



Thord Daniel Hedengren

PODREČZNIK

WordPressa

SMASHING MAGAZINE

Najlepszy podręcznik o WordPressie!

Tytuł oryginału: Smashing WordPress: Beyond the Blog

Tłumaczenie: Łukasz Piwko

ISBN: 978-83-246-6678-2

Translation copyright © 2013 by Helion S.A.

This edition first published 2012

© 2012 John Wiley & Sons, Ltd.

All Rights Reserved. Authorised translation from the English language edition published by John Wiley & Sons Limited. Responsibility for the accuracy of the translation rests solely with Helion S.A. and is not the responsibility of John Wiley & Sons Limited.

Wiley and the Wiley logo are trademarks or registered trademarks of John Wiley and Sons, Inc. and/ or its affiliates in the United States and/or other countries, and may not be used without written permission. WordPress is a registered trademark of Automattic, Inc. All other trademarks are the property of their respective owners. John Wiley & Sons, Ltd. is not associated with any product or vendor mentioned in the book.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz Wydawnictwo HELION dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Wydawnictwo HELION nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

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/podwsm.zip>

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

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

Spis treści

| | |
|---|-----------|
| O autorze | 13 |
| Wprowadzenie | 15 |
| CZĘŚĆ I PODSTAWY WORDPRESSA | 19 |
| Rozdział 1. Anatomia instalatora WordPressa | 21 |
| Podstawowa instalacja | 22 |
| Instalacja z kreatorem | 22 |
| Instalacja ręczna | 23 |
| Korzystanie z zewnętrznego serwera baz danych | 26 |
| Inne ustawienia bazy danych | 26 |
| Przydatne funkcje pliku wp-config.php | 27 |
| Kilka słów na temat instalatorów | 28 |
| Przenoszenie instalacji WordPressa do nowego katalogu | 29 |
| Modyfikowanie bazy danych | 30 |
| Struktura bazy danych WordPressa | 31 |
| Usuwanie problemów bezpośrednio w bazie danych | 31 |
| Robienie kopii zapasowej | 32 |
| Zmianie hosta | 34 |
| Narzędzia eksportu i importu | 34 |
| Problemy z importowaniem i eksportowaniem danych | 36 |
| Zabezpieczanie WordPressa | 38 |
| Użytkownicy i hasła | 39 |
| Zabezpieczenia po stronie serwera | 39 |
| W następnym rozdziale | 40 |
| Rozdział 2. Składnia WordPressa | 43 |
| WordPress i PHP | 44 |
| Dokumentacja WordPressa | 44 |
| Rdzeń WordPressa | 44 |
| Motywy i szablony | 46 |
| Tagi szablonowe | 49 |
| Tagi dołączania plików | 49 |
| Przekazywanie kilku parametrów do tagów szablonowych | 51 |
| Argumenty w stylu funkcji i łańcuchów zapytań | 52 |
| Typy danych | 54 |

| | |
|---|-----------|
| Tagi warunkowe | 55 |
| Co w nich takiego wyjątkowego | 56 |
| W następnym rozdziale: pętla | 57 |
| Rozdział 3. Pętla | 59 |
| Zasada działania pętli WordPressa | 60 |
| Najprostsza pętla | 60 |
| Zapisywanie pętli w pliku szablonowym loop.php | 61 |
| Kilka słów o WP_Query | 62 |
| Używanie pętli | 63 |
| Przyklejanie wpisów | 69 |
| Formaty wpisów | 72 |
| Tag get_template_part() i formaty wpisów | 74 |
| Funkcja query_posts() | 75 |
| Co zamiast pętli | 78 |
| Tworzenie wielu pętli | 79 |
| Wyświetlanie proponowanych artykułów | 80 |
| To było niezłe, ale cztery pętle to dopiero coś | 82 |
| Oswajanie się z pętlą | 86 |

CZĘŚĆ II PROJEKTOWANIE I PROGRAMOWANIE MOTYWÓW WORDPRESSA 89

| | |
|---|-----------|
| Rozdział 4. Motywy do WordPressa — wiadomości podstawowe | 91 |
| Podstawy budowy motywu | 92 |
| Podstawowe elementy motywu | 92 |
| Co będziemy robić | 93 |
| Kilka słów na temat języka HTML5 | 94 |
| Tworzenie plików szablonowych | 95 |
| Deklaracja motywu w pliku style.css | 95 |
| Plik header.php | 96 |
| Plik footer.php | 100 |
| Prawa kolumna: plik sidebar.php | 102 |
| Treść główna: plik index.php | 103 |
| Przenoszenie pętli do osobnego pliku | 109 |
| Pojedyncze wpisy i strony | 111 |
| Szablony archiwów | 114 |
| Błędy 404, wyszukiwarka i zrzuty ekranu | 116 |
| Plik functions.php | 117 |
| Pliki szablonowe | 118 |
| Kiedy używane są poszczególne pliki szablonowe | 120 |
| Hierarchia szablonów | 121 |
| Szablony stron | 121 |
| Korzystanie z pliku functions.php | 124 |
| Ustawianie domyślnej szerokości | 125 |
| Dodawanie elementów promocyjnych za pomocą pliku functions.php | 126 |

| | |
|--|------------|
| Widżety — czym są i do czego służą | 127 |
| Deklarowanie obszarów na widżety | 128 |
| Deklarowanie wielu obszarów na widżety | 128 |
| Dostosowywanie widżetów | 129 |
| Upiększanie komentarzy | 130 |
| Podział komentarzy na wątki | 131 |
| Wyróżnianie autora wpisu | 133 |
| Dodawanie własnych pól | 133 |
| Najczęstsze zastosowanie własnych pól | 133 |
| Kwestia użyteczności | 134 |
| Tworzenie motywu bazowego | 134 |
| Publikowanie motywu | 136 |
| Lista punktów do sprawdzenia przed publikacją motywu | 136 |
| Motywy komercyjne a licencja GPL | 139 |
| Zgłaszanie motywów do WordPress.org | 139 |
| W następnym rozdziale | 142 |
| Rozdział 5. Motywy potomne | 143 |
| Genialność motywów potomnych | 144 |
| Jak działają motywy potomne | 145 |
| Piękno techniki przesłaniania plików szablonowych | 147 |
| Wspaniały szablon loop.php | 147 |
| Motywy potomne do zastosowań specjalnych | 149 |
| Inne spojrzenie na kwestię dziedziczenia | 149 |
| Często spotykane problemy | 150 |
| Motywy potomne a zarządzanie wieloma witrynami | 150 |
| Zarządzanie projektami wielu witryn | 151 |
| Nie zapominaj o pliku functions.php | 152 |
| Nie należy przesadzać | 152 |
| Szkielety motywów | 153 |
| Dla odmiany kilka słów o motywach nadrzędnych | 153 |
| Wyższy poziom wtajemniczenia | 154 |
| Rozdział 6. Motywy dla zaawansowanych | 155 |
| Planowanie motywu | 156 |
| Zasada 1. Stylizuj według kategorii, sortuj według tagów, a dostosowuj według formatów wpisów | 157 |
| Zasada 2. Starannie przemyśl własne pola | 157 |
| Zasada 3. Używaj stron i własnych typów wpisów | 158 |
| Czy to wszystko? | 158 |
| Indywidualne techniki stylizacji | 158 |
| Stylizowanie wpisów | 159 |
| Klasy dla elementu body | 161 |
| Przyklejone wpisy | 163 |
| Używanie własnych pól | 164 |
| Podstawy własnych pól | 165 |
| Tworzenie modułów meta | 166 |

| | |
|--|-----|
| Ciekawe funkcje własne | 167 |
| Poprawne dodawanie funkcji w pliku functions.php | 168 |
| Ikony wpisów | 169 |
| Własne menu | 170 |
| Własne nagłówki | 170 |
| Własne obrazy tła | 171 |
| Haki akcji | 171 |
| Używanie haków | 172 |
| Tworzenie własnych haków | 173 |
| Odłączanie akcji od haków | 174 |
| Taksonomie | 174 |
| Zastosowania taksonomii | 175 |
| Myśl | 175 |
| Taksonomie a przenośność | 176 |
| Własne typy wpisów | 176 |
| Używanie własnych typów wpisów | 177 |
| Używanie własnych typów wpisów w motywach | 177 |
| Strony opcji motywu | 177 |
| Problemy dotyczące opcji motywów | 178 |
| Obsługa różnych języków | 179 |
| Praca z plikami językowymi | 180 |
| Problem z nazwami | 181 |
| Kanały RSS | 181 |
| Kanały WordPressa | 182 |
| Tworzenie własnego kanału RSS | 183 |
| Podstawowe kwestie SEO | 183 |
| Pozbywanie się niepotrzebnych rzeczy z motywu | 185 |
| Motywy a wtyczki | 186 |

CZĘŚĆ III WTYCZKI DO WORDPRESSA 189

| | |
|---|------------|
| Rozdział 7. Anatomia wtyczki do WordPressa | 191 |
| Rodzaje wtyczek | 192 |
| Zwykłe wtyczki | 192 |
| Wtyczki do rdzenia | 192 |
| Wtyczki obowiązkowe | 193 |
| Tworzenie wtyczek do użytku w sieciach witryn | 193 |
| Wtyczki dla całej sieci | 194 |
| Podstawy budowy wtyczek | 195 |
| Metody inkorporowania wtyczek | 197 |
| Używanie haków | 197 |
| Tworzenie własnych tagów szablonowych | 198 |
| Funkcje nadpisujące | 199 |
| Własne taksonomie i typy wpisów | 199 |
| Powody, aby użyć wtyczki | 200 |
| Tworzenie własnej taksonomii | 200 |
| Tworzenie własnego typu wpisów | 202 |

| | |
|---|------------|
| Co powinna mieć każda wtyczka | 204 |
| Ustawienia wtyczek | 204 |
| Baza danych a odinstalowywanie wtyczki | 209 |
| Po deinstalacji | 210 |
| Wtyczki tworzące widżety | 211 |
| Tworzenie widżetu | 212 |
| Widżety kokpitu | 214 |
| Kwestia korzystania z bazy danych we wtyczkach | 216 |
| Zgodność wsteczna wtyczek | 217 |
| Publikowanie wtyczek w portalu WordPress.org | 217 |
| Słowo ostrzeżenia na temat tworzenia wtyczek | 218 |
| Rozdział 8. Wtyczka czy plik functions.php? | 221 |
| Kiedy tworzyć wtyczki | 222 |
| Rozszerzanie funkcjonalności za pomocą wtyczek | 222 |
| Ostrzeżenie: wtyczki mogą spowolnić Twoją witrynę | 222 |
| Kiedy używać pliku functions.php | 223 |
| Dylemat ze skrótami kodowymi | 224 |
| Rozwiązanie problemu poprzez użycie motywu potomnego | 225 |
| Rozwiązanie uniwersalne: wtyczka na funkcje | 225 |
| Tworzenie wtyczki na funkcje | 225 |
| Jak ważna jest przenośność | 227 |
| Planowanie rozszerzania funkcjonalności WordPressa | 228 |
| CZĘŚĆ IV DODATKOWE FUNKCJE I ROZSZERZANIE FUNKCJONALNOŚCI WORDPRESSA | 229 |
| Rozdział 9. Używanie WordPressa jako systemu CMS | 231 |
| Czy WordPress jako CMS to dobry wybór | 232 |
| Lista punktów do sprawdzenia zanim wybierze się WordPressa | 233 |
| Ograniczanie WordPressa do minimum | 234 |
| Dostosowywanie panelu administracyjnego | 235 |
| Usuwanie funkcji typowych dla bloga | 236 |
| Idealna konfiguracja prostej statycznej witryny | 237 |
| Bardziej zaawansowane rozwiązania | 239 |
| Zastosowanie własnych typów wpisów i taksonomii | 239 |
| w WordPressie używanym jako CMS | 239 |
| Wykorzystanie widżetów w CMS-ie | 240 |
| Obsługa menu | 242 |
| Integracja treści spoza WordPressa | 242 |
| Nie zapomnij dodać instrukcji obsługi | 244 |
| Ostatnie słowo na temat używania WordPressa jako systemu CMS | 244 |
| Rozdział 10. Integracja WordPressa z mediami społecznościowymi | 245 |
| Integracja WordPressa z Facebookiem | 246 |
| Przycisk Lubię to | 246 |
| Widżety profilowe | 248 |

| | |
|---|------------|
| Integracja WordPressa z Twitterem | 248 |
| Dodawanie przycisków i widżetów Twittera do strony | 249 |
| Rozszerzenia Twittera | 250 |
| Integracja witryny z Google+ | 252 |
| Korzystanie z zewnętrznej obsługi komentarzy | 253 |
| Jeden login do wszystkich serwisów | 254 |
| Jak ważne są media społecznościowe | 256 |
| Rozdział 11. Sztuczki projektowe | 257 |
| Zwiększanie kontroli nad wpisami | 258 |
| Tworzenie projektów opartych na tagach | 258 |
| Używanie własnych pól | 259 |
| Podpinanie się do funkcji <code>body_class()</code> , <code>post_class()</code> oraz <code>comment_class()</code> | 261 |
| Dodawanie własnych taksonomii | 261 |
| Ulepszanie menu | 262 |
| Przesuwane drzewi | 263 |
| Menu rozwijane | 265 |
| Wstawianie reklam w pętli | 266 |
| Tworzenie pomocnych stron błędu 404 | 268 |
| Używanie bibliotek JavaScript w WordPressie | 268 |
| Rejestrowanie skryptów | 269 |
| Dostosowywanie stylu WordPressa do własnej marki | 270 |
| Własny formularz logowania | 271 |
| Motywy panelu administracyjnego | 272 |
| Dopieszczanie witryny | 274 |
| Rozdział 12. Zabawa z mediami | 275 |
| Tworzenie galerii obrazów | 276 |
| Stylizowanie galerii | 277 |
| Lepsze przeglądanie w lekkich okienkach | 280 |
| Galerie spoza WordPressa | 281 |
| Formaty wpisów | 282 |
| Osadzanie treści multimedialnej na stronach | 283 |
| Konfigurowanie ustawień | 283 |
| Magiczna technika <code>oEmbed</code> | 284 |
| Wyświetlanie losowych obrazów | 284 |
| Wyświetlanie losowych obrazów z galerii | 285 |
| Dodatkowe opcje losowania obrazów | 286 |
| Optymalne wykorzystanie serwisów do publikowania zdjęć | 287 |
| Przechowywanie obrazów w serwisie Flickr | 288 |
| Używanie pokazów slajdów z serwisu Flickr | 290 |
| Strzeż się bałaganu | 292 |
| Rozdział 13. Dodatkowe funkcje | 293 |
| Wyświetlanie treści na kartach | 294 |
| Inteligentne zastosowanie | 294 |
| Używać kart czy nie | 297 |

