

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

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

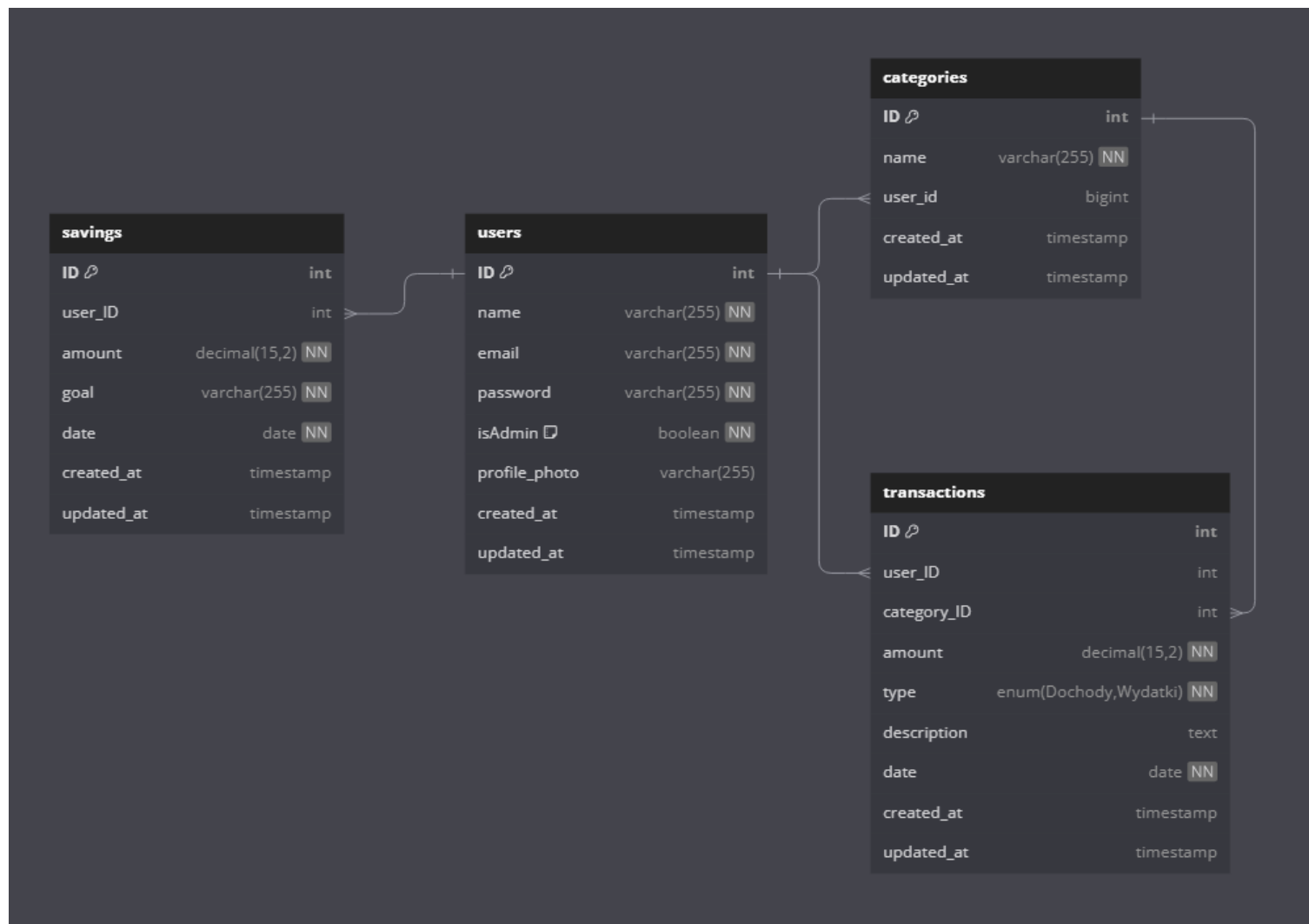
Wykorzystane technologie:

- **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.

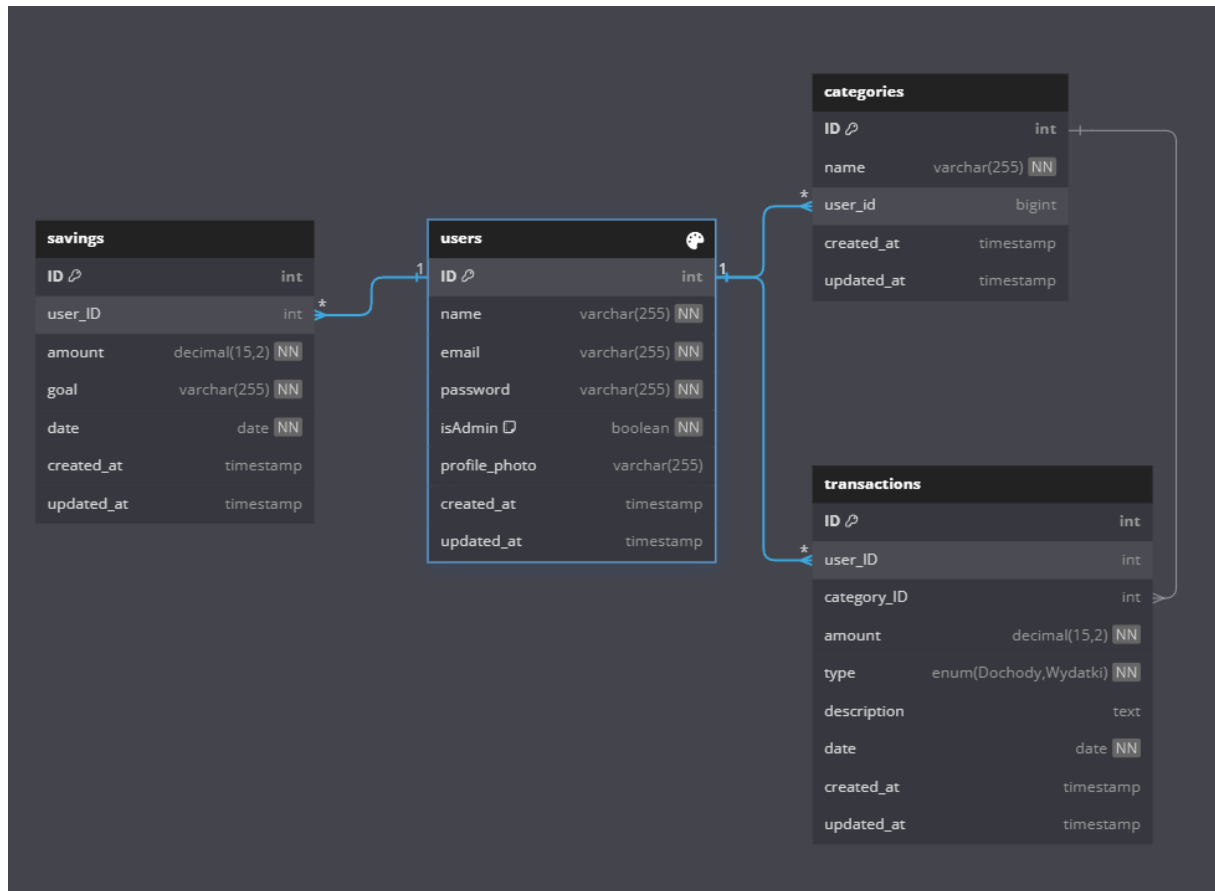
Baza danych

Schemat ERD:



Opis zawartości bazy danych:

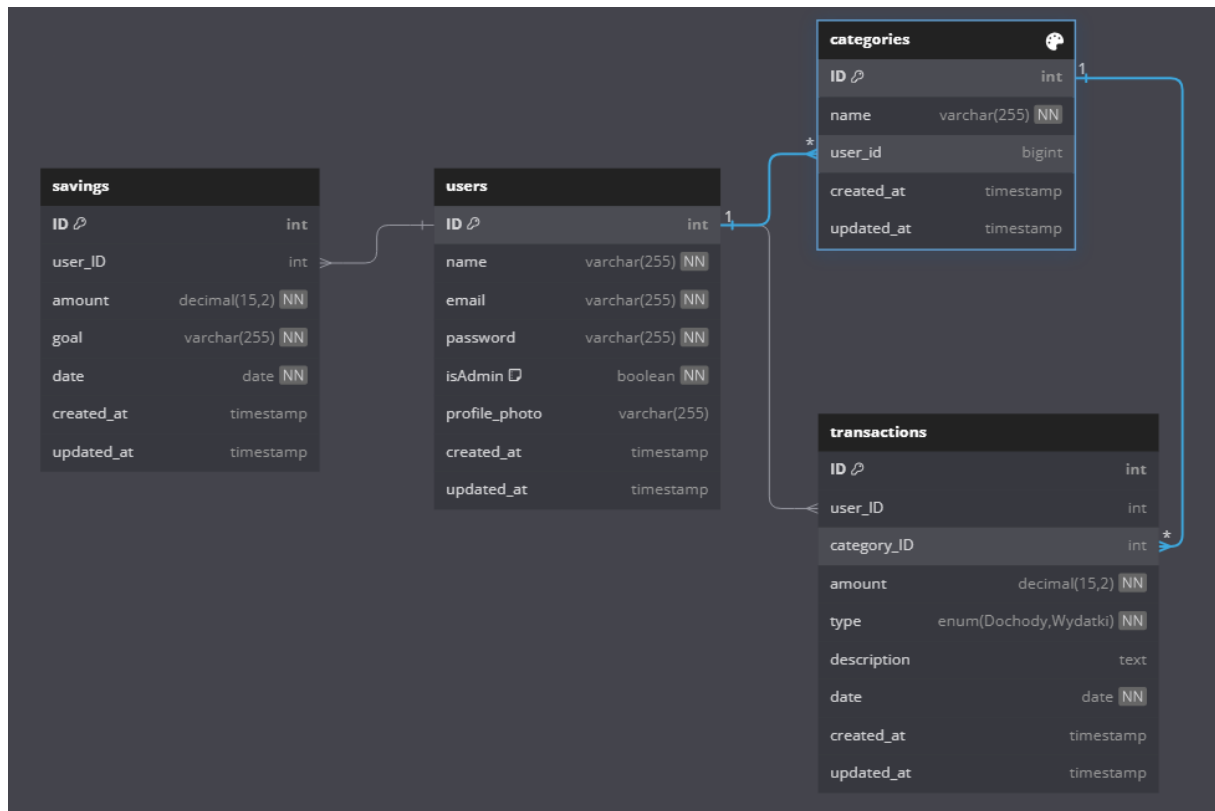
- Tabela **users**: Przechowuje informacje o użytkownikach, w tym ich dane logowania i role.



