99',

'date' => 'required|date',

'description' => 'nullable|string',

]);

**$transaction** = Transaction::findOrFail(**$id**);

**$transaction**->**update**(**$request**->all());

**return** redirect()->route('transactions.index')->with('success', 'Transakcja została zaktualizowana.');

}

?>

**Model Transaction**

Model Transaction definiuje właściwości transakcji oraz relacje z innymi modelami.

<?php

**namespace** App\Models;

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

**use** Illuminate\Database\Eloquent\Model;

**use** Carbon\Carbon;

**class** Transaction **extends** Model

{

**use** HasFactory;

**protected** **$fillable** = [

'user\_id', 'category\_id', 'amount', 'type', 'description', 'date',

];

**protected** **$dates** = [

'date',

];

**public** **function** category()

{

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

}

**public** **function** getDateAttribute(**$value**)

{

**return** Carbon::parse(**$value**);

}

}

**Proces Aktualizacji Transakcji**

1. **Pobranie Transakcji do Edycji**: Użytkownik wybiera transakcję do edycji, co powoduje wyświetlenie formularza edycji z bieżącymi danymi transakcji.
2. **Wyświetlenie Formularza**: Formularz edycji jest wypełniany aktualnymi danymi transakcji, które użytkownik może zmieniać.
3. **Przesłanie Formularza**: Użytkownik wprowadza zmiany i przesyła formularz. Formularz wysyła żądanie PUT do serwera z danymi zmienionej transakcji.
4. **Walidacja Danych**: Kontroler AdminController sprawdza, czy wszystkie pola są poprawnie wypełnione zgodnie z regułami walidacji.
5. **Aktualizacja Transakcji w Bazie Danych**: Po pomyślnej walidacji, kontroler znajduje odpowiednią transakcję w bazie danych za pomocą metody findOrFail i aktualizuje jej dane za pomocą metody update.
6. **Przekierowanie i Komunikat**: Po zaktualizowaniu transakcji, użytkownik jest przekierowany z powrotem do listy transakcji z komunikatem o sukcesie.

Usuwanie wybranej kategorii *test* przez użytkownika admin.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Strona internetowa

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Strona internetowa

Opis wygenerowany automatycznie

**Usuwanie kategorii przez użytkownika**

W aplikacji Budżet Studencki użytkownik ma możliwość usuwania kategorii, które utworzył. Usunięcie kategorii jest procesem, który obejmuje kilka kroków zarówno na froncie (w widoku), jak i na backendzie (w kontrolerze i modelu).

**Frontend**

**Widok kategorii (categories/index.blade.php)**

W widoku kategorii znajduje się lista wszystkich kategorii utworzonych przez użytkownika, wraz z przyciskiem do usuwania każdej z nich:

<?php

@**extends**('layouts.app')

@section('content')

<div **class**="container mx-auto mt-8 px-4">

<h1 **class**="text-2xl font-bold mb-6 text-center">Twoje Kategorie</h1>

<div **class**="bg-white shadow rounded-lg overflow-hidden">

<div **class**="overflow-x-auto">

<table **class**="min-w-full leading-normal">

<thead>

<tr>

<th **class**="px-5 py-3 border-b-2 border-gray-200 bg-gray-100 text-left text-xs font-semibold text-gray-600 uppercase tracking-wider">

Nazwa

</th>

<th **class**="px-5 py-3 border-b-2 border-gray-200 bg-gray-100 text-left text-xs font-semibold text-gray-600 uppercase tracking-wider">

Akcje

</th>

</tr>

</thead>

<tbody>

@**foreach**(**$categories** **as** **$category**)

<tr>

<td **class**="px-5 py-5 border-b border-gray-200 bg-white text-sm">{{ **$category**->**name** }}</td>

<td **class**="px-5 py-5 border-b border-gray-200 bg-white text-sm">

<form action="{{ route('categories.destroy', **$category**->id) }}" method="POST" onsubmit="return confirm('Czy na pewno chcesz usunąć tę kategorię?');">

@csrf

@method('DELETE')

<button **type**="submit" **class**="text-red-500 hover:text-red-700">Usuń</button>

</form>

</td>

</tr>

@**endforeach**

</tbody>

</table>

</div>

</div>

</div>

@endsection

?>

**Backend**

**Kontroler AdminController**

Kontroler zarządza logiką usuwania kategorii. Poniżej znajduje się fragment kodu odpowiedzialny za usuwanie kategorii.

<?php

**public** **function** destroyTransaction(**$id**)

{

**$transaction** = Transaction::findOrFail(**$id**);

**$transaction**->**delete**();

**return** redirect()->route('transactions.index')->with('success', 'Transakcja została usunięta.');

}

?>

**Model Category**

Model Category definiuje relację z modelem Transaction, co pozwala na sprawdzenie, czy dana kategoria ma powiązane transakcje.

