

IDŹ DO

PRZYKŁADOWY ROZDZIAŁ



SPIS TREŚCI

KATALOG KSIĄŻEK

KATALOG ONLINE

ZAMÓW DRUKOWANY KATALOG

TWÓJ KOSZYK

DODAJ DO KOSZYKA

CENNIK I INFORMACJE

ZAMÓW INFORMACJE  
O NOWOŚCIACH

ZAMÓW CENNIK

CZYTELNIA

FRAGMENTY KSIĄŻEK ONLINE

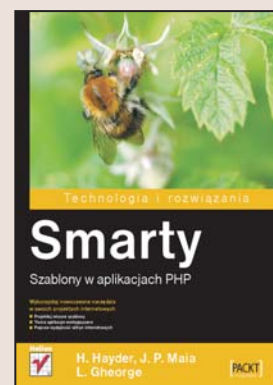
## Smarty. Szablony w aplikacjach PHP

Autorzy: H. Hayder, J. P. Maia, L. Gheorge  
Tłumaczenie: Radosław Meryk (rozdz. 1, 2, 4 – 11),  
Robert Polański (rozdz. 3)

ISBN: 83-246-0647-5

Tytuł oryginału: [Smarty PHP Template  
Programming and Applications](#)

Format: B5, stron: 296



### Wykorzystaj nowoczesne narzędzia w swoich projektach internetowych

- Projektuj własne szablony
- Twórz aplikacje wielojęzyczne
- Popraw wydajność witryn internetowych

Współczesne witryny WWW to nie tylko prezentacje podstawowych informacji o firmach. W sieci można znaleźć setki sklepów internetowych, portali informacyjnych i innych serwisów będących samodzielnymi przedsięwzięciami. Coraz częściej strony WWW wykorzystywane są również jako interfejs użytkownika dla aplikacji.

Tak rozbudowane projekty internetowe są realizowane przez zespoły składające się z programistów odpowiedzialnych za „zaplecze” serwisu i projektantów, których zadaniem jest opracowanie wyglądu witryny. W takich przypadkach niezbędny jest mechanizm pozwalający na oddzielenie treści i logiki stron WWW od ich prezentacji. Do tego celu stosowane są narzędzia umożliwiające skonstruowanie warstwy prezentacyjnej serwisu WWW w oparciu o szablony. PHP, jeden z najpopularniejszych języków programowania wykorzystywany do tworzenia witryn i aplikacji internetowych, oferuje mechanizm szablonów noszący nazwę Smarty.

Książka „Smarty. Szablony w aplikacjach PHP” to kompletny przewodnik po tej technologii. Zawiera zasady działania szablonów Smarty oraz możliwości wykorzystania ich w projektach internetowych. Czytając ją, dowiesz się, jak zainstalować i skonfigurować Smarty, jak zbudowane są witryny oparte o ten system oraz jak kreować własne szablony. Poznasz oferowane przez Smarty funkcje, metody i modyfikatory, które będziesz mógł wykorzystać podczas budowy złożonych projektów. Przeczytasz o buforowaniu, poprawie wydajności aplikacji, rozszerzeniu możliwości Smarty za pomocą wtyczek i tworzeniu aplikacji wielojęzycznych.

- Instalacja i konfiguracja systemu Smarty
- Struktura witryny opartej na szablonach Smarty
- Narzędzia do projektowania szablonów
- Tworzenie prostych szablonów
- Korzystanie z modyfikatorów w szablonach
- Funkcje w szablonach
- Wykrywanie i usuwanie błędów
- Buforowanie stron i optymalizacja wydajności witryn
- Rozbudowywanie Smarty za pomocą wtyczek

**Poznaj sposoby tworzenia profesjonalnych aplikacji sieciowych**

Wydawnictwo Helion  
ul. Kościuszki 1c  
44-100 Gliwice  
tel. 032 230 98 63  
e-mail: [helion@helion.pl](mailto:helion@helion.pl)