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**.

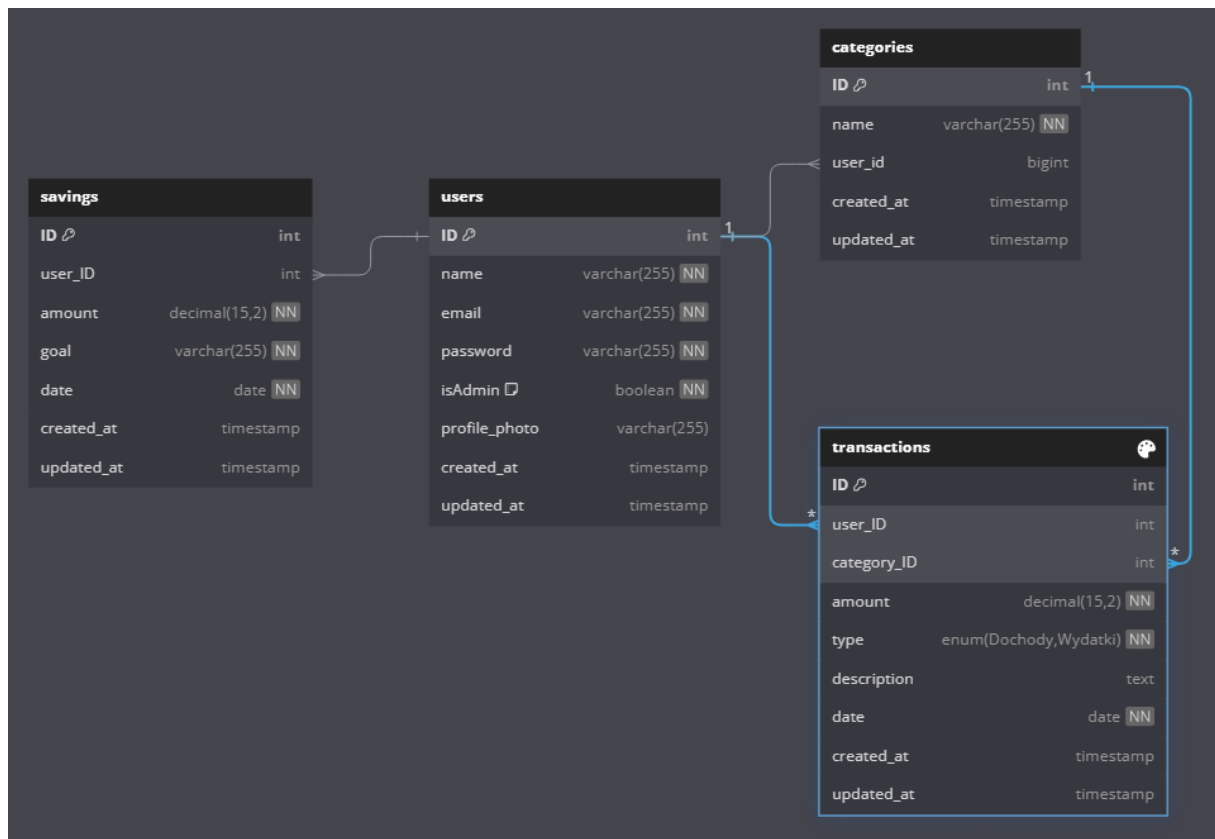
- **categories:** Przechowuje kategorie transakcji tworzone przez użytkowników.



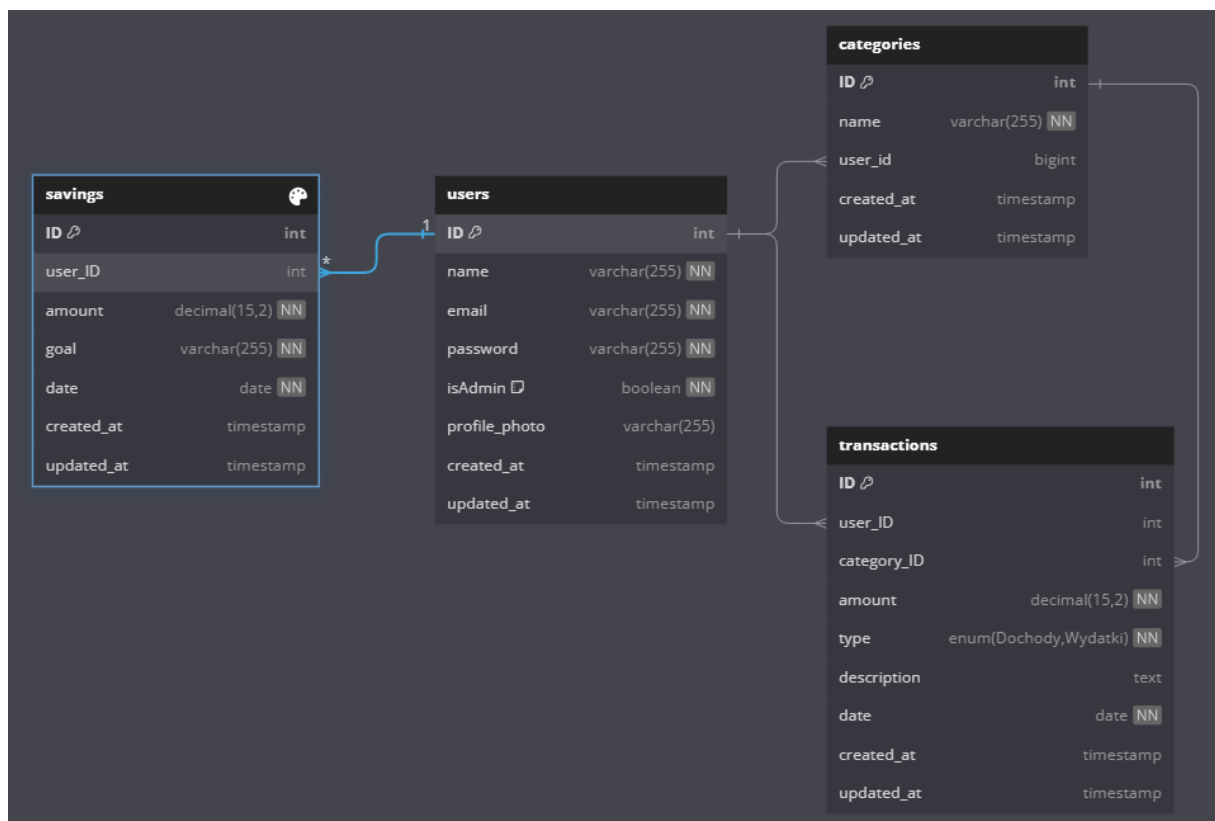
Powiązania między tabelami:

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.



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



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.

2. 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.

