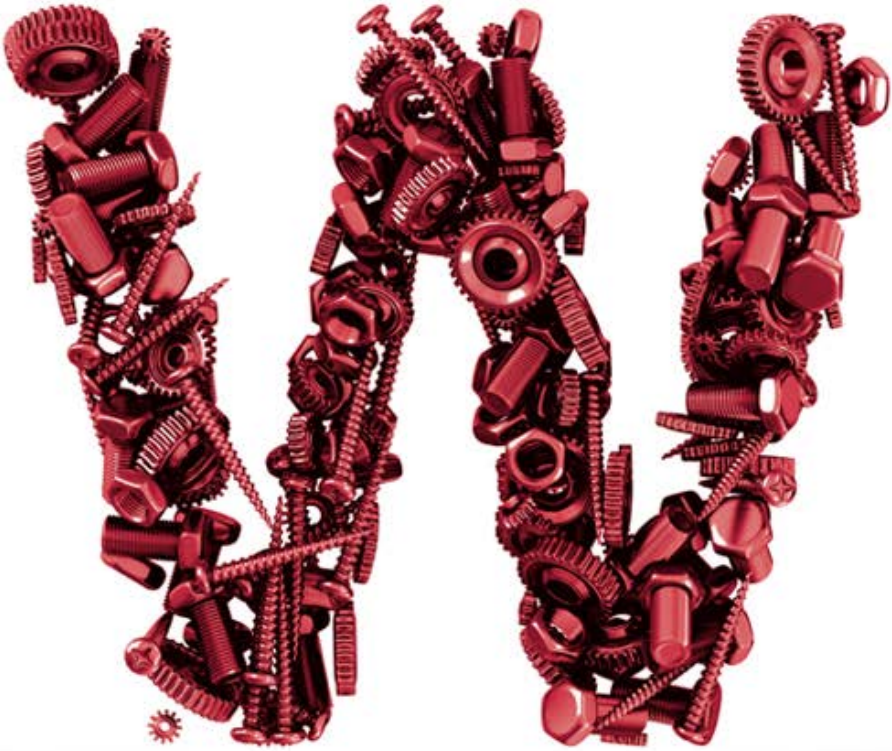
PROGRAMMER TO PROGRAMMER™



Wtyczki do WordPressa

Programowanie dla profesjonalistów

Brad Williams, Ozh Richard, Justin Tadlock

Helion

Tytuł oryginału: Professional WordPress Plugin Development

Tłumaczenie: Robert Górczyński

ISBN: 978-83-246-3564-1

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: helion@helion.pl
WWW: http://helion.pl (księgarnia internetowa, katalog książek)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/wtywor.zip

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/wtywor
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

10

API Shortcode

W TYM ROZDZIALE:

- Tworzenie własnego skrótu
- Rejestracja skomplikowanych i sparametryzowanych skrótów
- Zaawansowane wskazówki dotyczące skrótów
- Połączenie witryny z usługą Google Mapy

Skróty (ang. *shortcode*) to charakterystyczny dla WordPress kod pozwalający na wykonywanie różnych operacji przy minimalnym wysiłku, np. osadzanie treści lub tworzenie obiektów, które normalnie wymagałyby użycia dużej ilości skomplikowanego kodu.

W tym rozdziale dowiesz się, jak za pomocą jedynie niewielkiej liczby znaków umożliwić użytkownikom Twoich wtyczek rozbudowę wpisów bloga o zaawansowaną treść.

Tworzenie skrótu

W tym podrozdziale przekonasz się, czym są skróty oraz jak można je tworzyć; zaczniemy od zamienników w postaci prostych ciągów tekstowych, a skończymy na funkcjach zaawansowanych wraz z parametrami.

Czym jest skrót?

API Shortcode pozwala na tworzenie prostego kodu makro, czasami nazywanego BBCode z racji podobieństwa do popularnej składni w różnego rodzaju forach i serwisach komputerowych.

Ogólnie rzecz biorąc, skrót to prosta składnia znaczników w nawiasach kwadratowych, np. `[znacznik]`, stosowanych we wpisach bloga. W trakcie generowania i wyświetlania takiego wpisu bloga skrót jest dynamicznie zastępowany bardziej skomplikowaną i zdefiniowaną przez użytkownika treścią. Na rysunku 10.1 pokazano przykład wtyczki skrótu, która znacznik `[date]` zastępuje bieżącą datą i godziną.

Rysunek 10.1. Wtyczka skrótu w działaniu

Platforma WordPress domyślnie rejestruje wiele skrótów gotowych do wykorzystania. Przykładowo podczas wysyłania do serwera wielu obrazów dołączonych do danego wpisu bloga można w nim umieścić skrót `[gallery]`, co spowoduje zastąpienie obrazów ładnie sformatowaną galerią.

Z technicznego punktu widzenia skrót może być dowolnym ciągiem tekstowym odpowiednim do użycia w charakterze klucza tablicy. Przykładowo wymienione poniżej ciągi tekstowe można zastosować jako skróty:

- `[foo]`,
- `[Foo]`,
- `[123]`,
- `[133t]`,
- `[Witaj Jestem Jan Kowalski]`.

W praktyce, w celu zachowania prostoty i uniknięcia potencjalnych konfliktów pomiędzy różnymi skrótami skróty będziesz rejestrował jako proste ciągi tekstowe zapisane małymi literami.

UWAGA

Nie wolno rejestrować własnych skrótów o nazwach `[wp_caption]`, `[caption]`, `[gallery]` i `[embed]`, ponieważ są już zarejestrowane przez WordPress.

Rejestracja własnego skrótu

W tym punkcie dowiesz się, w jaki sposób rejestrować własne skróty. Ponadto poznasz praktyczne sposoby ich wykorzystania, od prostego zastępowania tagów aż po bardziej skomplikowane i sparametryzowane dane wyjściowe.

Przedstawione poniżej wtyczki skrótów używają prefiksu w postaci `boj_scX`, gdzie `X` oznacza liczbę.

[ksiazka]

Celem użycia skrótu jest przyśpieszenie wprowadzania danych i zastąpienie często stosowanych zdań ich skrótami, łatwymi do zapamiętania i szybszymi do wpisania.

Jeżeli np. często wspominasz tytuł książki, którą promujesz w serwisie Amazon, wówczas zamiast za każdym razem pisać `<a href="http://www.amazon.com/dp/0470560541">książka</a>`, o wiele szybciej i prościej będzie napisać po prostu `[ksiazka]`, prawda?

W tym celu trzeba użyć funkcji `add_shortcode()` wymagającej podania dwóch parametrów:

- wzorca znacznika (bez otaczających go nawiasów kwadratowych);
- wywoływanej funkcji odpowiedzialnej za zastąpienie znacznika.

```
<?php
/*
Plugin Name: Przykład skrótu nr 1
Plugin URI: http://przyklad.pl/
Description: Zastępuje znacznik [ksiazka] długim odnośnikiem prowadzącym do serwisu Amazon.
Version: 1.0
Author: Ozh
Author URI: http://wrox.com/
*/
// Rejestracja nowego skrótu: [ksiazka].
add_shortcode( 'ksiazka', 'boj_sc1_book' );
// Funkcja wywoływana w celu zastąpienia znacznika [ksiazka].
function boj_sc1_book() {
    return '<a href="http://www.amazon.com/dp/0470560541">książka<a/>';
}
?>
```

Powyższy fragment kodu pochodzi z pliku plugin_boj_sc1.php.

Jak to działa?

1. Za pomocą funkcji `add_shortcode()` zarejestrowano znacznik `[ksiazka]` jako nowy skrót, który będzie zastąpiony przez dane wyjściowe wygenerowane przez funkcję `boj_sc1_book()`.
2. Wywoływana funkcja zastępująca skrót (tutaj: `boj_sc1_book()`) musi na końcu zwrócić wartość za pomocą polecenia `return`. Błędem bardzo często popełnianym przez początkujących jest użycie do wyświetlenia wartości polecenia `echo` zamiast `return`. Trzeba pamiętać, że w tym przypadku nie można użyć polecenia `echo`.

Po aktywacji wtyczki można już napisać: „Kup moją `[ksiazka]`” na stronie wpisu bloga, a znacznik zostanie zastąpiony odnośnikiem prowadzącym na stronę w serwisie Amazon.

Warto zwrócić uwagę, że platforma WordPress jest bardzo elastyczna w zakresie składni skrótu: można stosować znaczniki mniej lub więcej przypominające znaczniki XHTML dowolnego typu, np. `[ksiazka]`, `[ksiazka ]`, `[ksiazka/]` i `[ksiazka /]`. Jedynym warunkiem jest, aby pomiędzy nawiasem otwierającym znacznik i samym znacznikiem nie było spacji.