<?php

**namespace** App\Models;

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

**use** Illuminate\Database\Eloquent\Model;

**class** Category **extends** Model

{

**use** HasFactory;

**protected** **$fillable** = ['name', 'user\_id'];

**public** **function** transactions()

{

**return** **$this**->hasMany(Transaction::**class**);

}

}

?>

**Proces Usuwania Kategorii**

1. **Inicjacja Usuwania**: Użytkownik klika przycisk "Usuń" obok wybranej kategorii. Formularz wysyła żądanie DELETE do serwera z ID kategorii do usunięcia.
2. **Obsługa Żądania w Kontrolerze**: Kontroler AdminController obsługuje żądanie i próbuje znaleźć kategorię w bazie danych za pomocą metody findOrFail.
3. **Sprawdzenie Powiązań**: Kontroler sprawdza, czy kategoria ma powiązane transakcje, używając metody transactions()->exists(). Jeśli tak, przekierowuje użytkownika z komunikatem o błędzie.
4. **Usunięcie Kategorii**: Jeśli kategoria nie ma powiązanych transakcji, zostaje usunięta z bazy danych za pomocą metody delete.
5. **Przekierowanie i Komunikat**: Po usunięciu kategorii, użytkownik zostaje przekierowany z powrotem do listy kategorii z komunikatem o sukcesie.

## Zarządzania użytkownikami przez administratora:

Dodanie nowego użytkownika *test* przez administratora:

Obraz zawierający tekst, zrzut ekranu, numer, Czcionka

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Ikona komputerowa

Opis wygenerowany automatycznie

**Jak admin dodaje użytkownika do bazy:**

W aplikacji "Budżet Studencki" administrator ma możliwość dodawania nowych użytkowników do systemu. Poniżej przedstawiono krok po kroku, jak to działa, wraz z odpowiednimi fragmentami kodu.

**1. Formularz dodawania użytkownika**

Najpierw musimy stworzyć formularz, który pozwoli administratorowi wprowadzić dane nowego użytkownika. Formularz ten znajduje się w widoku *admin/users/create.blade.php*:

<?php

@**extends**('layouts.app')

@section('content')

<div **class**="container mx-auto mt-8">

<h1 **class**="text-3xl font-bold mb-6 text-center">Dodaj Użytkownika</h1>

<div **class**="flex justify-center">

<div **class**="bg-white shadow-md rounded-lg overflow-hidden w-full max-w-2xl">

<div **class**="p-6">

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

@csrf

<div **class**="mb-4">

<label **for**="name" **class**="block text-gray-700 font-bold mb-2">Nazwa</label>

<input **type**="text" **name**="name" id="name" **class**="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight focus:outline-none focus:shadow-outline" required>

</div>

<div **class**="mb-4">

<label **for**="email" **class**="block text-gray-700 font-bold mb-2">Email</label>

<input **type**="email" **name**="email" id="email" **class**="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight focus:outline-none focus:shadow-outline" required>

</div>

<div **class**="mb-4">

<label **for**="password" **class**="block text-gray-700 font-bold mb-2">Hasło</label>

<input **type**="password" **name**="password" id="password" **class**="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight focus:outline-none focus:shadow-outline" required>

</div>

<div **class**="mb-4">

<label **for**="password\_confirmation" **class**="block text-gray-700 font-bold mb-2">Potwierdź Hasło</label>

<input **type**="password" **name**="password\_confirmation" id="password\_confirmation" **class**="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight focus:outline-none focus:shadow-outline" required>

</div>

<div **class**="mb-4">

<label **for**="isAdmin" **class**="block text-gray-700 font-bold mb-2">Czy Admin</label>

<select **name**="isAdmin" id="isAdmin" **class**="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight focus:outline-none focus:shadow-outline" required>

<option **value**="0">Nie</option>

<option **value**="1">Tak</option>

</select>

</div>

<div **class**="flex justify-end">

<button **type**="submit" **class**="bg-blue-500 text-white px-4 py-2 rounded shadow hover:bg-blue-600">Dodaj Użytkownika</button>

</div>

</form>

</div>

</div>

</div>

</div>

@endsection

?>

**2. Obsługa żądania w kontrolerze**

Następnie, musimy obsłużyć żądanie wysyłane przez formularz w kontrolerze. Kontroler AdminController ma metodę storeUser, która zajmuje się walidacją i tworzeniem nowego użytkownika:

<?php

**public** **function** storeUser(Request **$request**)

{

**if** (!auth()->**user**()->**isAdmin**) {

abort(403, 'Unauthorized action.');

}

**$request**->**validate**([

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

'email' => 'required|string|email|max:255|unique:users',

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

'isAdmin' => 'required|boolean',

]);

**User**::create([

'name' => **$request**->**name**,

'email' => **$request**->**email**,

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

'isAdmin' => **$request**->**isAdmin**,

]);

**return** redirect()->route('admin.users.index')->with('success', 'Użytkownik został dodany.');

}

?>

**3. Walidacja danych**

Dane wejściowe są walidowane, aby upewnić się, że wszystkie wymagane pola są wypełnione i mają odpowiednie formaty. Walidacja obejmuje:

* name: Wymagane, ciąg znaków, maksymalnie 255 znaków.
* email: Wymagane, poprawny adres email, maksymalnie 255 znaków, unikalny w tabeli użytkowników.
* password: Wymagane, ciąg znaków, co najmniej 8 znaków, musi być potwierdzone (password\_confirmation).
* isAdmin: Wymagane, wartość logiczna (0 lub 1).