3. transactions

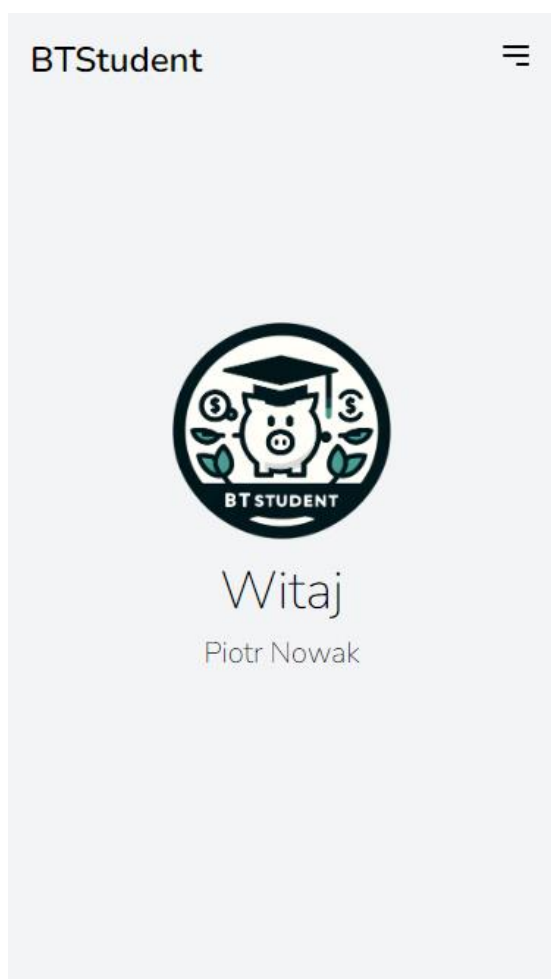
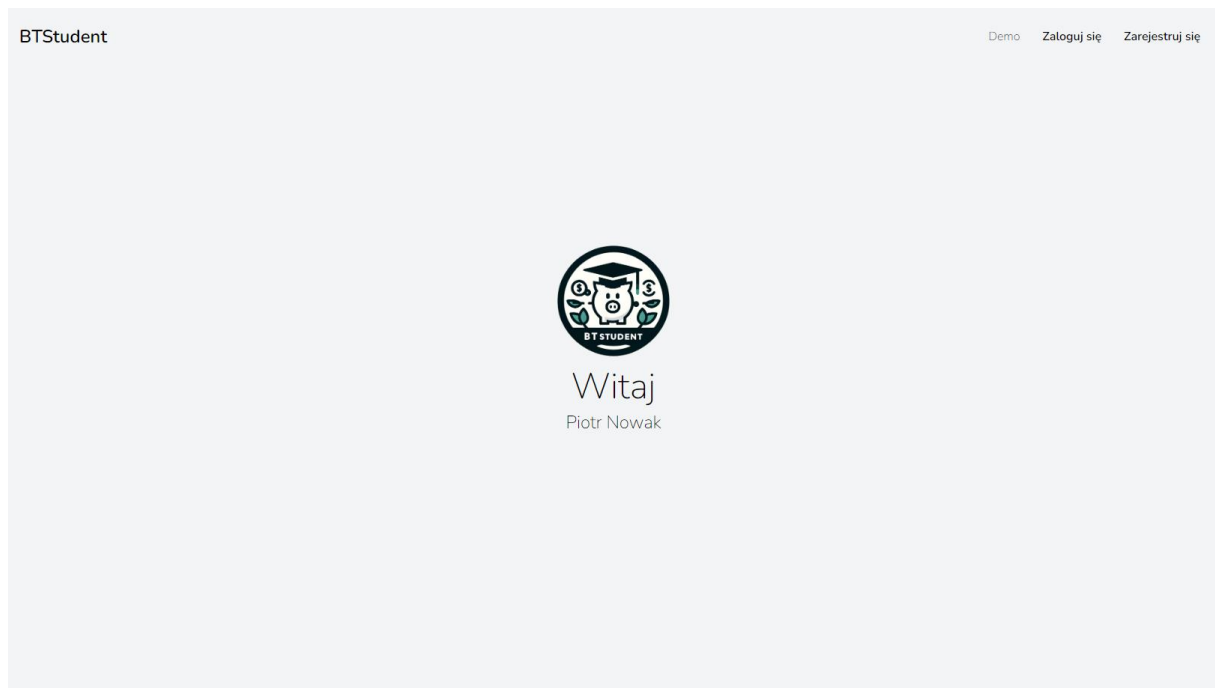
- **id:** (integer, primary key, auto-increment) Unikalny identyfikator transakcji.
- **user_id:** (integer, foreign key) Identyfikator użytkownika, który stworzył transakcję.
- **category_id:** (integer, foreign key) Identyfikator kategorii, do której należy transakcja.
- **amount:** (decimal) Kwota 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.

4. savings

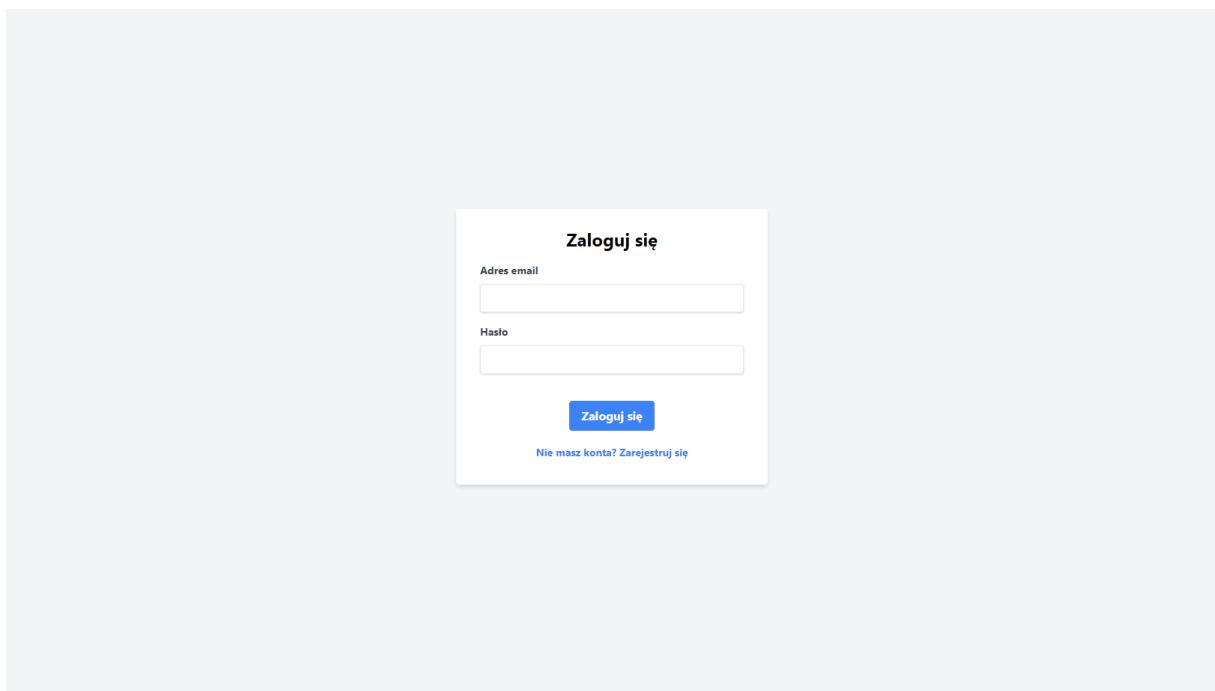
- **id:** (integer, primary key, auto-increment) Unikalny identyfikator oszczędności.
- **user_id:** (integer, foreign key) Identyfikator użytkownika, który stworzył oszczędność.
- **goal:** (string) Cel oszczędności.
- **amount:** (decimal) Kwota oszczędności.
- **date:** (date) Data, do której użytkownik 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)



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



A desktop login form titled "Zaloguj się" centered on a light gray background. It features two input fields: "Adres email" and "Hasło". Below the fields is a blue button labeled "Zaloguj się". At the bottom, there is a link "Nie masz konta? Zarejestruj się" in blue text.

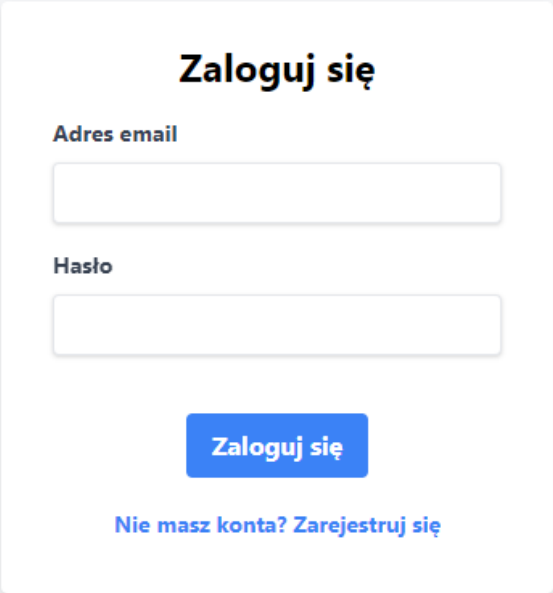
Zaloguj się

Adres email

Hasło

Zaloguj się

[Nie masz konta? Zarejestruj się](#)



A mobile login form titled "Zaloguj się" centered on a light gray background. It features two input fields: "Adres email" and "Hasło". Below the fields is a blue button labeled "Zaloguj się". At the bottom, there is a link "Nie masz konta? Zarejestruj się" in blue text.

Zaloguj się

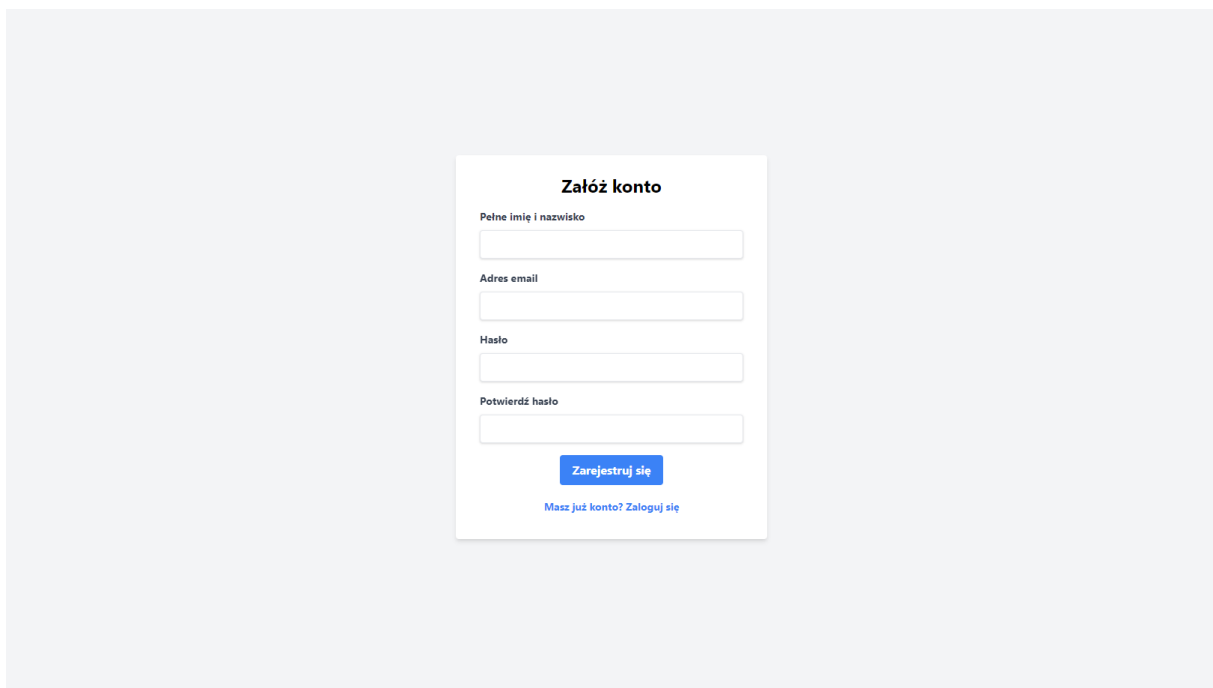
Adres email

Hasło

Zaloguj się

[Nie masz konta? Zarejestruj się](#)