[ksiazki tytul="xkcd"]

Co zrobić w sytuacji, gdy chcesz promować więcej niż jedną książkę?

Pierwszą opcją będzie utworzenie wielu skrótów w sposób przedstawiony powyżej, po jednym dla każdej książki (np. `[ksiazka1]`, `[ksiazka2]`, `[ksiazka3]` itd.). Jednak znacznie bardziej eleganckim rozwiązaniem będzie użycie atrybutu dla skrótu. W ten sposób można zastosować sprytniejszą składnię w postaci `[ksiazki tytul="prowp"]` lub `[ksiazki tytul="xkcd"]`.

W kodzie ponownie będzie użyta ta sama funkcja `add_shortcode()`, ale tym razem z nowym parametrem `$attr`, który pobiera tablicę atrybutu — parę wartości.

```
<?php
/*
Plugin Name: Przykład skrótu nr 2
Plugin URI: http://przyklad.pl/
Description: Zastępuje znacznik [ksiazki tytul="xxx"] różnymi odnośnikami prowadzącymi do serwisu
➥Amazon.
Version: 1.0
Author: Ozh
Author URI: http://wrox.com/
*/
// Rejestracja nowego skrótu: [ksiazki tytul="xxx"].
add_shortcode( 'ksiazki', 'boj_sc2_multiple_books' );
// Funkcja wywoływana w celu zastąpienia znacznika [ksiazki].
function boj_sc2_multiple_books( $attr ) {
    switch( $attr['tytul'] ) {
        case 'xkcd':
            $asin = '0615314465';
            $title = 'XKCD Volume 0';
            break;
        default:
        case 'prowp':
            $asin = '0470560541';
            $title = 'Professional WordPress';
            break;
    }
    return "<a href='http://www.amazon.com/dp/$asin'>$title<a/>";
}
?>
```

Powyższy fragment kodu pochodzi z pliku plugin_boj_sc2.php.

Jak to działa?

1. Za pomocą funkcji `add_shortcode()` zarejestrowano znacznik `[ksiazki]` jako nowy skrót.
2. Funkcja odpowiedzialna za zastąpienie znacznika `boj_sc2_multiple_books()` oczekuje podania parametru `$attr` w postaci tablicy atrybutu — pary wartości do użycia w skrócie. Przykładowo po użyciu znacznika `[ksiazki tytul="prowp"]` funkcja otrzyma parametr w postaci `array( 'tytul' => 'prowp' )`.
3. Funkcja odpowiedzialna za zastąpienie znacznika będzie zwracała różne wartości w zależności od przekazanego jej atrybutu.
4. Użycie znacznika `[ksiazki]` bez atrybutu spowoduje, że funkcja otrzyma pusty ciąg tekstowy. W takim przypadku zwrócona będzie wartość domyślna zdefiniowana w kodzie.

[amazon asin="12345"]tytuł książki[/amazon]

Przygotowaną powyżej wtyczkę można jeszcze bardziej usprawnić, czyli umożliwić sparametryzowanie tekstu w odnośniku prowadzącym do serwisu Amazon.

Użyta będzie ponownie ta sama funkcja `add_shortcode()`, ale tym razem z drugim parametrem — `$content` — który będzie przekazywał ciąg tekstowy wykorzystany następnie jako tekst w odnośniku.

```
<?php
/*
Plugin Name: Przykład skrótu nr 3
Plugin URI: http://przyklad.pl/
Description: Zastępuje znacznik [amazon isbn="xxx"]tytul ksiazki[/amazon].
Version: 1.0
Author: Ozh
Author URI: http://wrox.com/
*/
// Rejestracja nowego skrótu: [amazon isbn="123"]tytul ksiazki[/amazon].
add_shortcode( 'amazon', 'boj_sc3_amazon' );
// Funkcja wywoływana w celu zastąpienia znacznika [amazon].
function boj_sc3_amazon( $attr, $content ) {
    // Pobranie numeru ASIN (Amazon Standard Identification Number).
    if( isset( $attr['asin'] ) ) {
        $asin = preg_replace( '/[^\d]/', '', $attr['asin'] );
    } else {
        $asin = '0470560541';
    }
    // Oczyszczenie treści lub ustawienie domyślnej.
    if( !empty( $content ) ) {
        $content = esc_html( $content );
    } else {
        if( $asin == '0470560541' ) {
            $content = 'Professional WordPress';
        } else {
            $content = 'ta książka';
        }
    }
    return "<a href='http://www.amazon.com/dp/$asin'>$content<a/>";
}
?>
```

Powyższy fragment kodu pochodzi z pliku plugin_boj_sc3.php.

Jak to działa?

1. Zarejestrowano kolejny skrót, tym razem używając znacznika `[amazon]`.
2. Funkcja odpowiedzialna za zastąpienie znacznika `boj_sc3_amazon()` oczekuje dwóch opcjonalnych parametrów: tablicy atrybutu — pary wartości do użycia w skrócie oraz ciągu tekstowego umieszczonego pomiędzy znacznikami skrótu otwierającym i zamykającym.
3. Funkcja odpowiedzialna za zastąpienie znacznika musi obsłużyć różne kombinacje brakującego atrybutu w postaci numeru ASIN (ang. *Amazon Standard Identification Number*) i (lub) tytułu książki: znaczniki `[amazon]`, `[amazon asin="123"]` i `[amazon]super książka[/amazon]` działają bez problemów.
4. Skrót może zwrócić dowolną treść, więc trzeba pamiętać o zastosowaniu technik omówionych w rozdziale 6. Numer ASIN trzeba oczyścić, aby składał się jedynie z cyfr. Ponadto należy się upewnić, że tytuł książki może być bezpiecznie wyświetlony w Twoim blogu i nie zniszczy znacznika <a>, w którym będzie umieszczony.

Podsumowanie: funkcja add_shortcode() i funkcja odpowiedzialna za zastąpienie znacznika

Podczas rejestracji nowego skrótu dwa parametry definiują wzorzec znacznika w nawiasach kwadratowych oraz funkcję wywoływaną w celu zastąpienia znacznika:

```
<?php
add_shortcode( 'boj', 'boj_my_shortcode' );
?>
```

Funkcja odpowiedzialna za zastąpienie znacznika pobiera dwa parametry (puste, jeśli zostaną pominięte), czyli tablicę atrybutu — para wartości — oraz ciąg tekstowy definiujący treść umieszczaną pomiędzy znacznikami skrótu otwierającym i zamykającym. Podobnie jak w kodzie HTML, wielkość liter w atrybutach nie ma znaczenia.

Tak jak w każdej innej funkcji PHP, istnieje możliwość zdefiniowania wartości domyślnych. Trzeba pamiętać, że funkcja odpowiedzialna za zastąpienie znacznika musi zwrócić wartość.

```
<?php
function boj_my_shortcode( $attr = array( 'var' => 'wartość' ), $content =
↪'książka' ) {
    // $attr to tablica asocjacyjna.
    // $content to ciąg tekstowy.
    return $something;
}
?>
```

Atrybuty skrótu nie rozróżniają wielkości liter, mogą mieć dowolną wartość lub nie mieć jej wcale, a także obsługują znaki cudzysłowu lub akceptują ich brak. Przedstawione poniżej przykłady pokazują wartości tablicy `$attr` w funkcji odpowiedzialnej za zastąpienie znacznika, w zależności od sposobu użycia skrótu:

- `[boj] : $attr` będzie pustym ciągiem tekstowym;
- `[boj hello] : $attr` będzie tablicą `array( 'hello' );`
- `[boj name=ozh skillz='1337' MAP="q3dm6"] : $attr` będzie tablicą `array ( 'name' => 'ozh', 'skillz' => '1337', 'map' => 'q3dm6' ).`

Wskazówki dotyczące skrótów

Skróty to doskonały sposób wzbogacenia wpisu bloga skomplikowaną i dynamiczną treścią. W celu zagwarantowania dostarczenia użytkownikowi najlepszych wrażeń autor wtyczki powinien pamiętać o dwóch podstawowych zasadach:

- zbuduj prostą i niezawodną wtyczkę;
- pamiętaj, że jest dynamiczna.

Pomyśl o prostocie

Użytkownik jest zadowolony, kiedy otrzymuje nowe funkcje do bloga i jednocześnie może stosować skróty pozwalające na wyświetlanie znacznie bardziej skomplikowanej treści. Jednak zapamiętywanie składni parametrów skrótu jest niewygodne: powstaje wrażenie konieczności poznania nowego języka znaczników.

