

<b>AI1</b>	<b>Dokumentacja projektu</b>
<b>Autor</b>	Mateusz Bocak, 125104
<b>Kierunek, rok</b>	Informatyka, II rok, st. Stacjonarne (3,5-I)
<b>Temat Projektu</b>	Aplikacja do prowadzenia personalnego budżetu - myMoney

## Spis treści

Wstęp.....	3
Obszar działania aplikacji .....	3
Funkcje dostępne dla użytkownika.....	3
Narzędzia i technologie.....	3
Narzędzia.....	4
Technologie .....	4
Baza danych.....	5
Schemat ERD bazy danych.....	5
Opis zawartości bazy danych .....	5
Powiązania pomiędzy tabelami .....	6
GUI .....	6
Instrukcja uruchomienia aplikacji.....	7
Wymagane oprogramowanie .....	8
Konfiguracja.....	8
Przygotowanie bazy danych i seedów .....	8
Uruchomienie aplikacji.....	8
Serwer .....	8
Klient .....	9
Funkcjonalności aplikacji.....	9
Typy użytkowników.....	9
Administrator.....	9
Użytkownik anonimowy .....	9
Użytkownik .....	9
Logowanie i rejestracja .....	9
Logowanie.....	9
Rejestracja .....	10
CRUD administratora .....	10
Zarządzanie użytkownikami .....	10
Przeglądanie ogólnodostępnych zasobów .....	10
Zarządzanie danymi przez użytkownika .....	11
Wybrana logika biznesowa.....	11
Opis walidacji danych.....	13
Walidacja back-end.....	13
Walidacja front-end .....	16

## Wstęp

Projekt aplikacji "MyMoney" jest przeznaczony do zarządzania budżetem użytkowników poprzez system ewidencji przychodów i wydatków. Aplikacja umożliwia użytkownikom śledzenie swoich finansów, ustawianie celów oszczędnościowych oraz zarządzanie nimi.

Użytkownicy aplikacji mogą zalogować się lub zarejestrować, aby uzyskać dostęp do swoich kont i skorzystać z funkcjonalności aplikacji.

Głównym celem aplikacji jest ułatwienie użytkownikom śledzenia i zarządzania swoimi finansami, zapewniając przejrzyste interfejsy i podstawowe narzędzia do monitorowania budżetu.

Aplikacja obsługuje tryb ciemny i jest zaprojektowana z pomocą podejścia „Mobile First”.

## Obszar działania aplikacji

Obszar działania aplikacji obejmuje:

- Rejestrację nowych użytkowników oraz logowanie istniejących.
- Umożliwienie użytkownikom dodawania, edycji i usuwania przychodów oraz wydatków.
- Tworzenie i zarządzanie celami oszczędnościowymi.
- Zarządzanie danymi osobistymi i kontem użytkownika.
- Obsługa konta administratora, umożliwiająca zarządzanie użytkownikami i zasobami aplikacji.
- Zapewnienie interfejsu użytkownika, który jest czytelny, intuicyjny i responsywny.

## Funkcje dostępne dla użytkownika

- Rejestracja i logowanie: Użytkownicy mogą zarejestrować się w aplikacji, tworząc nowe konto, lub zalogować się, jeśli już posiadają konto.
- Dodawanie i zarządzanie przychodami: Użytkownicy mogą dodawać nowe wpisy dotyczące swoich przychodów, takie jak wynagrodzenia, premie, czy dochody z innych źródeł. Mogą również przeglądać, edytować i usuwać istniejące wpisy dotyczące przychodów.
- Dodawanie i zarządzanie wydatkami: Użytkownicy mogą dodawać nowe wpisy dotyczące swoich wydatków, takie jak rachunki, zakupy, czy inne wydatki. Mogą również przeglądać, edytować i usuwać istniejące wpisy dotyczące wydatków.
- Tworzenie i zarządzanie celami oszczędnościowymi: Użytkownicy mogą ustalać cele oszczędnościowe.
- Zarządzanie danymi osobistymi: Użytkownicy mogą zarządzać swoimi danymi osobowymi, takimi jak adres e-mail czy hasło, oraz edytować swoje preferencje konta.
- Wylogowanie: Użytkownicy mogą bezpiecznie się wylogować z aplikacji, aby zabezpieczyć swoje konto i dane.

Powyższe funkcje pozwalają użytkownikom efektywnie zarządzać swoimi finansami oraz planować przyszłe wydatki i oszczędności.

## Narzędzia i technologie

Aplikacja "MyMoney" została zbudowana przy użyciu różnorodnych narzędzi i technologii, które zapewniają stabilność, bezpieczeństwo i efektywność działania.

Proces produkcyjny był przeprowadzany na systemie Windows 10 w wersji 22H2 (komp. 190454412).