**4. Tworzenie użytkownika**

Po pomyślnej walidacji, nowy użytkownik jest tworzony w bazie danych za pomocą modelu User:

<?php

**namespace** App\Models;

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

**use** Illuminate\Foundation\Auth\**User** **as** Authenticatable;

**use** Illuminate\Notifications\Notifiable;

**class** **User** **extends** Authenticatable

{

**use** HasFactory, Notifiable;

**protected** **$fillable** = [

'name', 'email', 'password', 'isAdmin', 'profile\_photo'

];

**protected** **$hidden** = [

'password', 'remember\_token',

];

**public** **function** transactions()

{

**return** **$this**->hasMany(Transaction::**class**);

}

**public** **function** savings()

{

**return** **$this**->hasMany(Saving::**class**);

}

**public** **function** categories()

{

**return** **$this**->hasMany(Category::**class**);

}

}

Zmiana danych na koncie test przez administratora:

Obraz zawierający tekst, zrzut ekranu, numer, Czcionka

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Ikona komputerowa

Opis wygenerowany automatycznie

**Jak admin edytuje użytkownika w bazie danych:**

Administrator ma również możliwość edytowania danych użytkowników w systemie. Poniżej przedstawiono krok po kroku, jak to działa, wraz z odpowiednimi fragmentami kodu.

**1. Widok edycji użytkownika**

Widok edycji użytkownika znajduje się w pliku admin/users/edit.blade.php. Formularz edycji jest podobny do formularza dodawania użytkownika, z tą różnicą, że zawiera już wypełnione dane użytkownika, który ma być edytowany.

<?php

@**extends**('layouts.app')

@section('content')

<div **class**="container mx-auto mt-8">

<h1 **class**="text-3xl font-bold mb-6 text-center">Edytuj Użytkownika</h1>

<div **class**="flex justify-center">

<div **class**="bg-white shadow-md rounded-lg overflow-hidden w-full max-w-2xl">

<div **class**="p-6">

<form action="{{ route('admin.users.update', **$user**->id) }}" method="POST">

@csrf

@method('PUT')

<div **class**="mb-4">

<label **for**="name" **class**="block text-gray-700 font-bold mb-2">Nazwa</label>

<input **type**="text" **name**="name" id="name" **value**="{{ **$user**->name }}" **class**="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight focus:outline-none focus:shadow-outline" required>

</div>

<div **class**="mb-4">

<label **for**="email" **class**="block text-gray-700 font-bold mb-2">Email</label>

<input **type**="email" **name**="email" id="email" **value**="{{ **$user**->email }}" **class**="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight focus:outline-none focus:shadow-outline" required>

</div>

<div **class**="mb-4">

<label **for**="password" **class**="block text-gray-700 font-bold mb-2">Nowe Hasło</label>

<input **type**="password" **name**="password" id="password" **class**="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight focus:outline-none focus:shadow-outline">

</div>

<div **class**="mb-4">

<label **for**="password\_confirmation" **class**="block text-gray-700 font-bold mb-2">Potwierdź Hasło</label>

<input **type**="password" **name**="password\_confirmation" id="password\_confirmation" **class**="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight focus:outline-none focus:shadow-outline">

</div>

<div **class**="mb-4">

<label **for**="isAdmin" **class**="block text-gray-700 font-bold mb-2">Czy Admin</label>

<select **name**="isAdmin" id="isAdmin" **class**="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight focus:outline-none focus:shadow-outline" required>

<option **value**="0" {{ **$user**->**isAdmin** ? '' : 'selected' }}>Nie</option>

<option **value**="1" {{ **$user**->**isAdmin** ? 'selected' : '' }}>Tak</option>

</select>

</div>

<div **class**="flex justify-end">

<button **type**="submit" **class**="bg-blue-500 text-white px-4 py-2 rounded shadow hover:bg-blue-600">Zapisz Zmiany</button>

</div>

</form>

</div>

</div>

</div>

</div>

@endsection

?>

**2. Kontroler obsługujący edycję użytkownika**

W AdminController znajduje się metoda editUser, która pobiera dane użytkownika na podstawie jego identyfikatora oraz wyświetla formularz edycji. Metoda updateUser zajmuje się aktualizacją danych użytkownika po wysłaniu formularza:

<?php

**public** **function** editUser(**$id**)

{

**if** (!auth()->**user**()->**isAdmin**) {

abort(403, 'Unauthorized action.');

}

**$user** = **User**::findOrFail(**$id**);

**return** view('admin.users.edit', **compact**('user'));

}

**public** **function** updateUser(Request **$request**, **$id**)

{

**if** (!auth()->**user**()->**isAdmin**) {

abort(403, 'Unauthorized action.');

}

**$user** = **User**::findOrFail(**$id**);

**$request**->**validate**([

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

'email' => 'required|string|email|max:255|unique:users,email,' . **$user**->**id**,

'isAdmin' => 'required|boolean',

'password' => 'nullable|string|min:8|confirmed',

]);

**$user**->**name** = **$request**->**name**;

**$user**->**email** = **$request**->**email**;

**$user**->**isAdmin** = **$request**->**isAdmin**;

**if** (**$request**->**password**) {

**$user**->**password** = **Hash**::make(**$request**->**password**);

}

**$user**->**save**();

**return** redirect()->route('admin.users.index')->with('success', 'Użytkownik został zaktualizowany.');

}

?>

Usuwanie przez administratora użytkownika User4 z systemu:  
Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Ikona komputerowa

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Ikona komputerowa

Opis wygenerowany automatycznie

**Usuwanie użytkownika przez administratora**

Proces usuwania użytkownika przez administratora w aplikacji Laravel jest obsługiwany przez kontroler, model oraz odpowiedni widok. Poniżej przedstawiam szczegółowe kroki, jak to działa, wraz z odpowiednim kodem.