Powróćmy jeszcze na chwilę do skrótu `[amazon]`: istnieje możliwość przygotowania wtyczki, która będzie dodawała skrót `[amazonobraz]` wyświetlający obraz produktu z serwisu Amazon. Zadaniem użytkownika jest podanie numeru ASIN, typu obrazu (książka czy płyta CD) oraz jego wielkości.

Po zbudowaniu wtyczka będzie zezwalała na użycie skrótu, takiego jak `[amazonobraz asin='12345' typ='CD' wielkoscobrazu='maly']`.

Kiedy użytkownik posiada tę wtyczkę od dłuższego czasu, może zapomnieć nazwy i składni atrybutów. Czy to było `[amazonobraz]`, czy `[obrazamazon]`? Czy atrybutem jest `isbn`, czy `asin`? Czy `wielkoscobrazu='ogromny'`, czy `'duzy'`? Czy `typ='CD'`, czy `typ='dysk'`?

Wprawdzie posiadanie dużej liczby opcji może być zaletą, ale równocześnie nie chcesz, aby użytkownicy musieli nieustannie sięgać do dokumentacji, ponieważ to sprawia niekorzystne wrażenie. Warto sprawy uprościć i pozwolić użytkownikom na instynktowne używanie wtyczki.

Po wprowadzeniu modyfikacji kod wtyczki przedstawia się następująco:

```
<?php
<?php
/*
Plugin Name: Przykład skrótu nr 4
Plugin URI: http:/przyklad.pl/
Description: Zastąpienie znacznika [amazonobraz] obrazem z serwisu Amazon.
Version: 1.0
Author: Ozh
Author URI: http://wrox.com/
*/

// Rejestracja skrótów [amazonobraz] i [obrazamazon]
add_shortcode( 'amazonobraz', 'boj_sc4_amazonimage' );
add_shortcode( 'obrazamazon', 'boj_sc4_amazonimage' );

// Funkcja odpowiedzialna za zastąpienie skrótu [amazonobraz].
function boj_sc4_amazonimage( $attr, $content ) {

    // Pobranie numeru ASIN lub ustawienie domyślnego.
    $possible = array( 'asin', 'isbn' );
    $asin = boj_sc4_find( $possible, $attr, '0470560541' );

    // Pobranie powiązanego identyfikatora lub ustawienie domyślnego.
    $possible = array( 'aff', 'affiliate' );
    $aff = boj_sc4_find( $possible, $attr, 'aff_id' );

    // Pobranie wielkości obrazu, jeśli podano.
    $possible = array( 'wielkosc', 'obraz', 'wielkoscobrazu' );
    $size = boj_sc4_find( $possible, $attr, '' );

    // Pobranie typu, jeśli podano.
    if( isset( $attr['type'] ) ) {
        $type = strtolower( $attr['typ'] );
        $type = ( $type == 'cd' or $type == 'dysk' ) ? 'cd' : '';
    }

    // Utworzenie adresu URL prowadzącego do obrazu w serwisie Amazon.
    $img = 'http://images.amazon.com/images/P/';
```

```
    $img .= $asin;
    // Opcje obrazu: wielkość.
    if( $size ) {
        switch( $size ) {
            case 'maly':
                $size = '_AA100';
                break;
            default:
            case 'sredni':
                $size = '_AA175';
                break;
            case 'duzy':
            case 'ogromny':
                $size = '_SCLZZZZZZZ';
                break; // Dobra praktyka, nie zapomnij o ostatnim poleceniu break.
        }
    }
    // Opcje obrazu: typ.
    if( $type == 'cd' ) {
        $type = '_PF';
    }
    // Dołączenie opcji do adresu URL, o ile podano jakiekolwiek opcje.
    if( $type or $size ) {
        $img .= '.01.'.$type.$size;
    }
    // Zakończenie tworzenia adresu URL obrazu.
    $img .= '.jpg';

    // Trzeba pamiętać o zwróceniu obrazu.
    return "<a href='http://www.amazon.com/dp/$asin'><img src='$img' /></a>";
}

// Funkcja pomocnicza:
// Wyszukuje $find_keys w tablicy $in_array, zwraca $default, jeśli nie znajdzie $find_keys.
function boj_sc4_find( $find_keys, $in_array, $default ) {
    foreach( $find_keys as $key ) {
        if( isset( $in_array[$key] ) )
            return $in_array[$key];
    }
    return $default;
}
?>
```

Powyższy fragment kodu pochodzi z pliku plugin_boj_sc4.php.

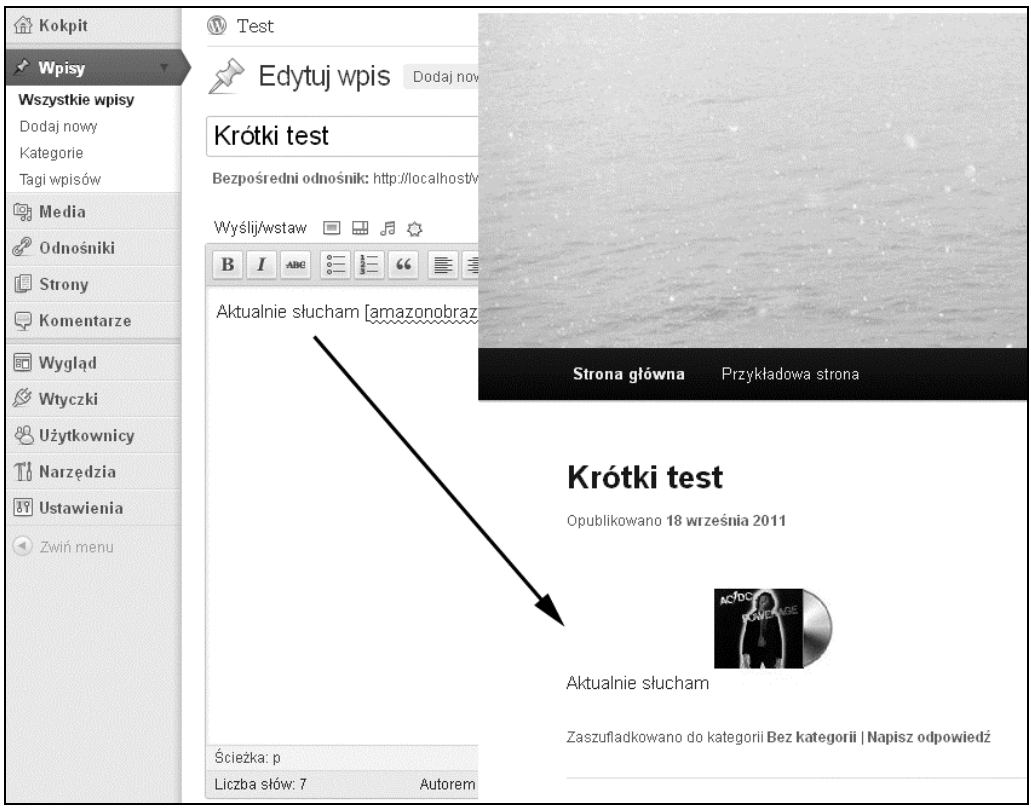
Najpierw uwagę należy zwrócić na rejestrację dwóch skrótów wywołujących tę samą funkcję: w ten sposób użytkownik może użyć skrótu zarówno `[amazonobraz]`, jak i `[obrazamazon]`.

Następnie warto zwrócić uwagę na to, jak wiele atrybutów jest traktowanych jako synonimy: przy użyciu funkcji pomocniczej o nazwie `boj_sc4_find()` główna funkcja odpowiedzialna za obsługę skrótu sprawdza wartości `$attr['asin']` i `$attr['isbn']`. Gdy ich brakuje powoduje ustawienie wartości domyślnej.

Kiedy informacje nie są związane z platformą WordPress, warto także przyjrzeć się sposobowi tworzenia przez wtyczkę odnośnika do obrazu w serwisie Amazon. Podstawowy adres URL to *http://images.amazon.com/images/P/*; do niego są dołączane następujące elementy:

- numer ASIN, np. w postaci `B0020EBMN4`;
- jeśli mają być użyte opcje, trzeba dołączyć `.01.`;
- pierwsza użyta opcja to wielkość: dołączenie `_AA100` powoduje pobranie obrazu o szerokości 100 pikseli, podczas gdy dołączenie `_SCLZZZZZZZ` pobiera dużą wersję obrazu;
- inna możliwa opcja to pobranie obrazu płyty CD: w tym celu do tworzonego adresu URL trzeba dołączyć `_PF`;
- na końcu adres URL obrazu trzeba zakończyć, dodając `.jpg`.

Po aktywacji wtyczki można utworzyć nowy wpis bloga o treści `Aktualnie słucham [amazonobraz asin="B00008WT5E" typ="cd" wielkosc= "maly"]`, a otrzymany wynik będzie podobny do pokazanego na rysunku 10.2.