# Spis treści

|                                                                                 |           |
|---------------------------------------------------------------------------------|-----------|
| <b>O autorach</b>                                                               | <b>11</b> |
| <b>O recenzencie</b>                                                            | <b>13</b> |
| <b>Przedmowa</b>                                                                | <b>15</b> |
| <b>Rozdział 1. Wprowadzenie do Smarty</b>                                       | <b>19</b> |
| <b>Systemy szablonów</b>                                                        | <b>19</b> |
| Po co stosuje się systemy szablonów?                                            | 20        |
| Inteligencja technologii Smarty                                                 | 21        |
| Alternatywy dla technologii Smarty                                              | 22        |
| <b>Krótki przewodnik po procesie projektowania oprogramowania</b>               | <b>23</b> |
| Praca w zespole — warstwy i podział obszarów zainteresowań                      | 24        |
| <b>Smarty — podstawowy system obsługi szablonów dla języka PHP</b>              | <b>26</b> |
| Czy aplikacje z szablonami Smarty są szybkie?                                   | 26        |
| Czy szablony Smarty są bezpieczne?                                              | 27        |
| <b>Główne własności technologii Smarty</b>                                      | <b>28</b> |
| Modyfikatory zmiennych                                                          | 28        |
| Funkcje szablonów                                                               | 29        |
| Debugowanie                                                                     | 29        |
| Wtyczki                                                                         | 29        |
| Filtry                                                                          | 30        |
| <b>Wnętrze systemu Smarty</b>                                                   | <b>31</b> |
| <b>Instalacja i konfiguracja systemu Smarty</b>                                 | <b>32</b> |
| Krok 1. Zdobycie systemu Smarty                                                 | 32        |
| Krok 2. Konfigurowanie PHP w sposób umożliwiający odnalezienie bibliotek Smarty | 33        |
| Alternatywa kroku 2. Korzystanie z bibliotek Smarty w sytuacji,                 |           |
| kiedy nie mamy pełnego dostępu do systemu                                       | 34        |
| Krok 3. Konfiguracja Smarty w aplikacji                                         | 35        |
| Krok 4. Sprawdzenie poprawności instalacji                                      | 35        |
| Wersja rozwojowa systemu Smarty w repozytorium CVS                              | 35        |
| <b>Aktualizacja witryny korzystającej z szablonów Smarty</b>                    | <b>37</b> |
| <b>Podsumowanie</b>                                                             | <b>37</b> |

|                                                                    |           |
|--------------------------------------------------------------------|-----------|
| <b>Rozdział 2. Architektura witryn Smarty</b>                      | <b>39</b> |
| <b>Rozdział pojęć</b>                                              | <b>40</b> |
| Co to jest „pojęcie”?                                              | 40        |
| Perspektywa rozwiązywania problemu                                 | 41        |
| Pojęcia przecinające                                               | 41        |
| <b>Osoby biorące udział w tworzeniu i utrzymywaniu witryny WWW</b> | <b>42</b> |
| <b>Rozpoczęcie projektu Smarty</b>                                 | <b>43</b> |
| Struktura katalogów                                                | 43        |
| Zabezpieczenia projektu Smarty                                     | 44        |
| Warstwa dostępu do danych                                          | 44        |
| Warstwa reguł biznesu                                              | 47        |
| Warstwa prezentacji                                                | 48        |
| Wynik                                                              | 50        |
| <b>Podsumowanie</b>                                                | <b>52</b> |

## *Smarty dla projektantów*

|                                                                         |           |
|-------------------------------------------------------------------------|-----------|
| <b>Rozdział 3. Co projektanci wiedzieć powinni?</b>                     | <b>53</b> |
| <b>Kłopoty programistów — częste scenariusze</b>                        | <b>53</b> |
| <b>Role projektanta szablonu i programisty</b>                          | <b>54</b> |
| <b>Definicje i pojęcia dla projektantów</b>                             | <b>55</b> |
| Pojęcie komponentów wielokrotnego użycia                                | 57        |
| Podział na komponenty                                                   | 59        |
| Jak tworzyć szablony pozbawione tabel?                                  | 61        |
| Podręczne, wbudowane znaczniki                                          | 64        |
| <b>Wybór edytora dla projektowania szablonów</b>                        | <b>65</b> |
| <b>Współpraca z innymi projektantami</b>                                | <b>66</b> |
| <b>Podsumowanie</b>                                                     | <b>67</b> |
| <b>Rozdział 4. Tworzenie szablonów</b>                                  | <b>69</b> |
| <b>Zagadnienia projektowe — od kodu HTML do TPL</b>                     | <b>69</b> |
| <b>Wprowadzenie do zmiennych Smarty</b>                                 | <b>73</b> |
| <b>Przystępujemy do tworzenia szablonów</b>                             | <b>75</b> |
| Tablice nieasocjacyjne                                                  | 75        |
| Tablice asocjacyjne                                                     | 75        |
| Przekazywanie tablic do szablonów Smarty i wykonywanie na nich operacji | 76        |
| <b>Proste szablony</b>                                                  | <b>78</b> |
| Warunki logiczne                                                        | 78        |
| Pętle                                                                   | 80        |
| <b>Szablony w praktyce</b>                                              | <b>84</b> |
| Kalendarz                                                               | 84        |
| Raport z bazy danych                                                    | 87        |
| Formularze do wprowadzania danych                                       | 91        |
| Biuletyn przesyłany pocztą elektroniczną                                | 95        |
| Uruchamianie kodu PHP wewnątrz szablonów                                | 98        |
| <b>Podsumowanie</b>                                                     | <b>99</b> |