## Narzędzia

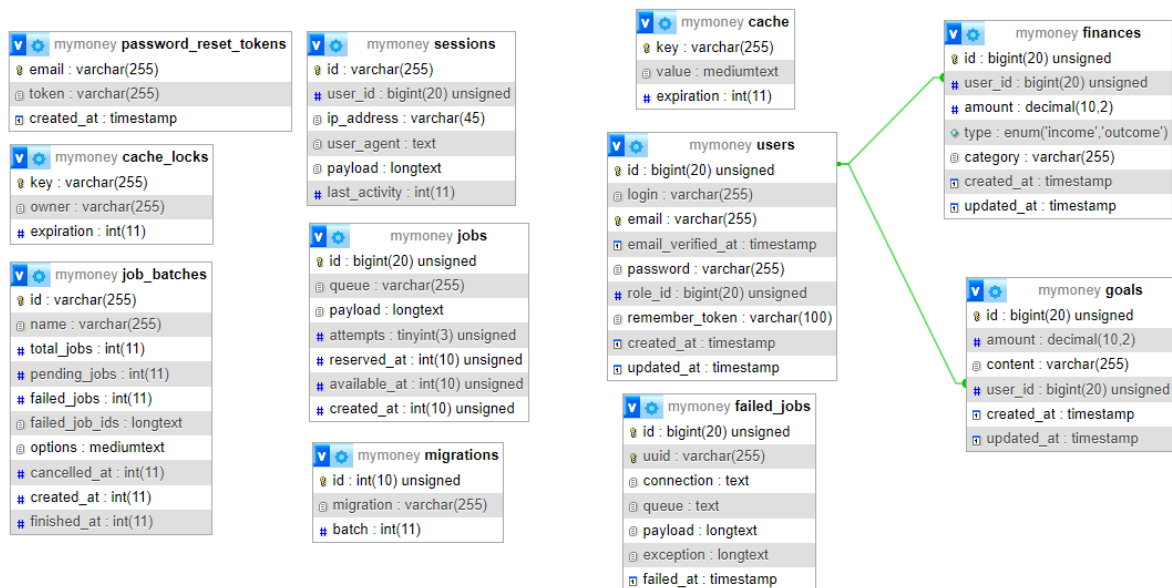
- [Git \(v2.37.2\)](#): System kontroli wersji używany do śledzenia zmian w kodzie aplikacji i współpracy programistów.
- [GitHub](#): Platforma do hostowania kodu źródłowego projektu i współpracy nad nim.
- [Visual Studio Code \(v1.89.1\)](#): Środowisko programistyczne do kodowania, testowania i debugowania aplikacji.
- [Composer \(v2.7.1\)](#): Menadżer zależności PHP używany do zarządzania zewnętrznymi bibliotekami i narzędziami.
- **Linux, Apache, MySQL, PHP (LAMP) stack**: Środowisko do uruchamiania aplikacji webowych na serwerze. W środowisku developerskim skonfigurowane za pomocą pakietu [XAMPP \(v8.2.12\)](#)

## Technologie

- [PHP \(v8.2.12\)](#): Język programowania serwerowego do obsługi żądań HTTP, przetwarzania danych i komunikacji z bazą danych.
- [MySQL](#): System zarządzania relacyjnymi bazami danych do przechowywania danych aplikacji. Wykorzystany silnik – [MariaDB](#) w wersji 10.4.32
- [Laravel \(v11.7.0\)](#): Framework PHP do budowy aplikacji webowych, zapewniający wiele gotowych funkcji i narzędzi.
- [Breeze \(v2.0\)](#): Pakiet narzędziowy Laravela do szybkiego tworzenia autentykacji.
- [Blade](#) - silnik szablonów używany w frameworku Laravel. Jest to narzędzie, które ułatwia tworzenie interfejsów użytkownika w aplikacjach webowych, poprzez umożliwienie integracji dynamicznych danych z kodem HTML
- [Flowbite \(v2.3.0\)](#): Framework CSS do stylizacji interfejsu użytkownika.
- [JavaScript](#): Język programowania używany do dodatkowej logiki aplikacji, interakcji z interfejsem użytkownika i walidacji formularzy.
- [TailwindCSS \(v3.1.0\)](#): Framework CSS do stylizacji HTML w prosty i powtarzalny sposób
- [Google Fonts](#) - [Chakra Petch](#): Czcionka używana w projekcie zapewniająca nowoczesny i estetyczny wygląd.