Rysunek 10.2. Wtyczka pobierająca dane z serwisu Amazon w działaniu

Pamiętaj o dynamiczności

Dane skrótu są generowane dynamicznie: za każdym razem, gdy platforma WordPress wyświetla stronę (pojedynczy wpis bloga lub archiwum), treść skrótu jest przetwarzana, a wszystkie skróty zastępowane są danymi zwracanymi przez wywołania funkcji obsługujących te skróty.

Zamienniki, takie jak wykorzystane w rozdziale, są bardzo szybkie, więc podczas rejestrowania nowego skrótu nie trzeba się przejmować wydajnością działania platformy WordPress.

Jednak wydajność nabierze większego znaczenia, gdy skróty będą pobierały informacje z bazy danych bądź ze zdalnych witryn internetowych.

- W pierwszym przypadku kod spowoduje wykonanie dodatkowych zapytań SQL, co może drastycznie zmniejszyć wydajność w wolniejszych serwerach.
- W drugim przypadku kod będzie wykonywał zewnętrzne żądania HTTP, które mogą spowolnić wygenerowanie całej strony, bo platforma WordPress będzie oczekiwała na odpowiedź ze zdalnego serwera.

W takich przypadkach warto rozważyć buforowanie wyniku przetworzenia skrótu, np. w metadanych wpisu bloga. W kolejnej wtyczce zostanie zaimplementowana taka technika buforowania.

Wewnętrzny sposób działania

Oprócz `add_shortcode()` służącej do rejestracji nowego skrótu istnieją także inne interesujące funkcje. Ponadto warto poznać kilka faktów dotyczących API Shortcode, które można wykorzystać w tworzonych wtyczkach.

$shortcode_tags

Wszystkie zarejestrowane skróty są przechowywane w tablicy globalnej o nazwie `$shortcode_tags` w postaci par `'nazwa_skrótu' => 'funkcja_obsługująca_dany_skrót'`:

```
<?php
global $shortcode_tags;
var_dump( $shortcode_tags );
/* Wynik:
array (
   'wp_caption'  => 'img_caption_shortcode',
   'caption' => 'img_caption_shortcode',
   'gallery' => 'gallery_shortcode',
   'embed' => '__return_false',
   'amazonimage' => 'boj_sc4_amazonimage',
   'amazonimg' => 'boj_sc4_amazonimage',
)
*/
?>
```

remove_shortcode()

Przy użyciu funkcji `remove_shortcode()` mamy możliwość dynamicznego wyrejestrowania skrótu, np.:

```
remove_shortcode( 'amazonobraz' );
```

remove_all_shortcodes()

Podobnie, w celu dynamicznego wyrejestrowania wszystkich skrótów należy użyć funkcji `remove_all_shortcodes()` bez parametrów. Z technicznego punktu widzenia wymieniona funkcja po prostu zeruje tablicę globalną `$shortcode_tags`, która staje się pusta.

strip_shorcodes()

Funkcja `strip_shorcodes()` powoduje usunięcie zarejestrowanych skrótów z treści, co przedstawiono w poniższym przykładzie:

```
<?php
$content = <<<S
Pewne istniejące skróty: [amazonobraz] [gallery]
Nieistniejące skróty: [bleh] [123]
S;
echo strip_shortcodes( $content );
/* Result:
Pewne istniejące skróty:
Nieistniejące skróty: [bleh] [123]
*/
?>
```

shortcode_atts()

Funkcja może być użyta do porównania atrybutów podanych przez użytkownika względem listy obsługiwanych atrybutów, a następnie do ustawienia wartości domyślnych, jeśli będzie trzeba.

Spójrz np. na sposób działania wbudowanego skrótu `[gallery]`. Odpowiedzialna za jego obsługę funkcja to `gallery_shortcode()`, która przetwarza atrybuty skrótu następująco:

```
<?php
function gallery_shortcode( $attr ){
    // Zdefiniowanie atrybutów obsługiwanych oraz ich wartości domyślne.
    $defaults = array(
        'order'      => 'ASC',
        'orderby'    => 'menu_order ID',
        'id'         => $post->ID,
        'itemtag'    => 'dl',
        'icontag'    => 'dt',
        'captiontag' => 'dd',
        'columns'    => 3,
        'size'       => 'thumbnail',
        'include'    => '',
        'exclude'    => ''
    );
    // Filtrowanie atrybutów podanych przez użytkownika oraz ustawienie ich wartości domyślnych,
    ↪jeśli zostały pominięte.
    $options = shortcode_atts( $defaults, $attr );
    // [... dalsza część kodu...]
    // Plik: wp-includes/media.php
}
?>
```

Dość długa lista obsługiwanych atrybutów oraz ich wartości domyślne zdefiniowane w tablicy `$defaults` są łączone z atrybutami dostarczonymi przez użytkownika w tablicy `$attr`. Wszystkie nieznane atrybuty są ignorowane.

do_shortcode()

Funkcja `do_shortcode()` przeszukuje pod kątem skrótów przekazany jej jako parametr ciąg tekstowy treści, a następnie przetwarza je. Podczas inicjalizacji platformy WordPress jest powiązana z filtrem `the_content`, więc zajmuje się treścią wpisu bloga:

```
<?php
// W pliku wp-includes/shortcodes.php
add_filter( 'the_content', 'do_shortcode', 11 );
?>
```

Skróty rekurencyjne

Może się zdarzyć, że treść skrótu będzie zawierała inne skróty. Przykładowo możesz zarejestrować skróty `[b]` i `[i]` do wyświetlania tekstu pogrubionego i zapisanego kursywą. Skróty powinny działać w zagnieżdżonej strukturze, takiej jak `[b]pewien [i]ciekawy[/i] tekst[/b]`.

Struktura taka nie stanowi problemu, ponieważ funkcja obsługująca dany skrót może rekurencyjnie wywołać funkcję `do_shortcode()`:

```
<?php
// Dodanie skrótów [b] i [i].
add_shortcode( 'i', 'boj_sc5_italic' );
add_shortcode( 'b', 'boj_sc5_bold' );
// Funkcja obsługująca skrót: zwraca pogrubiony tekst.
function boj_sc5_bold( $attr, $content ) {
    return '<strong>' . do_shortcode( $content ) . '</strong>';
}
// Funkcja obsługująca skrót: zwraca tekst zapisany kursywą.
function boj_sc5_italic( $attr, $content ) {
    return '<em>' . do_shortcode( $content ) . '</em>';
}
?>
```

Powyższy fragment kodu pochodzi z pliku plugin_boj_sc5.php.

Każda funkcja obsługująca skrót upewnia się, że przetworzony zostanie skrót znajdujący się w tekście danego skrótu.

Kod BBCode we wtyczce obsługującej komentarze

Na tym etapie można utworzyć nową wtyczkę, która umożliwi stosowanie w komentarzach znaczników typu BBCode. W większości forów zamiast zwykłych znaczników HTML, np. `<a>` lub `<b>`, komentujący muszą stosować znaczniki, takie jak `[url]` i `[b]`.

Wtyczka będzie ponadto charakteryzowała się następującymi cechami.

- Nie będzie zmieniać sposobu tworzenia wpisów bloga przez autorów (czyli z użyciem znaczników HTML).
- W komentarzach nie będą stosowane skróty zarejestrowane do używania w innych miejscach wpisu bloga, takie jak np. zdefiniowany wcześniej skrót `[amazonobraz]` lub `[gallery]`.