| | |
|--|------------|
| Wyświetlanie zawartości kanałów RSS | 297 |
| Wbudowany parser | 298 |
| Buforowanie przy użyciu API Transients | 299 |
| Mieszanie kanałów za pomocą SimplePie | 300 |
| Własne skróty kodowe | 301 |
| Dodawanie skrótów kodowych | 301 |
| Ciekawostki dotyczące skrótów kodowych | 302 |
| Wysyłanie e-maili z WordPressa | 303 |
| Dodawanie formularza logowania | 304 |
| Drukowanie treści | 306 |
| Więcej... | 308 |
| Rozdział 14. Nietypowe zastosowania WordPressa | 309 |
| Publikowanie treści dostarczanej przez użytkowników | 310 |
| Przyjmowanie wpisów od użytkowników | 311 |
| Obsługa wiadomości i recenzji publikowanych przez użytkowników | 312 |
| Tworzenie tablicy ogłoszeń o pracę | 313 |
| Funkcja wp_editor() | 315 |
| Ostatnie słowo na temat treści dodawanej przez użytkowników | 316 |
| WordPress jako baza wiedzy | 316 |
| Dodawanie funkcji | 317 |
| Dodatkowe ulepszenia | 318 |
| WordPress i handel elektroniczny | 319 |
| Prowadzenie sklepu opartego na WordPressie | 320 |
| Sprzedawanie produktów cyfrowych | 320 |
| Budowa sklepu | 321 |
| Tworzenie katalogu produktów | 321 |
| Tworzenie typu wpisów dla książek | 322 |
| Tworzenie strony dla książek | 323 |
| Promowanie produktów | 326 |
| Blog ze smakiem, czyli witryna z przepisami | 328 |
| Przystawka, czyli wybór motywu | 329 |
| Danie główne — przepisy jako typ wpisów | 329 |
| Deser — własne taksonomie | 330 |
| Ziółko na trawienie — podsumowanie | 334 |
| Tworzenie witryny z odnośnikami | 334 |
| Alternatywne rozwiązanie: odnośnikowy format wpisów | 336 |
| Kilka uwag na temat zastosowań | 337 |
| Mieszanie wpisów odnośnikowych ze zwykłą treścią | 337 |
| Inne zastosowania WordPressa | 338 |
| Strona z wydarzeniami i kalendarz | 338 |
| Intranet i współpraca | 339 |
| Społeczności i fora | 339 |
| Bazy danych | 340 |
| Statyczne witryny | 340 |
| Dzienniki i notatki | 340 |
| Możesz mieć wszystko, czego chcesz | 341 |

| | | |
|------------------|--|------------|
| CZĘŚĆ V | DODATKI | 343 |
| Dodatek A | Niezbędne wtyczki do WordPressa | 345 |
| | Wtyczki związane z treścią | 346 |
| | Wtyczki multimedialne | 347 |
| | Wtyczki administracyjne | 348 |
| | Wtyczki do zarządzania komentarzami i eliminowania spamu | 353 |
| | Wtyczki mediów społecznościowych | 354 |
| | Wtyczki subskrypcji i do obsługi urządzeń przenośnych | 355 |
| | Wtyczki dotyczące SEO i wyszukiwania | 356 |
| | Kod źródłowy i dane wyjściowe | 358 |
| | Przestroga na zakończenie: czy na pewno potrzebujesz tej wtyczki | 360 |
| Dodatek B | Motywy bazowe | 361 |
| | Jak wybrać motyw | 362 |
| | Znaczenie słowa szkielet | 362 |
| | Propozycje motywów | 363 |
| | Motywy Twenty Ten i Twenty Eleven | 363 |
| | Starkers | 364 |
| | Roots | 365 |
| | Toolbox | 366 |
| | Constellation | 367 |
| | Spectacular | 368 |
| | Bones | 369 |
| | Twój motyw, Twoje zasady | 369 |
| Skorowidz | | 371 |

7

ANATOMIA WTYCZKI
DO WORDPRESSA

NIE MA WĄTPLIWOŚCI, że wtyczki nie są tym samym co motywy, choć można znaleźć wiele łączących je podobieństw. Można powiedzieć, że gdy implementuje się jakąkolwiek funkcję w pliku *functions.php*, to w istocie pisze się wtyczkę.

Dzieli je jednak ogromna różnica. Motywy służą do prezentowania treści witryny przy użyciu dostępnych narzędzi. Natomiast wtyczki służą do rozszerzania funkcjonalności WordPressa o dodatkowe funkcje. Należy o tym pamiętać, ponieważ rozbudowywanie pliku *functions.php* w nieskończoność wcale nie jest najlepszym rozwiązaniem.

W tym rozdziale spojrzymy na wtyczki z nieco innej perspektywy niż do tej pory. Za pomocą wtyczki można zrobić wszystko. Ogólnie rzecz biorąc, wtyczki są metodą pozwalającą dodać do WordPressa dowolną funkcję bez żadnych ograniczeń. Porównaj to z kombinowaniem przy użyciu kilku tagów w plikach szablonowych motywu. W przypadku wtyczek kwestią nie jest, **co** można zrobić, lecz **po co** miałoby się coś robić.

RODZAJE WTYCZEK

Wyróżnia się trzy główne rodzaje wtyczek: zwykłe wtyczki, których na pewno nieraz używałeś, wtyczki do rdzenia (ang. *drop-in*), które zastępują rdzenne funkcje, oraz wtyczki obowiązkowe.

ZWYKŁE WTYCZKI

Pod pojęciem **zwykłych wtyczek** rozumiem wtyczki, do których używania jesteś przyzwyczajony. Są to wtyczki, które się pobiera z internetu oraz włącza, aby rozpocząć ich używanie. Taką zwykłą wtyczką jest np. Akismet (<http://wordpress.org/extend/plugins/akismet>). Aby taka wtyczka działała, wystarczy ją po prostu włączyć, ewentualnie skonfigurować jakieś drobne ustawienia. Zwykłe wtyczki są przechowywane w folderze *wp-content/plugins/*.

W zasadzie o tym rodzaju wtyczek wszystko już wiesz, a więc przejdźmy do omówienia następnych typów.

WTYCZKI DO RDZENIA

Wtyczki do rdzenia przesłaniają rdzenne funkcje systemu. Umieszcza się je bezpośrednio w folderze *wp-content* pod nazwą odpowiadającą plikowi, który mają zastąpić, np. *advanced-cache.php* albo *db.php*.

Poniżej podano dostępne wtyczki tego typu. Pamiętaj, że jeśli zdecydujesz się na ich użycie, musisz rzeczywiście napisać własny kod zastępujący standardowe skrypty. Jeśli tego nie zrobisz, prawie na pewno będziesz mieć problemy.

- *advanced-cache.php* — własne skrypty zaawansowanego buforowania;
- *db.php* — własna klasa bazy danych;
- *db-error.php* — własne powiadomienia o błędach bazy danych;
- *install.php* — własne skrypty instalacyjne;
- *maintenance.php* — własne wiadomości dotyczące spraw utrzymania serwisu;
- *object-cache.php* — zewnętrzne buforowanie;
- *sunrise.php* — skrypty, które mają zostać wykonane przed załadowaniem sieci witryn;
- *blog-deleted.php* — usuwanie blogów w sieci witryn;
- *blog-inactive.php* — wiadomości o nieaktywnych blogach w sieci witryn;
- *blog-suspended.php* — wiadomości o zawieszonych blogach w sieci witryn.

Wtyczek tych należy używać bardzo ostrożnie, chociaż dają one naprawdę bardzo duże możliwości. Możliwe, że niektórych z nich zdarzało Ci się używać, zapewne jako części innych wtyczek. W szczególności dotyczy to wtyczki *advanced-cache.php*. Zabawa z niektórymi jest mniej ryzykowna, np. wtyczka *maintenance.php* umożliwia wyświetlenie własnej wiadomości na temat uaktualniania instalacji systemu.

WTYCZKI OBOWIĄZKOWE

Wtyczki obowiązkowe różnią się od zwykłych wtyczek. Ich pliki przechowywane są w folderze *wp-content/mu-plugins/* i nie da się ich wyłączyć w panelu administracyjnym WordPressa. Jedyny sposób na ich dezaktywację to usunięcie ich z folderu na serwerze. Wtyczki obowiązkowe nie muszą zawierać specjalnego nagłówka. Bez niego również zostaną wykonane.

W folderze *wp-content/mu-plugins/* można umieścić dowolną wtyczkę, ale z niektórymi wtyczkami mogą być problemy, zwłaszcza w sieciach witryn. Dlatego postępuj ostrożnie. Najlepiej wtyczek używać w sposób zgodny z ich przeznaczeniem.

Wtyczki obowiązkowe są najlepszym rozwiązaniem, gdy chcemy mieć pewność, że jakieś funkcje nie zostaną przypadkowo wyłączone.

TWORZENIE WTYCZEK DO UŻYTKU W SIECIACH WITRYN

Od WordPressa 3.0 wersja systemu dla wielu użytkowników, zwana **WordPress MU**, stała się częścią podstawowej wersji. Teraz jest to tzw. funkcja tworzenia wielu witryn (ang. *multisite*), dzięki której można tworzyć sieci witryn. Większość wtyczek i motywów dobrze działa w tych sieciach. Problemy mogą się pojawić jedynie wtedy, gdy wtyczka będzie dodawać tabele do bazy danych albo modyfikować istniejące tabele rdzenia. Funkcję sieci witryn włącza się poprzez wpisanie kilku wierszy kodu w pliku *wp-config.php*. Na początek należy dodać poniższy wiersz kodu nad komentarzem */* To wszystko, zakończ edycję w tym miejscu!* Miłego blogowania! **/*:

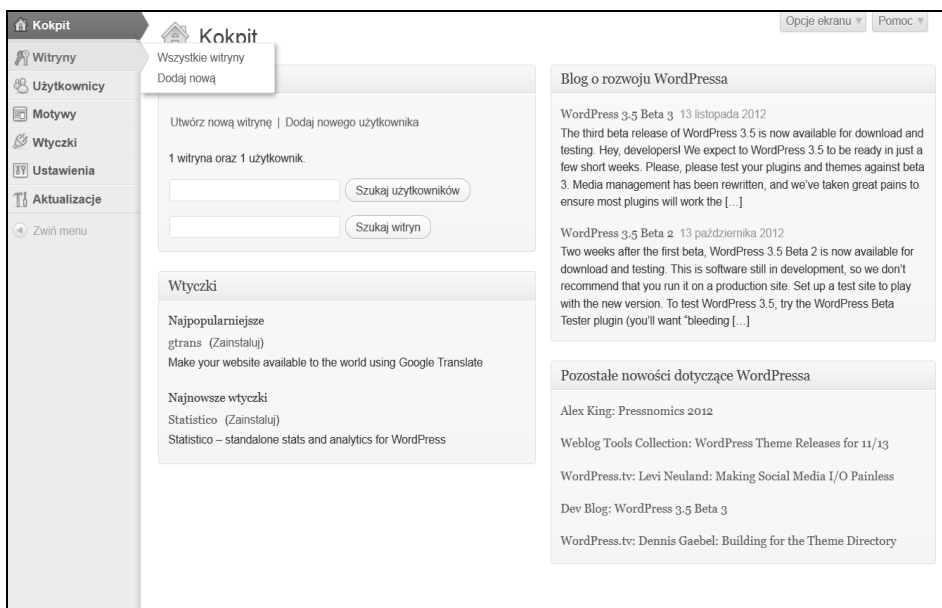
```
define( 'WP_ALLOW_MULTISITE', true );
```

Dodanie tego wiersza kodu do pliku *wp-config.php* spowoduje pojawienie się nowego odnośnika w panelu administracyjnym (rysunek 7.1). Gdy go klikniesz, zostaną wyświetlone proste instrukcje, według których należy postępować, aby uruchomić sieć witryn. Bardziej szczegółowe informacje na ten temat znajdują się na stronie http://codex.wordpress.org/Create_A_Network.

Sieć witryn pozwala uruchomić coś w rodzaju nadrzędnego panelu administracyjnego, w którym administrator może zarządzać wieloma witrynami utworzonymi w jego sieci. W serwisie WordPress.com każdy może założyć własnego bloga, ale sieć witryn niekoniecznie musi działać właśnie w ten sposób. Równie dobrze można ją wykorzystać do uruchomienia wielu witryn, nie pozwalając użytkownikom tworzyć własnych blogów.

Tworzenie wtyczek przeznaczonych do użytku w sieciach witryn niewiele różni się od tworzenia zwykłych wtyczek. Największa różnica dotyczy bazy danych i w mniejszym stopniu struktury katalogów.

Jeśli chodzi o strukturę katalogów, to prawie wszystko jest tak samo jak w standardowym WordPressie. Różnica polega na dodaniu katalogu *blogs.dir* do folderu *wp-content*, w którym przechowywane są wszystkie dane utworzonych witryn, takie jak obrazy i inne pliki. Z folderu



Rysunek 7.1. Panel administracji siecią witryn

tego nie będziesz często korzystać, ponieważ motywy i wtyczki należą do folderu *wp-content*, tak jak zawsze.

Jeśli chcesz, aby wybrane wtyczki były włączone w całej sieci, możesz je aktywować w panelu administracyjnym sieci. Możesz też skorzystać z wtyczek obowiązkowych, czyli po prostu umieścić wybrane wtyczki w folderze *wp-content/mu-plugins/*, chociaż to może przysporzyć Ci problemów, jeśli któraś z tych wtyczek nie będzie przystosowana do takiego sposobu użycia. Zwykle najlepszym rozwiązaniem jest aktywowanie wtyczki dla całej sieci.

Sam proces powstawania wtyczki wygląda tak samo, jak dla normalnego WordPressa. Trzeba tylko bardziej uważać podczas tworzenia nowych tabel w bazie danych oraz przy pobieraniu treści z tabel rdzenia. W większości przypadków czynności te nie sprawiają kłopotów, ale baza danych dla sieci witryn ma trochę inną strukturę, więc należy uważać.

Kolejną kwestią, jaką należy wziąć pod uwagę podczas pisania wtyczki dla sieci witryn, jest jej przewidywany sposób użycia. Sieć witryn może być prowadzona na wiele różnych sposobów, może być otwarta lub zamknięta, użytkownicy mogą mieć możliwość używania wtyczek lub nie itd. Trzeba to wszystko wziąć pod uwagę, gdy tworzy się wtyczkę.

WTYCZKI DLA CAŁEJ SIECI

Wtyczki można też włączać dla całej sieci witryn w panelu administracyjnym sieci. W ten sposób można włączać wtyczki znajdujące się w folderze *wp-content/plugins/* dla wszystkich witryn. To jest oczywiście bardzo wygodne rozwiązanie i należy z niego korzystać, zamiast używać wtyczek obowiązkowych. Skoro WordPress może być automatycznie aktualizowany, to im więcej ustawień można zdefiniować kliknięciem w panelu administracyjnym, tym lepiej.

PODSTAWY BUDOWY WTYCZEK

Podstawy budowy wtyczek są podobne do podstaw budowy motywów:

- Główny plik wtyczki musi być w formacie PHP i mieć niepowtarzalną nazwę albo znajdować się w folderze o niepowtarzalnej nazwie, jeśli wtyczka składa się z wielu plików.
- Główny plik PHP wtyczki musi mieć specjalny identyfikujący go nagłówek, podobny do pliku *style.css* w motywach.

Wtyczka może składać się z pliku głównego i wielu plików zawierających różne funkcje, podobnie jak motyw może składać się z wielu plików szablonowych i pliku *style.css*.

Zanim przejdziemy do szczegółowego omawiania podstaw budowy wtyczek, muszę Cię ostrzec, że tworzenie wtyczek jest znacznie bardziej wymagającym zadaniem niż tworzenie motywów. Do tego potrzebna jest solidna znajomość języka PHP i jeśli masz w tej kategorii braki, lepiej trochę się podszkół, zanim zaczniesz pisać jakąś poważniejszą wtyczkę.

Bardzo ważną kwestią jest wybór nazwy pliku lub folderu wtyczki, ponieważ wtyczka ta będzie zapisana w folderze *wp-content/plugins/* z innymi wtyczkami, a więc może dojść do konfliktów nazw. Nazwij zatem swoją wtyczkę w taki sposób, aby ktoś, kto będzie jej szukał na FTP, bez problemu mógł ją znaleźć, znając jej nazwę tylko z panelu administracyjnego WordPressa.