|                                                                                   |            |
|-----------------------------------------------------------------------------------|------------|
| <b>Rozdział 5. Szablony zaawansowane</b>                                          | <b>101</b> |
| <b>Smarty od środka</b>                                                           | <b>101</b> |
| Etapy kompilacji                                                                  | 103        |
| Filtry wstępne i końcowe                                                          | 104        |
| Czym są modyfikatory Smarty?                                                      | 104        |
| <b>Zaawansowane zastosowania technologii Smarty</b>                               | <b>104</b> |
| Arkusz ocen uczniów                                                               | 104        |
| Galeria zdjęć                                                                     | 108        |
| <b>Dostępne modyfikatory</b>                                                      | <b>111</b> |
| capitalize                                                                        | 111        |
| count_characters                                                                  | 111        |
| cat                                                                               | 112        |
| count_paragraphs                                                                  | 113        |
| count_sentences                                                                   | 113        |
| count_words                                                                       | 113        |
| date_format                                                                       | 114        |
| default                                                                           | 116        |
| escape                                                                            | 117        |
| indent                                                                            | 118        |
| lower                                                                             | 118        |
| upper                                                                             | 118        |
| nl2br                                                                             | 119        |
| regex_replace                                                                     | 120        |
| replace                                                                           | 120        |
| spacify                                                                           | 121        |
| string_format                                                                     | 121        |
| strip                                                                             | 122        |
| strip_tags                                                                        | 123        |
| truncate                                                                          | 123        |
| wordwrap                                                                          | 124        |
| <b>Łączenie modyfikatorów</b>                                                     | <b>125</b> |
| <b>Pliki konfiguracyjne</b>                                                       | <b>125</b> |
| <b>Podsumowanie</b>                                                               | <b>127</b> |
| <b>Rozdział 6. Funkcje Smarty</b>                                                 | <b>129</b> |
| <b>Typy funkcji Smarty</b>                                                        | <b>129</b> |
| <b>Funkcje w praktyce</b>                                                         | <b>130</b> |
| Operacja — wielokrotne wykorzystywanie elementów strony za pomocą funkcji include | 132        |
| Objaśnienie                                                                       | 133        |
| <b>Umieszczanie elementów dynamicznych</b>                                        | <b>133</b> |
| <b>Przekazywanie zmiennych do włączanych szablonów</b>                            | <b>135</b> |
| <b>Zapisywanie zmiennych w plikach konfiguracyjnych</b>                           | <b>137</b> |
| <b>Tworzenie sekcji pliku konfiguracyjnego dla każdej ze stron</b>                | <b>138</b> |
| <b>Obsługa list w szablonach</b>                                                  | <b>140</b> |
| <b>Usuwanie nadmiarowych spacji z szablonów</b>                                   | <b>143</b> |
| <b>Obsługa kodu JavaScript w szablonach</b>                                       | <b>144</b> |
| <b>Przetwarzanie zagnieżdżonych tablic</b>                                        | <b>146</b> |

|                                                              |     |
|--------------------------------------------------------------|-----|
| Cykliczne przetwarzanie listy wartości                       | 150 |
| Przeciwdziałanie automatom spamowym indeksującym witryny WWW | 151 |
| Funkcje obsługi formularzy                                   | 152 |
| Więcej funkcji obsługi formularzy                            | 155 |
| Podsumowanie                                                 | 157 |

---

## **Rozdział 7. Debugowanie dla projektantów** **159**

---

|                                                  |     |
|--------------------------------------------------|-----|
| Debugowanie szablonów Smarty                     | 159 |
| Błędy semantyczne                                | 161 |
| Częste błędy w szablonach Smarty                 | 163 |
| Inne często popełniane błędy w szablonach Smarty | 170 |
| Konsola debugowania                              | 171 |
| Podsumowanie                                     | 172 |