**1. Widok użytkowników**

W widoku użytkowników znajduje się przycisk lub link, który pozwala administratorowi usunąć danego użytkownika. Widok może wyglądać na przykład tak:

<?php

@**extends**('layouts.app')

@section('content')

<div **class**="container mx-auto mt-8">

<h1 **class**="text-3xl font-bold mb-6 text-center">Zarządzanie użytkownikami</h1>

<div **class**="flex justify-center">

<div **class**="bg-white shadow-md rounded-lg overflow-hidden w-full max-w-2xl">

<table **class**="min-w-full leading-normal">

<thead>

<tr>

<th **class**="px-5 py-3 border-b-2 border-gray-200 bg-gray-100 text-left text-xs font-semibold text-gray-600 uppercase tracking-wider">

Nazwa

</th>

<th **class**="px-5 py-3 border-b-2 border-gray-200 bg-gray-100 text-left text-xs font-semibold text-gray-600 uppercase tracking-wider">

Email

</th>

<th **class**="px-5 py-3 border-b-2 border-gray-200 bg-gray-100 text-left text-xs font-semibold text-gray-600 uppercase tracking-wider">

Akcje

</th>

</tr>

</thead>

<tbody>

@**foreach**(**$users** **as** **$user**)

<tr>

<td **class**="px-5 py-5 border-b border-gray-200 bg-white text-sm">{{ **$user**->**name** }}</td>

<td **class**="px-5 py-5 border-b border-gray-200 bg-white text-sm">{{ **$user**->**email** }}</td>

<td **class**="px-5 py-5 border-b border-gray-200 bg-white text-sm">

<a href="{{ route('admin.users.edit', **$user**->id) }}" **class**="text-indigo-600 hover:text-indigo-900">Edytuj</a>

<form action="{{ route('admin.users.destroy', **$user**->id) }}" method="POST" style="display: inline;">

@csrf

@method('DELETE')

<button **type**="submit" **class**="text-red-600 hover:text-red-900 ml-4">Usuń</button>

</form>

</td>

</tr>

@**endforeach**

</tbody>

</table>

</div>

</div>

</div>

@endsection

?>

**2. Kontroler obsługujący usuwanie użytkownika**

W AdminController znajduje się metoda destroyUser, która obsługuje usuwanie użytkownika na podstawie jego identyfikatora:

<?php

**public** **function** destroyUser(**$id**)

{

**if** (!auth()->**user**()->**isAdmin**) {

abort(403, 'Unauthorized action.');

}

**$user** = **User**::findOrFail(**$id**);

**$user**->**delete**();

**return** redirect()->route('admin.users.index')->with('success', 'Użytkownik został usunięty.');

}

?>

**4. Metoda delete w modelu User**

Model User dziedziczy metodę delete z klasy Eloquent w Laravel, co oznacza, że nie musimy dodawać dodatkowego kodu w modelu, aby móc usuwać użytkowników. Wystarczy wywołać metodę delete na instancji modelu.

## Przeglądania ogólnodostępnych zasobów:

Przeglądanie ogólnodostępnych zasobów odbywa się w widoku Demo. Można do niego przejść przez widok strony powitalnej.

System demo w aplikacji Budżet Studencki pozwala użytkownikom na przeglądanie przykładowych danych, takich jak transakcje, kategorie i oszczędności, bez konieczności logowania się. Demo zapewnia interaktywne doświadczenie, które symuluje pełną funkcjonalność aplikacji, ale wszelkie wprowadzone zmiany są zapisywane tylko lokalnie w przeglądarce użytkownika i nie wpływają na rzeczywiste dane w bazie.

Obraz zawierający tekst, zrzut ekranu, logo, projekt graficzny

Opis wygenerowany automatycznie

**Główne komponenty systemu demo:**

1. **Widoki demo**:
   * demo.blade.php: Główny panel demo wyświetlający podstawowe informacje o transakcjach, dochodach, wydatkach i saldzie.
   * demo\_transactions.blade.php: Widok przeglądania transakcji demo.
   * demo\_categories.blade.php: Widok przeglądania kategorii demo.
   * demo\_savings.blade.php: Widok przeglądania oszczędności demo.
2. **Kontroler DemoController**:

* demoDashboard(): Wyświetla główny panel demo.
* demoTransactions(): Wyświetla widok transakcji demo.
* demoCategories(): Wyświetla widok kategorii demo.
* demoSavings(): Wyświetla widok oszczędności demo.
* initializeDemoData(): Inicjalizuje przykładowe dane dla systemu demo.

1. **JavaScript**:

* Obsługuje lokalne przechowywanie danych w LocalStorage, aby umożliwić interaktywne zarządzanie danymi demo.

**Szczegółowy opis działania:**

**demo.blade.php**: Główny panel demo, który wyświetla całkowite dochody, wydatki i saldo oraz ostatnie transakcje.

Obraz zawierający tekst, zrzut ekranu, numer, oprogramowanie

Opis wygenerowany automatycznie

**demo\_transactions.blade.php**: Widok przeglądania transakcji demo, który pozwala na dodawanie, edytowanie i usuwanie transakcji.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Ikona komputerowa

Opis wygenerowany automatycznie

**demo\_categories.blade.php**: Widok przeglądania kategorii demo, który pozwala na dodawanie, edytowanie i usuwanie kategorii. Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Ikona komputerowa

Opis wygenerowany automatycznie

**demo\_savings.blade.php**: Widok przeglądania oszczędności demo, który pozwala na dodawanie, edytowanie i usuwanie oszczędności.