Blok identyfikacyjny wtyczki wygląda znajomo. Poniżej znajduje się przykład takiego bloku:

```
<?php
/*
Plugin Name: Moja wtyczka
Plugin URI: http://url-mojej-wtyczki.com/
Description: Opis mojej wtyczki.
Version: 1.0
Author: Thord Daniel Hedengren
Author URI: http://tdh.me/
*/
?>
```

Tak naprawdę obowiązkowo nagłówek musi zawierać tylko nazwę wtyczki, która w powyższym przykładzie została zapisana w pierwszym wierszu komentarza. Jednak pozostałe informacje również należy dodawać, aby użytkownik mógł przeczytać, co to za wtyczka, skąd pobierać aktualizacje, jaki jest numer wersji itd.

Powinno się także dodać informację o licencji. W dokumentacji WordPressa zalecane jest używanie poniższego standardowego wyciągu z tekstu licencji GPL:

```
<?php
/* Copyright ROK AUTOR (e-mail : E-MAIL AUTORA WTYCZKI)

This program is free software; you can redistribute it and/or modify it under the terms of the GNU
General Public License as published by the Free Software Foundation; either version 2 of the License,
or (at your option) any later version.
```

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA
*/
?>

Oczywiście napisy ROK, AUTOR i E-MAIL AUTORA WTYCZKI należy zastąpić prawdziwymi informacjami. Można także dołączyć cały tekst licencji GPL w pliku o nazwie *license.txt*. Tekst ten jest dostępny pod adresem www.gnu.org/copyleft/gpl.html.

To wystarczy, aby WordPress znalazł i poprawnie zidentyfikował wtyczkę. Jeśli umieścisz ją w folderze *wp-content/plugins/*, zostanie wyświetlona na liście wtyczek sekcji *Wtyczki* w panelu administracyjnym systemu. Można ją tam włączyć, aby była dostępna w motywie i akcjach WordPressa.

Od tego momentu zaczyna się prawdziwa zabawa, ponieważ teraz musisz zastanowić się, co wtyczka ma robić i jak sprawić, żeby to robiła.

Niezależnie od tego, czy planowana wtyczka ma zmienić sposób działania WordPressa, czy tylko dodać do niego jakąś funkcję, przed rozpoczęciem pracy powinienś przejrzeć poniższą listę punktów do sprawdzenia. Dzięki temu możesz zaoszczędzić sobie sporo czasu i problemów.

- Czy taka wtyczka już istnieje? Jeśli tak, zastanów się, czy nie lepiej jej użyć, zamiast tworzyć nową o prawie identycznym działaniu.
- Upewnij się, że nazwa Twojej wtyczki nie jest już zajęta. Sprawdź nie tylko w zasobach WordPress.org, ale również w Google. Dobrym sposobem na utworzenie niepowtarzalnej nazwy jest wstawienie na początku nazwy firmy, np. *acme_nazwawtyczki*.
- Nazwy wszystkich funkcji konsekwentnie zaczynaj od jakiegoś przedrostka. W ten sposób unikniesz konfliktów nazw z innymi funkcjami. To ważne: przedrostki stosuj przed wszystkimi nazwami!
- Czy chcesz umożliwić przetłumaczenie wtyczki na różne języki? Powinieneś. Przygotowywanie wtyczki do internacjonalizacji wygląda tak samo jak w przypadku motyów, a więc jest bardzo łatwe.
- Czy wtyczka będzie tworzyła widżety? Jeśli tak, to jakie ustawienia powinna udostępniać?
- Czy potrzebna będzie strona ustawień w panelu administracyjnym? Staraj się, aby ustawień było jak najmniej, ponieważ użytkownicy wolą proste rozwiązania.
- Na jakiej licencji będzie udostępniana wtyczka? Pamiętaj, że musi być to licencja zgodna z GPL, jeśli chcesz umieścić wtyczkę w zbiorach WordPress.org.
- Na koniec sprawdź, czy nagłówek jest aktualny, czy numer wersji jest poprawny, czy wszystkie łącza działają, czy w paczce są wszystkie pliki oraz czy nie ma błędów w tekście.

METODY INKORPOROWANIA WTYCZEK

Do budowy wtyczek o wiele częściej używa się zwykłego kodu PHP niż przy budowie motywów. Podczas gdy tagów szablonowych i warunkowych także można używać, to jednak w zdecydowanej większości przypadków będziesz pisać własne funkcje. Oczywiście dużo zależy też od rodzaju tworzonej wtyczki, jednak ogólnie rzecz biorąc, kod wtyczki jest w większości dziełem jej autora, a nie autorów WordPressa.

Działanie wtyczek zwykle opiera się na własnych funkcjach, które programista podłącza do haków i filtrów WordPressa. Umieszczając funkcję w odpowiednim miejscu, np. podłączając do `wp_head` lub komentarzy, możesz ją uruchomić w odpowiednim czasie.

W kolejnych trzech podrozdziałach znajduje się opis trzech sposobów pisania i używania wtyczek. Zaczniemy od jak zawsze ważnych haków.

UŻYWANIE HAKÓW

Pamiętasz opis haków z poprzedniego rozdziału? **Haki akcji** są wyzwalane przez określone zdarzenia podczas działania WordPressa, np. dla publikacji wpisu jest hak `publish_post`. Drugi rodzaj haków to **haki filtrów**. Są to funkcje, przez które WordPress przekazuje dane, a więc można ich używać do obróbki danych. Do przydatnych haków filtrów zaliczają się np. `the_excerpt` i `the_title`. Trzeba je odróżniać od tagów szablonowych.

Haki są przydatne także przy budowie wtyczek, nie tylko motywów. Dzięki nim można podłączyć wtyczki do odpowiednich części WordPressa, np. `wp_head` albo `wp_footer`. Dobrym tego przykładem jest dodanie funkcji w pliku *functions.php*, a następnie podpięcie jej do haka `wp_footer` za pomocą funkcji `add_action()`, opisane w rozdziale 6.

Podczas budowy wtyczek często pisze się funkcje, które następnie podłączają się do wybranych haków za pomocą funkcji `add_action()`:

```
add_action ( $nazwa_haka, $nazwa_funkcji, $priorytet, $parametry );
```

W miejsce `$nazwa_haka` należy wpisać nazwę haka, do którego chcemy dodać naszą akcję. Napotykając ten hak podczas przetwarzania kodu, WordPress sprawdza, czy nie ma dla niego zarejestrowanych żadnych funkcji. Jeśli są, wykonuje je. Funkcja, która zostanie wykonana, jest zdefiniowana w miejscu `$nazwa_funkcji`. Na przykład w poniższym fragmencie kodu zostanie wywołana funkcja `smashingsshortcode` podczas wykonywania funkcji `wp_head()`:

```
add_action ( 'wp_head', 'smashingsshortcode' );
```

Argumenty `$priority` i `$parameters` są nieobowiązkowe. Argument `$priority` jest liczbą całkowitą określającą priorytet (domyślnie ma wartość 10), według którego akcje są sortowane podczas dodawania do haka. Im mniejsza liczba, tym wcześniej dana funkcja zostanie wykonana. Jeśli więc chcesz, aby jedna funkcja została wykonana przed inną funkcją, za pomocą tego atrybutu możesz to ustawić. Natomiast argument `$parameters` określa liczbę argumentów

przyjmowanych przez naszą funkcję (domyślna wartość to 1). Jeśli chcesz, aby funkcja przyjmowała więcej niż jeden argument, możesz wpisać w tym miejscu dowolną liczbę całkowitą.

Argumenty priorytetu i liczby argumentów nie są często używane, ale w pewnych przypadkach są naprawdę bardzo przydatne. Ponieważ są opcjonalne, to jeśli się ich nie potrzebuje, można je po prostu opuścić. Jak już jednak napisałem, są sytuacje, w których argument priorytetu jest bardzo przydatny, ponieważ umożliwia uniknięcie konfliktów między wtyczkami. Jeśli masz taki problem, możesz ustawić priorytet swojej wtyczki tak, aby była ładowana jako pierwsza lub ostatnia.

Filtry działają mniej więcej tak samo, tylko zamiast `add_action()` używa się funkcji `add_filter()`. Parametry są takie same i tak samo się je przekazuje. Jedyna różnica polega na tym, że z funkcją `add_filter()` nie używa się haków akcji, tylko filtrów.

Wiemy już, że dodawanie akcji do haków każdego rodzaju jest łatwe, ale co z ich **usuwaniami**? Czasami nie chcemy, aby jakiś hak był uruchamiany, więc musimy go usunąć. Dla haków akcji istnieje funkcja `remove_action()`, a dla haków filtrów — `remove_filter()`. Składnia tych funkcji jest prosta:

```
remove_action( $hook_name, $function_name )
remove_filter( $hook_name, $function_name )
```

Funkcje te służą nie tylko do usuwania własnych funkcji, ale również funkcji samego systemu, a więc za ich pomocą można usunąć praktycznie każdy filtr i każdą akcję, od pingowania po usuwanie załączników. Dlatego niektóre wtyczki mogą tylko usuwać funkcje WordPressa, a nie rozszerzać jego funkcjonalność.

TWORZENIE WŁASNYCH TAGÓW SZABLONOWYCH

Innym sposobem na uzyskanie dostępu do funkcjonalności wtyczki jest utworzenie własnych tagów szablonowych, jak `bloginfo()` czy `the_title()`. Nie jest to wcale trudne. Wystarczy po prostu utworzyć funkcję we wtyczce (lub w pliku *functions.php*), a następnie ją wywoływać:

```
<?php nazwa_funkcji(); ?>
```

Proste, prawda? Może proste, ale to nie oznacza, że jest to najlepszy sposób dodawania funkcjonalności wtyczki do systemu. W tej metodzie nie musisz tworzyć żadnych haków, funkcja zostanie wykonana w chwili załadowania tagu szablonowego wtyczki, a więc można ją umieścić w plikach szablonowych motywu, gdzie się chce. Jest to szczególnie przydatne, gdy nie ma takiego haka, jaki jest w danej chwili potrzebny. Potem można utworzyć tag szablonowy. Oczywiście lepiej jest jednak używać istniejących haków, kiedy to tylko możliwe.

Zanim zdecydujesz się na takie rozwiązanie, powinieneś zastanowić się nad kwestią użyteczności. Nie każdy lubi i potrafi modyfikować pliki szablonowe, dlatego zwłaszcza jeśli planujesz przekazać wtyczkę do ogólnego użytku albo klientowi, zmuszanie użytkowników do grzebania w szablonach nie będzie najlepszym pomysłem. Jeśli wtyczki używać będziesz

tylko Ty, nie ma to żadnego znaczenia. Jeśli jednak inni użytkownicy będą samodzielnie wybierać miejsce do umieszczenia wtyczki albo będą zmieniać jej parametry, to powinieneś poszukać innego rozwiązania.

Jednak w niektórych sytuacjach samo dodanie tagu szablonowego nie wystarczy i trzeba nadpisać część funkcjonalności systemu. Wówczas należy użyć specjalnych funkcji nadpisujących (ang. *pluggable functions*).

FUNKCJE NADPISUJĄCE

Czasami trzeba nadpisać wybrane części rdzenia WordPressa, aby np. zastąpić je własnymi rozwiązaniami albo żeby po prostu się ich pozbyć, ponieważ chcemy używać WordPressa w niestandardowy sposób. Może nie chcesz, aby w panelu administracyjnym działała lokalizacja (wówczas pozbyć się funkcji `load_textdomain()`), albo chcesz zmienić stopkę panelu administracyjnego na własną. Tych rzeczy nie da się zrobić, usuwając tylko wybrany hak. Z tego typu problemami trzeba zwrócić się do pliku *pluggable.php* znajdującego się w folderze *wp-includes*. Oczywiście nie będziemy go modyfikować, bo mielibyśmy z nim same kłopoty przy każdej aktualizacji WordPressa. Zamiast tego napiszemy wtyczkę, która będzie go przesłaniać. Miej świadomość, że jest to niebezpieczna praca. Przede wszystkim każdą funkcję można nadpisać tylko raz, a więc jeśli dwie wtyczki nadpiszą tę samą funkcję w pliku *pluggable.php*, witryna w najlepszym wypadku będzie źle działać, a w najgorszym przestanie działać w ogóle. To oznacza, że nie można zainstalować dwóch wtyczek przesłaniających tę samą funkcję w pliku *pluggable.php*, co jest poważną wadą systemu. Z tego powodu funkcji nadpisujących najlepiej jest używać wyłącznie w witynach, nad którymi ma się pełną kontrolę.

Aby zapobiec wyświetlaniu niepotrzebnych powiadomień o błędach, dodatkowo można kod wtyczki umieścić w instrukcji warunkowej sprawdzającej, czy dana funkcja istnieje:

```
<?php if ( ! function_exists( 'function_name' ) ); ?>
```

Oczywiście funkcje nadpisujące są czasami przydatne. Aktualna lista funkcji, które można przesłaniać, znajduje się w dokumentacji na stronie http://codex.wordpress.org/Pluggable_Functions.

Pamiętaj, że samo poprawne napisanie wtyczki, o czym jest mowa dalej, nie wystarczy do zapewnienia jej poprawnego działania, gdy wtyczka ta usuwa wybrane części rdzenia WordPressa. Nie dziw się, jeśli coś przestanie działać albo pojawią się jakieś konflikty, których nigdy byś się nie spodziewał. Bądź co bądź funkcje WordPressa, które próbujesz obejść, po coś przecież są.

WŁASNE TAKSONOMIE I TYPY WPISÓW

Własne taksonomie i typy wpisów to bardzo przydatne narzędzia, zwłaszcza gdy planuje się używanie WordPressa do prowadzenia czegoś więcej niż prosty blog. Jeśli nie wiesz, do czego mogą Ci się one przydać, ciekawe przykłady ich zastosowań znajdziesz w rozdziale 14.

Dla przypomnienia: własne taksonomie są dodatkowym sposobem organizacji treści. Domyślne tagi i kategorie są przykładami taksonomii odpowiednio niehierarchicznej i hierarchicznej, natomiast konkretne tagi i kategorie to terminy tych taksonomii. Natomiast własne typy wpisów są dodatkowym rodzajem publikacji, podobnie jak wpisy i strony, które też są rodzajem wpisów.

POWODY, ABY UŻYĆ WTYCZKI

Powodem, dla którego do tworzenia własnych taksonomii i typów wpisów najlepiej używać wtyczek, jest przenośność. Jeśli kod umieścisz w pliku *functions.php* (co jest możliwe), to po przenosinach do innego motywu zostanie on utracony, chociaż oczywiście można go skopiować do tego nowego motywu. We wtyczkach powinno się implementować raczej funkcje dotyczące treści niż projektu, dzięki czemu można ich używać w różnych motywach. Wystarczy włączyć wtyczkę i gotowe.

Więcej na temat przenośności danych piszę w rozdziale 8.