Poniżej przedstawiono pełny kod wtyczki.

```
<?php
/*
Plugin Name: Przykład skrótu nr 6
```

```
Plugin URI: http://przyklad.pl/
Description: Umożliwia stosowanie skrótów [url] i [b] w komentarzach.
Version: 1.0
Author: Ozh
Author URI: http://wrox.com/
*/

// Rejestracja dla zaczepu 'comment_text' funkcji odpowiedzialnej za przetworzenie komentarza.
add_filter( 'comment_text', 'boj_sc6_comments' );

// Funkcja przetwarzająca treść komentarza.
function boj_sc6_comments( $comment ) {

    // Zapisanie zarejestrowanych skrótów.
    global $shortcode_tags;
    $original = $shortcode_tags;

    // Wyrejestrowanie wszystkich skrótów.
    remove_all_shortcodes();

    // Zarejestrowanie nowych skrótów.
    add_shortcode( 'url', 'boj_sc6_comments_url' );
    add_shortcode( 'b', 'boj_sc6_comments_bold' );
    add_shortcode( 'strong', 'boj_sc6_comments_bold' );

    // Usunięcie wszystkich znaczników HTML z komentarza.
    $comment = wp_strip_all_tags( $comment );

    // Przetworzenie treści komentarza z uwzględnieniem skrótów.
    $comment = do_shortcode( $comment );

    // Wyrejestrowanie skrótów komentarza, przywrócenie zapisanych skrótów.
    $shortcode_tags = $original;

    // Zwrócenie komentarza.
    return $comment;
}

// Skrót [b] lub [strong] powoduje wywołanie funkcji zastępującej skrót znacznikiem <strong>.
function boj_sc6_comments_bold( $attr, $text ) {
    return '<strong>' .do_shortcode( $text ). '</strong>';
}

// Skrót [url] powoduje wywołanie funkcji zastępującej skrót znacznikiem <a>.
function boj_sc6_comments_url( $attr, $text ) {
    $text = esc_url( $text );
    return "<a href=\"$text\">$text</a>";
}
?>
```

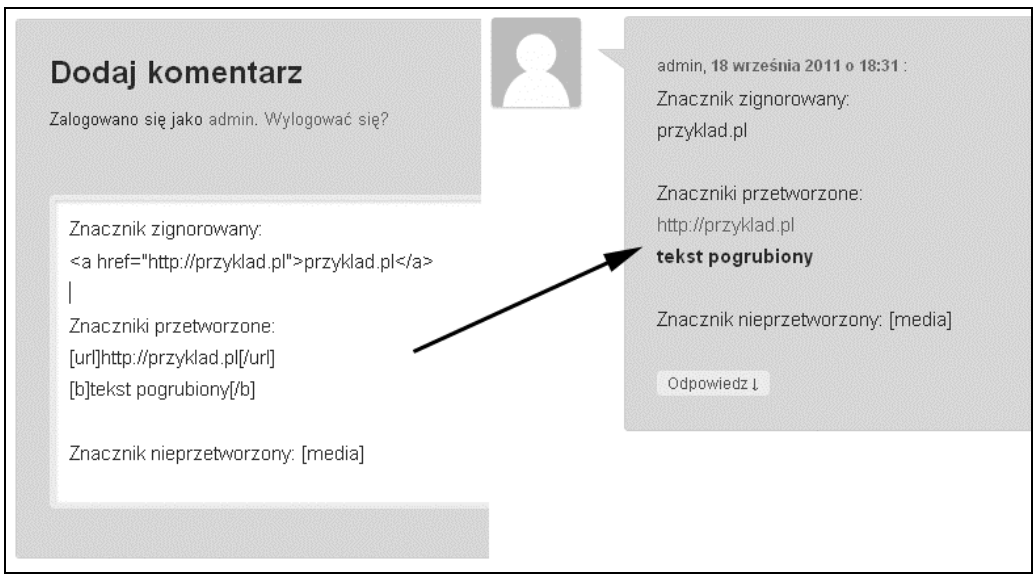
Powyższy fragment kodu pochodzi z pliku plugin_boj_sc6.php.

Jak to działa?

1. Jak możesz zobaczyć, wtyczka nie powoduje zarejestrowania nowych skrótów `[url]` i `[b]` od razu na początku, ponieważ w takim przypadku wpływałyby one na treść wpisu bloga. Pierwszym zadaniem wtyczki jest przechwycenie treści komentarza.

2. Funkcja przetwarzająca komentarz `boj_sc6_comments()` najpierw tworzy kopię wszystkich skrótów, a następnie je wyrejestrowuje.
3. Kolejnym krokiem jest rejestracja nowych skrótów: `[url]` i `[b]`. (W celu ułatwienia życia użytkownikowi `[strong]` jest odpowiednikiem `[b]`).
4. Z treści komentarza przechowywanej w zmiennej `$comment` następuje usunięcie zwykłych znaczników HTML, a następnie zastosowanie nowo zarejestrowanych skrótów.
5. Warto zwrócić uwagę, jak funkcja odpowiedzialna za pogrubienie tekstu rekurencyjnie wywołuje funkcję `do_shortcode()`, co pozwala na stosowanie zagnieżdżonych struktur.
6. Kolejny krok to przywrócenie oryginalnych skrótów, przy okazji następuje wyrejestrowanie skrótów komentarza `[url]` i `[b]`.
7. Sformatowana treść komentarza zostaje zwrócona w celu wyświetlenia.

Teraz możesz aktywować wtyczkę i dodać nowy komentarz. Spójrz na rysunek 10.3, aby przekonać się, jak ignorowane są znaczniki HTML. Skróty `[b]` i `[url]` są przetwarzane, ale zwykłe skróty, takie jak `[gallery]`, które są przetwarzane we wpisie bloga, są w komentarzach ignorowane.



Rysunek 10.3. Wtyczka umożliwiająca stosowanie znaczników typu BBCode w działaniu

Ograniczenia skrótów podczas obsługi struktur zagnieżdżonych

Jak się wcześniej dowiedziałeś, platforma WordPress potrafi obsługiwać zagnieżdżone struktury skrótów przy założeniu, że funkcje odpowiedzialne za ich obsługę rekurencyjnie wywołują `do_shortcode()`. Jednak tego rodzaju rozwiązanie ma swoje ograniczenia i czasami może po prostu nie działać, o czym się wkrótce przekonasz.

Struktura przedstawiona poniżej jest obsługiwana prawidłowo, ponieważ zagnieżdżone skróty są inne, a każdy z nich jest prawidłowo osadzony:

```
Działa:
[foo]
```

```
    [bar]
        [baz]
    [/bar]
[/foo]
```

Struktura przedstawiona niżej nie będzie prawidłowo obsłużona, bo osadzony skrót jest taki sam jak skrót, w którym się zawiera:

```
Nie działa:
[foo]
    [foo]
    [/foo]
[/foo]
```

Pamiętaj, że skróty mogą być samozamykające się (samodzielny skrót `[foo]` lub `[foo/]`) bądź mogą zawierać treść (`[foo]treść[/foo]`). Wspomniana treść również może uniemożliwiać prawidłowe przetwarzanie pewnych struktur:

```
Nie działa:
[foo]
[foo]
    treść
[/foo]
```

Integracja z usługą Google Mapy

Jako pełny i znacznie bardziej skomplikowany przykład użycia skrótów utworzymy wtyczkę pozwalającą na integrację usługi Google Mapy z Twoją witryną bazującą na platformie WordPress.

Google oferuje wiele różnych API pozwalających na uzyskanie dostępu do usług, szczególnie do usługi map. Usługa ta bazuje na dwóch powiązanych z nią usługach: API Google Geocoding i API Google Maps.

UWAGA

Google udostępnia dokładną dokumentację API Google Maps. Więcej informacji na ten temat można znaleźć na stronie *http://code.google.com/apis/maps/documentation/javascript/*.

W tym podrozdziale utworzymy wtyczkę pozwalającą na konwersję adresu zapisanego w postaci zwykłego tekstu (np. ul. Kościuszki 1c, 44-100 Gliwice) na dynamicznie generowaną, interaktywną mapę Google.

Uzyskanie dostępu do API Google Geocoding

Pierwszym krokiem podczas konwersji adresu na mapę jest utworzenie „geokodu" adresu. Tworzenie geokodu to proces opisany jako konwersja standardowego adresu (np. ul. Kościuszki 1c, 44-100 Gliwice) na współrzędne geograficzne (50.289, 18.660). Współrzędne są używane przez API Google Maps do znalezienia wskazanego położenia na mapie oraz umieszczenia na niej znaczników zależnych od podanych współrzędnych.

Obecnie API Google Geocoding zwraca wyniki w dwóch formatach: JSON i XML. W omawianym tutaj przykładzie zostanie wykorzystany format JSON. Ponadto będziemy stosować techniki omówione w rozdziale 9., który jest poświęcony wykonywaniu i obsłudze żądań HTTP.

Firma Google uprościła proces współpracy z API. W celu otrzymania współrzędnych można wykonać żądanie na następujący adres URL: *http://maps.google.com/maps/api/geocode/$output?$parameters*, gdzie `$output` oznacza format danych wyjściowych (np. `'json'`), natomiast `$parameter` to ciąg tekstowy zapytania zawierający parametry dodatkowe dla geokodu.

Do API trzeba przekazać tylko dwa wymagane parametry: `address` lub `latlng` i `sensor`.

- Ponieważ nie są znane współrzędne geograficzne, użyty będzie parametr `address`. Ten parametr to zapisany w postaci zwykłego tekstu pełny adres, który ma zostać zamieniony na geokod. Adres jest zakodowany w adresie URL.
- Parametr `sensor` wskazuje, czy żądanie pochodzi z urządzenia zawierającego sensor lokalizacji (np. smartfon). Tej zmiennej ustawimy wartość `false`.

