Fundusze Europejskie
Wiedza Edukacja Rozwój

Politechnika Wrocławska

Unia Europejska
Europejski Fundusz Społeczny

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Politechnika Wrocławska

Wydział Informatyki i Zarządzania

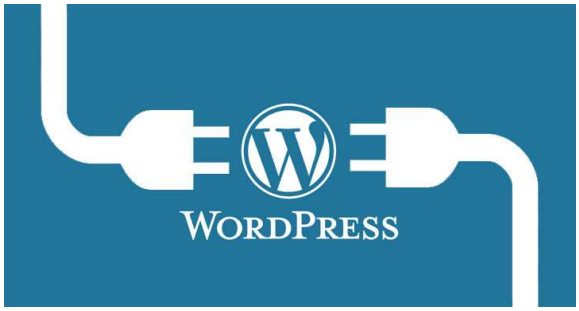
Zaawansowane Technologie Webowe

Laboratorium

Temat:	Implementacja pluginu dla WordPress
Opracował:	mgr inż. Piotr Jóźwiak
Data:	lipiec 2020
Liczba godzin:	2 godziny

Table of Contents

Wstęp



W kolejnym laboratorium zapoznamy się ze sposobem rozszerzenia standardowej funkcjonalności systemu WordPress w oparciu o własną wtyczkę (ang. *plugin*). Najważniejszym powodem do stworzenia własnego pluginu jest to, że pozwala on oddzielić własny kod od kodu podstawowego WordPress. Jeśli coś pójdzie nie tak z naszą wtyczką, reszta strony będzie na ogół wciąż funkcjonować poprawnie. Zmiana plików źródłowych WordPressa jest chyba najgorszą możliwością jaką można zrobić. Poza możliwością uszkodzenia naszego systemu odcinamy się tym sposobem od możliwości aktualizacji systemu do nowej wersji, gdyż aktualizacja automatycznie wymaże naszą zmianę w kodzie.

Pluginy są zwykłymi skryptami PHP. Jak się przekonamy implementacja nowej funkcjonalności do WP wcale nie jest tak trudnym procesem. Oczywiście wymagana jest choćby podstawowa wiedza o języku PHP jak i samym funkcjonowaniu systemu WordPress. W tym laboratorium zakładam, że Student posiada już podstawową wiedzę z zakresu PHP. Natomiast reszta niezbędnych wiadomości zostanie opisana w poniższym materiale.

Wprowadzenie do Pluginów WordPressa

Wtyczka WordPress to samodzielny zestaw kodu, który ulepsza i rozszerza funkcjonalność WordPress. Korzystając z dowolnej kombinacji PHP, HTML, CSS, JavaScript / jQuery, wtyczka może dodawać nowe funkcje do dowolnej części witryny, w tym do administracyjnego panelu sterowania. Możemy zmodyfikować domyślne zachowanie WordPressa lub całkowicie usunąć niechciane zachowanie. Wtyczki pozwalają łatwo dostosować i spersonalizować WordPress do własnych potrzeb.

Ponieważ wtyczki WordPress są samodzielne, nie zmieniają fizycznie żadnego z podstawowych plików źródłowych WordPress. Można je skopiować i zainstalować w dowolnej instalacji WordPress. Alternatywnym (i zdecydowanie odradzanym) sposobem wprowadzania zmian w WordPress jest zapisywanie nowych funkcji w pliku WordPress **`functions.php`**, który jest przechowywany w folderze **`/wp-include/`** lub pliku **`functions.php`**, który jest częścią motywu. Wiąże się to z wieloma potencjalnymi problemami.

WordPress i jego motywy są regularnie aktualizowane. Jeśli nie korzystamy z motywu podrzędnego (ang. *child*) WordPress, gdy **functions.php** zostanie nadpisana przez aktualizację, nowy kod zostanie usunięty i będziemy musieli napisać go od nowa. Jeśli piszemy wiele funkcji, a jedna z nich zawiera błąd, którego nie można debugować, może być konieczne zastąpienie całego pliku oryginalnym, usuwając wszystkie wprowadzone zmiany. Jeśli nasze funkcje zostaną usunięte z pliku, nasza strona będzie generować błędy PHP informujące o brakujących funkcjach.