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



The desktop view of the registration form is centered on a light gray background. It features a white card with a thin gray border. The title "Załącz konto" is centered at the top in a bold, black font. Below the title, there are four input fields, each preceded by a label: "Pełne imię i nazwisko", "Adres email", "Hasło", and "Potwierdź hasło". The input fields are white with a light gray border. At the bottom of the card, there is a blue button with the text "Zarejestruj się" in white, and a link "Masz już konto? Zaloguj się" in blue text below it.

Załącz konto

Pełne imię i nazwisko

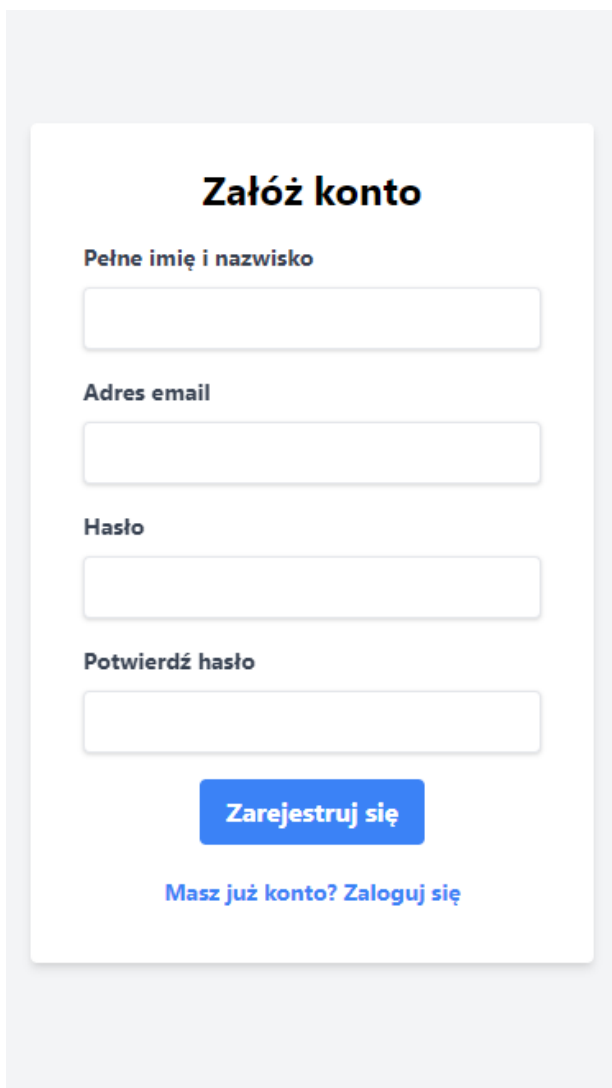
Adres email

Hasło

Potwierdź hasło

[Zarejestruj się](#)

[Masz już konto? Zaloguj się](#)



The mobile view of the registration form is shown on a light gray background. It features a white card with a thin gray border. The title "Załącz konto" is centered at the top in a bold, black font. Below the title, there are four input fields, each preceded by a label: "Pełne imię i nazwisko", "Adres email", "Hasło", and "Potwierdź hasło". The input fields are white with a light gray border. At the bottom of the card, there is a blue button with the text "Zarejestruj się" in white, and a link "Masz już konto? Zaloguj się" in blue text below it.

Załącz konto

Pełne imię i nazwisko

Adres email

Hasło

Potwierdź hasło

[Zarejestruj się](#)

[Masz już konto? Zaloguj się](#)

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

BTStudent

Dashboard

Transakcje

Kategorie

Oszczędności

Profil

Wyjdź

Panel Demo

Całkowity Dochód

700.00 PLN

Całkowite Wydatki

250.00 PLN

Łączne Saldo

450.00 PLN

Ostatnie Transakcje

ID	TYP	KATEGORIA	KWOTA	DATA	OPIS
1	Dochody	Jedzenie	500	2024-06-01	Wynagrodzenie
2	Wydatki	Transport	150	2024-06-03	Bilet miesięczny
3	Dochody	Rozrywka	200	2024-06-05	Zwrot podatku
4	Wydatki	Jedzenie	100	2024-06-07	Zakupy spożywcze

BTStudent

Panel Demo

Całkowity Dochód

700.00 PLN

Całkowite Wydatki

250.00 PLN

Łączne Saldo

450.00 PLN

Ostatnie Transakcje

ID	TYP	KATEGORIA	KWOTA
1	Dochody	Jedzenie	500

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

Admin:

BTStudent

DashboardTransakcjeKategorieOszczędnościProfilUżytkownicyWyloguj

Panel Administracyjny

Całkowity Dochód
180.00 PLN

Całkowite Wydatki
195.00 PLN

Łączne Saldo
-15.00 PLN

Ostatnie Transakcje

TYP	KATEGORIA	KWOTA	DATA	OPIS
Dochody	Jedzenie	80.00 PLN	2023-08-01	Sprzedaż gadżetów
Wydatki	Jedzenie	70.00 PLN	2023-07-01	Zakupy
Wydatki	Jedzenie	60.00 PLN	2023-06-01	Kolacja
Dochody	Rozrywka	100.00 PLN	2023-05-03	Sprzedaż starego podręcznika
Wydatki	Transport	15.00 PLN	2023-05-02	Bilet autobusowy

User:

BTStudent

DashboardTransakcjeKategorieOszczędnościProfilWyloguj

Panel Administracyjny

Całkowity Dochód
0.00 PLN

Całkowite Wydatki
60.00 PLN

Łączne Saldo
-60.00 PLN

Ostatnie Transakcje

TYP	KATEGORIA	KWOTA	DATA	OPIS
Wydatki	Jedzenie	60.00 PLN	2023-06-01	Kolacja

Admin:

User:

Panel Administracyjny

Całkowity Dochód
180.00 PLN

Całkowite Wydatki
195.00 PLN

Łączne Saldo
-15.00 PLN

Ostatnie Transakcje

TYP	KATEGORIA	KWOTA	DATA
Dochody	Jedzenie	80.00 PLN	2020-08-08

Panel Administracyjny

Całkowity Dochód
0.00 PLN

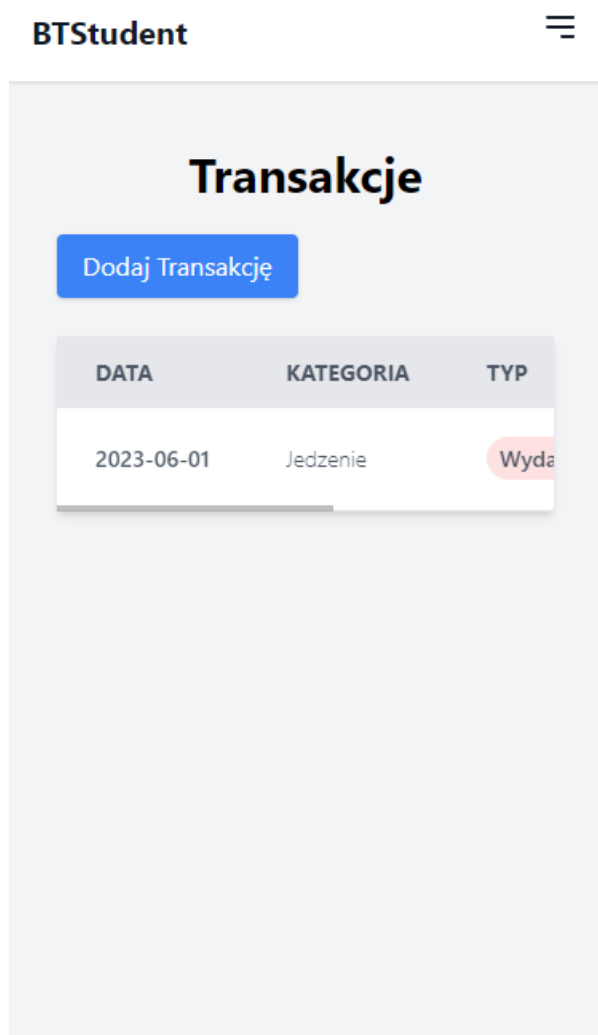
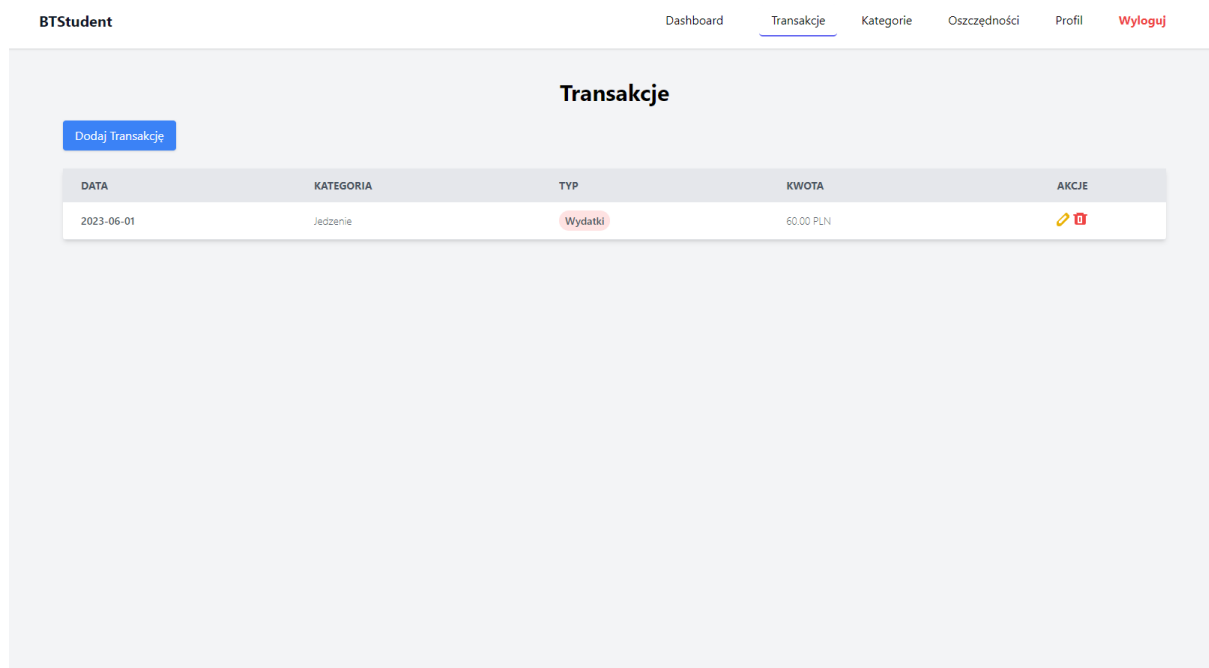
Całkowite Wydatki
60.00 PLN