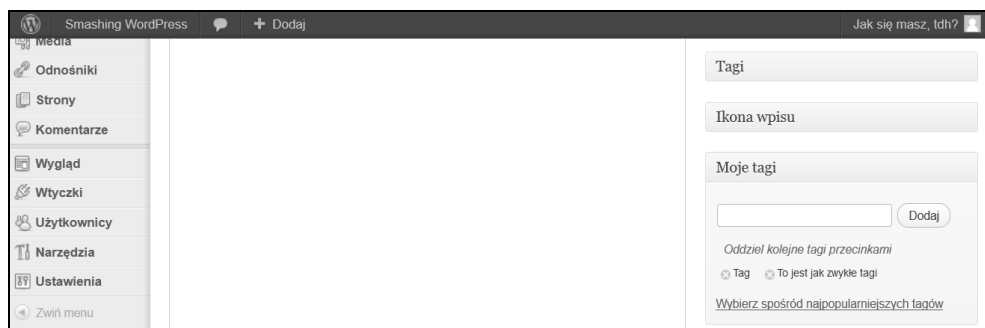
TWORZENIE WŁASNEJ TAKSONOMII

Aby utworzyć własną taksonomię, należy napisać funkcję zawierającą wywołanie funkcji `register_taxonomy()` i związać ją z hakiem `init`. Ustawienia funkcji `register_taxonomy()` są proste i dotyczą sposobu prezentacji taksonomii w panelu administracyjnym oraz tego, czy taksonomia powinna mieć własny bezpośredni odnośnik (argument `rewrite`), czy ma mieć strukturę hierarchiczną itd.

Poniżej znajduje się przykład tworzenia taksonomii *Moje tagi*, która jest niehierarchiczna, a więc bardzo podobna do domyślnej taksonomii tagów (rysunek 7.2):

```
// Powiązanie funkcji z hakiem init
add_action( 'init', 'smashing_tax', 0 );

// Funkcja taksonomii
function smashing_tax() {
    // Rejestracja taksonomii
    register_taxonomy( 'smashing_taxonomy', 'post',
        array(
            'hierarchical' => false,
            'labels' => array(
                'name' => 'Moje tagi',
                'singular_name' => 'Moje tagi',
                'search_items' => 'Przeszukuj Moje tagi',
                'popular_items' => 'Popularne Moje tagi',
                'add_new_item' => 'Dodaj nowy Mój tag'
            ),
            'query_var' => true,
            'rewrite' => true
        )
    );
}
```



Rysunek 7.2. Taksonomia Moje tagi

Niezbyt skomplikowane, prawda? Kod ten możesz zapisać w pliku *functions.php*, ale zapewne Twoja nowa taksonomia nie jest związana z konkretnym motywem, tylko z treścią, a więc lepiej byłoby utworzyć wtyczkę. W tym celu wystarczy przenieść ten kod do nowego pliku PHP zawierającego na początku odpowiedni nagłówek. Poniżej znajduje się treść takiego pliku:

```
<?php
/*
Plugin Name: Moje tagi
Plugin URI: http://tdh.me/wordpress/moje-tag/
Description: Dodaje taksonomię Moje tagi.
Version: 1.0
Author: Thord Daniel Hedengren
Author URI: http://tdh.me/
*/
// Powiązanie funkcji z hakiem init
add_action( 'init', 'smashing_tax', 0 );

// Funkcja taksonomii
function smashing_tax() {

    // Rejestracja taksonomii
    register_taxonomy( 'smashing_taxonomy', 'post',
        array(
            'hierarchical' => false,
            'labels' => array(
                'name' => 'Moje tagi',
                'singular_name' => 'Moje tagi',
                'search_items' => 'Przeszukuj Moje tagi',
                'popular_items' => 'Popularne Moje tagi',
                '_new_item' => 'Dodaj nowy Mój tag'
            ),
            'query_var' => true,
            'rewrite' => true
        )
    );
};
}
```

TWORZENIE WŁASNEGO TYPU WPISÓW

Własne typy wpisów również tworzy się w prosty sposób. Mimo że je również można tworzyć we wtyczkach (istnieją nawet wtyczki dodające interfejs pozwalający tworzyć własne typy wpisów), tutaj skoncentruję się na robieniu tego bezpośrednio w motywie. Kluczowe znaczenie w tym przypadku ma funkcja `register_post_type()`:

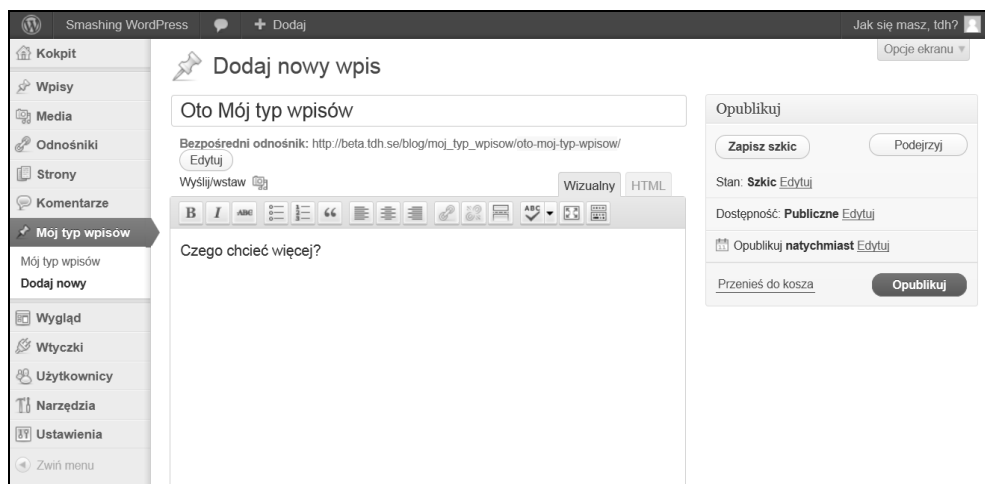
```
// Rejestracja nowego typu wpisów
register_post_type( 'moj_typ_wpisow',
    array(
        'labels' => array(
            'name' => 'Mój typ wpisów'
        ),
        'singular_label' => 'Mój typ wpisów',
        'public' => true,
        'show_ui' => true,
        'capability_type' => 'post',
        'has_archive' => true,
        'hierarchical' => false,
        'show_in_menu' => true,
        'supports' => array( 'title', 'editor', 'author', 'revisions', 'comments' )
    )
);
```

To spowoduje zarejestrowanie nowego typu wpisów o nazwie `moj_typ_wpisow` określonej w pierwszym parametrze. Drugi parametr to tablica zawierająca ustawienia wyglądu i właściwości tego typu wpisów, np. treść etykiet w panelu administracyjnym, czy ten typ jest publiczny, czy ma on być wyświetlany jako opcja w menu i które role użytkowników mogą z niego korzystać. Większości tych ustawień nie trzeba objaśniać, ale warto zwrócić uwagę na tablicę `supports` znajdującą się w tablicy ustawień. W niej definiuje się, co dany typ wpisów obsługuje, tu: tytuł, edytora tekstu, możliwość wyboru autora, wersje wpisu oraz komentarze. Żadnych taksonomii, wypisów, własnych pól — tylko to, co zostało wpisane.

Aby nowy typ wpisów pojawił się w panelu administracyjnym, wystarczy powyższy kod wkleić do pliku `functions.php` (rysunek 7.3). Prawdopodobnie będzie trzeba ponownie wygenerować wszystkie bezpośrednie odnośniki, aby adresy wpisów nowego typu zaczęły działać.

Treść własnych typów wpisów nie jest uwzględniana przez standardową pętlę. Aby je wyświetlić, trzeba tę pętlę zmodyfikować za pomocą funkcji `query_posts()` lub stosując jakąś inną metodę — więcej informacji o pętli znajduje się w rozdziale 3. Ale jeśli masz nowy typ wpisów, możesz bez przeszkód umieścić w menu łącze do strony ich archiwum.

Istnieje wiele opcji tworzenia wpisów. Ich kompletną listę można znaleźć w dokumentacji WordPressa na stronie http://codex.wordpress.org/Function_Reference/register_post_type. Własnych typów wpisów będziemy używać w projektach opisanych w rozdziale 14., a więc jeśli chcesz dowiedzieć się więcej na temat ich możliwości, zajrzyj do tego rozdziału.



Rysunek 7.3. Nowy typ wpisów

Jak kod tworzący własny typ wpisów wyglądałby we wtyczce? Bardzo podobnie do własnej taksonomii. Oto wtyczka Smashing Post Type w całej okazałości:

```
<?php
/*
```

```
Plugin Name: Smashing Post Type
```

```
Plugin URI: http://tdh.me/wordpress/smashing-post-type/
```

```
Description: Adding the Smashing Post Type.
```

```
Version: 1.0
```

```
Author: Thord Daniel Hedengren
```

```
Author URI: http://tdh.me/
```

```
*/
```

```
// Dodanie do haka init
```

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

```
// Dodanie własnych typów wpisów
```

```
function smashing_post_types() {
```

```
    // Rejestracja Mojego typu wpisów
```

```
    register_post_type( 'moj_typ_wpisow',
```

```
        array(
```

```
            'labels' => array(
```

```
                'name' => 'Mój typ wpisów',
```

```
                'menu_name' => 'Moje wpisy'
```

```
            ),
```

```
            'singular_label' => 'Mój typ wpisów',
```

```
            'public' => true,
```

```
            'show_ui' => true,
```

```
            'menu_position' => 5,
```

```
            'capability_type' => 'post',
```

```
            'has_archive' => true,
```

```
            'hierarchical' => false,
```

```
            'show_in_menu' => true,
```

```

        'supports' => array( 'title', 'editor', 'author', 'revisions', 'comments' )
    )
);
}

?>

```

CO POWINNA MIEĆ KAŻDA WTYCZKA

Tak naprawdę jedyną obowiązkową rzeczą, jaką musi mieć każda wtyczka, jest plik PHP zawierający nagłówek identyfikacyjny oraz kod potrzebny do wykonania zadania. W rzeczywistości jednak powinno się troszkę więcej popracować. Przecież wtyczki może używać jeszcze ktoś inny, a więc powinna ona być jak najłatwiejsza w użyciu i jak najbardziej dostępna. A to oznacza, że należy dołożyć wszelkich starań, aby wtopić wtyczkę w panel administracyjny WordPressa.

To samo dotyczy wszystkich elementów widocznych dla użytkownika witryny. Niektóre wtyczki dodają elementy wizualne i jeśli wtyczka ma być udostępniana publicznie, elementy te powinny nadawać się do wyświetlenia w jak największej ilości motywów. Oczywiście kwestii tej nie ma w przypadku wtyczek pisanych tylko dla konkretnych projektów. Podobnie jest z lokalizacją. Jeśli nie planujesz dodawać obsługi innych języków, nie ma sensu przygotowywać wtyczki do lokalizacji.

Moim zdaniem każda wtyczka powinna mieć dołączoną licencję i instrukcję obsługi. Nie mam nic przeciwko plikom *readme.txt*, ale wielu użytkowników nie otwiera ich, bo im się nie chce albo są zbyt niecierpliwi. Dlatego dobrze jest, jeśli podstawowe instrukcje znajdują się w samej wtyczce. Można też dodać informacje na karcie Pomoc WordPressa za pomocą funkcji `add_help_tab()`. Więcej informacji na ten temat znajduje się na stronie http://codex.wordpress.org/Function_Reference/add_help_tab.

USTAWIENIA WTYCZEK

Czasami trzeba zapisać jakieś informacje w bazie danych. Jeśli chodzi o bazę danych, to możesz z nią robić wszystko, co można zrobić przy użyciu skryptów PHP, a więc dodawać tabele itd. Nie będę się o tym rozpisywał.

Pokażę Ci natomiast, jak używać API ustawień (więcej informacji znajdziesz na stronie http://codex.wordpress.org/Settings_API), aby zlecić zapisywanie danych ustawień WordPressowi. To nie tylko pozwala zaoszczędzić mnóstwo czasu, ale również jest bezpieczne, ponieważ w WordPressie stosowane są różne zabezpieczenia, takie jak np. nonce (ang. *number used once* — liczba użyta tylko raz).

W celach testowych utworzymy prostą stronę ustawień, którą dodamy do menu *Ustawienia* na lewym pasku bocznym panelu administracyjnego. Strona ta będzie zawierała pole tekstowe i pole wyboru.

Najpierw musimy utworzyć wtyczkę. Wystarczy nam zwykły plik PHP o nazwie *smashing-settings.php* zaczynający się od poniższego nagłówka:

```
<?php
/*
Plugin Name: Ustawienia Smashing
Plugin URI: http://tdh.me/wordpress/ustawienia-settings/
Description: Prosta wtyczka ustawień.
Author: Thord Daniel Hedengren
Author URI: http://tdh.me/
*/
```

Teraz dodamy stronę ustawień do menu *Ustawienia*. W tym celu podłączymy funkcję do haka `admin_menu`. W tym przykładzie użyjemy funkcji `add_options_page()`, ale są też inne funkcje do wyboru:

```
// Dodaje funkcję strony ustawień do menu
add_action( 'admin_menu', 'smashings_settingsdemo_add_page' );
```

```
// Dodanie do menu
function smashings_settingsdemo_add_page() {
    add_options_page( 'Przykład ustawień Smashing',
        'Ustawienia Smashing',
        'manage_options',
        'smashings_settingsdemo',
        'smashings_settingsdemo_do_page' );
}
```

Za pomocą funkcji `add_options_page()` utworzyliśmy stronę o nazwie *Przykład ustawień Smashing*, która w menu po lewej stronie jest skrócona do *Ustawienia Smashing*. Strona ta jest dostępna tylko dla użytkowników mających rolę `manage_options`. Identyfikator tej strony to `smashings_settingsdemo`. Ponadto treść samej strony jest tworzona przez funkcję `smashings_settingsdemo_do_page()`, którą musimy jeszcze napisać:

```
//Rzeczywiste dodanie strony z ustawieniami
function smashings_settingsdemo_do_page() {
// Opuszczamy PHP na moment ?>

    <h2>Ustawienia Smashing</h2>
    <p>To jest nasza strona ustawień.</p>
    <form action="options.php" method="post">
    <?php settings_fields( 'smashings_settingsdemo' ); ?>
    <?php do_settings_sections( 'smashings_settingsdemo' ); ?>
    <?php submit_button(); ?>
    </form>

<?php
} // Powrót do PHP
```

Jest to cały kod HTML strony ustawień — na razie jest go niewiele, ale powoli go rozbudujemy. Zwróć uwagę na funkcję `settings_fields()` wskazującą pole ustawień o nazwie `smashings_settingsdemo` utworzone za pomocą funkcji `add_settings_field()`

oraz funkcję `do_settings_sections()`, która wyświetla sekcję o podanej nazwie. Nie trzeba też samodzielnie tworzyć pliku zatwierdzania formularza, ponieważ można go wstawić za pomocą wywołania `submit_button()`.

Następnie tworzymy funkcję dla naszych ustawień i dodajemy sekcję ustawień za pomocą funkcji `add_settings_section()` oraz pole wejściowe za pomocą funkcji `add_settings_field()`:

// Funkcja ustawień

```
function smashings_settingsdemo_init(){

    // Dodanie sekcji
    add_settings_section('smashing_settings_section',
        'Smashing Settings',
        'smashing_settings_section_callback',
        'smashings_settingsdemo');

    // Dodanie pola ustawień
    add_settings_field('smashing_sample_input',
        'Input sample',
        'smashing_sample_input_callback',
        'smashings_settingsdemo',
        'smashing_settings_section');

    // Rejestracja ustawień
    register_setting( 'smashings_settingsdemo', 'smashing_sample_input',
        'smashing_settingsdemo_validate' );
}

// Inicjacja smashings_settingsdemo_init() w panelu administracyjnym
add_action( 'admin_init', 'smashings_settingsdemo_init' );
```

Nie jest to wcale takie skomplikowane. Najpierw przy użyciu funkcji `add_settings_section()` tworzymy sekcję o nazwie `smashing_settings_section`. Zwróć uwagę na wartość `smashing_settingsdemo` będącą parametrem `$page`. Używaliśmy jej już w funkcji `do_settings_sections()` w kodzie strony. Do nowo utworzonej sekcji dodajemy pole ustawień `smashing_sample_input`, w ostatnim parametrze funkcji `add_settings_field()` wpisując `smashing_settings_section`. Listy wszystkich parametrów funkcji `add_settings_section()` i `add_settings_field()` znajdują się w dokumentacji na stronach http://codex.wordpress.org/Function_Reference/add_settings_section i http://codex.wordpress.org/Function_Reference/add_settings_field.

