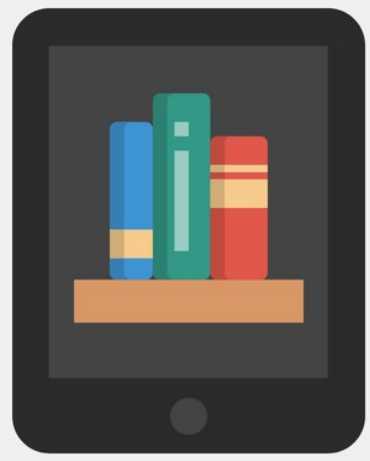
PRAKTYCZNE PHP

Zacznij przygodę z PHP tworząc praktyczny projekt od zera



Marcin Wesel
kursphp.com

PRZYKŁADOWY ROZDZIAŁ

KOMUNIKATY DLA UŻYTKOWNIKA I ROLE UŻYTKOWNIKÓW

KOMUNIKATY DLA UŻYTKOWNIKA

W większości aplikacji potrzebujesz mieć możliwość powiadomienia użytkownika o rezultacie wykonywanej akcji. Przykładowo, akcją taką może być zakładanie nowego konta na stronie aplikacji. Odwiedzający musi wiedzieć, czy rejestracja zakończyła się powodzeniem. Być może chcesz go poprosić, by odwiedził swoją skrzynkę mailową i kliknął w link w nowo otrzymanej wiadomości.

Sukcesy i prośby to jeden typ wiadomości. Drugim, ważnym rodzajem komunikatów są błędy. Błędy mogą wynikać z walidacji wprowadzonych danych lub z nieprawidłowości samego systemu. Walidacja, czyli sprawdzanie informacji wpisanych przez użytkownika pod kątem formatu, długości, dostępności nazw itp. (np. adres email jest błędnego formatu). Błędy systemu i jego dostępności to przykładowo: serwer SMTP nie odpowiada, padł serwer aplikacji lub sieć jest przeciążona.

Błędów może pojawić się cała masa, które często wychodzą długo po wypuszczeniu aplikacji w świat. Przykładowo, aplikacja może tak rosnąć, że skończy nam się dostępne miejsce na dysku (lub dobijemy do dopuszczalnego rozmiaru bazy danych). Wtedy wysyłanie wiadomości również zakończy się niepowodzeniem. Ważne, z punktu widzenia webmastera, byś był stale świadomym tego, co dzieje się na serwerze i w jakim stanie jest aplikacja.

Jeśli chodzi o komunikaty, zazwyczaj możemy podzielić je na cztery podstawowe typy:

Success Message:
powiadomienie o pomyślnym wykonaniu akcji (np. logowania lub rejestracji).

Info Message:
informacja o charakterze neutralnym (np. planowana niedostępność systemu w najbliższych godzinach lub kończący się abonament).

Warning Message:
ostrzeżenie dla użytkownika (np. za 60 sekund wygaśnie sesja).

Error Message:
informacja o błędzie, zazwyczaj podczas przetwarzania jakiejś akcji (błąd w logowaniu, brak użytkownika, błędy walidacji danych etc.)

W projektowanej przez nas aplikacji skorzystasz z dwóch typów powiadomień - sukcesu i błędu. Posłużysz się jednym, spójnym mechanizmem, by obsłużyć oba typy. Mianowicie, użyjesz sesji.

W zmiennej superglobalnej **$_SESSION** są trzymane wszystkie zmienne sesyjne (aktualne dla obecnej sesji użytkownika). Sesję możemy rozumieć jako pojedyncze odwiedziny użytkownika. Standardowa sesja trwa 24 minuty. Po tym czasie, gdy użytkownik nie odświeży strony, tworzona jest nowa sesja, a wraz z nią wszystkie zmienne sesyjne.

To było szybkie przypomnienie tego, jak działa sesja. Teraz napiszmy sobie własną klasę **Messages**, by nie używać stale zmiennej **$_SESSION**, a mieć swój własny dostęp do zmiennych sesyjnych. Dopiszemy do niej trzy ogólnodostępne metody statyczne, które będą dostępne z każdego miejsca w naszej aplikacji.