---

## *Smarty dla programistów*

---



---

## **Rozdział 8. Wbudowane zmienne i metody systemu Smarty** **173**

---

|                                         |            |
|-----------------------------------------|------------|
| <b>Wbudowane zmienne systemu Smarty</b> | <b>175</b> |
| \$template_dir                          | 175        |
| \$compile_dir                           | 175        |
| \$config_dir                            | 175        |
| \$plugins_dir                           | 176        |
| \$debugging                             | 176        |
| \$error_reporting                       | 176        |
| \$debug_tpl                             | 176        |
| \$debugging_ctrl                        | 176        |
| \$compile_check                         | 177        |
| \$force_compile                         | 177        |
| \$caching                               | 177        |
| \$cache_dir                             | 177        |
| \$cache_lifetime                        | 178        |
| \$cache_modified_check                  | 178        |
| \$php_handling                          | 178        |
| \$security                              | 178        |
| \$secure_dir                            | 179        |
| \$security_settings                     | 179        |
| \$trusted_dir                           | 179        |
| \$left_delimiter                        | 180        |
| \$right_delimiter                       | 180        |
| \$request_vars_order                    | 180        |
| \$request_use_auto_globals              | 180        |
| \$compile_id                            | 180        |
| \$use_sub_dirs                          | 180        |
| \$default_modifiers                     | 181        |
| \$default_resource_type                 | 181        |
| \$cache_handler_func                    | 181        |

|                                                              |            |
|--------------------------------------------------------------|------------|
| \$autoload_filters                                           | 181        |
| \$config_overwrite                                           | 181        |
| \$config_booleanize                                          | 182        |
| \$config_read_hidden                                         | 182        |
| \$config_fix_newlines                                        | 182        |
| \$default_template_handler_func                              | 182        |
| \$compiler_file                                              | 182        |
| \$compiler_class                                             | 182        |
| \$config_class                                               | 182        |
| <b>Podręczna tabela wbudowanych zmiennych systemu Smarty</b> | <b>183</b> |
| <b>Wbudowane metody Smarty</b>                               | <b>187</b> |
| assign                                                       | 187        |
| assign_by_ref                                                | 187        |
| Przykład — działanie metod assign i assign_by_ref            | 188        |
| append                                                       | 190        |
| append_by_ref                                                | 191        |
| clear_assign                                                 | 191        |
| register_function                                            | 192        |
| unregister_function                                          | 192        |
| register_object                                              | 193        |
| unregister_object                                            | 193        |
| register_block                                               | 193        |
| unregister_block                                             | 194        |
| register_compiler_function                                   | 194        |
| unregister_compiler_function                                 | 195        |
| register_modifier                                            | 195        |
| unregister_modifier                                          | 195        |
| register_resource                                            | 195        |
| unregister_resource                                          | 196        |
| register_prefilter                                           | 196        |
| unregister_prefilter                                         | 196        |
| register_postfilter                                          | 197        |
| unregister_postfilter                                        | 197        |
| register_outputfilter                                        | 197        |
| unregister_outputfilter                                      | 198        |
| load_filter                                                  | 198        |
| clear_cache                                                  | 198        |
| clear_all_cache                                              | 199        |
| is_cached                                                    | 199        |
| clear_all_assign                                             | 199        |
| clear_compiled_tpl                                           | 199        |
| template_exists                                              | 200        |
| get_template_vars                                            | 200        |
| get_config_vars                                              | 200        |
| trigger_error                                                | 200        |
| display                                                      | 201        |

|                       |            |
|-----------------------|------------|
| fetch                 | 201        |
| config_load           | 201        |
| get_registered_object | 201        |
| clear_config          | 202        |
| <b>Podsumowanie</b>   | <b>202</b> |

## **Rozdział 9. Buforowanie i wydajność** **203**

|                                                                  |            |
|------------------------------------------------------------------|------------|
| <b>Buforowanie w systemie Smarty</b>                             | <b>204</b> |
| <b>Dynamiczne buforowanie części szablonów</b>                   | <b>206</b> |
| <b>Czyszczenie bufora</b>                                        | <b>206</b> |
| <b>Zaawansowane możliwości buforowania</b>                       | <b>208</b> |
| <b>Zastosowanie grup buforów</b>                                 | <b>209</b> |
| <b>Czyszczenie grup buforów</b>                                  | <b>209</b> |
| <b>Blokowanie buforowania</b>                                    | <b>210</b> |
| <b>Tworzenie własnej procedury obsługi buforowania</b>           | <b>214</b> |
| <b>Optymalizacja aplikacji Smarty</b>                            | <b>220</b> |
| <b>Profilowanie kodu PHP</b>                                     | <b>221</b> |
| <b>Projektowanie witryn zapewniających skuteczne buforowanie</b> | <b>222</b> |
| Nagłówki Last-Modified i ETag                                    | 223        |
| Nagłówek Expires                                                 | 225        |
| Nagłówek Cache-Control                                           | 225        |
| <b>Narzędzie — ApacheBench (ab)</b>                              | <b>226</b> |
| <b>Narzędzie — Xdebug</b>                                        | <b>227</b> |
| <b>Narzędzie — WinCacheGrind</b>                                 | <b>228</b> |
| <b>Podsumowanie</b>                                              | <b>229</b> |