Następnie dodajemy funkcję `smashings_settingsdemo_init()` do haka `admin_init`.

Następnie dodamy coś do sekcji tylko po to, aby pokazać, że jest to możliwe:

// Funkcja wykonywana na początku sekcji

```
function smashing_settings_section_callback() {
    echo '<p>Informacja na początku sekcji.</p>';
}
```

Nazwę funkcji `smashing_settings_section_callback()` widziałeś już w funkcji `add_settings_section()`. Jest to funkcja zwrotna, która będzie wywoływana na początku sekcji i będzie wyświetlała element `p` z tekstem.

W podobny sposób utworzymy funkcję zwrotną dla funkcji `add_settings_field()` o nazwie `smashing_sample_input_callback()`:

```
// Implementacja funkcji smashing_sample_input_callback()
function smashing_sample_input_callback() {
// Opuszczamy PHP na chwilę ?>
    <input type="text" name="smashing_sample_input" value="<?php echo
        get_option( 'smashing_sample_input' ); ?>" />
    <?php }
// Powrót do PHP
```

Funkcja ta zawiera tylko pole wejściowe, którego wartość `smashing_sample_input` jest przekazywana przez funkcję `get_option()`. Wartość tę będziemy zapisywać i jest ona identyfikatorem pola, które utworzyliśmy.

Na koniec musimy trochę oczyścić dane wprowadzane do formularza. W tym przykładzie użyjemy tylko funkcji `esc_attr()`, ale jeśli chcesz, możesz skorzystać też z innych metod oczyszczających:

```
// Oczyszczanie
function smashing_settingsdemo_validate($input) {

    // Kodowanie
    $newinput = esc_attr($input);
    return $newinput;
}
```

Co robi ten kod? Utworzyliśmy stronę ustawień zawierającą pola ustawień tworzone za pomocą funkcji `settings_fields()` (w tym przykładzie tylko jedno, ale można dodać więcej) i sekcje utworzone za pomocą funkcji `do_settings_section()`. Poniżej znajduje się kompletny kod wtyczki.

```
<?php
/*
Plugin Name: Ustawienia Smashing
Plugin URI: http://tdh.me/wordpress/ustawienia-settings/
Description: Prosta wtyczka ustawień.
Author: Thord Daniel Hedengren
Author URI: http://tdh.me/
*/

// Dodaje stronę ustawień do menu
add_action( 'admin_menu', 'smashings_settingsdemo_add_page' );

// Dodanie do menu
function smashings_settingsdemo_add_page() {
    add_options_page( 'Przykład ustawień Smashing',
```

```

        'Ustawienia Smashing',
        'manage_options',
        'smashings_settingsdemo',
        'smashings_settingsdemo_do_page' );
    }

```

// Rzeczywiste dodanie strony ustawień

```
function smashings_settingsdemo_do_page() {
```

// Opuszczamy PHP na moment ?>

```

    <h2>Ustawienia Smashing</h2>
    <p>To jest nasza strona ustawień.</p>
    <form action="options.php" method="post">
    <?php settings_fields( 'smashings_settingsdemo' ); ?>
    <?php do_settings_sections( 'smashings_settingsdemo' ); ?>
    <?php submit_button(); ?>
    </form>

```

```
<?php
```

```
} // Powrót do PHP
```

// Funkcja ustawień

```
function smashings_settingsdemo_init(){
```

// Dodanie sekcji

```

add_settings_section('smashing_settings_section',
    'Ustawienia Smashing',
    'smashing_settings_section_callback',
    'smashings_settingsdemo');

```

// Dodanie pola ustawień

```

add_settings_field('smashing_sample_input',
    'Przykładowe pole',
    'smashing_sample_input_callback',
    'smashings_settingsdemo',
    'smashing_settings_section');

```

// Rejestracja ustawień

```

register_setting( 'smashings_settingsdemo', 'smashing_sample_input',
    'smashing_settingsdemo_validate' );

```

```
}
```

// Inicjacja smashings_settingsdemo_init() w panelu administracyjnym

```
add_action( 'admin_init', 'smashings_settingsdemo_init' );
```

// Funkcja wykonywana na początku sekcji

```

function smashing_settings_section_callback() {
    echo '<p> Informacja na początku sekcji.</p>';
}

```

// Implementacja funkcji smashing_sample_input_callback()

```
function smashing_sample_input_callback() {
```

```
// Opuuszczamy PHP na moment ?>
```

```
    <input type="text" name="smashing_sample_input" value="<?php echo
        get_option( 'smashing_sample_input' ); ?>" />
<?php }
```

```
// Powrót do PHP
```

```
// Oczyszczanie
```

```
function smashing_settingsdemo_validate($input) {
```

```
    // Kodowanie
```

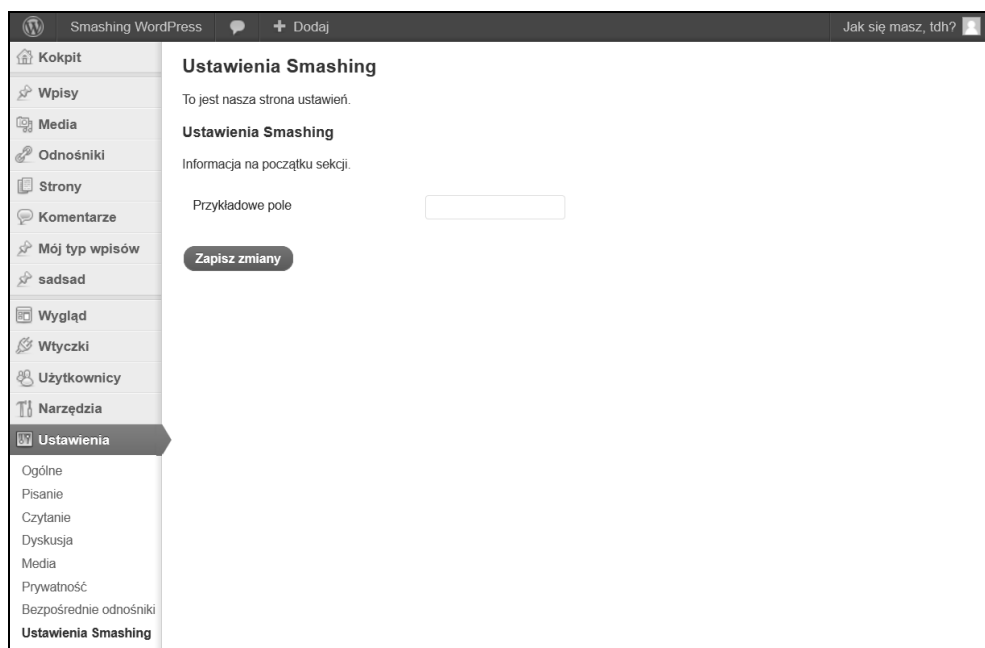
```
    $newinput = esc_attr($input);
```

```
    return $newinput;
```

```
}
```

```
?>
```

W ten sposób utworzyliśmy bardzo prostą stronę ustawień widoczną na rysunku 7.4. Za jej pomocą można zapisywać ustawienia w bazie danych.



Rysunek 7.4. Prosta strona ustawień z tekstem i polem tekstowym

BAZA DANYCH A ODINSTALOWYWANIE WTYCZKI

Tworząc wtyczkę przechowującą w bazie danych jakieś informacje, należy zastanowić się, co zrobić, gdy ktoś, kto ją zainstalował, zechce z niej po pewnym czasie zrezygnować. Czy wtyczka powinna po sobie sprzątać? W większości przypadków tak, zwłaszcza jeśli zapisała w bazie danych sporo informacji, które nie powinny tam zalegać bezużytecznie.

Jest kilka sposobów na usunięcie niepotrzebnych danych. Jednym z nich jest utworzenie pliku *uninstall.php* zawierającego kod usuwający z bazy danych treść, która została dodana tam przez wtyczkę:

```
delete_option( 'my-data' );
```

Powyższa instrukcja spowoduje usunięcie pola *my-data* z tabeli *option* bazy danych. Jako że wiele wtyczek zapisuje różne opcje w bazie danych, tabela ta szybko może stać się bardzo zabałaganiona, a to niczemu dobremu nie służy. Oczywiście w swoim pliku *uninstall.php* powinienś zapisać własne instrukcje usuwające niepotrzebne dane. To samo dotyczy deinstalacji wykonywanych poprzez panel administracyjny.

Oto przykładowa zawartość pliku *uninstall.php*:

```
<?php
    // Dla starych wersji
    if ( !defined( 'WP_UNINSTALL_PLUGIN' ) ) {
        exit();
    }
    // Usuwanie danych opcji
    delete_option( 'myplugindata_post' );
    delete_option( 'myplugindata_feed' );
?>
```

W pierwszej części skryptu sprawdzamy, czy funkcja deinstalacji jest dostępna. W starszych wersjach WordPressa nie było jej, a więc jeśli używasz systemu w wersji starszej niż 3.0 (choć nie powinienś!), skrypt zakończy działanie, nic nie robiąc. Dzięki temu ten plik jest zgodny ze starszymi wersjami WordPressa. Pozostały kod służy do usuwania danych z bazy: *myplugindata_post* i *myplugindata_feed*. Instrukcje te są wykonywane podczas usuwania wtyczki z panelu administracyjnego, dzięki czemu po deinstalacji wtyczki baza danych jest od razu posprzątana.

Ważne jest, aby pamiętać o zaimplementowaniu funkcji deinstalacji, jeśli wtyczka zapisuje cokolwiek w bazie danych. Można też pozwolić użytkownikowi wybrać, czy chce usunąć dane, czy woli je pozostawić, aby móc ich użyć w przyszłości. Dobrym pomysłem jest też usuwanie danych przy wyłączaniu, ponieważ zaleca się zrobienie tego dla wszystkich wtyczek podczas ręcznego uaktualniania WordPressa.

PO DEINSTALACJI

Łatwo jest zapomnieć, że po odinstalowaniu wtyczki w systemie mogą jeszcze być jakieś jej pozostałości. Dane w bazie danych to jedno (użytkownik powinien mieć przynajmniej możliwość wyboru, czy chce je usunąć), ale jest jeszcze jedna rzecz, która może być nawet bardziej kłopotliwa: tzw. skróty kodowe (ang. *shortcode*).

Co się dzieje, gdy zostanie odinstalowana wtyczka tworząca skróty kodowe? Skróty kodowe to specjalne ciągi znaków, które powodują wyświetlenie treści we wpisie w miejscu, w którym zostaną umieszczone. Jednym z najczęściej używanych w WordPressie skrótów tego typu jest

[gallery]. Można go zobaczyć, gdy doda się galerię do wpisu, a następnie przełączy się edytor w tryb HTML.

Co dzieje się ze skrótami wtyczki, gdy wtyczka ta zostanie odinstalowana? Nie zostaną przetworzone przez system jako skróty, tylko wyświetlone jak zwykły tekst. W tekście wówczas pojawią się napisy typu [mojshortcode].

To nie będzie dobrze wyglądać.

Dlatego na wypadek gdyby wtyczka została wyłączona albo odinstalowana, powinniśmy udostępnić jakieś rozwiązanie kwestii skrótów kodowych. Rozwiązanie to musi umożliwiać użytkownikowi łatwe pozbycie się nieprzetwarzanych ciągów z tekstu wpisów. Jednym ze sposobów jest napisanie zapytania SQL po prostu usuwającego wszystkie wystąpienia danego skrótu, ale to jest dość drastyczne posunięcie i nie da się przewidzieć, co się stanie, gdy coś w trakcie procesu usuwania się nie powiedzie. W ten sposób można uszkodzić całą bazę danych. Poza tym nie wiadomo, czy z powodu błędu użytkownika wtyczka nie usunie za dużego fragmentu tekstu.

Sposób radzenia sobie z niepotrzebnymi skrótami kodowymi zależy od sposobu działania wtyczki. Bardziej szczegółowo kwestią tą zajmują się w rozdziale 8.

Oczywiście nie z każdą wtyczką jest ten problem. Na przykład wtyczki tworzące tylko widżety nie są problematyczne, ponieważ po ich wyłączeniu widżety po prostu znikają.

WTYCZKI TWORZĄCE WIDŻETY

Za pomocą widżetów można łatwo dostosować sposób wyświetlania treści w blogu lub witrynie. Widżety umieszcza się w specjalnie wyznaczonych do tego obszarach za pośrednictwem panelu administracyjnego. W WordPressie dostępnych jest kilka standardowych widżetów, np. wyświetlający kanały RSS, najnowsze wpisy, listę stron, listę kategorii itp. Funkcjonalność tych widżetów może być niewystarczająca dla użytkownika i dlatego tworząc wtyczkę, można dać użytkownikom możliwość skorzystania z niej także w formie widżetu. Jest to o wiele lepsze niż zmuszanie użytkownika do wpisywania tagów szablonowych w plikach PHP motywu. Jeżeli więc funkcjonalność Twojej wtyczki ku temu przemawia, warto umożliwić używanie jej jako widżetu.

Tworzenie widżetów dla wtyczek nie jest trudne, głównie dzięki API widżetów, którego szczegółowy opis można znaleźć na stronie http://codex.wordpress.org/Widgets_API. Polega to na rozszerzeniu wbudowanej klasy WP_Widget, przekazaniu kilku instrukcji i zarejestrowaniu widżetu, aby mógł być wyświetlany, na przykład:

```
class SmashingWidget extends WP_Widget {
    function SmashingWidget() {
        // Kod widżetu
    }
    function widget( $args, $instance ) {
        // Zwrócenie treści widżetu
    }
}
```

```

    }
    function update( $new_instance, $old_instance ) {
        // Przetworzenie i zapisanie opcji widżetu
    }
    function form( $instance ) {
        // Wyświetlenie formularza opcji w panelu administracyjnym
    }
}
register_widget( 'SmashingWidget' );

```

W tym przykładzie utworzyliśmy podklasę klasy `WP_Widget` o nazwie `SmashingWidget`. Pierwsza funkcja, `function SmashingWidget()`, zawiera rzeczywisty kod widżetu, a więc to ona jest odpowiedzialna za jego działanie. Funkcje `widget()`, `update()` i `form()` pozwalają sprawić, aby widżet zachowywał się tak, jak chcemy. Oczywiście widżet należy zarejestrować za pomocą funkcji `register_widget()`. Łączy *Anuluj* i *Zapisz* są wbudowane w API widżetów, a więc nie musimy implementować ich obsługi, aby użytkownik mógł zapisać lub anulować swoje ustawienia.

TWORZENIE WIDŻETU

W tym podrozdziale przedstawiony jest krok po kroku proces tworzenia widżetu. Opisany tu widżet będzie wyświetlał tekst powitalny oraz będzie można zmienić jego tytuł w panelu administracyjnym:

1. Pamiętaj, że cały opisywany kod powinien znajdować się w pliku PHP wtyczki zawierającym nagłówek identyfikacyjny. Jeśli nie masz żadnej wtyczki, którą mógłbyś teraz rozbudować, utwórz nową.

Zacznijmy od utworzenia klasy widżetu:

```
class SmashingHello extends WP_Widget {
```

Ten widżet będzie nazywał się `SmashingHello`, dzięki czemu od razu wiadomo, co prawdopodobnie będzie robił.