# Baza danych

## Schemat ERD bazy danych



## Opis zawartości bazy danych

Baza danych aplikacji "MyMoney" składa się z jedenastu tabel:

- **users**: Tabela przechowująca informacje o użytkownikach, takie jak identyfikator, login, adres e-mail, zahaszkowane hasło, identyfikator roli użytkownika, znacznik weryfikacji adresu e-mail, znacznik zapamiętania sesji, oraz znaczniki czasowe tworzenia i aktualizacji rekordu.
- **finances**: Tabela przechowująca transakcje finansowe użytkowników, zawierająca identyfikator transakcji, identyfikator użytkownika, kwotę transakcji, typ transakcji (dochód lub wydatek), kategorię transakcji oraz znaczniki czasowe dodania i aktualizacji rekordu.
- **goals**: Tabela przechowująca cele użytkowników, zawierająca identyfikator celu, kwotę celu, opis celu oraz identyfikator użytkownika, do którego cel należy, oraz znaczniki czasowe dodania i aktualizacji rekordu.
- **password\_reset\_tokens**: Tabela przechowująca tokeny resetowania hasła, zawierająca adres e-mail użytkownika oraz wygenerowany token resetowania hasła.
- **sessions**: Tabela przechowująca sesje użytkowników, zawierająca identyfikator sesji, identyfikator użytkownika, adres IP użytkownika, agenta użytkownika oraz dane sesji.
- **jobs**: Tabela przechowująca informacje o zadaniach wykonywanych przez system kolejkowania, zawierająca identyfikator zadania, kolejkę, dane zadania oraz znaczniki czasowe.
- **failed\_jobs**: Tabela przechowująca informacje o zadaniach, które nie zostały wykonane pomyślnie, zawierająca identyfikator zadania, kolejkę, dane zadania, wyjątek oraz znacznik czasowy niepowodzenia.
- **migrations**: Tabela przechowująca informacje o migracjach bazy danych wykonanych w aplikacji Laravel.
- **job\_batches**: Tabela przechowująca informacje o grupach zadań w systemie kolejkowania, zawierająca identyfikator grupy, informacje o zadaniach w grupie oraz znaczniki czasowe.
- **cache**: Tabela przechowująca dane tymczasowe, zawierająca klucz, wartość oraz czas wygaśnięcia danych.

- **cache\_locks:** Tabela przechowująca blokady dla danych tymczasowych w tabeli cache, zawierająca klucz, właściciela blokady oraz czas wygaśnięcia.

### Powiązania pomiędzy tabelami

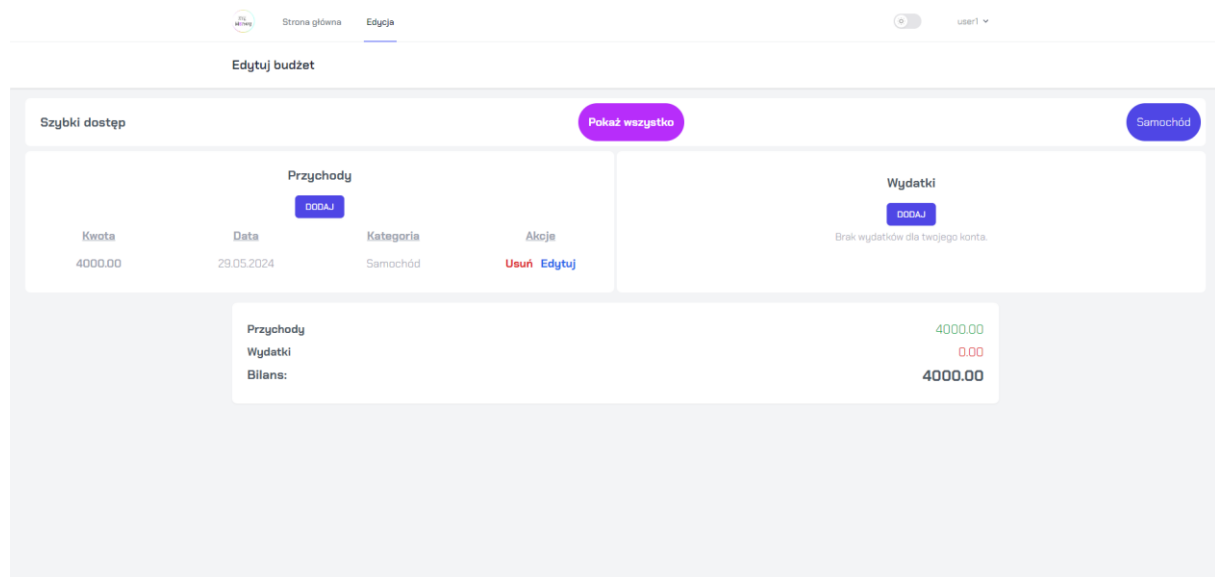
Tabela *finances* jest powiązana z tabelą *users* za pomocą klucza obcego *user\_id*, co oznacza, że każda transakcja finansowa jest przypisana do konkretnego użytkownika.