Wtyczki nigdy nie są automatycznie nadpisywane ani usuwane podczas instalowania aktualizacji WordPress. Jeśli we wtyczce występują błędy kodowania, zwykle można ją po prostu dezaktywować w administracyjnym panelu sterowania podczas naprawy. Jeśli we wtyczce wystąpi poważny błąd, WordPress czasami automatycznie ją dezaktywuje, umożliwiając dalsze działanie witryny.

Czym są WordPress Hooks?

Wtyczki WordPressa współdziałają z podstawowym kodem za pomocą tzw. **Hooks**, czyli pewnych punktów zaczepienia. Istnieją dwa typy hooks.

1. **Action hooks** (wykorzystywane do dodawania/usuwania funkcji)
2. **Filter hooks** (wykorzystywane do modyfikowania danych zwracanych przez funkcje)

Action Hooks

Gdy użytkownik odwiedza dowolną witrynę opartą o WordPress, to w tle serwer wykonuje w różnych punktach wywoływania serię funkcji PHP (nazywanych **actions**), które są dołączone do różnych punktów zaczepienia zwanych **Action Hooks**. Wykorzystując Action Hooks dostarczone przez WordPressa, można dodać własne funkcje do listy akcji, które są uruchamiane po wywołaniu pewnego zdarzenia, a także można usunąć istniejące funkcje z dowolnego miejsca. Action Hooks określają, kiedy akcje są wywoływane w procesie generowania kodu HTML strony. Np. przed zamykającym tagiem **</head>** wywoływana jest Action Hook o nazwie **wp_head()**, która to wywołuje po koleji wszystkie akcje zaczepione do **wp_head()**.

Action Hooks są kontekstowe - niektóre są wywoływane na każdej stronie naszej witryny, inne są wywoływane tylko podczas przeglądania Administracyjnego panelu sterowania i tak dalej. Pełna lista dostępnych Action Hooks wraz z opisem kontekstu, w którym są one wywoływane, można znaleźć na stronie Plugin API/Action Reference.

Przypięcie własnej funkcji Action do Action Hook

Aby przypiąć swoją funkcję do wybranego punktu zaczepienia, plugin musi wykonać funkcję silnika WordPress o nazwie **add_action()**. Funckja ta oczekuje dwóch parametrów. Przeanalizujmy poniższy przykład, aby poznać zasadę działania tej funkcjonalności:

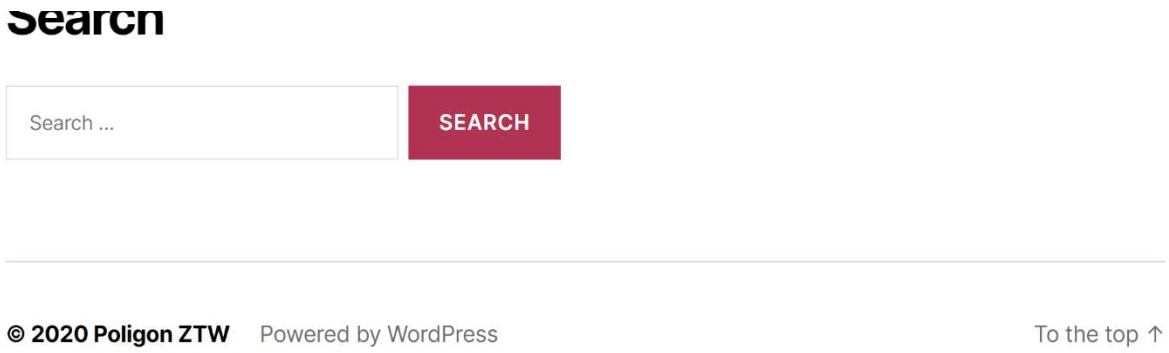
```
// Hook to the 'init' action, which is called after WordPress is finished loading
 the core code
add_action( 'init', 'add_Cookie' );

// Set a cookie with the current time of day
function add_Cookie() {
  setcookie("last_visit_time", date("r"), time()+60*60*24*30, "/");
}
```

- Pierwszym wymaganym parametrem jest nazwa Action Hook, do którego chcemy się podłączyć
- Drugi wymagany parametr to nazwa funkcji, którą chcemy uruchomić
- Trzeci parametr (opcjonalny) jest priorytetem funkcji, którą chcemy uruchomić. Możemy podłączyć dowolną liczbę różnych funkcji do tej samej Action Hook. Parametr ten pozwala uszeregować akcje w żądanej kolejności. Domyślny priorytet to 10, umieszczając niestandardową funkcję po dowolnej z wbudowanych funkcji WordPress.
- Czwarty parametr (opcjonalny) to liczba argumentów funkcji. Domyślna wartość to 1.