Łączne Saldo
-60.00 PLN

Ostatnie Transakcje

TYP	KATEGORIA	KWOTA	DATA
Wydatki	Jedzenie	60.00 PLN	2020-06-06

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



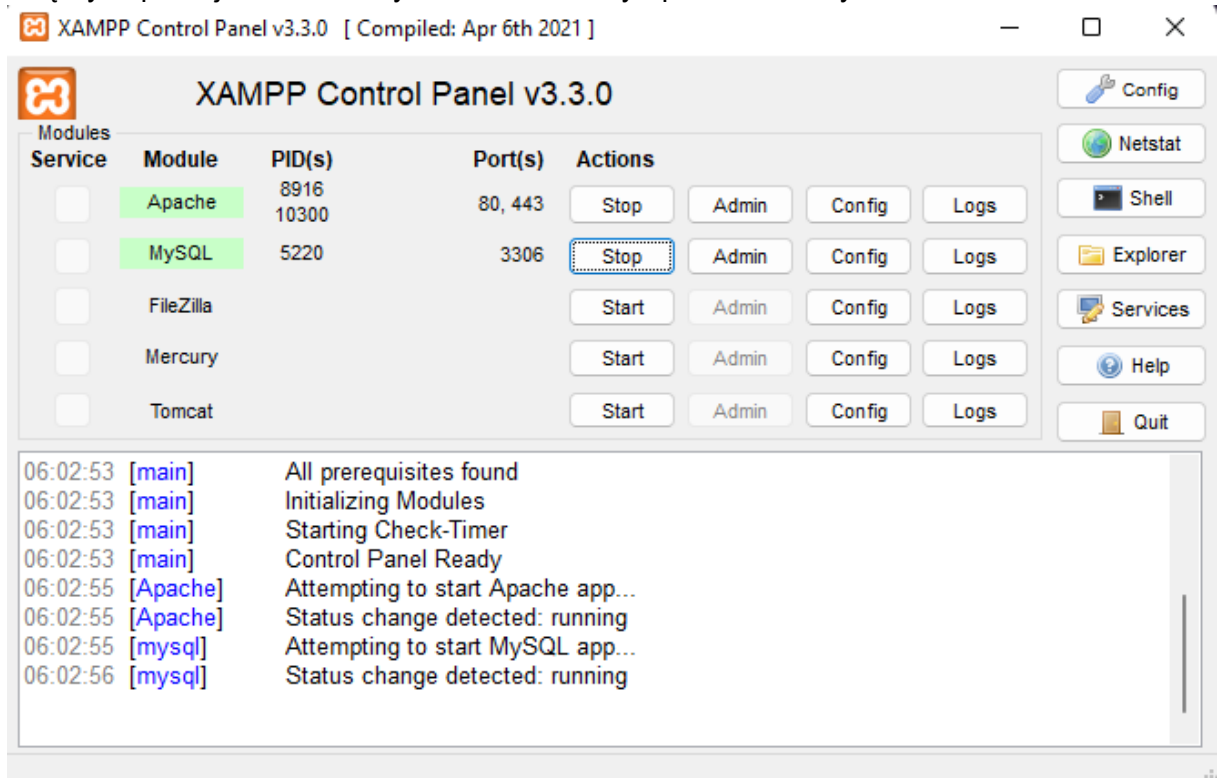
Uruchomienie aplikacji:

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 aplikację XAMPP i wystartować moduły Apache oraz MySQL:



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.

3. W lewym górnym rogu kliknąć zakładkę Terminal >> New Terminal (lub Ctrl + Shift + `).

4. Wywołać komendę `php artisan serve`.
5. Otworzyć kolejny terminal i wywołać komendę `npm run dev`.
6. 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

Hasło: password

User:

User1

Login: user1@example.com

Hasło: password

User2

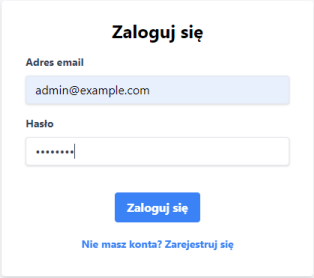
Login: user2@example.com

Hasło: password

User3

Login: user3@example.com

Hasło: password



The screenshot shows a login form titled "Zaloguj się" (Log in) centered on a light gray background. The form contains two input fields: "Adres email" (Email address) with the value "admin@example.com" and "Hasło" (Password) with masked characters "*****". Below the fields is a blue button labeled "Zaloguj się". At the bottom of the form, there is a link that says "Nie masz konta? Zarejestruj się" (Don't have an account? Register).

Załącz konto

Pełne imię i nazwisko

Adres email

Hasło

Potwierdź hasło

Zarejestruj się

[Masz już konto? Zaloguj się](#)

CRUD (admin):

Dodawanie transakcji przez administratora:

BTStudent

Dashboard

Transakcje

Kategorie

Oszczędności







Profil

Użytkownicy

Wyloguj

Transakcje

Dodaj Transakcję

DATA	KATEGORIA	TYP	KWOTA	AKCJE
2023-05-01	Jedzenie	Wydatki	50.00 PLN	 
2023-05-02	Transport	Wydatki	15.00 PLN	 
2023-05-03	Rozrywka	Dochody	100.00 PLN	 