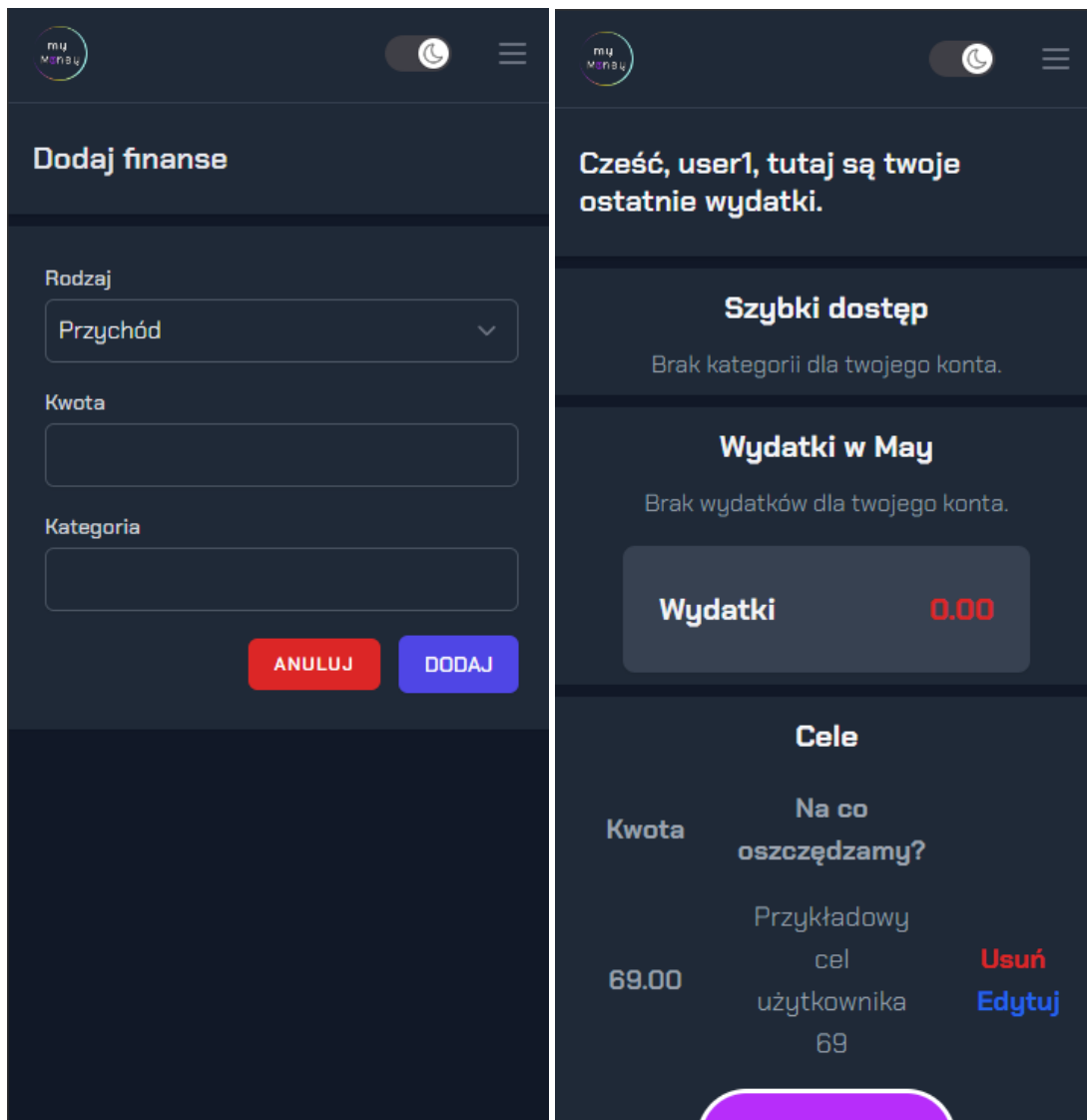
Tabela *goals* również jest powiązana z tabelą *users* za pomocą klucza obcego *user\_id*, co oznacza, że każdy cel jest przypisany do konkretnego użytkownika.

## GUI

Interfejs aplikacji "MyMoney" został zaprojektowany w sposób intuicyjny i responsywny, aby zapewnić użytkownikom wygodne korzystanie z aplikacji na różnych urządzeniach, takich jak komputery, tablety i smartfony. Oto ogólne cechy interfejsu:

- **Menu nawigacyjne:** Aplikacja posiada menu nawigacyjne umieszczone na górze lub lewym górnym rogu ekranu
- **Strona powitalna:** Na stronie głównej użytkownik ma dostęp do podstawowych funkcji aplikacji, takich jak logowanie, rejestracja oraz wyświetlanie przykładowego budżetu dla anonimowych użytkowników. Jest również logo aplikacji oraz krótki opis.
- **Strona główna:** Po zalogowaniu użytkownik ma dostęp do swojego konta, gdzie może zobaczyć powitanie z własnym imieniem, edytować swój budżet, przeglądać szybki dostęp do kategorii wydatków, tabelkę z transakcjami danego miesiąca oraz cele związane z oszczędzaniem. Użytkownik może tu także przeglądać cele związane z oszczędzaniem oraz edytować je, dodawać nowe cele, aktualizując kwotę celu.
- **Strona "Edytuj":** Ta strona umożliwia użytkownikowi zarządzanie swoim budżetem poprzez dodawanie, edycję i usuwanie transakcji finansowych, zarówno przychodów, jak i wydatków.
- **Panel Administratora:** Dostępny tylko dla użytkowników o roli administratora, ten panel umożliwia zarządzanie użytkownikami, przeglądanie ogólnodostępnych zasobów oraz pełne CRUD zarządzanie zasobami aplikacji.