## **Rozdział 10. Rozszerzanie systemu Smarty za pomocą wtyczek** **231**

|                                                 |            |
|-------------------------------------------------|------------|
| <b>Wyszukiwanie i instalacja wtyczek</b>        | <b>232</b> |
| <b>Przydatne wtyczki</b>                        | <b>232</b> |
| Listy HTML                                      | 232        |
| Formatowanie rozmiaru pliku                     | 234        |
| Podświetlanie słów kluczowych w stylu Google    | 236        |
| <b>Pisanie własnych wtyczek</b>                 | <b>239</b> |
| <b>Typy wtyczek</b>                             | <b>240</b> |
| Funkcje                                         | 240        |
| Modyfikatory                                    | 241        |
| Funkcje blokowe                                 | 241        |
| Funkcje kompilatora                             | 241        |
| Filtry wstępne, końcowe i wynikowe              | 241        |
| Zasoby                                          | 241        |
| Wstawki                                         | 242        |
| <b>Rejestracja wtyczek</b>                      | <b>243</b> |
| <b>Przykładowa wtyczka — kalendarz</b>          | <b>245</b> |
| <b>Przykładowa wtyczka — automatyczne łącza</b> | <b>248</b> |
| <b>Podsumowanie</b>                             | <b>250</b> |

|                                                          |            |
|----------------------------------------------------------|------------|
| <b>Rozdział 11. Filtry</b>                               | <b>251</b> |
| Filtry wstępne                                           | 251        |
| Filtry końcowe                                           | 252        |
| Filtry wynikowe                                          | 252        |
| Tworzenie filtrów                                        | 252        |
| Rejestracja filtra w fazie wykonywania aplikacji         | 253        |
| Ręczne załadowanie filtra                                | 254        |
| Automatyczne ładowanie filtrów                           | 255        |
| Filtr #1 — usuwanie komentarzy HTML                      | 256        |
| Filtr #2 — mierzenie wydajności aplikacji                | 259        |
| Filtr #3 — kompresja wyniku za pomocą gzip               | 262        |
| Filtr #4 — wyróżnianie wyszukiwanych słów kluczowych     | 263        |
| Podsumowanie                                             | 266        |
| <b>Rozdział 12. Tworzenie aplikacji wielojęzycznych</b>  | <b>267</b> |
| Infrastruktura tłumaczenia — Gettext                     | 268        |
| Konfiguracja rozszerzenia Gettext w PHP                  | 268        |
| Prosty przykład w PHP                                    | 269        |
| Konfiguracja plików Gettext                              | 270        |
| Zastosowanie rozszerzenia Gettext wraz z systemem Smarty | 273        |
| Generowanie plików PO                                    | 276        |
| Zaawansowane własności pakietu Smarty Gettext            | 280        |
| Podsumowanie                                             | 282        |
| <b>Skorowidz</b>                                         | <b>283</b> |



# Wprowadzenie do Smarty

Współczesne witryny WWW daleko wykraczają poza poziom prezentacji sumarycznych danych i informacji kontaktowych dla firm i projektów. Internet przeszedł ewolucję, w wyniku której większość współczesnych witryn to samodzielne przedsięwzięcia, a nie prezentacje firm. Co więcej, projektanci złożonych aplikacji zaczęli wykorzystywać witryny WWW jako interfejs dla swoich programów.

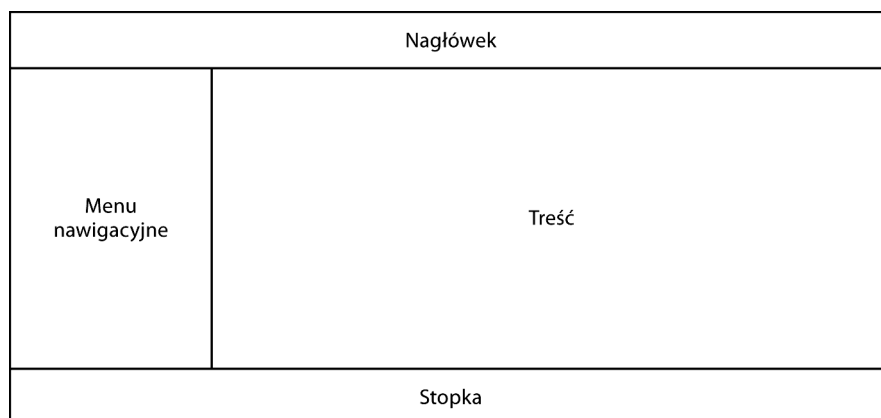
Od ponad dziesięciu lat PHP jako najbardziej kompletny język programowania aplikacji WWW o otwartym kodzie jest jedną z najlepszych technologii tworzenia witryn w internecie. Wraz ze wzrostem złożoności aplikacji PHP pojawił się nowy problem — w jaki sposób oddzielić kod pisany przez programistów (PHP) od kodu pisanego przez projektantów witryn (HTML) lub, posługując się bardziej precyzyjnymi sformułowaniami, jak oddzielić treść od jej prezentacji.