2. Następnie definiujemy funkcję widżetu:

```
function SmashingHello() {
    parent::WP_Widget( false, $name = 'Witajcie, cześć i czołem' );
}
```

3. Do wykonania wielu czynności potrzebne są też funkcje `widget()`, `update()` i `form()`. Zacznijmy od definicji funkcji `widget()`:

```
function widget($args, $instance) {
    extract( $args );

    ?>
    <?php echo $before_widget; ?>
    <?php echo $before_title
        . $instance['title']
        . $after_title; ?>
    Cześć! Czy to nie jest wspaniałe?
```



```

        <?php echo $after_widget; ?>
    <?php
}

```

W funkcji tej pobieramy argumenty. Zwróć uwagę na ustawienia `$before_widget`, `$after_widget`, `$before_title` oraz `$after_title`. Nie należy ich zmieniać, jeśli nie jest to konieczne. Są one kontrolowane przez API widżetów i domyślne funkcje motywów i dzięki nim widżety dobrze wyglądają.

Wartości zmiennych `$before_widget` i `$before_title` po prostu wysyłamy na wyjście, nie robiąc z nimi niczego szczególnego, a więc po prostu otrzymamy domyślny kod. Następną jest zmienna `$instance` reprezentująca tytuł widżetu, który użytkownik może wpisać w polu tekstowym dostępnym w panelu administracyjnym. Dalej znajduje się zmienna `$after_title`, a za nią tekst, który zostanie wyświetlony jako treść widżetu: *Cześć! Czy to nie jest wspaniałe?*. To jest oczywiście tylko przykład, więc nie ma tu żadnych fajerwerków, ale w tym miejscu możesz umieścić dowolny kod, choćby pętlę WordPressa. Na końcu widżet zamyka zmienna `$after_widget`.

Przypomnę, że zmienne ze słowem `before` i `after` w nazwie umożliwiają zmuszenie widżetu do zachowywania się zgodnie z konstrukcją motywu. Jest to kwestia zależna od projektanta motywu, a więc jeśli chcesz, aby Twój widżet był prawidłowo wyświetlany we wszystkich motywach, pozostaw ustawienia domyślne.

4. Następnie trzeba zadbać o to, aby widżet był poprawnie zapisywany w przypadku aktualizacji:

```

function update($new_instance, $old_instance) {
    return $new_instance;
}

```

Funkcja `update()` pobiera tylko argumenty `$new_instance` i `$old_instance`. Oczywiście zwraca `$new_instance`, ponieważ jest to reprezentacja zmian. Jeśli obawiasz się nieprzyjemnego kodu HTML, możesz zastosować filtrowanie znaczników przy użyciu funkcji `strip_tags()`. Użycie tej funkcji jest bardzo łatwe, poniżej znajduje się przykład dla pola wejściowego o nazwie `music`:

```
$instance['music'] = strip_tags( $new_instance['music'] );
```

W tym kodzie funkcja `strip_tags()` pilnuje, aby nie przedostał się żaden niepożądany kod HTML — jest to bardzo przydatne.

5. Teraz dodamy jeszcze jedno ustawienie pozwalające zmienić tytuł widżetu:

```

function form( $instance ) {
    $title = esc_attr( $instance['title'] );
    ?>
    <p>
        <label for="<?php echo $this->get_field_id( 'title' ); ?>">
            Tytuł: <input class="widefat" id="<?php echo $this->
                get_field_id( 'title' ); ?>" name="<?php echo $this->
                get_field_name( 'title' ); ?>" type="text" value="<?php
                    echo $title; ?>" />
        </label>

```

```
</p>
<?php
}
```

Kluczowe w tym przypadku są funkcje `get_field_name()` i `get_field_id()`. Pierwsza określa nazwę, a druga identyfikator elementu. W ten sposób został utworzony formularz ustawień widżetu, który można zapisać za pomocą przycisku *Zapisz* automatycznie tworzonego przez API widżetów.

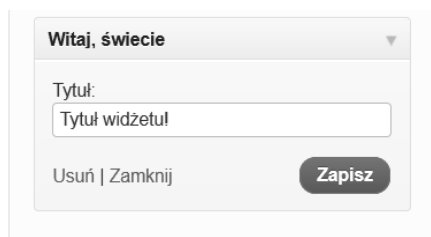
- Na koniec zamykamy klasę klamrą i rejestrujemy widżet:

```
}

function smashing_widget_init() {
    register_widget( 'SmashingHello' );
}

add_action( 'widgets_init', 'smashing_widget_init' );
```

Gotowy widżet jest przedstawiony na rysunku 7.5.



Rysunek 7.5. Utworzony widżet umieszczony na pasku bocznym w panelu administracyjnym

Masz już gotowy widżet, któremu możesz ustawiać tytuł i który wyświetla tekst. Oczywiście zamiast tekstu można wyświetlić cokolwiek, ponieważ wynik działania widżetu jest po prostu wynikiem działania skryptu PHP.

Należy też pamiętać, że nie wszystkie widżety muszą przyjmować opcje. Jeśli chcesz tylko umożliwić umieszczenie widżetu w obszarze widżetów, to nie twórz formularza ustawień. Nie ma sensu dodawać niepotrzebnych funkcji.

WIDŻETY KOKPITU

Można tworzyć nie tylko zwykłe widżety, ale również widżety kokpitu, czyli takie, które umieszcza się w panelu administracyjnym WordPressa potocznie nazywanym **kokpitem**. Widżetami są wszystkie ramki, które widać po wejściu do panelu administracyjnego; można też tworzyć własne takie ramki.

Aby utworzyć widżet kokpitu, należy utworzyć wtyczkę, a więc też i nowy plik. Poniżej znajduje się krótkie przypomnienie dla użytkowników grupowego bloga, żeby weszli na wewnętrzną stronę. Składa się ono tylko z tekstu i odnośników. Najpierw należy utworzyć odpowiednią funkcję:

```
function dashboard_reminder() {
    echo '
        Hej! Nie zapomnijcie przeczytać ważnych informacji na wewnętrznych
        stronach:<br />
        &larr; <a href="http://domain.com/internal/forum">Forum</a><br />
        &larr; <a href="http://domain.com/internal/docs">Dokumentacja</a><br />
        &larr; <a href="http://domain.com/internal/staff">Obsada</a><br />
        DZIĘKI!
    ';
```

Ta prosta funkcja o nazwie `dashboard_reminder()` wysyła na wyjście kod HTML, który stanowi treść widżetu. Następnym krokiem jest dodanie samego widżetu:

```
function dashboard_reminder_setup() {
    wp_add_dashboard_widget( 'dashboard_reminder_widget', 'Przypomnienie',
        'dashboard_reminder' );
}
```

Najważniejsza w tym kodzie jest funkcja `wp_add_dashboard_widget()`, której przekazaliśmy identyfikator widżetu (`dashboard_reminder_widget`), etykietę tekstową widżetu oraz nazwę funkcji zawierającej treść widżetu (`dashboard_reminder()`). Warto też wiedzieć, że identyfikator widżetu będący pierwszym parametrem funkcji `wp_add_dashboard_widget()` zostanie również użyty jako klasa tego widżetu, za pomocą której można go dowolnie sformatować przy użyciu CSS.

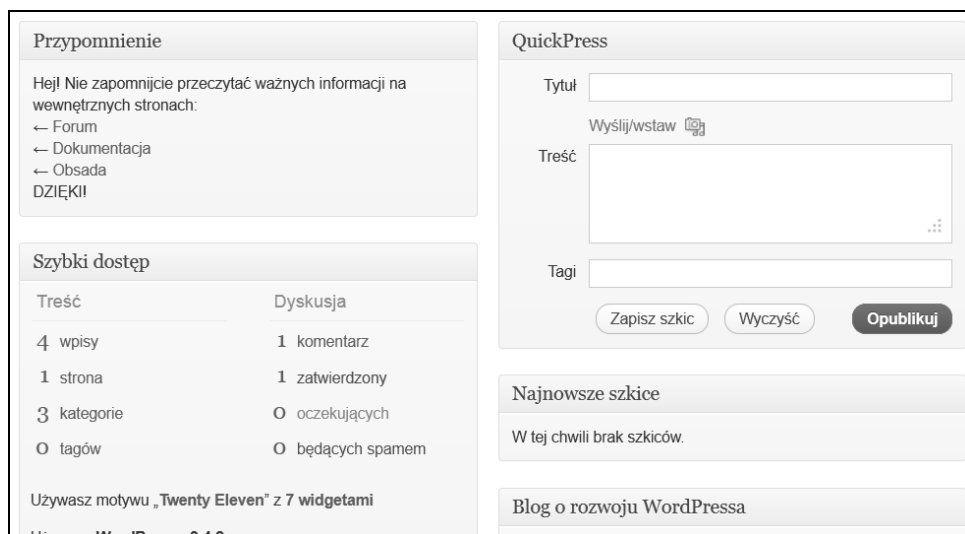
Zatrzymamy się na moment przy funkcji `wp_add_dashboard_widget()`. Ma ona jeszcze jeden parametr, o nazwie `$control_callback`, który jest opcjonalny i domyślnie przyjmuje wartość `null`. Nie został użyty w tym przykładzie, ale warto wiedzieć o istnieniu jeszcze jednego parametru, który również może być przydatny.

Wracając do przykładu, musimy jeszcze dodać akcję widżetu do haka `wp_dashboard_setup` za pomocą funkcji `add_action()`:

```
add_action( 'wp_dashboard_setup', 'dashboard_reminder_setup' );
```

To wszystko, widżet kokpitu jest już gotowy (rysunek 7.6)! Na razie nie istnieje żadne API pozwalające określać kolejność widżetów w panelu administracyjnym, a więc nasz widżet zostanie wyświetlony na dole. Użytkownik może go przesunąć w dowolne miejsce, ale są też sposoby na wyniesienie widżetu na górę automatycznie. Jeśli Cię to interesuje, zajrzyj np. do dokumentacji — na stronę http://codex.wordpress.org/Dashboard_Widgets_API.

Wiesz już, jak tworzyć wtyczki i widżety, a więc czas zająć się kwestią bazy danych. Zapisać informacje w bazie danych jest bardzo łatwo, ale nie zawsze jest to najlepsze rozwiązanie.



Rysunek 7.6. Widżet kokpitu w całej okazałości

KWESTIA KORZYSTANIA Z BAZY DANYCH WE WTYCZKACH

Czasami aby wtyczka działała prawidłowo, konieczne jest umożliwienie jej zapisywania informacji w bazie danych. Baza danych to bardzo przydatny magazyn, w którym można przechowywać wszystko — od prostych ustawień konfiguracyjnych po całe tabele danych do użytku użytkowników. Dane umieszczone w bazie danych bardzo łatwo jest pobrać i jest to bardzo wygodne rozwiązanie.

Niestety ta wygoda ma swoją cenę. Brak porządku w bazie utrudnia jej utrzymanie i dlatego trzeba pamiętać o tym, aby użytkownik mógł wraz z wtyczką usuwać dane tej wtyczki z bazy danych. Ponadto trzeba zdecydować, gdzie należy zapisywać dane. Można do tego użyć API ustawień, dzięki czemu mamy pewność, że dane trafią tam gdzie trzeba — jest to dobre rozwiązanie w przypadku niewielkich ilości danych konfiguracyjnych — albo utworzyć własną tabelę przeznaczoną tylko dla naszej wtyczki. Drugie rozwiązanie jest zwykle lepsze w przypadkach, gdy trzeba zapisywać duże ilości danych. Rozszerzanie bazy danych może powodować problemy z innymi wtyczkami również korzystającymi z tej bazy. Ponadto taka dodatkowa tabela nie należy do WordPressa i trzeba oddzielnie tworzyć jej kopię zapasową, a ponadto jeśli zechcesz przenieść instalację w inne miejsce, nie zostanie ona standardowo uwzględniona przez narzędzia eksportu i importu.

Pozostaje też kwestia buforowania, którą warto rozważyć w przypadku niektórych wtyczek, zwłaszcza regularnie pobierających dane z zewnętrznego źródła. Oczywiście do buforowania można używać plików, ale można też wykorzystać API Transients, które zostało właśnie do tego stworzone. Opis tego API znajduje się na stronie http://codex.wordpress.org/Transients_API.

Które rozwiązanie jest najlepsze? Zapisywać dane w opcjach, czy we własnej tabeli? Może należałoby użyć tabeli metadanych albo wpisów? Wybór zależy od tego, co chcesz przechowywać. Na podstawie własnego doświadczenia mogę powiedzieć, że ustawienia najlepiej zapisywać w tabeli opcji; realną treść i większe ilości danych często umieszczam w osobnej tabeli.

Pamiętaj tylko, aby poinformować użytkownika o sytuacji, i dodaj możliwość zrobienia porządku w bazie danych po odinstalowaniu wtyczki.

ZGODNOŚĆ WSTECZNA WTYCZEK

Kolejnym aspektem, jaki należy rozważyć podczas budowy wtyczki, jest to, czy wtyczka ta ma być zgodna ze starszymi wersjami systemu. Do WordPressa wciąż są dodawane nowe funkcje i haki i jeśli będziemy używać tych najnowszych, to naturalnie w starszych wersjach systemu nasza wtyczka nie będzie działać. Dlatego trzeba ostrożnie korzystać z najnowszych dodatków do systemu, ponieważ mogą one nie działać w jego starszych wersjach.

Co więcej, nie tylko funkcjonalność WordPressa się zmienia. Także wymagania systemowe co jakiś czas podlegają modyfikacjom. W tej chwili do działania systemu wymagany jest PHP 5.2.4. Wtyczka powinna obsługiwać te same wersje oprogramowania co sam WordPress, ale jeśli chcesz użyć nowszych wersji PHP lub MySQL, koniecznie wyświetl informację o błędzie, gdy ktoś spróbuje uruchomić ją w systemie ze starszymi wersjami PHP lub MySQL.

Jak daleko chcesz sięgać w przeszłość w swojej wtyczce, zależy tylko od Ciebie. Od kiedy wprowadzono możliwość automatycznej aktualizacji systemu, częstotliwość aktualizowania wzrosła. Jednak wciąż jest zaskakująco dużo witryn działających w oparciu o stare wersje WordPressa, co jest niekorzystne zarówno dla programistów, jak i właścicieli witryn. W każdym razie przestarzałe oprogramowanie jest mniej bezpieczne i powoduje problemy ze zgodnością z nowszymi modułami.

PUBLIKOWANIE WTYCZEK W PORTALU WORDPRESS.ORG

Dla wtyczek, podobnie jak dla motywów, istnieje oficjalny serwis w portalu WordPress.org, w którym można je publikować. Oczywiście nie musisz tego robić, ale dzięki temu użytkownicy otrzymują automatycznie powiadomienia o dostępności aktualizacji i mogą ich dokonywać na bieżąco.

Aby wtyczka została opublikowana w portalu WordPress.org, musi spełniać pewne wymagania:

- musi być na licencji zgodnej z GPL;
- nie może wykonywać nielegalnych ani „niemoralnych” działań;

- musi zostać wysłana do repozytorium Subversion WordPress.org;
- musi zawierać poprawny plik *readme.txt*.

Aby móc opublikować wtyczkę, trzeba być użytkownikiem portalu WordPress.org. Następnie wysyła się wtyczkę (<http://wordpress.org/extend/plugins/add>) i czeka na jej zatwierdzenie. Czas oczekiwania zależy od tego, ile aktualnie pracy ma zespół zajmujący się weryfikowaniem wtyczek.