Kolejny przykład ma za zadanie wyświetlać pewien tekst po wygenerowaniu standardowej stopki strony. W tym celu przypniemy naszą akcję do Action Hook o nazwie **wp_footer()**. Spowoduje to wywołanie naszej funkcji za każdym razem, gdy generator WordPressa będzie na chwilę przed tagiem zamykającym **</body>**. Przypniemy funkcję o nazwie **mfp_Add_Text()**. Pytanie tylko gdzie wpisać nasz kod? Na tę chwilę dla uproszczenia zmodyfikujemy aktualny szablon. Zmiany wykonamy we wspomnianym już pliku **functions.php**. Do zrobienia tego przykładu nie jest wymagane nawet zaglądanie w pliki źródłowe zapisane na serwerze WWW. Wykorzystamy edytor wbudowany w panel administracyjny WordPressa. Aby go otworzyć przejdźmy do panelu administratora. Następnie z menu bocznego wybierzmy **Appearance** -> **Theme Editor**. Po ostrzeżeniu o konsekwencjach modyfikacji ukaże się edytor kodu źródłowego aktualnego motywu. Po prawej stronie wyszukajmy plik o nazwie **functions.php** i wybierzmy go do edycji. Na samej górze tego pliku wstawmy poniższy kod:

```
// Define 'mfp_Add_Text'
function mfp_Add_Text()
{
  echo "<p style='color: red;'>Ten tekst wyświetla się po wygenerowaniu stopki st
rony</p>";
}

// Hook the 'wp_footer' action hook, add the function named 'mfp_Add_Text' to it
add_action("wp_footer", "mfp_Add_Text");
```

Zapiszmy zmiany i przejdźmy do widoku strony. Tekst został wstawiony po wygenerowaniu stopki:



Ten tekst wyświetla się po wygenerowaniu stopki strony

Usunięcie funkcji Action z Action Hook

Aby usunąć akcję z Action Hoop, musimy napisać nową funkcję, która wywoła funkcję **remove_action()**. Ta nowa funkcja najpierw oczywiście musi zostać zarejestrowana. Wyjaśni to poniższy przykład.

```
// Hook the 'init' action, which is called after WordPress is finished loading th
e core code, add the function 'remove_My_Meta_Tags'
add_action( 'init', 'remove_My_Meta_Tags' );
```

```
// Remove the 'add_My_Meta_Tags' function from the wp_head action hook
function remove_My_Meta_Tags()
{
  remove_action( 'wp_head', 'add_My_Meta_Tags');
}
```

Funkcja **remove_action()** oczekuje przynajmniej dwóch parametrów.

- Pierwszym wymaganym parametrem jest nazwa Action Hook, do którego zaczepiona jest funkcja
- Drugi wymagany parametr to nazwa funkcji, którą chcemy usunąć
- Trzeci parametr (opcjonalny) jest priorytetem oryginalnej funkcji. Ten parametr musi być identyczny z priorytetem, który został pierwotnie zdefiniowany podczas dodawania akcji do zaczepu akcji. Jeśli nie zdefiniowaliśmy priorytetu dla swojej funkcji, nie podawajmy tego parametru.

Zatem rozwińmy nasz przykład z dodanym tekstem w stopce. Wyobraźmy sobie, że nie chcemy z jakiś powodów, aby ten tekst wyświetlał się w poniedziałki. Naturalnie najprostszym sposobem byłoby dodanie odpowiedniego wyrażenia **if** w ciele naszej funkcji Action. Jednakże chcemy przekonać się jak działa usuwanie Action. Dlatego nasz przykład będzie usuwał akcję w poniedziałki. Przeanalizuj poniższy kod:

```
// Define 'mfp_Add_Text'
function mfp_Add_Text()
{
  echo "<p style='color: red;'>Ten tekst wyświetla się po wygenerowaniu stopki st
rony</p>";
}

// Hook the 'wp_footer' action hook, add the function named 'mfp_Add_Text' to it
add_action("wp_footer", "mfp_Add_Text");

// Define the function named 'mfp_Remove_TextOnMondays()' to remove our previous
function from the 'wp_footer' action
function mfp_Remove_TextOnMondays()
{
  if (date("l") === "Monday") {
    // Target the 'wp_footer' action, remove the 'mfp_Add_Text' function from it
    remove_action("wp_footer", "mfp_Add_Text");
  }
}

// Hook the 'wp_head' action, run the function named 'mfp_Remove_Text()'
add_action("wp_head", "mfp_Remove_TextOnMondays");
```