Technologię Smarty — system szablonów dla języka PHP — opracowano po to, by rozwiązać ten problem.

---

## Systemy szablonów

Podstawowa rola systemu szablonów polega na znalezieniu sposobu oddzielenia prezentacji od treści przy niewielkiej interakcji pomiędzy programistami a projektantami. Jako przykład przeanalizujemy witrynę WWW o układzie pokazanym na rysunku 1.1.


**Rysunek 1.1.** Przykładowy układ witryny WWW

Projektanci tworzą wizerunek witryny WWW, wprowadzając ilustracje, style tekstu, tabelki itp. Tworzą szablony pokazujące, w jaki sposób będzie prezentowana treść na stronie.

Jedyną informacją, jakiej potrzebują projektanci, jest to, w jaki sposób powinna wyglądać witryna oraz czy treść powinna być zaprezentowana w postaci aktualności, opisu produktu, żartów itp.

Z kolei programiści tworzą kod aplikacji, którego zadaniem jest przetwarzanie występujących w niej danych (reguły biznesowe). Nie interesuje ich to, jak witryna WWW wygląda (kolory, ilustracje, style tekstu) lub jaki jest układ treści na stronie. Programiści muszą jedynie przekazać treść do szablonów, posługując się nazwami zmiennych uzgodnionych z projektantami.

Z grubsza w ten sposób działa system szablonów. Oprócz tej podstawowej funkcji każdy system szablonów zawiera zbiór własności funkcjonalnych, które jeśli są dobrze udokumentowane, ułatwiają posługiwanie się systemem przez projektantów i programistów.

Celem niniejszej książki jest pokazanie optymalnego sposobu wykorzystania wszystkich własności technologii Smarty.

## Po co stosuje się systemy szablonów?

Tworzenie witryny WWW przypomina tworzenie oprogramowania. Można przystąpić do pracy natychmiast — po prostu zacząć pisać kod, a wszystkie problemy rozwiązywać po drodze — w takim przypadku efekt końcowy będzie trudny do przewidzenia. Można też wyróżnić komponenty projektu, podzielić je na warstwy i stworzyć witrynę WWW, biorąc je pod uwagę. Drugie z podejść określa się terminem oprogramowania o architekturze wielowarstwowej lub wielopoziomowej. W przypadku prostych witryn WWW, składających się z zaledwie kilku linii kodu PHP, można zastosować podejście pierwsze. Wystarczy utworzyć kilka tabel w bazie danych i napisać kod w PHP: utworzyć kilka banerów, wygenerować kilka tabel i spróbować go uruchomić po każdym etapie. Ta metoda ma wielu zwolenników — jest szybka i łatwa.

W przypadku bardziej złożonych projektów internetowych, takich jak portale, witryny e-commerce, systemy ERP itp. również można zastosować pierwszą opcję, ale w większości przypadków, przystępując od razu do kodowania, niewiele się osiągnie. Co więcej, biorąc pod uwagę fakt, że na rynku jest bardzo wiele projektów internetowych, nasza witryna WWW będzie miała szanse tylko wtedy, gdy będzie miała doskonały układ, odpowiedni do tworzonego projektu, oraz oczywiście stabilny kod, przetestowany we wszystkich możliwych sytuacjach. Dlatego właśnie osoby zajmujące się tworzeniem witryn WWW specjalizują się w tworzeniu doskonałego układu witryn WWW (projektanci) albo w ich oprogramowywaniu (programiści). Niezbyt często spotyka się dobrych programistów witryn WWW tworzących doskonałe projekty lub dobrych projektantów, którzy piszą dobry, wolny od błędów kod.

Firmy programistyczne zajmujące się tworzeniem witryn WWW, zazwyczaj mają wydział projektowy i wydział programowania. Nad większością średnich i dużych projektów internetowych pracują zarówno programiści, jak i projektanci.

Programiści stosują bardzo różne sposoby kodowania aplikacji. Gdyby posługiwali się wyłącznie językiem PHP, bez szablonów, projektanci musieliby znać wszystkie sposoby tworzenia kodu stosowanego przez programistów, nazywania zmiennych itp. Z kolei programiści musieliby rozumieć szablony projektantów i generować kod PHP, który wyświetla kod HTML w miejscu, w którym życzą sobie tego projektanci.