Gdy wtyczka zostanie zatwierdzona, otrzymasz dostęp do katalogu Subversion, do którego możesz wysłać wtyczkę z plikiem *readme.txt*. Poprawność pliku *readme.txt* sprawdza specjalne narzędzie (<http://wordpress.org/extend/plugins/about/validator>). Weryfikuje ono, czy w pliku znajdują się wszystkie informacje potrzebne do publikacji wtyczki.

Przed wysłaniem wtyczki należy przeczytać odpowiedzi na najczęściej zadawane pytania dotyczące wtyczek opublikowane pod adresem <http://wordpress.org/extend/plugins/about/faq>. To na pewno usprawni proces zatwierdzania wtyczki.

Korzyścią z opublikowania wtyczki w zbiorach WordPress.org są nie tylko automatyczne aktualizacje, ale również prowadzenie statystyk. Dowiesz się, ile osób pobrało Twoją wtyczkę, oraz otrzymasz oceny i komentarze od użytkowników. Ponadto witryna WordPress.org jest centrum społeczności WordPressa, a więc stwarza największą szansę, że użytkownicy w ogóle znajdą wtyczkę. Jeśli opublikujesz ją na własnym serwerze, to z jej znalezieniem może być różnie, tymczasem dzięki narzędziu wyszukiwania wtyczek w panelu administracyjnym Twoją wtyczkę będzie mógł znaleźć i zainstalować dosłownie każdy użytkownik WordPressa.

Ale warunkiem jest obecność w zbiorach WordPress.org. Dlatego postaraj się tam zaistnieć ze swoją wtyczką!

Szukasz innego miejsca do opublikowania swojej wtyczki niż WordPress.org? Sprawdź GitHub (<http://github.com>). Jest to serwis do publikowania programów z otwartym kodem źródłowym. Bardzo wygodnie się z niego korzysta, zwłaszcza jeśli zna się obsługę programu Git. Jeśli go nie znasz, zawsze możesz poznać.

SŁOWO OSTRZEŻENIA NA TEMAT TWORZENIA WTYCZEK

Wtyczka to nie to samo co motyw. Oczywiście można znaleźć jakieś podobieństwa w procesie powstawania, ale jednak w przypadku wtyczek tak naprawdę pisze się zwykle skrypty PHP podłączone do WordPressa. W związku z tym podczas gdy każdy posiadający podstawową wiedzę na temat pisania skryptów może nagiąć WordPressa do swoich potrzeb w motywie, z pisaniem kodu wtyczek nie będzie już miał tak łatwo. Do tego potrzebna jest praktyczna umiejętność pisania skryptów PHP i trzeba przy tym bardzo uważać, ponieważ łatwo można coś zepsuć, zwłaszcza jeśli szpera się w bazie danych.

Biorąc pod uwagę te kwestie i znając się na programowaniu w PHP, za pomocą wtyczek możesz sprawić, że Twoja witryna będzie działała dokładnie tak, jak sobie zaplanujesz.

W następnym rozdziale dowiesz się, kiedy używać wtyczek, a kiedy lepiej jest pozostać przy pliku *functions.php*.

Skorowidz

A

archive.php, 114
attachment.php, 120

B

baza danych, 23
 kasowanie widżetów, 31
 kopia zapasowa, 33
 masowa edycja wpisów, 32
 modyfikowanie, 30
 ograniczanie uprawnień użytkownika, 39
 struktura, 31
 tabele, 32
 ustawienia, 26
 wersje wpisów, 27
 wtyczki, 216
 odinstalowywanie, 209
 zewnętrzny serwer, 26
 zmiana hasła użytkownika, 32

C

category.php, 47, 120
CMS, 231
 funkcje społecznościowe, 232
 menu, 242
 modularność, 232
 pomoc, 232
 widżety, 240
 własne
 taksonomie, 240
 typy wpisów, 239
 WordPress, 232, 238
 WP-CMS Post Control, 235
 wybór, 233
Codex, 44
comments.php, 46, 50, 93, 109, 130
content.php, 110
content-single.php, 112

F

Facebook, 246
 Facebook Connect, 254
 Get Code, 247
 Lubię to, 246
 narzędzia, 246
 widżet, 248
Flickr, 288
 pokaz slajdów, 290
 publikowanie zdjęć, 288
footer.php, 46, 50, 92, 93, 100
formaty wpisów, 72, 282
functions.php, 47, 67, 73, 117, 124, 125, 126, 127,
 128, 146, 152, 168, 170, 171, 172, 191, 221,
 223, 225
funkcja
 add_action(), 173, 197, 236
 add_filter(), 127, 198
 add_help_tab(), 204
 add_image_size(), 169, 324
 add_meta_box(), 166
 add_options_page(), 205
 add_post_meta(), 167
 add_settings_field(), 206
 add_settings_section(), 206
 add_theme_support(), 73, 74, 169, 324
 admin_url(), 305
 bloginfo(), 99, 182
 body_class(), 261
 comment_class(), 261
 comments_template(), 109
 dashboard_reminder(), 215
 delete_post_meta(), 167
 do_action(), 173
 do_settings_sections(), 206
 do_shortcode(), 302
 esc_attr(), 207
 esc_url(), 336
 fetch_feed(), 300

funkcja

form(), 212
 get_field_id(), 214
 get_field_name(), 214
 get_footer(), 105
 get_header(), 105
 get_option(), 207
 get_post(), 285
 get_post_format(), 74
 get_sidebar(), 105
 get_stylesheet_directory_uri(), 150, 271
 get_template_part(), 111, 148
 get_transient(), 299
 has_post_format(), 338
 load_textdomain(), 199
 mt_rand(), 284
 next_image_link(), 279
 next_post_link(), 278
 post_class(), 258, 260, 261, 326
 previous_image_link(), 279
 previous_post_link(), 278
 query_posts(), 75, 77, 124, 202
 rand(), 284
 register_nav_menus(), 118
 register_post_type(), 202
 register_sidebar(), 128, 295
 register_taxonomy(), 200
 register_widget(), 212
 remove_action(), 174, 198
 remove_filter(), 198
 remove_meta_box(), 235
 rewind_posts(), 79
 set_transient(), 299
 settings_fields(), 205
 simpleblog_load_scripts(), 118
 simpleblog_themesetup(), 118
 simpleblog_register_menus(), 118
 simpleblog_register_sidebars(), 118
 simpleblog_theme_setup(), 118
 smashing_post_demo_meta_box(), 167
 smashing_register_sidebars(), 128
 smashing_rss_promotion(), 302
 smashing_text_example(), 301
 smashings_settingsdemo_do_page(), 205
 smashings_settingsdemo_init(), 206
 smashingshortcode, 197
 smashingtheme_setup(), 74
 SmashingWidget(), 212
 strip_tags(), 213
 superfunction(), 174

the_content, 127
 the_date(), 109
 the_excerpt(), 336
 the_post_thumbnail(), 169
 update(), 212, 213
 update_post_meta(), 167
 widget(), 212
 wp_add_dashboard_widget(), 215
 wp_admin_css_color(), 273
 wp_editor(), 315
 wp_enqueue_script(), 98, 252, 269, 294
 wp_enqueue_style(), 226, 272, 307
 wp_footer(), 93, 101, 172
 wp_head(), 93, 97, 98, 172, 197
 wp_header(), 101
 wp_list_comments(), 114, 133
 wp_login_form(), 305
 wp_mail(), 303
 wp_nav_menu(), 99, 101, 265
 wp_register_script(), 270
 wp_register_style(), 307
 wp_reset_postdata(), 79, 81, 84
 wp_tag_cloud(), 318
 WPLANG, 25

G

galeria, 276
 lightbox, 280
 strona
 ustawień mediów, 276
 załącznika, 276
 stylizowanie, 278
 tworzenie, 276
 wyświetlanie losowych obrazów, 285
 gettext GNU, 181
 GlotPress, 181
 Google, 252
 przycisk +1, 252
 WordPress, 252

H

haki, 153
 add_meta_boxes, 235
 admin_init, 206, 273
 admin_menu, 205
 after_setup_theme, 118, 168, 179, 173, 181
 akcji, 171, 197
 dodawanie akcji, 173
 excerpt_length, 64

- filtrów, 197
- lista haków, 172
- login_head, 271
- odłączanie akcji, 174
- post_class, 261
- the_content, 126, 174
- the_excerpt, 197
- the_title, 197
- tworzenie własnych, 173
- user_register, 304
- używanie, 172
- widgets_init, 128, 236
- wp_enqueue_script, 252, 307
- wp_footer, 172
- header.php, 46, 50, 92, 93, 96
- HTML5, 94

I

- ikony wpisów, 169
- image.php, 120
- index.php, 30, 46, 50, 92, 93, 103, 107, 109, 236
- instalacja, 22
 - formularz z danymi witryny, 24
 - instalatory, 28
 - interfejs instalatora, 23
 - klucze
 - tajne, 25
 - uwierzytelniania, 25
 - multisite, 28
 - przycisk
 - Wyślij, 23
 - Zainstaluj WordPressa, 23
 - ręczna, 23
 - serwer baz danych, 26
 - ustawianie
 - adresu URL, 27
 - ścieżki, 27
 - w podfolderze, 29
 - wersje wpisów, 27
 - wp-config-sample.php, 24
 - zmiana języka, 25
 - z kreatorem, 22
- instrukcja
 - echo, 150, 180, 260
 - endif, 61
 - endwhile, 60
 - have_posts(), 62
 - if, 98, 127
 - return, 180

- Template, 145
- while, 60

J

- JavaScript, 268
 - biblioteki, 268
 - rejestrowanie skryptów, 270

K

- kanały RSS, 181, 245, 297
 - mieszanie zawartości kanałów, 300
 - parser kanałów, 298
 - tworzenie własnego, 183
 - wtyczki, 355
 - wyświetlanie, 297
- kanały subskrypcji, 182
- klucze
 - tajne, 25, 39
 - uwierzytelniania, 25
- komentarze, 130, 253
 - comments.php, 130
 - projektowanie, 130
 - w wątkach, 131
 - wtyczki, 353
 - wyróżnienie autora wpisu, 133
 - zakorzenianie, 132
 - zarządzanie, 353
 - zewnętrzny system komentarzy, 254

L

- lightbox, 280
- linki partnerskie, 319
- loop.php, 48, 61
- loop-category.php, 62
- loop-index.php, 61
- loop-single.php, 65, 67

M

- media społecznościowe, 245, 256
 - Facebook, 246
 - Google, 252
 - Twitter, 248
 - wtyczki, 354
- menu
 - CMS, 242
 - motywy, 262
 - Narzędzia, 34

menu

- przesuwane drzwi, 263
- rozwijane, 265
- ulepszanie, 262
- Ustawienia/Bezpośrednie odnośniki, 29
- własne, 170
- WordPress, 263

metadane, 164

- moduł meta, 166, 168

motyw, 46, 91, 118

- bazowy, 134, 153
- Bones, 369
- budowa, 92
- Constellation, 367
- deklarację motywu, 95
- elementy promocyjne, 126
- kategorie, 157
- komentarze, 130

- obsługa, 93

komercyjny, 136

kontrolowanie, 56

lista klas, 162

menu, 262

- własne, 170

modyfikowanie panelu administracyjnego, 272

nadrzędny, 144

nagłówki, 92, 96

Notes Blog, 145

obszary

- widżetów, 103
- właściwej treści, 92

pasek boczny, 92, 102

pętla, 60

planowanie, 156

pliki szablonowe, 46, 118

prosty projekt bloga, 93

przyspieszanie działania, 185

publikowanie, 136

Roots, 365

Spectacular, 368

Starkers, 364

stopka, 92, 100

strona opcji, 178

szablony, 46, 144

- archiwów, 114

- stron, 158

szkieletowy, 153, 362

tagi, 157, 258

taksonomie, 157

techniki stylizacji, 158

Toolbox, 366

treść główna, 103

- szerokość, 125

Twenty Eleven, 93, 329, 363

Twenty Ten, 93, 363

Twenty Ten Five, 363

wiadomość o błędzie, 116

własne pola, 133, 259

wpisy

- ikony, 169
- stylizowanie, 159
- wstępy, 65

wybór, 362

wyświetlanie wyników wyszukiwania, 116

zarządzanie, 48

multikanał, 300

multimedia, 275

ikony, 287

lightbox, 280

obrazy nagłówkowe, 287

osadzanie treści multimedialnej, 283

pokaz slajdów, 290

publikowanie zdjęć, 287

tworzenie galerii obrazów, 276

ustawienia treści osadzonych, 283

wtyczki, 347

wyświetlanie losowych obrazów, 284

multisite, 28, 193

0

obszar widżetów, 103

deklarowanie, 128

tworzenie, 118

oEmbed, 284

opcje

echo, 55

Indexes, 40

multisite, 28

WP_CONTENT_URL, 27

WP_DEBUG, 28, 139

WP_DEBUG_DISPLAY, 28

WP_DEBUG_LOG, 28

WP_HOME, 27

OpenID, 254

P

page.php, 46, 111, 112, 120, 121

parametr

author, 183

capability_type, 322

cat, 76

day, 183

echo, 52

exclude, 52

format, 52

hour, 183

include, 52

keyword, 183

large, 169

largest, 52

link, 52

medium, 169

minute, 183

monthnum, 183

number, 52

order, 52

orderby, 52

p, 183

pagedcat, 76

posts_per_page, 77

post-thumbnails, 169

public, 322

second, 183

separator, 52

smallest, 52

tag, 76

taxonomy, 52

thumbnail, 169

topic_count_text_callback, 52

unit, 52

year, 183

pasek boczny, 102, 148

pętla, 59, 60

else, 61

endif, 61

endwhile, 60

powiadomienie o błędzie, 60

promowanie produktów, 327

struktura, 60

tworzenie wielu pętli, 79

while, 60

klasa WP_Query, 62

wpisy, 64

przyklejanie, 69

wstawianie reklam, 266

wypisy, 63

zapisywanie, 61

PHP, 44, 49

pętla, 59

tagi

dołączania plików, 49

szablonowe, 49

wtyczki, 197

phpMyAdmin, 35, 38

pliki

.htaccess, 40, 45

.mo, 179

.po, 179

footer.php, 46, 50, 92, 93, 100

functions.php, 47, 67, 73, 117, 124, 125, 126,
127, 128, 146, 152, 168, 170, 171, 172, 191,
221, 223, 225

językowe, 179, 180

loop.php, 48, 61

loop-category.php, 62

loop-index.php, 61

loop-single.php, 65, 67

POT, 179

print.css, 306

sidebar.php, 46, 50, 92, 102, 128

style.css, 69, 92, 145, 146

szablonowe, 95, 118

404.php, 120

archive.php, 120

attachment.php, 120

category.php, 120

comments.php, 113

content.php, 110

content-single.php, 112

front-page.php, 120

header.php, 96

hierarchia, 121

image.php, 120

index.php, 30, 46, 50, 92, 93, 103, 107,
109, 236

motywy potomne, 146

page.php, 46, 111, 112, 120, 121

search.php, 46, 116, 120

single.php, 46, 65, 111, 120, 160

single-attachment.php, 120

style.css, 69, 92, 95, 145, 146

szablony stron, 122

taxonomy.php, 120, 174

text.php, 120

video.php, 120

pliki

- uninstall.php, 210
- wp-blog-header.php, 30
- wp-config.php, 26, 27, 33, 38, 40, 193
- wp-config-sample.php, 24, 26