Obraz zawierający zrzut ekranu, tekst, oprogramowanie, System operacyjny

Opis wygenerowany automatycznie

**Przechowywanie Danych w Demo**

1. **Inicjalizacja Danych**: Kiedy użytkownik po raz pierwszy otwiera stronę demo, aplikacja pobiera wstępnie zdefiniowane dane z serwera i zapisuje je w Local Storage. Dane te obejmują kategorie, oszczędności i transakcje. Kod ten znajduje się w metodzie initializeDemoData w DemoController.

<?php

**public** **function** initializeDemoData()

{

**$categories** = [

['name' => 'Jedzenie'],

['name' => 'Transport'],

['name' => 'Rozrywka'],

];

**$savings** = [

['goal' => 'Nowy laptop', 'amount' => 1500, 'date' => '2024-06-30'],

['goal' => 'Wakacje', 'amount' => 2000, 'date' => '2024-07-15'],

];

**$transactions** = [

['type' => 'Dochody', 'category' => 'Jedzenie', 'amount' => 500, 'date' => '2024-06-01', 'description' => 'Wynagrodzenie'],

['type' => 'Wydatki', 'category' => 'Transport', 'amount' => 150, 'date' => '2024-06-03', 'description' => 'Bilet miesięczny'],

['type' => 'Dochody', 'category' => 'Rozrywka', 'amount' => 200, 'date' => '2024-06-05', 'description' => 'Zwrot podatku'],

['type' => 'Wydatki', 'category' => 'Jedzenie', 'amount' => 100, 'date' => '2024-06-07', 'description' => 'Zakupy spożywcze'],

];

**return** response()->json([

'categories' => **$categories**,

'savings' => **$savings**,

'transactions' => **$transactions**,

]);

}

?>

1. **Zapisywanie Danych w Local Storage**: Kiedy użytkownik otwiera stronę demo, dane są pobierane i zapisywane w Local Storage za pomocą JavaScript.

<?php

document.addEventListener('DOMContentLoaded', **function**() {

**fetch**("{{ route('demo.initialize') }}")

.then(response => response.json())

.then(**data** => {

localStorage.setItem('demo\_transactions', JSON.stringify(**data**.transactions));

localStorage.setItem('demo\_savings', JSON.stringify(**data**.savings));

localStorage.setItem('demo\_categories', JSON.stringify(**data**.categories));

loadRecentTransactions();

loadCategories();

loadSavings();

calculateTotals();

});

});

?>

**Dodawanie Danych**

1. **Formularz Dodawania**: Użytkownik może dodać nowe transakcje, kategorie lub oszczędności za pomocą odpowiednich formularzy w interfejsie użytkownika.
2. **Przechwycenie Danych**: Kiedy użytkownik wypełnia formularz i go wysyła, dane są przechwytywane przez JavaScript i zapisywane w Local Storage.

<?php

**function** saveTransaction(event) {

event.preventDefault();

**const** transactions = JSON.parse(localStorage.getItem('demo\_transactions')) || [];

**const** **type** = document.**getElementById**('type').**value**;

**const** category = document.**getElementById**('category').**value**;

**const** amount = parseFloat(document.**getElementById**('amount').**value**);

**const** **date** = document.**getElementById**('date').**value**;

**const** **description** = document.**getElementById**('description').**value**;

**const** transaction = { **type**, category, amount, **date**, **description** };

transactions.**push**(transaction);

localStorage.setItem('demo\_transactions', JSON.stringify(transactions));

loadTransactions();

hideTransactionModal();

}

?>

**Aktualizowanie Danych**

1. **Formularz Edycji**: Użytkownik może edytować istniejące dane za pomocą formularza edycji, który jest wypełniany aktualnymi danymi do edycji.
2. **Zapisanie Zmian**: Po edycji danych i przesłaniu formularza, zmiany są zapisywane w Local Storage.

<?php

**function** editTransaction(index) {

**const** transactions = JSON.parse(localStorage.getItem('demo\_transactions')) || [];

**const** transaction = transactions[index];

document.**getElementById**('transaction-id').**value** = index;

document.**getElementById**('type').**value** = transaction.**type**;

document.**getElementById**('category').**value** = transaction.category;

document.**getElementById**('amount').**value** = transaction.amount;

document.**getElementById**('date').**value** = transaction.**date**;

document.**getElementById**('description').**value** = transaction.**description**;

document.**getElementById**('modal-title').textContent = 'Edytuj Transakcję';

document.**getElementById**('transaction-modal').classList.**remove**('hidden');

}

**function** saveTransaction(event) {

event.preventDefault();

**const** transactions = JSON.parse(localStorage.getItem('demo\_transactions')) || [];

**const** id = document.**getElementById**('transaction-id').**value**;

**const** **type** = document.**getElementById**('type').**value**;

**const** category = document.**getElementById**('category').**value**;

**const** amount = parseFloat(document.**getElementById**('amount').**value**);

**const** **date** = document.**getElementById**('date').**value**;

**const** **description** = document.**getElementById**('description').**value**;

**const** transaction = { **type**, category, amount, **date**, **description** };

**if** (id) {

transactions[id] = transaction;

} **else** {

transactions.**push**(transaction);

}

localStorage.setItem('demo\_transactions', JSON.stringify(transactions));

loadTransactions();

hideTransactionModal();

}

?>

**Usuwanie Danych**

1. **Usuwanie Elementu**: Użytkownik może usunąć dane klikając na odpowiedni przycisk w interfejsie użytkownika. Element jest usuwany z Local Storage.