BTStudent

Dashboard

Transakcje

Kategorie

Oszczędności

Profil

Użytkownicy

Wyloguj

Dodaj Transakcję

Typ

Wydatki

Kategoria

Transport

Kwota

10

Data

03.06.2024

Opis

Autobus

Dodaj

BTStudent

Dashboard

Transakcje

Kategorie

Oszczędności









Profil

Użytkownicy

Wyloguj

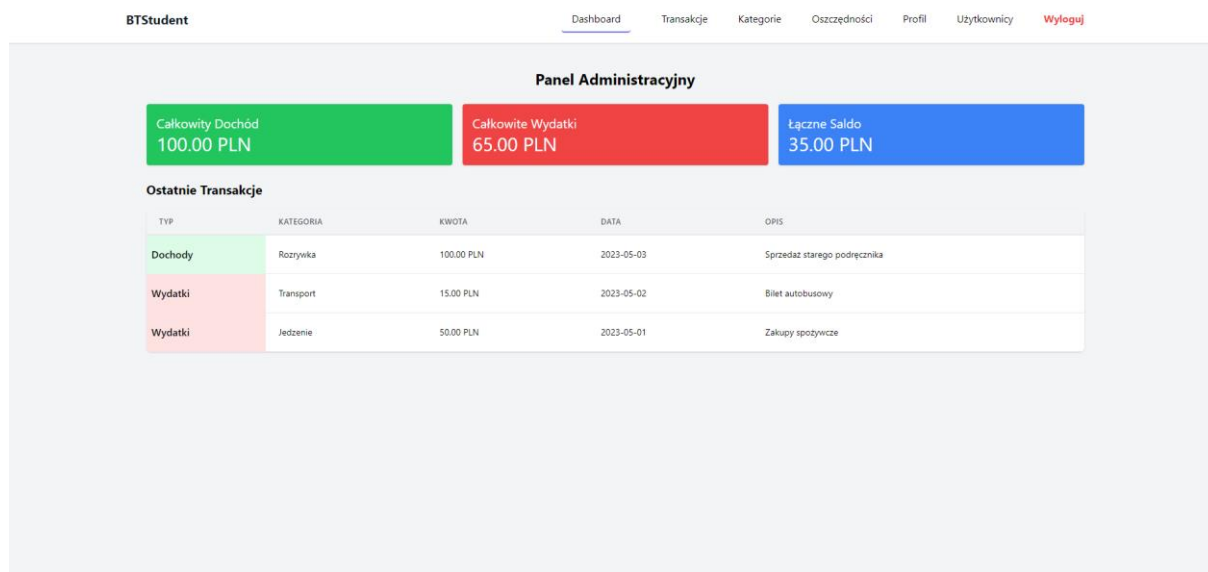
Transakcje

Dodaj Transakcję

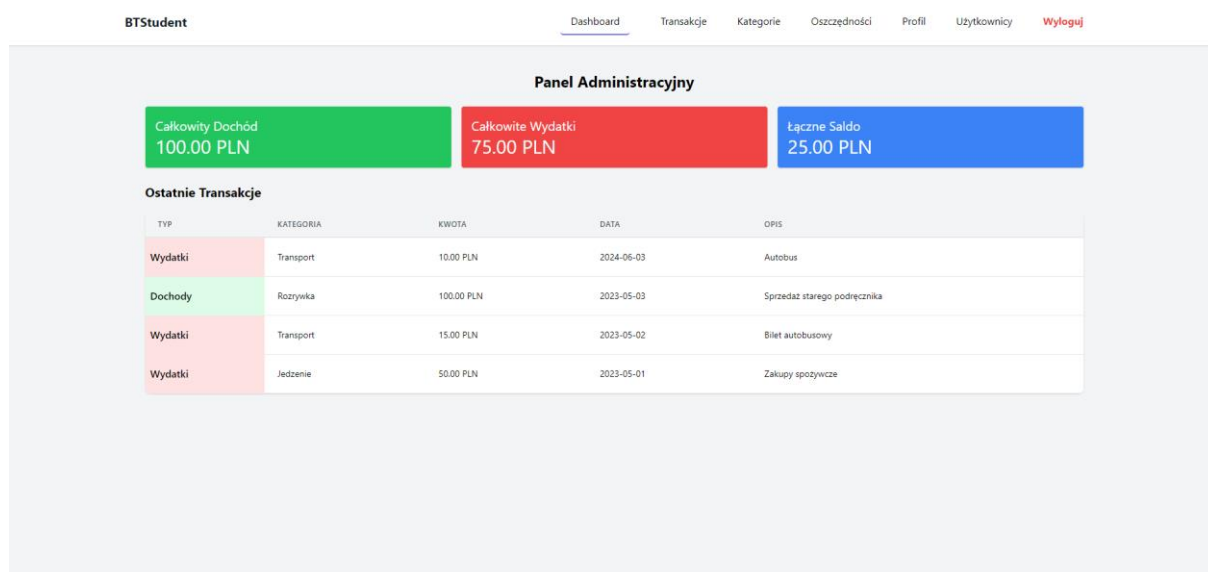
DATA	KATEGORIA	TYP	KWOTA	AKCJE
2023-05-01	Jedzenie	Wydatki	50.00 PLN	 
2023-05-02	Transport	Wydatki	15.00 PLN	 
2023-05-03	Rozrywka	Dochody	100.00 PLN	 
2024-06-03	Transport	Wydatki	10.00 PLN	 

Użytkownik admin od razu zobaczy swoje transakcje w widoku Dashboard.

Przed dodaniem transakcji:



Po dodaniu transakcji:



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
?>

```


Kontroler TransactionController

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

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\Transaction;
use App\Models\Category;
use Illuminate\Support\Facades\Auth;

class TransactionController extends Controller
{
    public function create()
    {
        $categories = Category::where('user_id', Auth::id())->get();
        return view('transactions.create', compact('categories'));
    }

    public function store(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::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 TransactionController 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):

BTStudent

DashboardTransakcjeKategorieOszczędnościProfilUżytkownicyWyloguj

Transakcje

Dodaj Transakcję

DATA	KATEGORIA	TYP	KWOTA	AKCJE
2023-05-01	Jedzenie	Wydatki	50.00 PLN	 
2023-05-02	Transport	Wydatki	15.00 PLN	 
2023-05-03	Rozrywka	Dochody	100.00 PLN	 
2024-06-03	Transport	Wydatki	10.00 PLN	 

127.0.0.1:8000/transakcje/1/edit

BTStudent

DashboardTransakcjeKategorieOszczędnościProfilUżytkownicyWyloguj

Edytuj Transakcję

Typ

Wydatki

Kategoria

Jedzenie

Kwota

50.00

Data

dd.mm.rrrr

Opis

Zakupy spożywcze

Zapisz

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

BTStudent

Dashboard

Transakcje

Kategorie

Oszczędności

Profil

Użytkownicy

Wyloguj

Edytuj Transakcję

Typ

Wydatki

Kategoria

Jedzenie

Kwota

150,00

Data

03.06.2024

Opis

Zakupy spożywcze

Zapisz

BTStudent

Dashboard

Transakcje

Kategorie

Oszczędności









Profil

Użytkownicy

Wyloguj

Transakcje

Dodaj transakcję

DATA	KATEGORIA	TYP	KWOTA	AKCJE
2024-06-03	Jedzenie	Wydatki	150.00 PLN	 
2023-05-02	Transport	Wydatki	15.00 PLN	 
2023-05-03	Rozrywka	Dochody	100.00 PLN	 
2024-06-03	Transport	Wydatki	10.00 PLN	 

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 TransactionController

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

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\Transaction;
use App\Models\Category;
use Illuminate\Support\Facades\Auth;

class TransactionController extends Controller
{
    public function edit($id)
    {
        $transaction = Transaction::where('user_id', Auth::id())->findOrFail($id);
        $categories = Category::where('user_id', Auth::id())->get();
        return view('transactions.edit', compact('transaction', 'categories'));
    }

    public function update(Request $request, $id)
    {
        $request->validate([
            'type' => 'required|in:Dochody,Wydatki',
            'category_id' => 'required',
            'amount' => 'required|numeric|min:0.01|max:999999999.99',
            'date' => 'required|date',
            'description' => 'nullable|string',
        ]);

        $transaction = Transaction::where('user_id', Auth::id())->findOrFail($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',
    ];

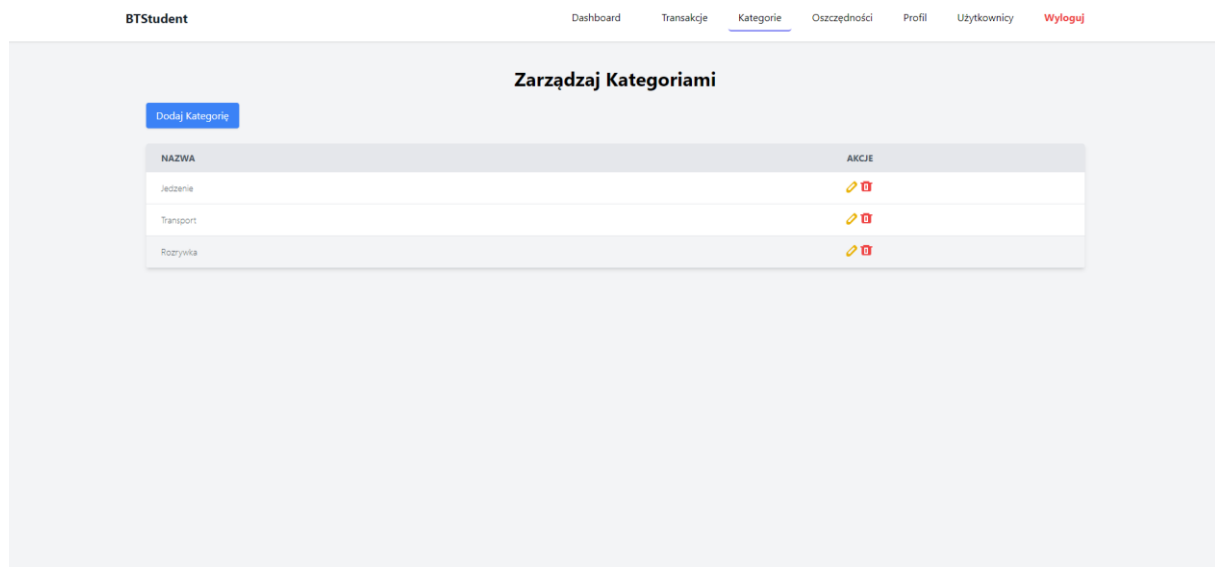
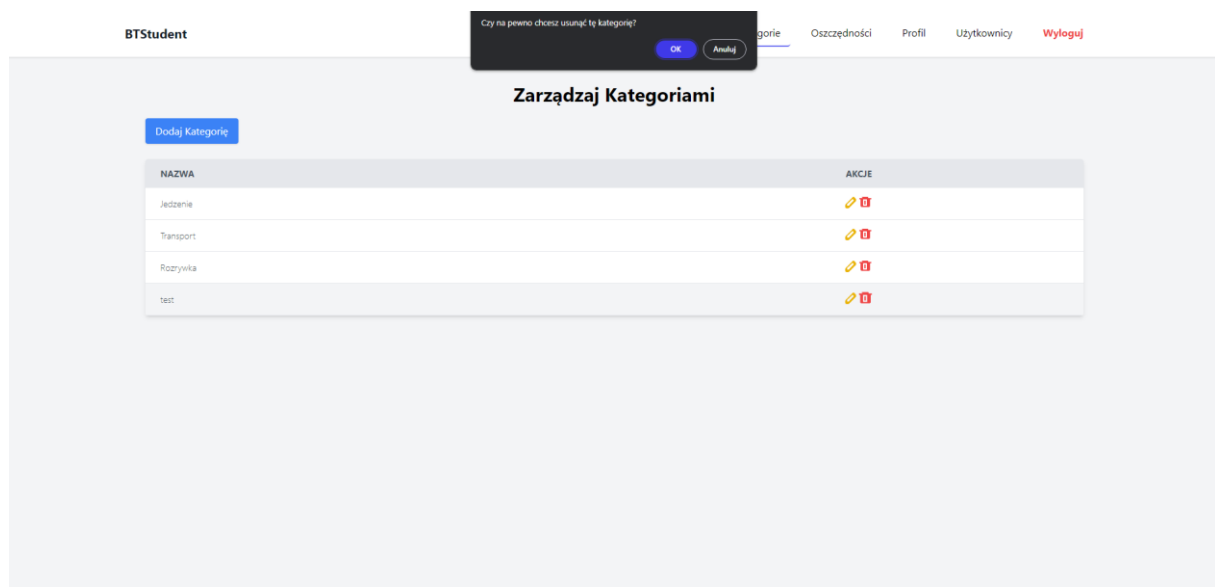
    public function category()
    {
        return $this->belongsTo(Category::class);
    }

    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 TransactionController 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.



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 CategoryController

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