W powyższym przykładzie zarejestrowaliśmy akcję w **wp_head**. Wewnątrz tej akcji jeśli dzisiejszy dzień jest poniedziałkiem wykonujemy usunięcie innej akcji, dodającej napis pod stopką. Pytanie brzmi, dlaczego

naszą akcję obsługującą usuwanie innej akcji wstawiliśmy do **wp_head** zamiast do **wp_footer**? Odpowiedź jest dość oczywista. Akcja **wp_head** jest wykonywana wcześniej niż akcje **wp_footer**. Zatem musimy zapewnić usunięcie danej akcji, zanim się ona wykona. W taki oto sposób dokonaliśmy tego uszeregowania.

Filtry oraz Filter Hooks

Funkcja filtrująca pozwala modyfikować dane wynikowe zwracane przez istniejące funkcje i musi być podpięta do jednego z **Filter Hooks** – podobnie jak było to w przypadku Action Hooks. Filter Hooks zachowują się podobnie do Action Hooks, ponieważ są wywoływane w różnych punktach skryptu oraz także są kontekstowe. Pełna lista punktów filtrujących oraz kontekst, w którym są one wywoływane jest opisana w **WordPress Plugin API/Filter Reference page**.

Przypinanie Filtrów

Aby dodać funkcję filtrującą do dowolnego Filter Hook, wtyczka musi wywołać funkcję WordPress o nazwie **add_filter()**, z co najmniej dwoma parametrami. Przeanalizuj poniższy przykład:

```
// Hook the 'the_content' filter hook (content of any post), run the function nam
ed 'mfp_Fix_Text_Spacing'
add_filter("the_content", "mfp_Fix_Text_Spacing");

// Automatically correct double spaces from any post
function mfp_Fix_Text_Spacing($the_Post)
{
 $the_New_Post = str_replace("  ", " ", $the_Post);

 return $the_New_Post;
}
```

- Pierwszym wymaganym parametrem jest nazwa Filter Hook
- Drugim wymaganym parametrem jest nazwa funkcji filtrowania, którą chcemy uruchomić
- Trzeci parametr (opcjonalny) jest priorytetem funkcji, którą chcemy uruchomić.
- Czwarty parametr (opcjonalny) to liczba argumentów, co oznacza, ile parametrów może przyjąć nasza funkcja filtru. Domyślna wartość to 1.

Implementacja Plugina

Skoro poznaliśmy już podstawy Hooks w WordPress to możemy przejść do omówienia zasad tworzenia pluginu. Sposób implementacji rozszerzeń WP omówimy sobie na podstawie przykładowego plugina. Prześledzimy proces jego powstawania.

Podstawowym działaniem naszego plugina będzie zbadanie czy dany post jest nowym postem (opublikowanym nie dalej jak X dni temu). I dla takich nowych postów plugin będzie generował odpowiednią etykietę przy tytule postu. Plugin będzie posiadał także prosty panel administracyjny, w którym będziemy określać co to znaczy, że post jest nowy – inaczej jak długo post jest traktowany jako nowy.

Krok 1 – nazwa pluginu

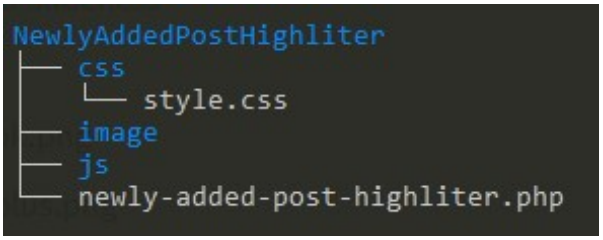
Zaczynamy od wymyślenia nazy dla naszego pluginu. Nazwa pluginu ma niewielkie znaczenie gdyż jedynymi odbiorcami będziemy my sami. W takiej sytuacji należy zadbać, aby plugin nie kolidował z zainstalowanymi już rozszerzeniami. Jednakże jeśli mamy w planach opublikowanie naszego pluginu w publicznym repozytorium WordPressa, to nazwa powinna być unikalna. Najlepiej sprawdzić unikalność nazwy w wyszukiwarce tutaj: https://wordpress.org/plugins/.