zasady używania, 120

Poedit, 180

pola własne, *Patrz* własne pola

print.css, 306

przenośność, 227

przesuwane drzwi, 263

przyciski

- +1, 252

- Dodaj własne pole, 165

- Get Code, 247

- Lubię to, 246

- Pobierz skrócony odnośnik, 251

- Tweetnij, 249

- Więcej, 63

- Wyślij, 23

- Zainstaluj WordPressa, 23

przyklejanie wpisów, 69, 163

- dodatkowy obszar nagłówkowy, 163

publikowanie

- ogłoszeń, 163

- wtyczki, 346

R

rdzeń, 44

roadblocks, 241

S

screenshot.png, 117

search.php, 46, 116, 120

SEO, 183

- wtyczki, 356

sidebar.php, 46, 50, 92, 102, 128

single.php, 46, 65, 111, 120, 160

skróty kodowe, 210, 224, 301

- dodawanie, 301

- zagnieżdżanie, 302

strona załącznika, 276

strony błędu 404, 268

strony opcji motywu, 177

style.css, 69, 92, 95, 145, 146

system zarządzania treścią, *Patrz* CMS

szablony, 46

- hierarchia, 121

szkielet, 362

T

tagi

- ciąg znaków, 54

- dołączania plików, 49

- `comments_template()`, 50

- `get_calendar()`, 54

- `get_footer()`, 50, 93

- `get_header()`, 50, 93

- `get_sidebar()`, 93

- `get_template_directory_uri()`, 49

- `get_template_part()`, 50, 61, 62, 65, 74

- domyślne parametry chmury tagów, 52

- kontrolowanie motywu, 56

- liczby całkowite, 54

- lista tagów, 49

- łańcuch zapytań, 53

- metoda funkcyjna, 53

- pobieranie treści, 51

- sposoby wykorzystania, 258

- szablonowe, 49, 63

- `bloginfo()`, 49

- `body_class()`, 72, 99, 161

- `comments_template()`, 93

- `edit_comment_link()`, 51

- `edit_post_link()`, 51

- `header_image()`, 171

- `in_category()`, 258

- `post_class()`, 71, 158, 159, 163

- `query_posts()`, 80

- `sticky_class()`, 71

- `the_content()`, 64, 66, 172

- `the_date()`, 55

- `the_excerpt()`, 64, 65

- `the_meta()`, 165

- `the_permalink()`, 247

- `the_time()`, 55

- `the_title()`, 63

- `the_title_attribute()`, 63

- `wp_list_comments()`, 131

- `wp_get_attachment_link()`, 286

- `wp_nav_menu()`, 170

- `wp_tag_cloud()`, 52, 53

- tworzenie własnych, 198

- typy danych, 54

- wartości logiczne, 54

- warunkowe, 55

- `get_sidebar()`, 56

- `has_post_thumbnail()`, 169

- `is_attachment()`, 279

- is_category(), 56
- is_front_page(), 55, 57
- is_home(), 77, 119
- is_single(), 112, 119
- is_sticky(), 71, 163
- is_tag(), 258
- taxonomy_exists(), 261
- taksonomie, 174, 200
 - terminy, 174
 - własne, 174, 240, 261, 330
 - tworzenie, 200
 - zastosowania, 175
- taxonomy.php, 120, 174
- text.php, 120
- Twitter, 248
 - Sign In with Twitter, 254
 - skraccanie adresów URL, 251
 - TweetMeme, 251
 - Tweetnij, 249
 - Twitterfeed, 251
 - widżet, 249
- WordPress, 248

U

- uninstall.php, 210

V

- video.php, 120

W

- widżety, 47, 127
 - CMS, 240
 - deklarowanie obszarów widżetów, 128
 - dla wtyczki, 211
 - dodawanie dynamizmu, 241
 - Facebook, 248
 - kokpitu, 214, 216
 - obszarów
 - deklarowanie, 128
 - tworzenie, 118
 - proces tworzenia, 212
 - rejestracja, 214
 - SmashingHello, 212
 - Tekst, 241
 - Twitter, 249
 - zmiana sposobu wyświetlania, 129
- własne pola, 133, 157, 160, 259

- nagłówki w stylu czasopism, 133
- użyteczność, 134
- wtyczki, 222
- WordPress, 15, 22
 - API Transients, 216, 299
 - API ustawień, 204
 - BuddyPress, 232
 - buforowanie treści, 299
 - CMS, 232, 238
 - dokumentacja, 44
 - dostosowywanie stylu, 270
 - drukowanie treści, 306
 - eksportowanie danych, 35
 - elementy multimedialne, 275
 - osadzanie treści, 283
 - Facebook, 246
 - formaty wpisów, 72, 282
 - formularze
 - logowania, 271, 304
 - z danymi witryny, 24
 - funkcje, 293
 - komentarzy, 130
 - społecznościowe, 232
 - Google, 252
 - import danych, 36
 - instalacja, 22
 - Interfejs instalatora, 23
 - JavaScript, 269
 - kanały
 - RSS, 181, 297
 - subskrypcji, 182
 - kokpit, 214
 - komentarze, 253
 - konfiguracja statycznej witryny, 237
 - menu, 263
 - modularność, 232
 - modyfikowanie bazy danych, 30
 - motywy, 46
 - planowanie, 156
 - potomne, 145
 - publikowanie, 136
 - zarządzanie, 48
 - MU, 193
 - multikanal, 300
 - multisite, 28, 193
 - oEmbed, 284
 - optymalizacja, 186
 - panel administracyjny, 235
 - pętla, 60
 - pliki szablonowe, 95, 236

WordPress

- pola własne, 164
- pomoc, 232
- promowanie produktów, 326
- protokół Atom, 181
- przesuwane drzwi, 263
- publikowanie
 - bazy wiedzy, 316
 - linków partnerskich, 319
 - motywu, 136
 - przepisów, 328
 - recenzji przez użytkowników, 312
 - wiadomości przez użytkowników, 312
 - wpisów przez użytkowników, 311
- rdzeń, 44
- rejestrowania skryptów, 270
- rozszerzenie funkcjonalności, 228
- SimplePie, 298, 300
- sklep internetowy, 320
- strony
 - szablony, 121, 158
 - ustawień bezpośrednich odnośników, 30
 - ustawień mediów, 277
- stylizowanie galerii, 277
- tagi
 - dołączania plików, 49
 - szablonowe, 49
- taksonomie, 174
- TinyMCE, 315
- Twenty Eleven, 363
- Twenty Ten, 363
- Twitter, 248
- tworzenie
 - galerii obrazów, 276
 - katalogu, 321
 - strony wpisów, 323
 - tablicy ogłoszeń, 313
- typowy układ bloga, 92
- ustawienia
 - mediów, 125
 - ścieżki do instalacji, 27
 - bazy danych, 26
- użytkownik z uprawnieniami administratora, 39
- wiadomości e-mail, 303
- widżety, 47, 211
- własne
 - pola, 133, 157
 - taksonomie, 240
 - typy wpisów, 239
 - komponenty z kartami, 294

- wpisy, 61, 258
 - edycja, 68
 - formaty, 72, 157, 282
 - niestandardowego typu, 326
 - przyklejanie, 69
 - wersje, 27
- wtyczki, 191, 345
 - identyfikacja, 196
- wykonywanie kopii zapasowej, 32
- wymuszenie szyfrowania SSL, 40
- wypisy, 63
- wyświetlanie
 - proponowanego artykułu, 80
 - treści zewnętrznych, 242
 - edytora wpisów, 315
- zabezpieczanie, 38
- zmiana hasła użytkownika, 32
- zmiana hostingu, 34
- wp_commentmeta, 31
- wp_comments, 31
- wp_links, 31
- wp_options, 31, 32
- wp_postmeta, 31
- wp_posts, 31, 32
- WP_Query, 62, 79
- wp_term_relationships, 31
- wp_term_taxonomy, 31
- wp_terms, 31
- wp_usermeta, 31
- wp_users, 31
- wp-blog-header.php, 30
- wp-config.php, 26, 27, 38, 40, 193
- wp-config-sample.php, 24, 26
- wp-content, 33, 45
- wpisy, 61
 - chwytlive wstępy, 65
 - drukowanie, 306
 - edycja, 68
 - formaty, 72, 157, 282
 - ikony, 169
 - kategorie, 157
 - kontrolowanie treści, 258
 - liczba wyświetlanych wpisów, 77
 - lista wpisów, 65
 - niestandardowego typu, 326
 - nowy typ, 203
 - początkowy fragment wpisu, 64
 - przyklejanie, 69, 163
 - publikowanie przez użytkowników, 311
 - ramka z najnowszymi wpisami, 79

- słowa kluczowe, 157
- stylizowanie, 159
- tagi, 258
- taksonomia, 261
- tytuł wpisu, 63
- wersje, 27
- własne pola, 259
- własne typy, 176, 202, 239
 - tworzenie, 329
- wstępna segregacja, 157
- zwiększenie kontroli, 258, 261
- wtyczki, 186, 191
 - administracyjne
 - Activate Update Services, 352
 - bbPress, 351
 - Broken Link Checker, 349
 - BuddyPress, 351
 - Custom Admin Branding, 349
 - Custom Post Type UI, 352
 - Disabler, 348
 - Download Monitor, 350
 - Editorial Calendar, 352
 - Fast Secure Contact Form, 349
 - FeedWordPress, 243, 351
 - Google Analyticator, 349
 - Google Analytics for WordPress, 349
 - Jigoshop, 350
 - Members, 352
 - Members Only, 351
 - More Fields, 350
 - Pods, 350
 - Post Editor Buttons, 352
 - Pretty Link Lite, 350
 - Random Redirect, 352
 - Redirection, 350
 - Revision Control, 352
 - Theme My Login, 349
 - TinyMCE Advanced, 352
 - Viper's Video Quicktags, 352
 - Widget Context, 352
 - Woopra Analytics Plugin, 352
 - WordPress.com Stats, 349
 - WP Bannerize, 351
 - WP e-Commerce, 349
 - WP Event Ticketing, 352
 - WP Mail SMTP, 352
 - WP Maintenance Mode, 348
 - WP No Category Base, 348
 - WP-DB-Backup, 348
 - advanced-cache.php, 192
 - aktywacja dla całej sieci, 194
 - argumenty priorytetu, 198
 - bazy danych, 216
 - Blog Time, 358
 - blog-deleted.php, 192
 - blog-inactive.php, 192
 - blog-suspended.php, 192
 - blok identyfikacyjny wtyczki, 195
 - budowa, 195
 - db.php, 192
 - db-error.php, 192
 - do publikowania treści, 346
 - GD Star Rating, 347
 - Polldaddy, 347
 - WordPress Popular Posts, 346
 - WP Greet Box, 346
 - WP-Polls, 347
 - WP-PostRatings, 347
 - Yet Another Related Posts Plugin, 346
 - do rdzenia, 192
 - dodawanie funkcjonalności, 198
 - elementy wizualne, 204
 - funkcje nadpisujące, 199
 - identyfikacja w WordPressie, 196
 - informacja o licencji, 195
 - install.php, 192
 - instrukcja obsługi, 204
 - kanały RSS
 - Align RSS Images, 355
 - Disable RSS, 356
 - MobilePress, 356
 - RSS Footer, 356
 - Subscribe2, 356
 - WordPress Mobile Edition, 356
 - WordPress Mobile Pack, 356
 - WPtouch, 356
 - licencja, 204
 - maintenance.php, 192
 - media społecznościowe, 354
 - Lifestream, 354
 - SexyBookmarks, 355
 - Share Buttons by Lockerz / AddToAny, 355
 - ShareThis, 355
 - Simple Social Bookmarks, 355
 - Sociable, 355
 - Tweet Old Post, 355
 - Twitter for WordPress, 355
 - Twitter Tools, 354
 - Wickett Twitter Widget, 355

wtyczki

metody inkorporowania, 197
 multimedialne, 347
 Featured Articles Lite, 348
 Lightbox Gallery, 347
 Podcasting, 348
 Slimbox, 348
 na funkcje, 225
 object-cache.php, 192
 obowiązkowe, 192, 193
 odinstalowywanie, 209
 One Quick Post, 313
 PHP Snippets, 359
 plik PHP, 204
 Post From Site, 311
 przenośność, 227
 publikowanie, 217
 Query Posts, 359
 rodzaje, 192
 rozszerzanie funkcjonalności, 222
 SEO
 All in One SEO Pack, 357
 Better Search, 357
 Breadcrumb Trail, 358
 GD Press Tools, 358
 Global Translator, 357
 Google XML Sitemaps, 357
 HeadSpace2 SEO, 357
 Robots Meta, 357

Search Everything, 358
 WordPress SEO by Yoast, 357
 skróty kodowe, 210
 Smashing Post Type, 203
 strona ustawień, 209
 sunrise.php, 192
 SyntaxHighlighter Evolved, 358
 taksonomie, 200
 Theme Check, 139
 tworzenie, 193
 tworzenie widżetów, 211
 ustawienia, 204
 używanie haków, 197
 Widget Logic, 358
 własne pola, 222
 WP Super Cache, 359
 WP-Cirrus, 358
 WP-CMS Post Control, 235
 WP-DBManager, 359
 wpisy, 202
 WP-PageNavi, 359
 WP-Typography, 358
 Your Classified Ads, 314
 zgodność wsteczna, 217
 zwykłe, 192

Z

znacznik warunkowy, 95

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW
w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

<http://program-partnerski.helion.pl>

GRUPA WYDAWNICZA

 **Helion SA**

KOMPLETNE I PRZYJAZNE ŹRÓDŁO INFORMACJI O PLATFORMIE WORDPRESS!

Dwóch gigantów postanowiło połączyć swoje potencjały. Razem stworzyli niezwykłą książkę, którą właśnie trzymasz w rękach. Jednym z nich jest Smashing Magazine, należący do najpopularniejszych serwisów poświęconych tworzeniu stron WWW i nie tylko, a drugim WordPress, lider wśród systemów do prowadzenia blogów. Obok tej pozycji nie możesz przejść obojętnie!

W tej niepowtarzalnej i wyśmienitej książce znajdziesz zbiór najświeższych, najciekawszych i najlepszych informacji poświęconych systemowi WordPress. W czasie lektury dowiesz się, jak używać tej platformy jako systemu CMS oraz narzędzia do blogowania czy projektowania witryn dowolnego typu. Ponadto nauczysz się tworzyć genialne motywy, przydatne wtyczki oraz integrować Wordpressa z portalami społecznościowymi. Szczególną uwagę powinieneś zwrócić na zagadnienia poświęcone SEO — przestrzeganie dobrych zasad z pewnością ułatwi Ci zaistnienie w sieci. Książka ta jest obowiązkową pozycją dla każdego użytkownika platformy WordPress!

Tylko krok dzieli Cię od:

- szczegółowego poznania platformy Wordpress
- przygotowania własnego motywu graficznego
- poprawnej konfiguracji, zgodnej z SEO
- odniesienia sukcesu w sieci WWW

helion.pl
księgarnia
internetowa

Nr katalogowy: 13903



Księgarnia internetowa:
<http://helion.pl>



Zamówienia telefoniczne:

0 801 339900



0 601 339900

Informatyka w najlepszym wydaniu

 **WILEY**



Helion

Sprawdź najnowsze promocje:

• <http://helion.pl/promocje>

Książki najchętniej czytane:

• <http://helion.pl/bestsellery>

Zamów informacje o nowościach

• <http://helion.pl/novosci>

Helion SA

ul. Kościuszki 1c, 44-100 Gliwice

tel.: 32 230 98 63

e-mail: helion@helion.pl

<http://helion.pl>

sięgnij po WIĘCEJ



KOD KORZYŚCI

ISBN 978-83-246-6678-2



9 788324 666782

Cena: 59,00 zł