```

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\Category;

class CategoryController extends Controller
{
    public function destroy($id)
    {
        // Znajdź kategorię do usunięcia
        $category = Category::findOrFail($id);

        // Sprawdź, czy kategoria ma powiązane transakcje
        if ($category->transactions()->exists()) {
            return redirect()->route('categories.index')->with('error', 'Nie można usunąć kategorii, która
ma powiązane transakcje.');
```

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 CategoryController 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:

BTStudent Dashboard Transakcje Kategorie Oszczędności Profil Użytkownicy Wyloguj

Dodaj Użytkownika

Nazwa:

Email:

Hasło:

Potwierdź Hasło:

Typ Konta:

[Dodaj Użytkownika](#)

BTStudent Dashboard Transakcje Kategorie Oszczędności Profil Użytkownicy Wyloguj

Zarządzaj Użytkownikami

[Dodaj Użytkownika](#)

Użytkownik został dodany.

| ID | NAZWA | EMAIL | TYP KONTA | AKCJE |
|----|-------|-------------------|-----------|-------------------------------------|
| 1 | Admin | admin@example.com | Admin | ✎ ✖ |
| 2 | User1 | user1@example.com | User | ✎ ✖ |
| 3 | User2 | user2@example.com | User | ✎ ✖ |
| 4 | User3 | user3@example.com | User | ✎ ✖ |
| 5 | test | test@example.com | User | ✎ ✖ |

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:

BTStudent

Dashboard

Transakcje

Kategorie

Oszczędności

Profil

Użytkownicy

Wyloguj

Edytuj Użytkownika

Nazwa:

User4

Email:

user4@example.com

Nowe Hasło:

Potwierdź Hasło:

Typ Konta:

User

Zaktualizuj Użytkownika

BTStudent

Dashboard

Transakcje

Kategorie

Oszczędności

Profil











Użytkownicy

Wyloguj

Zarządzaj Użytkownikami

Dodaj Użytkownika

Użytkownik został zaktualizowany.

| ID | NAZWA | EMAIL | TYP KONTA | AKCJE |
|----|-------|-------------------|-----------|---|
| 1 | Admin | admin@example.com | Admin |   |
| 2 | User1 | user1@example.com | User |   |
| 3 | User2 | user2@example.com | User |   |
| 4 | User3 | user3@example.com | User |   |
| 5 | User4 | user4@example.com | User |   |

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;

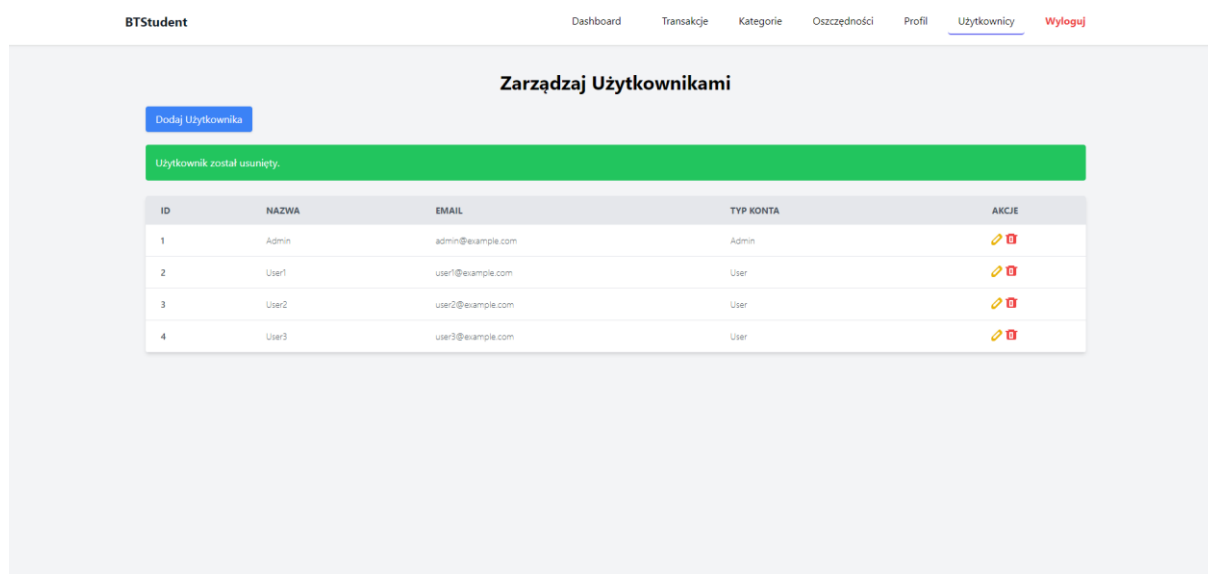
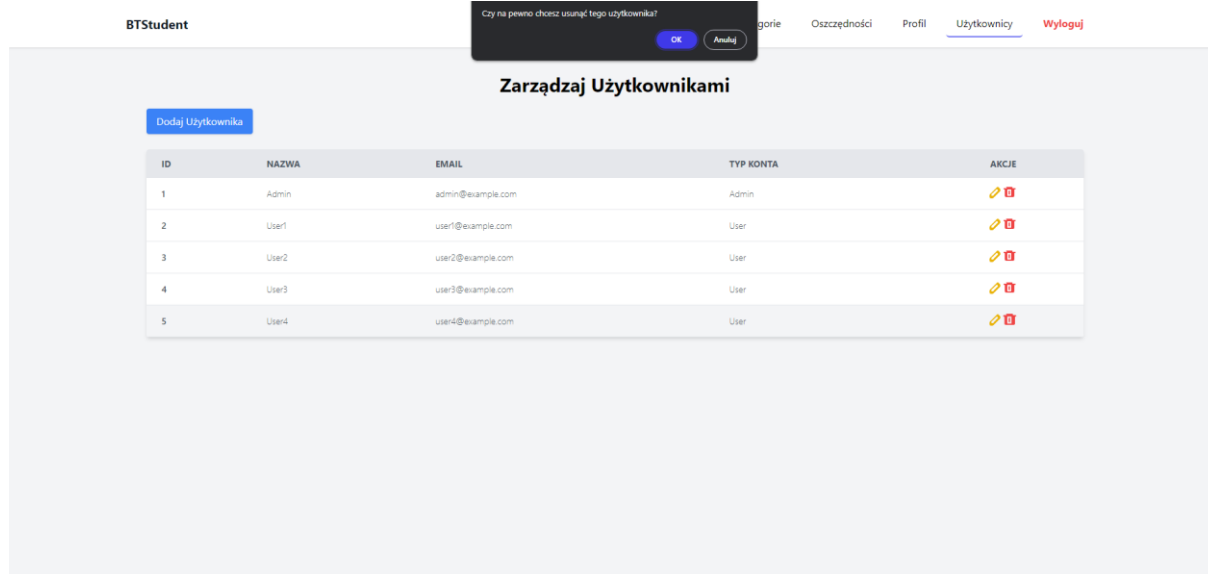
    if ($request->password) {
        $user->password = Hash::make($request->password);
    }

    $user->save();

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

??

Usuwanie przez administratora użytkownika User4 z systemu:



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.');
```

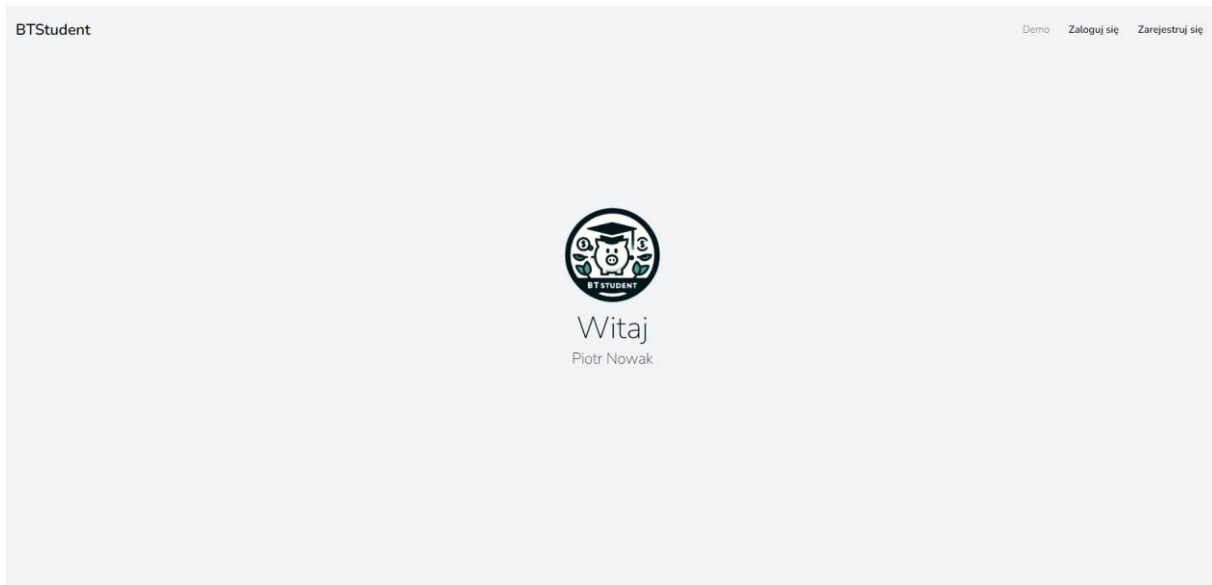
4. Metoda delete w modelu User

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.