W naszym przypadku będziemy wykorzystywać nazwę **Newly Added Post Highliter**.

Krok 2 – przygotowanie struktury folderów pluginu

Plugin jak już wspominaliśmy to zestaw plików PHP, CSS, JavaScript i innych resourców. W najprostszym podejściu plugin może być pojedyńczym plikiem PHP. Nazwa tego pliku powinna korespondować z nazwą pluginu. Całość najlepiej jest zamknąć w jednym folderze nadrzędnym o nazwie naszego pluginu.

Wszystkie pluginy są umieszczone w folderze **`wp-content/plugins/`**. Zatem przygotujmy naszą strukturę folderów pluginu w poniższy sposób:



Jest to oczywiście tylko pewna propozycja. Nie musimy ściśle trzymać się takiej hierarchii, jeśli inna będzie bardziej odpowiadać naszym potrzebom, to spokojnie można to zmienić. W powyższym przykładzie umieściłem także od razu dwa pliki. Pierwszy **`newly-added-post-highliter.php`** będzie głównym plikiem naszego plugina. Natomiast plik **`css/style.css`** jak sama nazwa wskazuje, będzie definiował szatę graficzną.

Krok 3 – meta informacje o pluginie

Opis naszego pluginu umieszczamy w główny pliku PHP jako komentarz. Wymaga on użycia określonego formatu. Na podstawie tego komentarza silnik WordPressa wczytuje niezbędne informacje o pluginie. O wymaganiach dla tego nagłówka można więcej przeczytać tutaj: https://developer.wordpress.org/plugins/plugin-basics/header-requirements/. W naszym przypadku w pliku **`newly-added-post-highliter.php`** wstawimy poniższy opis:

```php
<?php
/**
 * Plugin Name:       Newly Added Post Highliter
 * Plugin URI:        https://example.com/plugins/Newly Added Post Highliter/
 * Description:       Highlight newly posts with tag.
 * Version:           1.0
 * Requires at least: 5.0
 * Requires PHP:      7.2
 * Author:            Piotr Jóźwiak
 * Author URI:        https://darksource.pl/
```

```
 * License:           GPL v2 or later
 * License URI:       https://www.gnu.org/licenses/gpl-2.0.html
 */
```

Niezbędne minimum to wpisanie pola Plugin Name. Jednakże warto tutaj uwzględnić także inne pola.

Krok 4 – uruchomienie pluginu

Zanim napiszemy jakikolwiek kod warto uruchomić plugin na stronie internetowej. Ponieważ nie posiada on jeszcze w zasadzie nawet jednej linijki kodu, to nie powinien powodować błędów. Zatem mamy pewność, że po uruchomieniu nic złego nie stanie się z naszą stroną. Plugin będziemy rozwijać przyrostowo, aby widzieć, który kod powoduje błędy skryptu.

Zatem wgrajmy folder **`NewlyAddedPostHighliter`** wraz z jego zawartością do folderu **`wp-content/plugins/`** na serwerze WWW. Następnie zalogujmy się do dashboardu WordPressa i przejdźmy do sekcji **Plugins**. Na liście pluginów powinniśmy zobaczyć nasz nowy plugin:

Aktywujmy go klikając w link ***Activate***.

Krok 5 – implementacja funkcjonalności pluginu

Implementację naszego pluginu zaczniemy od umieszczenia styli. Ponieważ nie jest celem tego laboratorium omawianie zasad działania CSS, zatem jedynie dla kompletności przykładu przedstawiam zawartość pliku **`css/style.css`**. Pamiętajmy, aby pisząc własny plugin, umieścić definicję własnych styli odpowiadających założeniom graficznym strony. Na potrzeby tego przykładu plik ten posiada poniższą zawartość:

```
.naph_marker{
    display:none;
}


.naph_marker{
    color: white;
    margin-left: 10px;
    background-color: red;
    padding: 1px 10px;
    border-radius: 5px;
    font-size: 0.5em;
    font-style: italic;
```

```
}

article .naph_marker{
    display: inline;
}
```

Przejdźmy zatem do napisania samego pluginu. Cały kod będzie zawarty w głównym pliku PHP. Kod będziemy omawiać w kolejności jego umieszczania.