Pierwszymi dwoma metodami są **setSuccess($message)**, i **setError($message),** które będą dodawały komunikaty odpowiedniego typu do naszych zmiennych sesyjnych. Trzecią jest f**lashMessages()**, która zaprezentuje wszystkie komunikaty zebrane w zbiorze komunikatów i zaprezentuje je użytkownikowi. Następnie zbiór komunikatów zostanie wyczyszczony, by nie pokazywać tej samej wiadomości wielokrotnie temu samemu użytkownikowi.

Z racji, że korzystamy z mechanizmu sesji, nie ma ryzyka, że komunikat przeznaczony dla jednego użytkownika zostanie omyłkowo zaprezentowany innemu odwiedzającemu. Dla każdego z odwiedzających tworzona jest osobna sesja na serwerze i tylko do tego obiektu sesji są dodawane zbiory komunikatów.

Korzystamy ze zmiennej superglobalnej **$_SESSION**, gdzie dodajemy własne zdefiniowane indeksy:

```
$_SESSION["message"]["success"]

$_SESSION["message"]["error"]
```

Teraz, gdy korzystamy z metody **flashMessages()**, metoda ta przeczesuje tablicę **["message"]** i sprawdza, czy istnieją indeksy **success** i **error**. Jeśli tak, wyświetla ich

zawartość w odpowiedni dla nich sposób. Dla **success** będzie to zielona belka, dla **error** będzie to belka czerwona.

To, jak wygląda komunikat, zależy ściśle od tego, z którego szablonu HTML skorzystamy do tworzenia własnej aplikacji webowej. Może się okazać, że twórca wcale nie przewidział takich komunikatów (co zdarza się często w przypadku darmowych szablonów) i będziemy musieli sami zatroszczyć się o zaprojektowanie ich wyglądu.

Kod klasy Messages:

```
<?php

namespace Wesel\Shortener;

class Messages
{
    public static function setError($message)
    {
        $_SESSION['messages']['error'] = $message;
    }

    public static function setSuccess($message)
    {
        $_SESSION['messages']['success'] = $message;
    }

    public static function flashMessages()
    {
        if (isset($_SESSION['messages']['error']))
        {
            echo '<div class="row"><div class="alert alert-danger alert-
dismissable"><button type="button" class="close" data-dismiss="alert" aria-
hidden="true">&times;</button>'.$_SESSION['messages']['error'].'</div></div>';
            unset($_SESSION['messages']['error']);
        }

        if (isset($_SESSION['messages']['success']))
        {
            echo '<div class="row"><div class="alert alert-success alert-
dismissable"><button type="button" class="close" data-dismiss="alert" aria-
hidden="true">&times;</button>'.$_SESSION['messages']['success'].'</div></div>';
            unset($_SESSION['messages']['success']);
        }
    }
}
```

Przeanalizujmy sobie klasę Messages.

Klasa Messages, jak nasze pozostałe klasy, znalazła się w przestrzeni nazw **\Wesel\Shortener.** Dzięki temu, dodając inny komponent zawierający klasę Messages, nie napotkamy konfliktu nazewnictwa. Nie używając namespace'ów (przestrzeni nazw) moglibyśmy mieć tylko jedną klasę o konkretnej nazwie w całym naszym projekcie.

Metody **setError** i **setSuccess** przyjmują jako argument jeden parametr **$message.** Przy jego pomocy będziemy przekazywać wiadomość do wyświetlenia czy to błędu, czy pozytywnego rezultatu. Wewnątrz tych metod odwołujemy się do zmiennej superglobalnej **$_SESSION.** W niej korzystamy z tablicy **['messages'],** do której dodajemy wiadomości w indeksach **['error']** oraz **['success'].**

Metoda **flashMessages()** wyświetla wiadomości z indeksów **['error']** oraz **['success']**, po czym, korzystając z wbudowanego polecenia **unset,** usuwamy indeksy z tablicy **$_SESSION['messages'].** Teraz użyjesz metody **flashMessages()** w klasie **Page**, która jest odpowiedzialna za wyświetlenie górnej części strony. Dzięki temu, każda strona renderowana przy pomocy klasy **Page** będzie wyświetlać zawarte w sesji komunikaty. Dodatkowo, będzie to robić w sposób spójny z pozostałymi stronami.

ROLE UŻYTKOWNIKÓW

W niemal każdej rozbudowanej aplikacji potrzebujemy rozróżnić typy użytkowników. W zależności od typu, musimy nadać użytkownikowi odpowiednie uprawnienia do wykonywania danych akcji.