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.

3. 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.

BTStudent

Dashboard Transakcje Kategorie Oszczędności Profil Wyjść

Panel Demo

| | | |
|---------------------------------------|--|-----------------------------------|
| Całkowity Dochód
700.00 PLN | Całkowite Wydatki
250.00 PLN | Łączne Saldo
450.00 PLN |
|---------------------------------------|--|-----------------------------------|

Ostatnie Transakcje

| ID | TYP | KATEGORIA | KWOTA | DATA | OPIS |
|----|---------|-----------|-------|------------|------------------|
| 1 | Dochody | Jedzenie | 500 | 2024-06-01 | Wynagrodzenie |
| 2 | Wydatki | Transport | 150 | 2024-06-03 | Bilet miesięczny |
| 3 | Dochody | Rozrywka | 200 | 2024-06-05 | Zwrot podatku |
| 4 | Wydatki | Jedzenie | 100 | 2024-06-07 | Zakupy spożywcze |

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

BTStudent

Dashboard Transakcje Kategorie Oszczędności Profil Wyjść

Transakcje Demo

[Dodaj transakcję](#)

| ID | TYP | KATEGORIA | KWOTA | DATA | OPIS | AKCJE |
|----|---------|-----------|-------|------------|------------------|-------------------------------------|
| 1 | Dochody | Jedzenie | 500 | 2024-06-01 | Wynagrodzenie | ✎ ✖ |
| 2 | Wydatki | Transport | 150 | 2024-06-03 | Bilet miesięczny | ✎ ✖ |
| 3 | Dochody | Rozrywka | 200 | 2024-06-05 | Zwrot podatku | ✎ ✖ |
| 4 | Wydatki | Jedzenie | 100 | 2024-06-07 | Zakupy spożywcze | ✎ ✖ |

demo_categories.blade.php: Widok przeglądania kategorii demo, który pozwala na dodawanie, edytowanie i usuwanie kategorii.

BTStudent

DashboardTransakcjeKategorieOszczędnościProfilWyjdź

| Kategorie Demo | | | Dodaj Kategorię |
|----------------|-----------|---|---------------------------------|
| ID | NAZWA | AKCJE | |
| 1 | Jedzenie | Edytuj Usuń | |
| 2 | Transport | Edytuj Usuń | |
| 3 | Rozrywka | Edytuj Usuń | |

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

BTStudent

DashboardTransakcjeKategorieOszczędnościProfilWyjdź

| Oszczędności Demo | | | | | Dodaj Oszczędność |
|-------------------|-------------|-------|------------|---|-----------------------------------|
| ID | CEL | KWOTA | DATA | AKCJE | |
| 1 | Nowy laptop | 1500 | 2024-06-30 | Edytuj Usuń | |
| 2 | Wakacje | 2000 | 2024-07-15 | Edytuj Usuń | |

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,
    ]);
}
```

2. **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();
}

?>
```

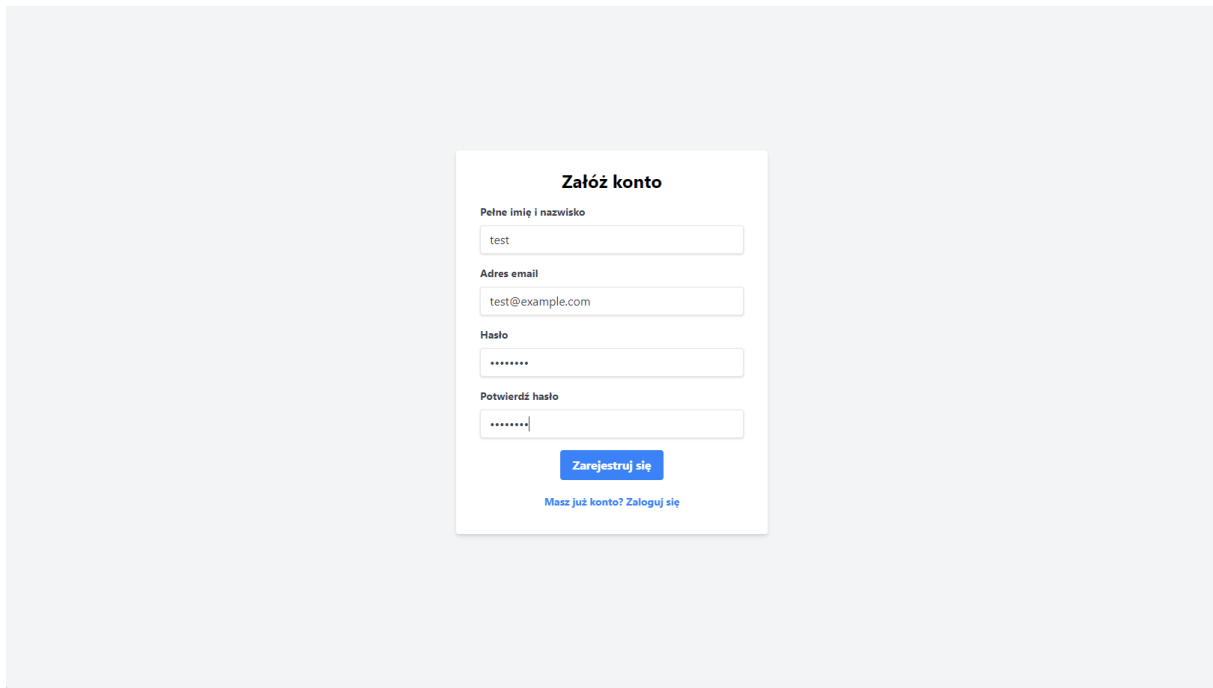
Proces rejestracji i logowania:

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.



```
<?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">Załącz 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);
    }
}
```

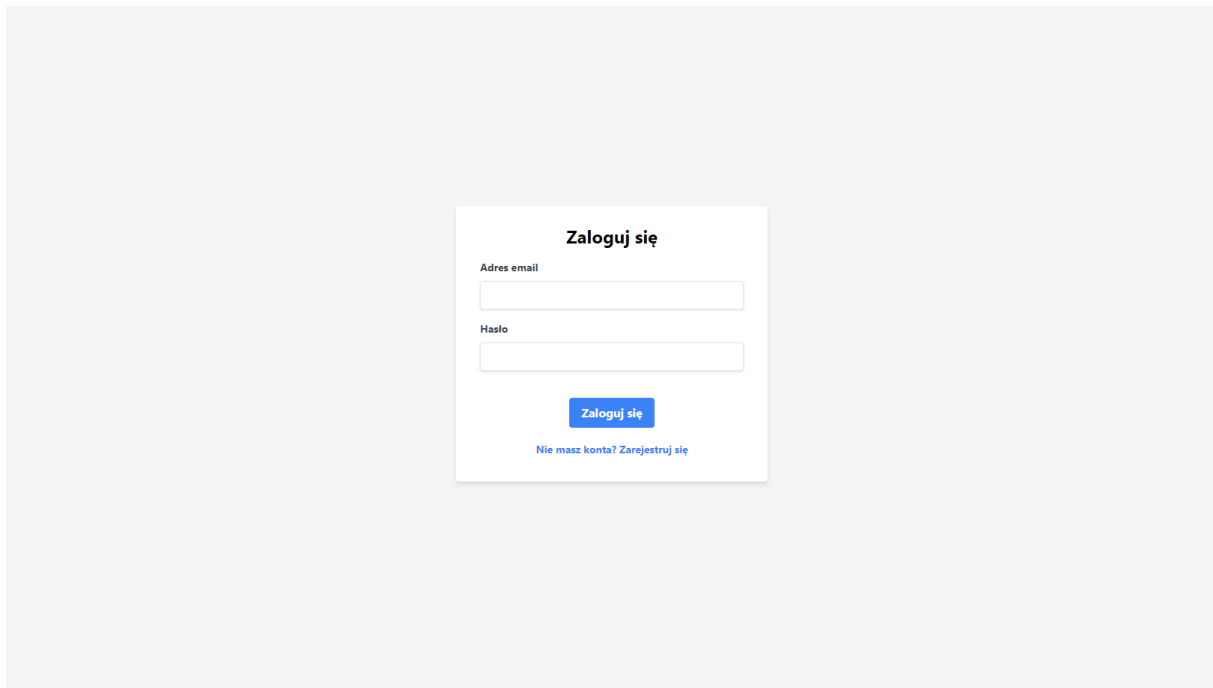
Proces Rejestracji Użytkownika

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.



```
<?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.');
```

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>

?>
```

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.');
```

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.');
```

??>

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),
]);

??>
```