## Instrukcja uruchomienia aplikacji

Instrukcja uruchomienia aplikacji na przygotowanym środowisku:

1. Pobierz kod źródłowy:
  - Pobierz kod źródłowy aplikacji "MyMoney" z repozytorium Git lub skopiuj go z lokalnego środowiska.
2. Instalacja zależności:
  - Otwórz terminal i przejdź do katalogu głównego aplikacji.
  - Uruchom polecenie `composer install` w celu zainstalowania wszystkich zależności PHP.
  - Następnie wykonaj polecenie `npm install` w celu zainstalowania zależności JavaScript.
3. Konfiguracja pliku `.env`:
  - Skopiuj plik `.env.example` i nazwij go jako `.env`.
  - W pliku `.env` skonfiguruj ustawienia bazy danych, takie jak nazwa bazy danych, nazwa użytkownika i hasło.
4. Generowanie klucza aplikacji:
  - Uruchom polecenie `php artisan key:generate`, aby wygenerować unikalny klucz aplikacji.
5. Migracja bazy danych:

- Uruchom polecenie *php artisan migrate*, aby uruchomić migracje i utworzyć strukturę bazy danych.
- 6. Uruchomienie serwera:
  - Uruchom serwer PHP za pomocą polecenia *php artisan serve*.
  - Aplikacja będzie dostępna pod adresem `http://localhost:8000`.

## Wymagane oprogramowanie

Oprogramowanie wymagane przed rozpoczęciem instalacji aplikacji:

- PHP w wersji co najmniej 8.2
- Composer
- Node.js i npm
- Serwer WWW (Apache, Nginx)
- Baza danych MySQL
- Git (opcjonalnie, jeśli korzystasz z repozytorium Git)

## Konfiguracja

- Skonfiguruj plik *.env* zgodnie z danymi dostępowymi do bazy danych i innymi ustawieniami aplikacji.

## Przygotowanie bazy danych i seedów

- Użyj polecenia *php artisan db:seed*, aby wypełnić bazę danych przykładowymi danymi, jeśli takie zostały zdefiniowane.

## Uruchomienie aplikacji

### Serwer

Aby uruchomić aplikację "MyMoney" na serwerze, należy upewnić się, że są dostępne wymagane zależności aplikacji:

#### 1. XAMPP

Skrypt automatyzujący instalację jest przygotowany pod tą aplikację i jej domyślne ścieżki.

Można skorzystać z innych serwisów jednak, należy wtedy ręcznie skonfigurować plik *.env*, oraz zainstalować potrzebne serwisy (MySQL, serwer Aplikacji webowych, PHP). W takim wypadku ręcznie trzeba zainstalować zależności za pomocą *composer'a* oraz *npm*.

Dodatkowo należy przeprowadzić ręcznie migracje, seedowanie bazy, generowanie klucza oraz linkowanie pamięci. Wszystkie komendy potrzebne do przeprowadzenia tych kroków są w plikach *.bat/.sh*.

#### 2. Composer:

Narzędzie do zarządzania zależnościami w PHP. Wymagane do instalacji zależności aplikacji.

#### 3. Node.js i npm:

Potrzebne do instalacji pakietów npm używanych w aplikacji.

#### 4. Serwer bazy danych:

Skonfiguruj dane dostępowe do bazy danych w pliku konfiguracyjnym aplikacji. Na serwerze powinna znajdować się pusta baza danych o nazwie „mymoney”



Należy pamiętać o podmienieniu danych do serwera bazy danych w pliku konfiguracyjnym .env

#### 5. Uruchomienie skryptu startowego na wybrany system operacyjny.

Mając wszystkie zależności i skonfigurowany plik .env można przejść do uruchomienia automatycznego ustawienia aplikacji przez plik .sh lub .bat w zależności od systemu.

Skrypt utworzy bazę danych w domyślnej ścieżce XAMPP, zainstaluje zależności oraz obsłuży tworzenie bazy danych.

#### Klient

Aby uruchomić aplikację "MyMoney" na kliencie poprzez przeglądarkę internetową, nie są wymagane dodatkowe zależności na poziomie klienta. Wystarczy mieć dostęp do przeglądarki internetowej.

## Funkcjonalności aplikacji

### Typy użytkowników

Pole *role\_id* rozstrzyga typ użytkownika.

*role\_id*=1 – administrator

*role\_id*=2 – użytkownik

*role\_id*=3 – użytkownik anonimowy

Domyślny administrator dodawany jest w seedzie bazy, tak samo jak użytkownik anonimowy i użytkownicy przykładowi.

#### Administrator

Admini mają dostęp do specjalnych panelów CRUD służących do zarządzania wszystkimi zasobami bazy danych.