<?php

**function** deleteTransaction(index) {

**const** transactions = JSON.parse(localStorage.getItem('demo\_transactions')) || [];

transactions.splice(index, 1);

localStorage.setItem('demo\_transactions', JSON.stringify(transactions));

loadTransactions();

}

?>

## Proces rejestracji i logowania:

**Rejestracja Nowego Użytkownika**

**Frontend**

**Widok Rejestracji (register.blade.php)**

Formularz rejestracji umożliwia użytkownikom wprowadzenie swoich danych, takich jak pełne imię i nazwisko, adres email i hasło.

Obraz zawierający tekst, zrzut ekranu, design

Opis wygenerowany automatycznie

<?php

<!**DOCTYPE** html>

<html lang="{{ str\_replace('\_', '-', app()->getLocale()) }}">

<head>

<meta charset="utf-8">

<meta **name**="viewport" **content**="width=device-width, initial-scale=1">

<**title**>Zarejestruj się - BTStudent</**title**>

<**link** href="https://cdn.jsdelivr.net/npm/tailwindcss@2.2.19/dist/tailwind.min.css" rel="stylesheet">

</head>

<body **class**="bg-gray-100 flex items-center justify-center h-screen">

<div **class**="w-full max-w-md px-4">

<form method="POST" action="{{ route('register') }}" **class**="bg-white shadow-md rounded px-8 pt-6 pb-8 mb-4">

@csrf

<div **class**="mb-4">

<h2 **class**="text-center text-2xl font-bold mb-4">Załóż konto</h2>

<label **class**="block text-gray-700 text-sm font-bold mb-2" **for**="name">

Pełne imię i nazwisko

</label>

<input **class**="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight focus:outline-none focus:shadow-outline" id="name" **type**="text" **name**="name" **value**="{{ old('name') }}" required autocomplete="name" autofocus>

</div>

<div **class**="mb-4">

<label **class**="block text-gray-700 text-sm font-bold mb-2" **for**="email">

Adres email

</label>

<input **class**="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight focus:outline-none focus:shadow-outline" id="email" **type**="email" **name**="email" **value**="{{ old('email') }}" required autocomplete="email">

</div>

<div **class**="mb-4">

<label **class**="block text-gray-700 text-sm font-bold mb-2" **for**="password">

Hasło

</label>

<input **class**="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight focus:outline-none focus:shadow-outline" id="password" **type**="password" **name**="password" required autocomplete="new-password">

</div>

<div **class**="mb-4">

<label **class**="block text-gray-700 text-sm font-bold mb-2" **for**="password\_confirmation">

Potwierdź hasło

</label>

<input **class**="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight focus:outline-none focus:shadow-outline" id="password\_confirmation" **type**="password" **name**="password\_confirmation" required autocomplete="new-password">

</div>

<div **class**="flex items-center justify-center">

<button **class**="bg-blue-500 hover:bg-blue-700 text-white font-bold py-2 px-4 rounded focus:outline-none focus:shadow-outline" **type**="submit">

Zarejestruj się

</button>

</div>

@**if** (Route::has('login'))

<div **class**="text-center mt-4">

<a **class**="inline-block align-baseline font-bold text-sm text-blue-500 hover:text-blue-800" href="{{ route('login') }}">

Masz już konto? Zaloguj się

</a>

</div>

@**endif**

</form>

</div>

</body>

</html>

?>

**Backend**

**Kontroler Rejestracji (RegisterController.php)**

Kontroler RegisterController zarządza procesem rejestracji. Obejmuje on wyświetlanie formularza rejestracji oraz obsługę przesłanych danych.

<?php

**namespace** App\Http\Controllers\Auth;

**use** App\Http\Controllers\Controller;

**use** App\Models\**User**;

**use** Illuminate\Support\Facades\**Hash**;

**use** Illuminate\Support\Facades\Validator;

**use** Illuminate\Foundation\Auth\RegistersUsers;

**class** RegisterController **extends** Controller

{

**use** RegistersUsers;

**protected** **$redirectTo** = '/home';

**public** **function** **\_\_construct**()

{

**$this**->middleware('guest');

}

**protected** **function** validator(**array** **$data**)

{

**return** Validator::make(**$data**, [

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

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

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

]);

}

**protected** **function** create(**array** **$data**)

{

**return** **User**::create([

'name' => **$data**['name'],

'email' => **$data**['email'],

'password' => **Hash**::make(**$data**['password']),

'profile\_photo' => 'default.png', // domyślne zdjęcie profilowe

]);

}

}

**Model User**

Model User definiuje właściwości użytkownika oraz relacje z innymi modelami.

<?php

**namespace** App\Models;

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

**use** Illuminate\Foundation\Auth\**User** **as** Authenticatable;

**use** Illuminate\Notifications\Notifiable;

**class** **User** **extends** Authenticatable

{

**use** HasFactory, Notifiable;

**protected** **$fillable** = [

'name', 'email', 'password', 'isAdmin', 'profile\_photo'

];

**protected** **$hidden** = [

'password', 'remember\_token',

];

**public** **function** transactions()

{

**return** **$this**->hasMany(Transaction::**class**);

}

**public** **function** savings()

{

**return** **$this**->hasMany(Saving::**class**);

}

**public** **function** categories()

{

**return** **$this**->hasMany(Category::**class**);

}

}

**Proces Rejestracji Użytkownika**

1. **Wyświetlenie Formularza**: Użytkownik otwiera stronę rejestracji, gdzie wyświetlany jest formularz rejestracyjny.
2. **Wprowadzenie Danych**: Użytkownik wprowadza swoje dane, takie jak imię, adres email, hasło i potwierdzenie hasła.
3. **Przesłanie Formularza**: Użytkownik przesyła formularz, wysyłając dane do serwera.
4. **Walidacja Danych**: Kontroler RegisterController waliduje wprowadzone dane zgodnie z zdefiniowanymi regułami.
5. **Tworzenie Nowego Użytkownika**: Po pomyślnej walidacji, kontroler tworzy nowego użytkownika w bazie danych z domyślnym zdjęciem profilowym.
6. **Przekierowanie**: Nowo zarejestrowany użytkownik jest automatycznie logowany i przekierowany na stronę główną.