Typów może być wiele. Przykładowo, możemy mieć:

- zwykłego użytkownika,
- użytkownika, który opłaca abonament,
- moderatora,
- administratora.

Każdy z nich będzie widział nieco inną zawartość aplikacji niż pozostali. Będzie miał dostęp do dodatkowych funkcji i akcji. W jaki sposób osiągnąć taki efekt w programowaniu?

Zazwyczaj stosuje się do tego mechanizm ról.

JAK TO DZIAŁA?

Każdy profil użytkownika (UserID) będzie miał odniesienie do tabeli z rolami. Tabelę nazwiemy **role**. Następnie, podczas logowania, pobierzemy z tabeli **role** odpowiadającą użytkownikowi rolę. Na jej bazie będziemy mogli prezentować mu dostępne opcje.

Jakie role będą nam potrzebne w systemie do skracania linków?

Tak naprawdę tylko dwie:

- regular_user,
- admin.

Moglibyśmy to ograć bardzo łatwo, dodając do tabeli **user** pole "**isAdmin**". Wtedy, ustawialibyśmy 0 (jeśli nie jest adminem) lub 1 (jeśli jest). Jednak, by sprawić, że system będzie bardziej elastyczny na przyszłość, podejdziemy książkowo do tematu. Stworzymy osobą tabelę z rolami, a tabelę **user** wzbogacimy o odnośnik do **RoleID**.

Tabela **role** będzie zawierała tylko dwie kolumny:

- ID
- Name

Do tabeli **user** dodajemy kolumnę **RoleID**, która jest kluczem obcym dla tabeli **role**.

Teraz, w pobieranym obiekcie **user**, po zalogowaniu, będziemy posiadać również jego **RoleID**, które determinuje nam rolę użytkownika.

Dodajmy do naszego modelu **User.php** dodatkową metodę, która sprawdzi, czy użytkownik jest adminem. Metoda **isAdmin()** będzie zwracała **true**, jeśli **RoleID** jest 1, w przeciwnym wypadku **false**. Dzięki temu, obiekt zalogowanego usera będzie w łatwy sposób przedstawiał się czy ma uprawnienia administratora.

Wystarczy, że użyjemy **if (SESSION['user']->IsAdmin())** we wszystkich funkcjach wymagających uprawnień administratora, by łatwo zdecydować, czy może ją wykonać akurat ten użytkownik. Tyczy się to zarówno logiki aplikacji, jak i samych widoków (wyświetlanie danego przycisku tylko adminowi).

Dopiszmy teraz metodę **getUsers**, gdzie sprawdzimy czy user jest adminem. Metoda posłuży nam do wyświetlenia listy userów na osobnej stronie, gdzie admin będzie mógł każdego z userów zablokować:

```
private function getUsers() {
        if (empty($_SESSION['user']) || !$_SESSION['user']->isAdmin())
        {
            Messages::setError("Błąd autoryzacji");
            header("Location:https://" . ROOT_APP_URL . "/loginForm");
            return;
        }

        $db = new DB();
        $db->query("SELECT * FROM user;");
```

```
        $results = $db->resultSet();
        return $results;
    }
```

Patrząc na powyższy kod, przejdźmy sobie po nim linijka po linijce:

1. Sprawdzamy, czy zmienna **$_SESSION['user']** (ta, gdzie zapisujemy obiekt usera po zalogowaniu) jest pusta lub (jeśli nie jest) sprawdzamy, czy metoda isAdmin zwraca false (user nie jest adminem)
2. Jeśli warunek jest SPEŁNIONY:
   - ustawiamy komunikat dla użytkownika o błędzie autoryzacji,
   - przekierowujemy na stronę logowania,
   - wychodzimy z funkcji.
3. Tworzymy nowy obiekt połączenia z bazą danych i pobieramy kompletną listę userów.
4. Zwracamy listę userów.

Kluczowym tutaj jest krok pierwszy. To on odnosi się bezpośrednio do uprawnień i ról użytkownika. Jeśli zalogowany użytkownik nie jest adminem, wyrzuci go do strony logowania z odpowiednim komunikatem.

Tym sposobem możesz już rozróżnić typ zalogowanego użytkownika i pisać metody wymagające wyższych uprawnień do wykonania. W kolejnym rozdziale napiszesz widok z listą użytkowników, którym będzie można zablokować dostęp. Będzie to widok dostępny wyłącznie dla administratora.