Domyślny admin posiada hasło „P@ssw0rd”

#### Użytkownik anonimowy

Posiada przykładowy budżet wyświetlający się na stronie powitalnej. Jest on unikatowy, nie można go wyświetlać ani zmieniać jego zasobów. Nie można też dodawać użytkowników anonimowych ani zmieniać roli obecnych użytkowników na anonimów.

Użytkownik anonimowy oraz użytkownicy pokazowi mają silne hasła, praktycznie niemożliwe do odgadnięcia (na dzień 29.05.2024). Składają się z 50 losowych znaków. Nie mają służyć do logowania, tylko do kwestii poglądowych – budżetu na stronie powitalnej oraz panelu CRUD administratorów.

Nie jest przeznaczony do logowania się

#### Użytkownik

Zwykli użytkownicy posiadają *role\_id* = 2 i mogą obsługiwać podstawowe funkcje aplikacji przeznaczone dla publiczności.

## Logowanie i rejestracja

### Logowanie

#### 1. Strona logowania/rejestracji:

Użytkownik odwiedzający stronę główną aplikacji może przejść do formularza logowania, klikając na odpowiedni przycisk "Zaloguj się".

## 2. Weryfikacja danych:

Po przejściu do formularza logowania użytkownik wprowadza swój adres e-mail i hasło.

Dane logowania są weryfikowane po stronie serwera, a następnie użytkownik jest uwierzytelniany.

## 3. Błędy logowania:

W przypadku nieprawidłowego adresu e-mail lub hasła, użytkownik zostanie poinformowany o błędzie i poproszony o ponowne wprowadzenie poprawnych danych.

## 4. Sesja użytkownika:

Po poprawnym uwierzytelnieniu, użytkownik otrzymuje sesję, która pozwala mu na dostęp do chronionych zasobów aplikacji.

## Rejestracja

### 1. Strona logowania/rejestracji:

Użytkownik, który nie ma jeszcze konta, może przejść do formularza rejestracji, klikając na odpowiedni przycisk "Utwórz konto".

### 2. Wprowadzenie danych:

W formularzu rejestracji użytkownik musi wprowadzić swoje dane, takie jak login, adres e-mail oraz hasło.

### 3. Walidacja danych:

Przed zapisaniem danych nowego użytkownika do bazy danych, aplikacja dokonuje walidacji wprowadzonych informacji, np. sprawdza poprawność formatu adresu e-mail oraz sprawdza unikalność adresu e-mail.

### 4. Zapis do bazy danych:

Po pomyślnej walidacji danych, nowe konto użytkownika zostaje zapisane do bazy danych, a użytkownik otrzymuje dostęp do aplikacji. Przypisane zostaje pole `role_id = 2` oznaczające zwykłego użytkownika.

### 5. Błędy rejestracji:

W przypadku wystąpienia błędów podczas rejestracji, użytkownik zostanie poinformowany o przyczynach niepowodzenia i poproszony o ponowne wprowadzenie poprawnych danych.

## CRUD administratora

Administratorzy posiadają pełen wgląd do encji bazy danych za pomocą opcji w menu rozwijanym. Mogą przeprowadzać pełne operacje CRUD na dostępnych zasobach.

## Zarządzanie użytkownikami

W jednej z dostępnych dla administratorów zakładce znajdują się wszyscy dostępni użytkownicy. Można przeprowadzać na tych zasobach pełny CRUD – dodawać, usuwać, edytować (np. role, login, email, itd.).

## Przeglądanie ogólnodostępnych zasobów

Każdy użytkownik posiada dostęp do ogólnodostępnych zasobów po zalogowaniu się na swoje konto, gdzie może przeprowadzać operacje CRUD na:

- Finansach – wyświetlać, dodawać, usuwać oraz edytować przychody/wydatki
- Celach – wyświetlać, dodawać, usuwać oraz edytować cele

### Zarządzanie danymi przez użytkownika

Każdy użytkownik ma dostęp do panelu ustawień gdzie może zarządzać swoim loginem, mailem oraz hasłem

### Wybrana logika biznesowa

Wyświetlanie przykładowego budżetu za pomocą kontrolera użytkownika.

```
<?php
```

```
namespace App\Http\Controllers;
```

```
use Illuminate\Http\Request;
```

```
use App\Models\User;
```

```
use App\Models\Finance;
```

```
class UserController extends Controller
```

```
{
```

```
    /**
```

```
     * Display a listing of the resource.
```

```
     *
```

```
     * @return \Illuminate\Http\Response
```

```
     */
```

```
    public function index()
```

```
    {
```

```
        // Get current user
```