Implementacja panelu administratora

Pierwszą funkcjonalnością jaką się zajmiemy będzie napisanie panelu administratora, w którym będziemy ustawiać parametry naszego pluginu. Konfiguracja będzie bardzo prosta i sprowadzać się będzie do ustawienia jednego parametru określającego liczbę dni od opublikowania przez jaką dany post będziemy oznaczać naszym tagiem.

Zacznijmy od zarejestrowania linku w menu Settings. Do tego celu w głównym pliku wstawiamy poniższy kod:

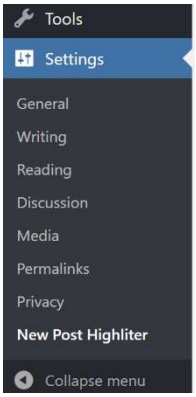
```
function naph_admin_actions_register_menu(){
    add_options_page("Newly Added Post Highliter", "New Post Highliter", 'manage_
options', "naph", "naph_admin_page");
}

add_action('admin_menu', 'naph_admin_actions_register_menu');
```

Ten prosty przykład wykonuje dwie czynności. W pierwszym kroku rejestruje Action Hook na zdarzenie **`admin_menu`**. Jest to zdarzenie, które jest wykonywane w momencie generowania menu administratora. Przypinamy do tego zdarzenia funkcję o nazwie **`naph_admin_actions_register_menu`**. Funkcja ta wywołuje funkcję **`add_options_page`**. Funkcja ta odpowiada za dodawanie elementów menu w sekcji **Settings**. Szczegóły tej funkcji można przeczytać tutaj: https://developer.wordpress.org/reference/functions/add_options_page/. Przyjmuje ona do 6 parametrów:

- Pierwszy parametr to Page Title.
- Drugi parametr odpowiada za nazwę/tekst wyświetlany w menu administratora
- Trzeci parametr to wymagane uprawnienie do tego elementu, tzw. ***capability***. Więcej o rolach i uprawnieniach można przeczytać tutaj: https://wordpress.org/support/article/roles-and-capabilities/. W naszym przypadku wstawiamy wymagane uprawnienie do zmian w ustawieniach zapisane w capability o nazwie: **manage_options**.
- Czwarty parametr to ciąg znaków, który będzie budował URL do strony administracyjnej. Musi on być unikalny w całym serwisie.
- Piąty parametr to nazwa funkcji jaka ma zostać uruchomiona do wygenerowania strony panelu administratora.

Ponieważ zarejestrowaliśmy już odwołanie do funkcji generującej panel administratora to zapiszmy na razie jej uproszczoną wersję, aby móc przetestować działanie naszego kodu. W naszym przypadku będzie ona wyglądała tak (jeszcze będziemy ją modyfikować):

```
function naph_admin_page(){
?>
    <h1>Newly Added Post Highliter</h1>
<?php
}
```



Zapiszmy zmiany w pliku i odświeżmy stronę z panelem administratora. Kliknijmy w sekcję Settings. Teraz powinien zostać wygenerowany link do naszego panelu administratora.

Po kliknięciu na niego pojawi się pusta strona z nagłówkiem zapisanym w funkcji generującej zawartość strony.

Przejdźmy zatem do napisania funkcji obsługującej panel administratora. W naszym przypadku wygląda ona następująco:

```
function naph_admin_page(){
  // get _POST variable from globals
  global $_POST;

  // process changes from form
  if(isset($_POST['naph_do_change'])){
    if($_POST['naph_do_change'] == 'Y'){
      $opDays = $_POST['naph_days'];
      echo '<div class="notice notice-success is-
dismissible"><p>Settings saved.</p></div>';
      update_option('naph_days', $opDays);
    }
  }

  //read current option value
  $opDays = get_option('naph_days');

  //display admin page
?>
  <div class="wrap">
    <h1>Newly Added Post Highliter</h1>
     <form name="naph_form" method="post">
       <input type="hidden" name="naph_do_change" value="Y">
```

```
        <p>Highlight post title for
        <input type="number" name="naph_days" min="0" max="30" value="<?php echo $
opDays ?>"> days
        </p>
        <p class="submit"><input type="submit" value="Submit"></p>
      </form>
    </div>
<?php
}
```