Możesz to bardzo łatwo przetestować: wczytaj API Google Geocoding poprzez uruchomienie dowolnej przeglądarki internetowej i przejdź na stronę *http://maps.google.com/maps/api/geocode/json?address=Kościuszki+1c+Gliwice&sensor=false*.

Jak widać, zwrócone dane JSON zawierają współrzędne geograficzne podanego adresu, a także dane dodatkowe, takie jak kod pocztowy (który nie został podany w żądaniu).

Teraz możemy przystąpić do tworzenia funkcji `boj_gmap_geocode()`, która wygeneruje geokod dla wskazanego adresu:

```
<?php
// Utworzenie geokodu na podstawie adresu: wartością zwrotną jest tablica zawierająca współrzędne
  ↪geograficzne.
function boj_gmap_geocode( $address ) {
    // Utworzenie adresu URL do API Google Geocoding.
    $map_url = 'http://maps.google.com/maps/api/geocode/json?address=';
    $map_url .= urlencode( $address ).' & sensor=false';
    // Wykonanie żądania GET.
    $request = wp_remote_get( $map_url );
    // Pobranie obiektu JSON.
    $json = wp_remote_retrieve_body( $request );
    // Upewnienie się, że żądanie zakończyło się powodzeniem. W przeciwnym razie trzeba zwrócić false.
    if( empty( $json ) )
        return false;
    // Odkodowanie obiektu JSON.
    $json = json_decode( $json );
    // Pobranie współrzędnych geograficznych.
    $lat = $json->results[0]->geometry->location->lat;      // Szerokość geograficzna.
    $long = $json->results[0]->geometry->location->lng;     // Długość geograficzna.
    // Zwrócenie tablicy zawierającej współrzędne geograficzne.
    return compact( 'lat', 'long' );
}
?>
```

Funkcja wykonuje żądanie do API Google Geocoding i otrzymuje odpowiedź w formacie JSON, która po zdekodowaniu będzie zawierała współrzędne geograficzne. Można to sprawdzić, analizując otrzymaną wartość zwrotną:

```
<?php
$coords = boj_gmap_geocode( 'ul. Kościuszki 1c, Gliwice' );
var_dump( $coords );
/* Wynik:
array(2) {
```

```
 ["lat"]=> float(50.28891549)
 ["long"]=> float(18.65953990)
}
*/
?>
```

Więcej informacji wraz z objaśnieniem funkcji użytych w powyższej funkcji można znaleźć w rozdziale 9.

Przechowywanie wyników

Jednym z ważniejszych aspektów skrótów jest fakt, że za każdym razem dynamicznie generują swoją treść. Jednak wykonywanie żądania HTTP do API Google Geocoding podczas każdego wyświetlania wpisu bloga nie będzie efektywnym rozwiązaniem, ponieważ doprowadzi do spowolnienia wczytywania każdej strony.

Alternatywą jest przechowywanie współrzędnych danego adresu w metadanych dołączanych do wpisu bloga. W ten sposób podczas kolejnego wyświetlenia wpisu bloga współrzędne geograficzne zostaną pobrane z bazy danych wraz z pozostałymi metadanymi wpisu bloga. W ten sposób unika się wykonywania dodatkowego żądania HTTP.

UWAGA

Metadane wpisu bloga dostępne w interfejsie WordPress są pobierane razem z samym wpisem bloga. Tak więc odczyt tych informacji nie powoduje wykonania dodatkowego zapytania SQL. Więcej informacji na temat metadanych znajduje się w rozdziale 11.

Zamiast pobierać współrzędne z API Google za pomocą funkcji `boj_gmap_geocode()`, można użyć funkcji proxy o nazwie `boj_gmap_get_coords()`, która sprawdza dostępność tych informacji w metadanych wpisu bloga. Jeżeli informacje nie zostaną znalezione w metadanych, będą pobrane z API Google, a następnie zapisane w metadanych w celu ich późniejszego wykorzystania.

Poniżej przedstawiono kod funkcji proxy:

```
<?php
// Konwersja adresu zapisanego w postaci zwykłego tekstu na współrzędne geograficzne.
// Współrzędne będą pobrane z metadanych, o ile to możliwe. W przeciwnym razie będą pobrane z API Google.
function boj_gmap_get_coords( $address = 'ul. Kościuszki 1c, Gliwice' ) {
    // Identyfikator bieżącego wpisu bloga.
    global $id;
    // Sprawdzenie, czy współrzędne geograficzne znajdują się w bazie danych.
    $saved = get_post_meta( $id, 'boj_gmap_addresses' );
    foreach( (array)$saved as $_saved ) {
        if( isset( $_saved['address'] ) && $_saved['address'] == $address ) {
            extract( $_saved );
            return compact( 'lat', 'long' );
        }
    }
    // Współrzędne nie są jeszcze buforowane, więc trzeba je pobrać z Google.
    $coords = boj_gmap_geocode( $address );
    if( !$coords )
        return false;
```

```
    // Buforowanie wyniku w metadanych wpisu bloga.
    add_post_meta( $id, 'boj_gmap_addresses', array(
        'address' => $address,
        'lat' => $coords['lat'],
        'long' => $coords['long']
        )
    );
    extract( $coords );
    return compact( 'lat', 'long' );
}
?>
```

Gdy adres będzie po raz pierwszy konwertowany na geokod, wywołanie `add_post_meta()` spowoduje wstawienie do metadanych (o nazwie `boj_gmap_addresses`) wpisu bloga tablicy, podobnej do przedstawionej poniżej:

```
array(
    "address" => "ul. Kościuszki 1c, Gliwice",
    "lat"     => "50.28891549",
    "long"    => "18.65953990"
)
```

Podczas kolejnego wczytywania strony współrzędne geograficzne powinny już być znalezione i pobrane wraz z metadanymi wpisu bloga.

Uzyskanie dostępu do API Google Maps

Kiedy wiadomo, w jaki sposób skonwertować adres na współrzędne geograficzne przy użyciu API Google Geocoding, można przystąpić do wykorzystania tych współrzędnych w usłudze Google Mapy za pomocą API Google Maps.

Koncepcje API

Interaktywne mapy usługi Google Mapy są tworzone za pomocą kodu JavaScript. Ten kod musi być wstawiony na stronie, na której ma zostać wyświetlona mapa. Przed rozpoczęciem integracji usługi z wtyczką warto dowiedzieć się, jak można osadzić mapę na stronie HTML.

Na początek trzeba dodać skrypt główny:

```
<script type="text/javascript" src="http://maps.google.com/maps/api/js?sensor=false">
```

Następnie należy wstawić kod JavaScript dotyczący map i mieszczący się we własnej funkcji:

```
function initialize_map() {
```

Obiekt przechowuje nowy egzemplarz Google Maps wraz z podanymi parametrami szerokości i długości geograficznej:

```
var myLatlng = new google.maps.LatLng(45.124099,-123.113634);
```

Inny obiekt może zdefiniować opcje mapy: poziom przybliżenia, wycentrowanie oraz rodzaj mapy (teren, mapa, satelita lub widok hybrydowy):

```
var myOptions = {
    zoom: 4,
    center: myLatlng,
    mapTypeId: google.maps.MapTypeId.SATELLITE
}
```

Teraz mapę można już dołączyć do obiektu HTML, takiego jak ; w omawianym przykładzie obiekt ma atrybut o identyfikatorze map_canvas:

```
var map = new google.maps.Map( document.getElementById("map_canvas"), myOptions );
```

Przedstawiony poniżej ciąg tekstowy przechowuje własny tekst wyświetlający informacje w oknie, które się pokazuje po kliknięciu znacznika znajdującego się na mapie:

```
var contentString = '<div id="content">'+
    '<p><b>Ikona Firefox</b>: Gdzieś w Oregonie, ta ikona o średnicy 67 '+
    'metrów została utworzona przez grupę użytkowników systemu Linux Uniwersytetu '+
    'Stanowego w celu świętowania wydania przeglądarki Firefox 2</p>'+
    '</div>'+
    '</div>';
```

Ten ciąg tekstowy jest teraz dołączony do nowego egzemplarza obiektu InfoWindow:

```
var infowindow = new google.maps.InfoWindow({
    content: contentString
});
```

Pozostało umieszczenie znacznika na mapie:

```
var marker = new google.maps.Marker({
    position: myLatlng,
    map: map,
    title: 'Ikona Firefox'
});
```

Trzeba jeszcze zdefiniować odpowiednie zachowanie, czyli wyświetlenie okna z informacjami po kliknięciu znacznika:

```
    google.maps.event.addListener(marker, 'click', function() {
        infowindow.open(map,marker);
    });
}
```

Prawie gotowe! Teraz tworzymy pusty obiekt HTML, który otrzyma mapę, i wywołujemy funkcję JavaScript odpowiedzialną za wyświetlenie mapy:

```
<p>Mapa jest wyświetlona poniżej:</p>
<div id="map_canvas" style="width:600px;height:600px"></div>
<script type="text/javascript">initialize_map()</script>
```

Powyższy fragment kodu pochodzi z pliku google_map_api_example.html.