**Logowanie Użytkownika**

**Widok Logowania (login.blade.php)**

Formularz logowania umożliwia użytkownikom wprowadzenie adresu email i hasła.

Obraz zawierający tekst, zrzut ekranu, design, logo

Opis wygenerowany automatycznie

<?php

<!**DOCTYPE** html>

<html lang="{{ str\_replace('\_', '-', app()->getLocale()) }}">

<head>

<meta charset="utf-8">

<meta **name**="viewport" **content**="width=device-width, initial-scale=1">

<**title**>Zaloguj się - BTStudent</**title**>

<**link** href="https://cdn.jsdelivr.net/npm/tailwindcss@2.2.19/dist/tailwind.min.css" rel="stylesheet">

</head>

<body **class**="bg-gray-100 flex items-center justify-center h-screen">

<div **class**="w-full max-w-md px-4">

<form method="POST" action="{{ route('login') }}" **class**="bg-white shadow-md rounded px-8 pt-6 pb-8 mb-4">

@csrf

<div **class**="mb-4">

<h2 **class**="text-center text-2xl font-bold mb-4">Zaloguj się</h2>

<label **class**="block text-gray-700 text-sm font-bold mb-2" **for**="email">

Adres email

</label>

<input **class**="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight focus:outline-none focus:shadow-outline" id="email" **type**="email" **name**="email" **value**="{{ old('email') }}" required autocomplete="email" autofocus>

</div>

<div **class**="mb-4">

<label **class**="block text-gray-700 text-sm font-bold mb-2" **for**="password">

Hasło

</label>

<input **class**="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight focus:outline-none focus:shadow-outline" id="password" **type**="password" **name**="password" required autocomplete="current-password">

</div>

<div **class**="mb-4">

<input **class**="mr-2 leading-tight" **type**="checkbox" **name**="remember" id="remember" {{ old('remember') ? 'checked' : '' }}>

<label **class**="text-sm text-gray-600" **for**="remember">

Zapamiętaj mnie

</label>

</div>

<div **class**="flex items-center justify-between">

<button **class**="bg-blue-500 hover:bg-blue-700 text-white font-bold py-2 px-4 rounded focus:outline-none focus:shadow-outline" **type**="submit">

Zaloguj się

</button>

@**if** (Route::has('password.request'))

<a **class**="inline-block align-baseline font-bold text-sm text-blue-500 hover:text-blue-800" href="{{ route('password.request') }}">

Zapomniałeś hasła?

</a>

@**endif**

</div>

</form>

</div>

</body>

</html>

?>

**Kontroler Logowania (LoginController.php)**

Kontroler LoginController zarządza procesem logowania.

<?php

**namespace** App\Http\Controllers\Auth;

**use** App\Http\Controllers\Controller;

**use** Illuminate\Foundation\Auth\AuthenticatesUsers;

**use** Illuminate\Http\Request;

**class** LoginController **extends** Controller

{

**use** AuthenticatesUsers;

**protected** **$redirectTo** = '/home';

**public** **function** **\_\_construct**()

{

**$this**->middleware('guest')->except('logout');

}

**public** **function** showLoginForm()

{

**return** view('auth.login');

}

**protected** **function** credentials(Request **$request**)

{

**return** **$request**->only(**$this**->username(), 'password');

}

**protected** **function** sendFailedLoginResponse(Request **$request**)

{

**return** redirect()->back()

->withInput(**$request**->only(**$this**->username(), 'remember'))

->withErrors([

**$this**->username() => [trans('auth.failed')],

]);

}

}

**Proces Logowania Użytkownika**

1. **Wyświetlenie Formularza**: Użytkownik otwiera stronę logowania, gdzie wyświetlany jest formularz logowania.
2. **Wprowadzenie Danych**: Użytkownik wprowadza swoje dane logowania, takie jak adres email i hasło.
3. **Przesłanie Formularza**: Użytkownik przesyła formularz, wysyłając dane do serwera.
4. **Walidacja Danych**: Kontroler LoginController sprawdza, czy wprowadzone dane są poprawne i zgodne z danymi w bazie.
5. **Logowanie Użytkownika**: Po pomyślnej walidacji, użytkownik jest logowany do systemu.
6. **Przekierowanie**: Zalogowany użytkownik jest przekierowany na stronę główną lub dashboard.

# Walidacja:

Walidacja w systemie Budżet Studencki jest wielopoziomowym procesem, który zapewnia integralność danych i zabezpieczenia aplikacji. Składa się z walidacji na poziomie frontendu, backendu oraz bezpośredniej walidacji w bazie danych. Poniżej przedstawiam szczegółowe omówienie każdej z tych warstw walidacji.