Rozważmy następujący przykład:

```
for ( $col = 1; $col < $i; $col++ ) {
    print "<tr> <td> $procesid[$col]</td> <td>$data[$col] </td>
    <td>$value[$col]</td> </tr>"; }
```

Jest to kod PHP wyświetlający w tabelce wartości trzech tablic. Nawet w przypadku tak prostego przykładu jak ten, który pokazano powyżej, trudno розміścić informacje na stronie WWW dokładnie tak, jak oczekiwano.

W przypadku technologii Smarty kod PHP nie zawiera instrukcji print. Zamiast tego programista przekazałby te trzy tablice do projektanta poprzez przypisanie ich do szablonów Smarty. Dalej to już zmartwienie projektanta, który, nie przejmując się kodem PHP, musi zadbać o to, by tabelka dobrze wyglądała na stronie. Jest to jedna z wielkich zalet technologii Smarty. W tej książce dowiemy się, w jaki sposób ją uzyskano.

## Inteligencja technologii Smarty

Technologia Smarty umożliwia bardziej efektywną współpracę pomiędzy programistami, a projektantami bez zagłębiania się jednych w szczegóły działalności drugih. Projektant tworzy szablony stron WWW i pobiera dane ze skryptów PHP utworzonych przez programistów. Programista przekazuje dane do szablonów, nie przejmując się generowaniem kodu HTML. Dzięki temu wszyscy są zadowoleni i bardziej wydajni, ponieważ robią to, w czym są dobrzy.

Przeanalizujmy przypadek witryny e-commerce zajmującej się sprzedażą laptopów. W bazie danych i na stronie WWW powinny się znaleźć takie dane jak nazwa producenta, numer modelu, jego charakterystyka i cena.

Dzięki zastosowaniu technologii Smarty praca projektanta, a także programisty, jest bardzo prosta. Najważniejsze zadania, które wykonują, wymieniono poniżej:

### Zadania programisty:

- Odczytać dane z bazy danych za pomocą prostego zapytania.
- Sprawdzić poprawność danych i przetworzyć je, stosując reguły biznesowe.
- Jeśli jest taka potrzeba, zmodyfikować metody dostępu do danych oraz implementację reguł biznesowych bez ingerencji w pracę projektanta. Na przykład migracja systemu z bazy danych MySQL do PostgreSQL nie powinna wymagać od projektanta żadnych zmian.

### Zadania projektanta:

- Utworzyć projekt HTML, który nie będzie miał wpływu na kod PHP tworzony przez programistę. Jedynym zadaniem projektanta jest rozmieszczenie elementów danych w miejscach uzgodnionych z programistą.
- Wprowadzać zmiany w projekcie bez konieczności konsultacji z programistą i niewymagające od niego wprowadzania zmian.
- Przestać się martwić o techniczne modyfikacje w witrynie, które burzą sposób, w jaki jest ona prezentowana użytkownikom.

W pokazanym powyżej przykładzie można zauważyć, że zadania przydzielone osobom biorącym udział w projekcie dotyczą części, nad którymi te osoby pracują — prezentacji, reguł biznesowych i dostępu do danych.

W dalszej części rozdziału przeanalizujemy proces tworzenia aplikacji (w tym również internetowych) z zastosowaniem architektury wielowarstwowej.

## Alternatywy dla technologii Smarty

Dla większości osób, którym jest potrzebny system obsługi szablonów w PHP, Smarty jest naturalnym wyborem ze względu na jego popularność. Jak łatwo się domyślić, Smarty nie jest jedynym dostępnym na rynku mechanizmem obsługi szablonów dla języka PHP. Jest jednak mechanizmem najbardziej kompletnym i niezawodnym.

Spośród wielu innych mechanizmów obsługi szablonów dla języka PHP warto wymienić następujące:

- PHP Savant: <http://phpsavant.com/yawiki/>,
- PHPLib: <http://phplib.sourceforge.net/>,
- Yats: <http://yats.sourceforge.net/>,

- FastTemplate: <http://www.thewebmasters.net/php/FastTemplate.phtml>,
- SimpleTemplate: <http://simplet.sourceforge.net/>,
- Yapter: <http://yapter.sourceforge.net/>,
- patTemplate: <http://www.php-tools.de/site.php?file=/patTemplate/overview.xml>.

Większość z wymienionych systemów obsługi szablonów ma przewagę nad technologią Smarty polegającą głównie na większej szybkości działania i łatwości korzystania. Wynika to z faktu, iż systemy te nie są tak złożone jak Smarty. Smarty ma jednak znacznie więcej własności od większości systemów konkurencji, a przy odpowiedniej konfiguracji zapewnia przyzwoitą szybkość działania.

Moim zdaniem prawdziwą konkurencją dla technologii Smarty jest PHP Savant, który jest łatwym w użyciu obiektywnym systemem obsługi szablonów dla języka PHP o rozbudowanych możliwościach. System Savant ma wiele własności technologii Smarty.