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

```
        // Check if the user is authenticated
```

```
        if ($user) {
```

```
            // Fetch user finances
```

```
            $finances = $user->finances;
```

```
            // Fetch user goals
```

```
            $goals = $user->goals;
```

```
            // Fetch expenses and incomes
```

```
            $expences = Finance::where('user_id', $user->id)
```

```
                ->where('type', 'outcome')
```

```
                ->orderBy('created_at', 'DESC')
```

```
                ->paginate(10);
```

```

$incomes = $finances->where('type', 'income');

// Calculate totals
$totalExpenses = $expences->sum('amount');
$totalIncomes = $incomes->sum('amount');
$balance = $totalIncomes - $totalExpenses;

// Fetch unique categories from expenses
$categories = $expences->pluck('category')->unique();

// Return view with finances and categories
return view('home', [
    'incomes' => $incomes,
    'expences' => $expences,
    'categories' => $categories,
    'goals' => $goals,
    'totalExpenses' => $totalExpenses,
    'totalIncomes' => $totalIncomes,
    'balance' => $balance
]);
} else {
    // Redirect guests
    $user = User::where("role_id", 3)->first();

    // Fetch user finances
    $finances = $user->finances;

    // Fetch user goals
    $goals = $user->goals;

    // Fetch expenses and incomes
    $expences = $finances->where('type', 'outcome');
    $incomes = $finances->where('type', 'income');

    // Calculate totals
    $totalExpenses = $expences->sum('amount');
    $totalIncomes = $incomes->sum('amount');
    $balance = $totalIncomes - $totalExpenses;

    // Fetch unique categories from expenses
    $categories = $expences->pluck('category')->unique();

```

```

        // Return view with finances and categories
        return view('welcome', [
            'incomes' => $incomes,
            'expences' => $expences,
            'categories' => $categories,
            'goals' => $goals,
            'totalExpenses' => $totalExpenses,
            'totalIncomes' => $totalIncomes,
            'balance' => $balance
        ]);
    }
}
}

```

Funkcja index z wybranego kontrolera wywoływana jest na stronie powitalnej przez plik web.php. W zależności od tego czy użytkownik jest zalogowany wyświetla ona stronę powitalną dla gości z przykładowym budżetem. Dla zalogowanych użytkowników następuje przekierowanie do strony głównej aplikacji z przekazaniem danych o finansach obecnie zalogowanego użytkownika.

Działanie ścieżek dla powyższego kodu:

```

Route::get('/', [UserController::class, 'index']);
Route::get('/home', [UserController::class, 'index'])->
    middleware(['auth', 'verified'])->name('home');

```

## Opis walidacji danych

### Walidacja back-end

W aplikacji została zaimplementowana walidacja danych w kontrolerach, co zapewnia poprawność i kompletność danych wprowadzanych przez użytkowników. W każdym kontrolerze, gdzie istnieje konieczność walidacji danych, zastosowano podobny wzorzec działania. Poniżej przedstawiono ogólny opis zrealizowanej walidacji:

#### 1. Sprawdzenie poprawności danych:

W kontrolerach wykorzystano mechanizm walidacji dostępny w Laravelu, który umożliwia sprawdzenie poprawności przesyłanych danych.

Walidacja obejmuje weryfikację obecności oraz poprawności wartości przesłanych przez użytkownika, łącznie z ustawieniem minimum w danych polach formularzy.

#### 2. Wymagane pola:

Walidacja obejmuje sprawdzenie obecności wszystkich wymaganych pól w danych przesłanych przez użytkownika.

### 3. Poprawność danych:

Sprawdzana jest poprawność danych, takich jak numeryczność, wartości logiczne (np. income lub outcome dla typu finansów), a także długość ciągów znaków.

### 4. Zabezpieczenie przed błędnymi danymi:

Walidacja zapobiega wprowadzeniu do systemu błędnych lub niekompletnych danych, co zapewnia integralność danych oraz bezpieczeństwo aplikacji.

### 5. Zastosowanie uniwersalnego wzorca:

Dzięki zastosowaniu podobnego wzorca walidacji w kontrolerach, proces weryfikacji danych jest jednolity i spójny w całej aplikacji. Pola przesyłane do bazy np. w *AdminController::store()* są sprawdzane na podstawie ich typu, za pomocą wymienionej funkcji *validate()*:

```
$request->validate([
    'login' => 'required|string|max:255',
    'email' => 'required|string|email|max:255|unique:users',
    'password' => 'required|string|min:5',
    'role_id' => 'required|in:1,2,3',
]);
```

Jak można zauważyć pola mają ograniczenie typu. Pole email jest sprawdzane wg. Ogólnego wzorca email. Rola musi być jedną z trzech wybranych liczb.

Walidacja w pozostałych kontrolerach odbywa się analogicznie, w zależności od potrzebnych pól.

Dodatkowo przekazywane są informacje o statusie danej operacji, umożliwiając użytkownikowi dokładny wgląd w obecny status.



## Użytkownicy

POMYŚLNIE DODANO UŻYTKOWNIKA.