#### Zabezpieczenia przycisku "Użytkownicy"

1. **Dostępność przycisku "Użytkownicy"**:
   * Przycisk "Użytkownicy" jest widoczny i dostępny tylko dla użytkowników z uprawnieniami administratora. Kontrola dostępu odbywa się zarówno po stronie serwera, jak i na poziomie interfejsu użytkownika.
   * Po stronie serwera, dostęp do widoku listy użytkowników i powiązanych akcji jest zabezpieczony za pomocą middleware, które sprawdza, czy zalogowany użytkownik jest administratorem.

<?php

**public** **function** **\_\_construct**()

{

**$this**->middleware(**function** (**$request**, **$next**) {

**if** (!auth()->check() || !auth()->**user**()->**isAdmin**) {

**return** redirect('/home')->with('error', 'Nie masz uprawnień do tej strony.');

}

**return** **$next**(**$request**);

});

}

?>

#### Walidacja Pól w Formularzach

1. **Frontend Walidacja**:
   * Walidacja na poziomie frontend jest realizowana przy użyciu standardowych atrybutów HTML5 oraz dodatkowo za pomocą JavaScriptu.
   * Atrybuty takie jak required, min, max, pattern zapewniają podstawową walidację na poziomie przeglądarki.

<?php

<input **type**="email" **name**="email" id="email" required>

<input **type**="number" **name**="amount" id="amount" **min**="0.01" **max**="99999999.99" step="0.01" required>

?>

Obraz zawierający tekst, zrzut ekranu, linia, Czcionka

Opis wygenerowany automatycznie

JavaScript jest używany do bardziej złożonych walidacji, takich jak sprawdzanie poprawności formularzy przed ich wysłaniem.

<?php

document.**getElementById**('amount').addEventListener('input', **function** () {

**if** (this.**value** > 99999999.99) {

this.setCustomValidity('Kwota nie może przekraczać 99999999.99 PLN.');

} **else** {

this.setCustomValidity('');

}

});

?>

**Backend Walidacja**:

* Backend walidacja odbywa się w kontrolerach Laravel, gdzie używane są reguły walidacyjne dostarczane przez framework.

<?php

**$request**->**validate**([

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

'email' => 'required|string|email|max:255|unique:users',

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

'isAdmin' => 'required|boolean',

]);

**$request**->**validate**([

'type' => 'required|in:Dochody,Wydatki',

'category\_id' => 'required',

'amount' => 'required|numeric|min:0.01|max:99999999.99',

'date' => 'required|date',

'description' => 'nullable|string',

]);

?>

**Walidacja w Bazie Danych**:

* Walidacja na poziomie bazy danych jest realizowana za pomocą ograniczeń i reguł zapisanych w migracjach bazy danych.

<?php

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

**$table**->id();

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

**$table**->foreignId('category\_id')->constrained('categories')->onDelete('cascade');

**$table**->decimal('amount', 15, 2)->check('amount > 0');

**$table**->enum('type', ['Dochody', 'Wydatki']);

**$table**->text('description')->nullable();

**$table**->**date**('date');

**$table**->timestamps();

});

?>

#### Przykłady Walidacji

1. **Walidacja Przycisku "Użytkownicy"**:
   * Dostęp do funkcji zarządzania użytkownikami jest kontrolowany za pomocą middleware oraz sprawdzenia uprawnień użytkownika. Przycisk jest renderowany w interfejsie użytkownika tylko dla administratorów.
2. **Walidacja Formularza Dodawania Transakcji**:
   * W formularzu dodawania transakcji używane są atrybuty HTML5 oraz JavaScript do wstępnej walidacji danych.
   * Po przesłaniu formularza, dane są walidowane ponownie na serwerze za pomocą metod walidacyjnych w Laravel.

<?php

**public** **function** storeTransaction(Request **$request**)

{

**$request**->**validate**([

'type' => 'required|in:Dochody,Wydatki',

'category\_id' => 'required',

'amount' => 'required|numeric|min:0.01|max:99999999.99',

'date' => 'required|date',

'description' => 'nullable|string',

]);

**$data** = **$request**->all();

**$data**['user\_id'] = auth()->**user**()->**id**;

Transaction::create(**$data**);

**return** redirect()->route('transactions.index')->with('success', 'Transakcja została dodana.');

}

?>

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Ikona komputerowa

Opis wygenerowany automatycznie

**Walidacja Formularza Rejestracji**:

* W formularzu rejestracji użytkownika pola są walidowane zarówno na frontendzie, jak i na backendzie. Atrybuty HTML5 oraz JavaScript zapewniają wstępną walidację, natomiast ostateczna walidacja odbywa się na serwerze przy użyciu metod walidacyjnych Laravel.

<?php

**$request**->**validate**([

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

'email' => 'required|string|email|max:255|unique:users',

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

]);

**User**::create([

'name' => **$request**->**name**,

'email' => **$request**->**email**,

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

]);

?>

#### Przykłady stron błędów

1. **Strona błędu 404 (**<http://127.0.0.1:8000/test-404>**)**:

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

1. **Strona błędu 500 (**<http://127.0.0.1:8000/test-500>**)**:

Obraz zawierający tekst, zrzut ekranu, Czcionka, design

Opis wygenerowany automatycznie

1. **Strona błędu 403 (**<http://127.0.0.1:8000/test-403>**)**:

Obraz zawierający tekst, zrzut ekranu, Czcionka, design

Opis wygenerowany automatycznie