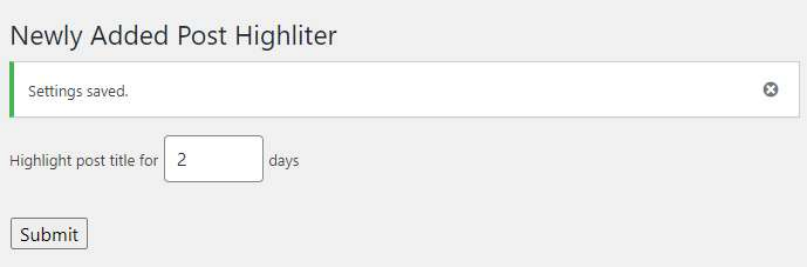
Jak widać z komentarzy do kodu, funkcja ta zajmuje się kolejno kilkoma sprawami. Pierwsza część odpowiada za przetworzenie formularza i zapisanie nowych danych do bazy danych WordPress. Wykorzystujemy do tego wbudowaną funkcjonalność silnika CMS w postaci dwóch funkcji:

- **update_option**(string $option, mixed $value, string|bool $autoload = null)
- **get_option**(string $option, mixed $default = false)

Pierwsza z nich zapisuje wartość pod identyfikatorem **naph_days** w bazie danych. Nie musimy przejmować się tym jak przechować te dane. WordPress sam zajmuje się tym za nas. Druga funkcja wczytuje aktualną wartość tej opcji. Jak widać jest to bardzo łatwe i przyjemne w użyciu rozwiązanie.

Kolejna część omawianej funkcji generuje formularz z aktualnymi ustawieniami.

To całość funkcjonalności odpowiedzialnej za obsługę administracyjną plugina. Zapiszmy zmiany i przejdźmy do przeglądarki. Postarajmy się ustawić wartość dla naszej opcji i przetestujmy, czy wszystko się zapisuje. Rezultat powinien przypominać ten z poniższego obrazka:



Implementacja funkcjonalności frontendowej

Przyszła kolej na napisanie części plugina odpowiedzialnej za wygenerowanie markera dla tytułów najnowszych postów. W tym celu musimy wpisać kod obsługujący te wymaganie do pliku głównego plugina. Można dopisać to na końcu tego pliku. W naszym przykładzie kod ten wygląda tak:

```
function naph_mark_new_post_title($content, $id){
    //read post publish date
    $date = get_the_date('Ymd', $id);
    //get current date
    $now = date('Ymd');
    //get setting for how long post is a new post
    $opDays = get_option('naph_days');
```

```
    //generate proper post title
    if($now - $date <= $opDays)
        return $content."<sup class=\"naph_marker\">new</sup>";
    return $content;
}

add_filter("the_title", "naph_mark_new_post_title", 10, 2);
```

Podobnie jak do tej pory rejestrujemy funkcję **naph_mark_new_post_title** do filtra o nazwie **the_title**. Filtr ten odpowiada za wyświetlanie tytułu posta. Będziemy go modyfikować w zależności od ustawień naszego plugina.

Logika markowania została zaimplementowana w funkcji **naph_mark_new_post_title**; Funkcja ta otrzymuje dwa parametry:

- Pierwszy to aktualna wartość tytułu posta
- Drugi to identyfikator posta

Na początku wczytujemy informację o dacie publikacji posta funkcją **get_the_date**, przekazując jej format oraz identyfikator naszego posta. Następnie wczytujemy aktualną datę. Dalej wczytujemy nasze ustawienia plugina.

Na końcu generujemy tytuł posta z markerem jeśli post jest świeższy niż liczba dni wczytana z ustawień plugina lub tytuł bez zmian.

Zapiszmy zmiany i sprawdźmy czy plugin działa. Oczywiście na tę chwilę jeszcze nie podłączyliśmy styli. Dlatego efekt jest mało imponujący. Zajmiemy się tym w kolejnym kroku.