DODAJ

ID	Login	Email	Rola	Akcje
2	user1	user1@example.com	Użytkownik	<a href="#">Usuń</a> <a href="#">Edytuj</a>
3	user2	user2@example.com	Użytkownik	<a href="#">Usuń</a> <a href="#">Edytuj</a>
4	user3	user3@example.com	Użytkownik	<a href="#">Usuń</a> <a href="#">Edytuj</a>
5	user4	user4@example.com	Użytkownik	<a href="#">Usuń</a> <a href="#">Edytuj</a>
6	user5	user5@example.com	Użytkownik	<a href="#">Usuń</a> <a href="#">Edytuj</a>
7	user6	user6@example.com	Użytkownik	<a href="#">Usuń</a> <a href="#">Edytuj</a>
9	essa	mateusz.bocak9@gmail.com	Admin	<a href="#">Usuń</a> <a href="#">Edytuj</a>
14	test2137	test2137@example	Użytkownik	<a href="#">Usuń</a> <a href="#">Edytuj</a>



## Dodaj użytkownika

The email has already been taken.

Login

Email


Hasło


Rola



ANULUJ

STWÓRZ

Strona głównaEdycja

admin

### Edytuj Finanse

Ups! Coś poszło nie tak.  
The selected user id is invalid.

Kwota

20,00

Kategoria

Zabawa

Typ

Przychód


ID użytkownika


420

The selected user id is invalid.

ANULUJ

ZAPISZ

Strona głównaEdycja

admin

### Dodaj Finanse

The selected user id is invalid.

Kwota

4000

Typ

Przychód

Kategoria

Wypłata

ID właściciela

2115

The selected user id is invalid.

ANULUJ

DODAJ

## Walidacja front-end

W aplikacji zaimplementowano walidację danych również po stronie klienta, co pozwala na szybką informację zwrotną dla użytkownika dotyczącą poprawności wprowadzonych danych. Poniżej przedstawiono ogólny opis walidacji frontendowej:

### 1. Obsługa błędów:

W przypadku błędnych danych przesłanych przez użytkownika, aplikacja wyświetla komunikaty informujące o popełnionych błędach.



Komunikaty są wyświetlane powyżej formularza, aby użytkownik mógł natychmiast zidentyfikować i poprawić błędy.

## 2. Zastosowanie wymaganych pól:

Formularz zawiera atrybuty required dla pól, które muszą być wypełnione przez użytkownika przed wysłaniem formularza.

Dzięki temu użytkownik jest zobligowany do wypełnienia wszystkich wymaganych pól, co zapewnia kompletność danych.

## 3. Walidacja typu danych:

Dla pól takich jak adres email czy hasło, zastosowano odpowiednie typy danych (np. type="email"), aby przeglądarka mogła automatycznie sprawdzić ich poprawność.

## 4. Stosowanie komunikatów błędów:

Komunikaty błędów są dynamicznie generowane w oparciu o błędy zwrócone przez serwer lub walidację przeglądarki.

Dzięki temu użytkownik otrzymuje klarowne informacje dotyczące błędów, co ułatwia szybką korektę danych.

## 5. Interaktywność formularza:

Formularz umożliwia natychmiastową reakcję na błędy, co pozwala użytkownikowi na ich korektę bez potrzeby odświeżania strony.

Po poprawnym wypełnieniu formularza, użytkownik może go wysłać, co uruchamia proces przetwarzania danych po stronie serwera.

Przykład:

```
<div class="mb-4">
  <label class="formLabel" for="amount">
    Kwota
  </label>
  <input required type="number" name="amount" id="amount"
step="0.01" min="0.01" value="{{ old('amount') }}"
class="formField">
</div>
```

Pola formularza mają ograniczenia – minimum, niektóre mają atrybut „required” nakazując użytkownikowi ich wypełnienie.



## Użytkownicy

[DODAJ](#)


ID	Login	Email	Rola	Akcje
2	user1	user1@example.com	Użytkownik	<a href="#">Usuń</a> <a href="#">Edytuj</a>
3	user2	user2@example.com	Użytkownik	<a href="#">Usuń</a> <a href="#">Edytuj</a>
4	user3	user3@example.com	Użytkownik	<a href="#">Usuń</a> <a href="#">Edytuj</a>
5	user4	user4@example.com	Użytkownik	<a href="#">Usuń</a> <a href="#">Edytuj</a>
6	user5	user5@example.com	Użytkownik	<a href="#">Usuń</a> <a href="#">Edytuj</a>
7	user6	user6@example.com	Użytkownik	<a href="#">Usuń</a> <a href="#">Edytuj</a>
9	essa	mateusz.bocak9@gmail.com	Admin	<a href="#">Usuń</a> <a href="#">Edytuj</a>



## Dodaj użytkownika

Login

Email

 Wypełnij to pole.

Hasło

Rola

Użytkownik ▾

[ANULUJ](#)[STWÓRZ](#)



## Dodaj użytkownika

Login

Email

Hasło

Rola



Uwzględnij znak „@” w adresie e-mail. W adresie „test2137example” brakuje znaku „@”.

ANULUJ

STWÓRZ