W pliku przedstawiono koncepcję używania API Google Maps w celu wyświetlenia mapy (zobacz rysunek 10.4).

Teraz jesteś przygotowany do implementacji dynamicznej mapy Google we własnej wtyczce.

Implementacja wtyczki

Trzeba przygotować główną część wtyczki: funkcję rejestrującą skrót [googlemap] i wyświetlającą mapę Google. Skrót będzie używany w następujący sposób:

```
[googlemap width=500 height=300 zoom=12]ul. Kościuszki 1c, Gliwice[/googlemap]
```

Rysunek 10.4. Gotowa mapa wyświetlona w przeglądarce internetowej

Najpierw należy zarejestrować sam skrót. Można również zarejestrować podobne skróty, np. `[googlemaps]`, `[google_map]` i `[google_maps]`, które będą obsługiwane przez tę samą funkcję.

```
<?php
// Dodanie obsługi skrótu [googlemap].
add_shortcode( 'googlemap', 'boj_gmap_generate_map' );
```

Kolejny krok to rozpoczęcie definiowania funkcji odpowiedzialnej za analizę i przetworzenie atrybutów skrótu oraz jego treści:

```
// Funkcja odpowiedzialna za obsługę skrótu.
function boj_gmap_generate_map( $attr, $address ) {
    // Ustawienie wartości domyślnych mapy.
    $defaults = array(
        'width'  => '500',
        'height' => '500',
        'zoom'   => 12,
    );
    // Pobranie atrybutów mapy (w przypadku ich pominięcia będą miały wartości domyślne).
    extract( shortcode_atts( $defaults, $attr ) );
```

W części pierwszej następuje przygotowanie tablicy wartości domyślnych połączonych z rzeczywistymi atrybutami przy użyciu funkcji `shortcode_atts()`, która zwraca tablicę. Wywołanie `extract()` powoduje zaimportowanie zmiennych z tablicy w taki sposób, że dla egzemplarza `array( 'size' => 300 )` otrzymamy `$size = 300`.

```
// Pobranie współrzędnych.
$coord = boj_gmap_get_coords( $address );
// Upewniamy się, że mamy współrzędne. W przeciwnym razie trzeba zwrócić pusty ciąg tekstowy.
if( !$coord )
    return '';
```

W tym miejscu utworzono geokod na podstawie adresu (albo pobrano z API albo z bazy danych wraz z metadanymi wpisu bloga). Gdy utworzenie geokodu zakończy się niepowodzeniem (np. na skutek tymczasowego problemu między serwerami Twoim i Google), należy zwrócić pusty ciąg tekstowy.

```
// Dane wyjściowe skrótu.
$output = '';
// Przypisanie wartości zmiennym $lat i $long.
extract( $coord );
```

Po przygotowaniu wszystkich wymaganych zmiennych, a przed wygenerowaniem danych wyjściowych, trzeba je oczyścić. Niektóre będą umieszczone w ciągach tekstowych JavaScript, jeszcze inne użyte jako atrybuty elementów HTML. Konieczne jest więc prawidłowe ich oczyszczenie, zgodnie z opisem przedstawionym w rozdziale 6.

```
// Oczyszczenie zmiennych w zależności od kontekstu, w którym będą użyte.
$lat     = esc_js( $lat );
$long    = esc_js( $long );
$address = esc_js( $address );
$zoom    = esc_js( $zoom );
$width   = esc_attr( $width );
$height  = esc_attr( $height );
```

Teraz przechodzimy do kodu JavaScript.

Główny skrypt najczęściej umieszczany jest w elemencie `<head>` dokumentu HTML, ale w tym przypadku to nie jest najlepsze rozwiązanie, ponieważ skrypt będzie wówczas wstawiony nawet na tej stronie, na której nie jest potrzebny.

Lepiej skrypt osadzić w kodzie jako fragment kodu skrótu. W ten sposób znajdzie się tylko na wymagającej go stronie.

```
// Wygenerowanie unikalnego identyfikatora mapy, co pozwala mieć wiele map na stronie.
$map_id = 'boj_map_'.md5( $address );
// Tylko jednokrotne dodanie głównego skryptu Google Maps na stronie.
static $script_added = false;
if( $script_added == false ) {
    $output .= '<script type="text/javascript"
    src="http://maps.google.com/maps/api/js?sensor=false" ></script>';
    $script_added = true;
 }
```

Teraz można wstawić kod JavaScript odpowiedzialny za obsługę mapy. Każda funkcja i każde miejsce zarezerwowane dla mapy będą miały unikalne nazwy, uzyskamy to, używając wcześniej wygenerowanej zmiennej `$map_id`, co pozwala na umieszczenie wielu map na tej samej stronie.

```
// Dodanie kodu dotyczącego mapy.
$output .= <<<CODE
<div id="$map_id" ></div>
<script type="text/javascript" >
    function generate_$map_id() {
        var latlng = new google.maps.LatLng( $lat, $long );
        var options = {
            zoom: $zoom,
            center: latlng,
            mapTypeId: google.maps.MapTypeId.ROADMAP
        }
```

```
        var map = new google.maps.Map(
            document.getElementById("$map_id"),
            options
        );
        var legend = '<div class="map_legend"><p> $address </p></div> ';
        var infowindow = new google.maps.InfoWindow({
            content: legend,
        });
        var marker = new google.maps.Marker({
            position: latlng,
            map: map,
        });
        google.maps.event.addListener(marker, 'click', function() {
            infowindow.open(map,marker);
        });
    }
    generate_$map_id();
</script>
```

Do danych wyjściowych dodajemy proste style w postaci atrybutów zdefiniowanych przez użytkownika:

```
    <style type"text/css">
    .map_legend{
        width:200px;
        max-height:200px;
        min-height:100px;
    }
    #$map_id {
        width: {$width}px;
        height: {$height}px;
    }
    </style>
CODE;
```

Oczywiście, nie wolno zapomnieć o zwróceniu treści zamiennika dla skrótu:

```
        return $output;
    }
    ?>
```

Powyższy fragment kodu pochodzi z pliku plugin_boj_sc7.php.

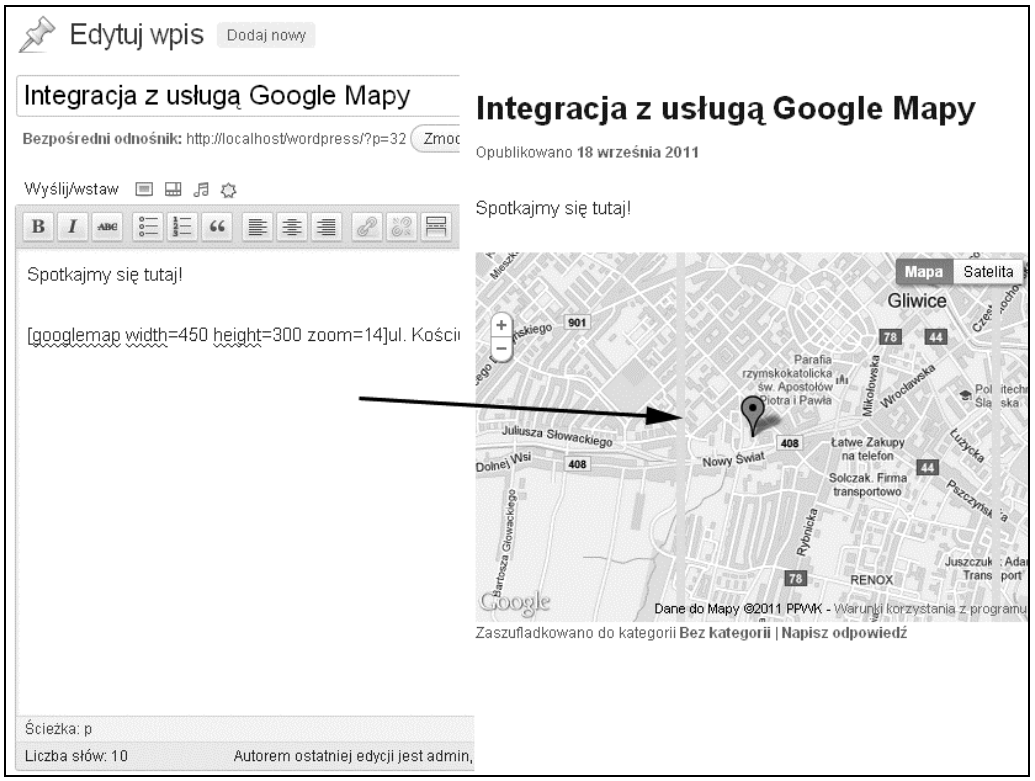
Na tym etapie wtyczka jest już gotowa do użycia! Utwórz nowy wpis bloga i zastosuj np. poniższy skrót:

```
[googlemap width=450 height=300 zoom=14]ul. Kościuszki 1c, Gliwice[/googlemap]
```

Tak utworzony wpis bloga powinien być podobny do pokazanego na rysunku 10.5.