Rejestracja plików styli

Ostatnią rzeczą jaką musimy zrobić to zarejestrować wcześniej zdefiniowane style. Dokonujemy tego także w naszym głównym pliku PHP. Na samym końcu dodajemy poniższy kod:

```
function naph_register_styles(){
    //register style
    wp_register_style('naph_styles', plugins_url('/css/style.css', __FILE__));
    //enable style (load in meta of html)
    wp_enqueue_style('naph_styles');
}

add_action('init', 'naph_register_styles');
```

Do tego celu znowu posłużymy się przypięciem akcji do Action Hook o nazwie **init**. Akcja **init** jest uruchamiana po załadowaniu silnika WordPressa, ale przed wysłaniem jakiegokolwiek nagłówka HTTP. W przypiętej funkcji wykorzystujemy dwie funkcje do:

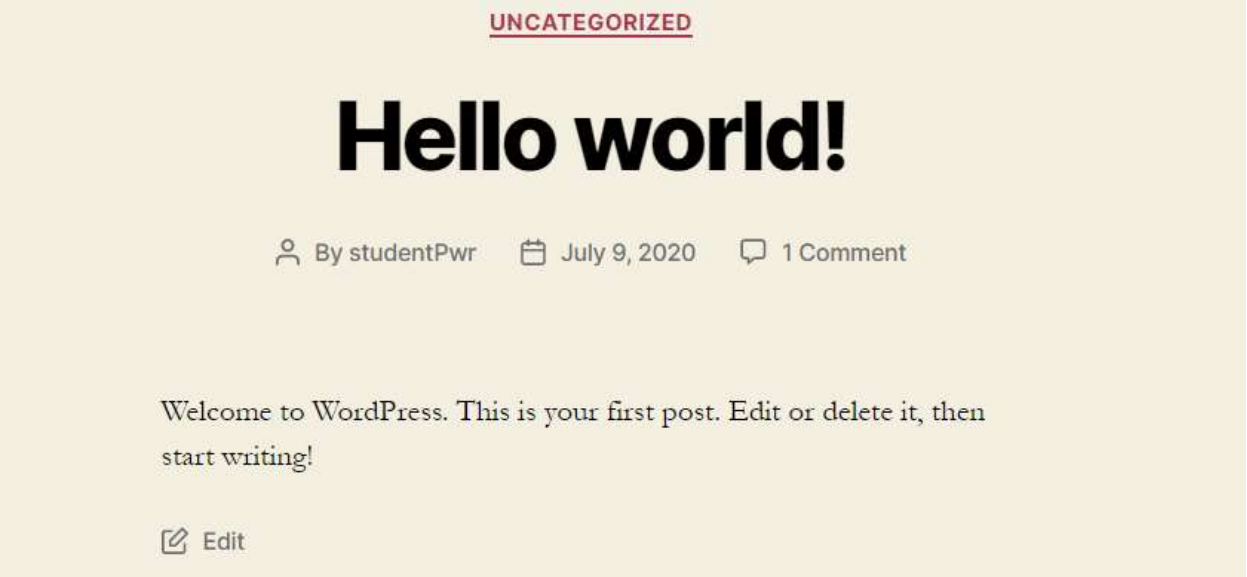
- **wp_register_style()** – rejestruje nasz plik styli pod nazwą z pierwszego argumentu

- **`wp_enqueue_style()`** – wskazuje aby dołączyć plik zarejestrowany pod nazwą z argumentu do htmla strony.

Powyższy mechanizm daje pełną kontrolę nad zarządzaniem stylami. Poprzez rozdzielenie tej funkcjonalności na rejestrację i kolejkę, mamy możliwość dołączania plików styli (tak samo i plików skryptów) tylko w wymaganych miejscach. Nie musimy ładować tych plików zawsze – przez co oszczędzamy transfer i przyspieszamy działanie serwisu WWW.

Zapiszmy zmiany w pliku. Pora sprawdzić jak działa plugin. Przejdźmy do przeglądarki. W wyniku działania powinniśmy otrzymać rezultat podobny do poniższego:

Natomiast dla starszego postu tytuł pozostał niezmieniony:



Debugowanie WordPressa

Jeszcze tylko jedna dygresja na koniec tego laboratorium. Pisząc jakikolwiek kod, czymś naturalnym jest, że pojawiają się błędy. WordPress domyślnie nie wyświetli informacji o błędzie, aby chronić te wrażliwe informacje przed atakującymi. Jednakże, w trakcie implementowania są one nieoceniona pomocą dla dewelopera.

Aby uruchomić te informacje wystarczy dodać w pliku **`wp-config.php`** poniższą linię kodu:

```
define( 'WP_DEBUG', true );
```

Od tej pory komunikaty o błędach powinny być dużo bardzie pomocne przy rozwiązywaniu problemów. Więcej na temat debugowania WordPressa znajdziemy tutaj: https://wordpress.org/support/article/debugging-in-wordpress/