Więcej pomysłów dotyczących skrótów

Przy użyciu skrótów można bardzo łatwo dodawać interesujące i praktyczne funkcje, tutaj ograniczeniem jest jedynie Twoja wyobraźnia. Istnieje np. możliwość zarejestrowania skrótu wyświetlającego treść tylko dla zalogowanych użytkowników, wyświetlającego treść zależną ograniczoną czasowo, zaciemniającego adres e-mail oraz wiele innych. W kolejnych punktach zostaną przedstawione niektóre opcje.



Rysunek 10.5. Wpis bloga zintegrowany z mapą Google

Wyświetlanie treści jedynie dla zalogowanych użytkowników

Pierwszy krótki implementowany skrót to elegancki sposób wyświetlenia treści jedynie zalogowanym użytkownikom. Jego sposób użycia we wpisie bloga może być następujący:

```
Dzisiejsza porada Jedi:
[czlonkowie]Używaj mocy[/czlonkowie]
```

Funkcja odpowiedzialna za obsługę wspomnianego skrótu znajduje się w poniższym fragmencie kodu:

```
<?php
add_shortcode( 'czlonkowie', 'boj_sc8_loggedin' );
function boj_sc8_loggedin( $attr, $content ) {
    if( is_user_logged_in() ) {
        return $content;
    } else {
        return "<p>Tylko dla zalogowanych użytkowników</p>";
    }
}
?>
```

Powyższy fragment kodu pochodzi z pliku plugin_boj_sc8.php.

Wynikiem jest treść umieszczona pomiędzy znacznikami skrótu `[czlonkowie]`. Będzie ona wyświetlona tylko zalogowanym użytkownikom. Przykład tej samej koncepcji, jednak omówiony bardziej szczegółowo, znajduje się w rozdziale 8.

Wyświetlenie treści ograniczonej czasowo

Innym prostym i przydatnym skrótem jest kod pozwalający na wyświetlenie treści ograniczonej czasowo, np. odnośnik promocyjny ważny przez jedynie 24 godziny:

```
Ten odnośnik promocyjny jest ważny przez jedynie 24 godziny:
[24godziny] http://przyklad.pl/promo/ [/24godziny]
```

W celu zaimplementowania takiego skrótu trzeba wykorzystać przedstawiony poniżej fragment kodu, który po prostu sprawdza bieżącą godzinę oraz godzinę opublikowania wpisu bloga:

```
<?php
add_shortcode( '24godziny', 'boj_sc8_24hours' );
function boj_sc8_24hours( $attr, $content ) {
    $now = time();
    $post_time = get_the_date( 'U' );
    if( ( $now - $post_time ) > 86400 ) {
        return 'Oferta wygasła!';
    } else {
        return $content;
    }
}
?>
```

Powyższy fragment kodu pochodzi z pliku plugin_boj_sc8.php.

Jeżeli bieżący wpis bloga był opublikowany wcześniej niż 86 400 sekund wcześniej (czyli 24 godziny), to tekst znajdujący się między znacznikami `[24godziny]` nie zostanie wyświetlony.

Zaciemnienie adresu e-mail

W kolejnym krótkim i użytecznym skrócie pokazano praktyczny sposób konwersji adresu e-mail zapisanego w postaci zwykłego tekstu na odnośnik `mailto:`, który będzie zaciemniony (tzn. mniej czytelny) dla robotów spamowych. We wpisie bloga wystarczy po prostu użyć skrótu:

```
Napisz do mnie na adres [email]ozh@ozh.org[/email]
```

Funkcja obsługująca skrót wykorzystuje funkcję WordPress o nazwie `antispambot()`, która po prostu konwertuje znaki na encje HTML, które są trudniejsze do odczytania przez roboty spamowe i roboty kolekcjonujące adresy e-mail:

```
<?php
add_shortcode( 'email', 'boj_sc8_email' );
function boj_sc8_email( $attr, $content ) {
    if( is_email( $content ) ) {
        $content = antispambot( $content );
        return sprintf( '<a href="mailto:%s" > %s <a/>', $content, $content );    }
else {
        return '';
    }
}
?>
```

Powyższy fragment kodu pochodzi z pliku plugin_boj_sc8.php.

Umieszczenie adresu e-mail między znacznikami `[email][/email]` spowoduje zaciemnienie tego adresu przez skrót. Przykładowo po podaniu adresu *ozh@ozh.org* wartością zwrotną skrótu będzie `&#122;&#104;&#64;&#111;z&#104;&#46;org`.

Podsumowanie

Skróty otwierają użytkownikom końcowym drzwi do zaawansowanej, dostosowanej do własnych potrzeb i dynamicznej treści. Ci użytkownicy nie muszą znać skomplikowanego kodu HTML, JavaScript, CSS lub PHP albo mogą nie mieć ochoty na jego tworzenie.

Za pomocą skrótów można przygotować dla klientów zaawansowane makra i znacznie zwiększyć wartość swojej pracy przy niewielkim wysiłku.

Skorowidz

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

Ś

T

U

V

W

Y

Z

Ż

Dowiedz się, jak tworzyć własne wtyczki — przejdź na wyższy poziom używania systemu WordPress!

WordPress to jeden z najpopularniejszych systemów zarządzania treścią. Jest fundamentem niezliczonych blogów, portali i stron WWW. Jego największe atuty to łatwa instalacja, przyjemna konfiguracja i niewygórowane wymagania. Jak to się stało, że zdobył aż taką popularność? Jest coś jeszcze — rozbudowany system wtyczek zwiększających funkcjonalność, dzięki którym możliwości WordPress są w zasadzie nieograniczone!

W trakcie czytania tej książki dowiesz się, jak tworzyć wtyczki i dostosowywać system WordPress do własnych potrzeb, choćby były najbardziej wymyślne. Na początku przygotujesz się do stworzenia własnej wtyczki, a także dowiesz się, jak ją instalować i odinstalowywać oraz dokumentować jej kod. W kolejnych rozdziałach nauczysz się integrować wtyczkę z systemem WordPress — tworzyć widgety, modyfikować menu, obsługiwać pola użytkownika oraz formularze. Jeżeli chcesz, aby Twoja wtyczka zdobyła popularność na całym świecie, koniecznie zapoznaj się ze sposobami tworzenia wtyczek wielojęzycznych. To jednak nie wszystko! Książka porusza wiele innych istotnych kwestii, takich jak bezpieczeństwo wtyczek, przygotowanie strony administracyjnej dla wtyczki, obsługa API. Trzymasz w rękach kompendium wiedzy na temat wtyczek w systemie WordPress. Zacznij przygodę z nimi już dziś!

- Szczegóły tworzenia wtyczek o różnym poziomie złożoności — począwszy od bardzo prostych, a skończywszy na wyjątkowo skomplikowanych pluginach typu e-commerce
- skończywszy.
- Sposoby integracji wtyczek z platformą WordPress, zapisywania ustawień, tworzenia
- widgetów i skrótów, a także implementacji funkcji deinstalacji wtyczki.
- Stosowanie poprawnych technik przechowywania danych, dostosowywanie ról użytkowników
- oraz najlepsze praktyki z zakresu implementacji zabezpieczeń we wtyczkach.
- Wykorzystanie procedur pozwalających na używanie własnych typów wpisów na blogu, a także na tworzenie i używanie własnych taksonomii.
- Prezentacja API Http, JavaScript, Ajax, Cron, API Rewrite i wiele innych.

Brad Williams to CEO oraz współzałożyciel witryny *webdevstudios.com*, a także autor książki *Professional WordPress* (Wrox, 2010).
Ozh Richard to autor wielu popularnych wtyczek dla WordPressa, a także laureat konkursu Annual *WordPress Plugin Competition*.
Justin Tadlock opracował wiele wtyczek. Na swojej stronie *ThemeHybrid.com* poświęconej platformie WordPress uczy użytkowników, w jaki sposób używać wtyczek i motywów.

ISBN 978-83-246-3564-1
9 788324 635641
Cena: 89,00 zł