Jest jednak jedna duża różnica pomiędzy systemem Smarty i Savant, którą w zależności od punktu widzenia można postrzegać jako wielką zaletę lub wielką wadę systemu Savant. Różnica ta polega na tym, że szablony w systemie Savant są napisane w PHP, a nie w innym języku, tak jak w przypadku technologii Smarty. Własność tę można uznać za:

- Zaletę systemu Savant — projektanci pracują z PHP jako językiem szablonów. Nie muszą uczyć się innych języków.
- Wadę systemu Savant i wielką zaletę systemu Smarty jeśli chodzi o bezpieczeństwo. W systemie Savant szablony nie są kompilowane do PHP, ponieważ same są plikami PHP. Dzięki temu projektanci mają dostęp do pełnych możliwości języka PHP. Jeśli nie ufamy projektantowi, zastosowanie systemu Savant nie wchodzi w rachubę. W takim przypadku lepiej stosować technologię Smarty.

W większości przypadków ufanie projektantom w zakresie udzielenia im dostępu do systemu, w którym działają strony WWW, jest ryzykowne, dlatego Smarty jest w dalszym ciągu najlepszym systemem obsługi szablonów dla języka PHP. System Savant jest przyjemny, jednak jest niemal całkowicie pozbawiony mechanizmów bezpieczeństwa. Z kolei inne systemy obsługi szablonów mają tylko niewielki zbiór własności funkcjonalnych dostępnych w systemie Smarty.

## Krótki przewodnik po procesie projektowania oprogramowania

Proces tworzenia witryny internetowej obejmuje kilka czynności, które w różnych projektach mogą być różne, choć w większości przypadków są do siebie podobne.

Najpierw trzeba określić potrzeby klienta, dla którego tworzymy bazę danych. Potrzeby te należy później uzgodnić. Można to zrobić ustnie, ale znacznie lepiej sporządzić odpowiedni dokument. W takim dokumencie powinien znaleźć się opis planowanych własności funkcjonalnych

oprogramowania, a także definicje tabel bazy danych wraz z opisem poszczególnych kolumn. Po uzgodnieniu tego dokumentu z klientem oraz ewentualnym wprowadzeniu zmian można przystąpić do właściwego procesu tworzenia witryny internetowej.

Jeśli piszemy aplikację sami dla siebie, nie musimy sporządzać wspomnianego wyżej dokumentu. W pozostałych przypadkach gorąco zalecam, aby to zrobić. W jednym z moich pierwszych projektów zaniedbałem etap jego tworzenia, co spowodowało, że projekt zakończył się całkowitym fiaskiem. Klient chciał coraz więcej funkcji, za które nie chciał płacić, a gdyby istniał dokument, nie miałby innego wyjścia.

Następna czynność polega na utworzeniu i zaprezentowaniu klientowi kilku ekranów. To zadanie należy do projektanta. Powinien on wspólnie z klientem opracować układ witryny, który spodoba się klientowi. Po uzgodnieniu projektu rozpoczyna się właściwy proces tworzenia witryny WWW. Po zakończeniu projektu, klient ma pewien czas na testowanie, w którym wprowadza się kilka modyfikacji i usuwa znalezione błędy. Na tym projekt się kończy.

Wszystko wygląda pięknie w teorii, jednak w rzeczywistości programiści posługują się takimi językami jak PHP i SQL, natomiast projektanci — HTML i CSS. W związku z tym pomiędzy nimi często występują różnice.

## Praca w zespole

### — warstwy i podział obszarów zainteresowań

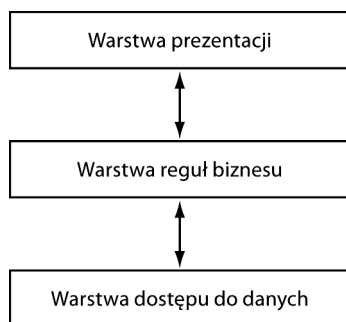
W przypadku, gdy istnieje zagrożenie niepowodzeniem interakcji pomiędzy ludźmi, trzeba jak najbardziej rozdzielić ich zadania. Można to zrobić, stosując architekturę wielowarstwową oprogramowania. Najczęściej spotyka się architekturę trójwarstwową. Należą do niej następujące warstwy:

- warstwa prezentacji,
- warstwa reguł biznesu,
- warstwa dostępu do danych.

Rozdzielenie tych warstw na etapie projektowania aplikacji pozwala na osiągnięcie szybkiego tempa tworzenia aplikacji przy zachowaniu łatwości zarządzania projektem.

Interakcje pomiędzy warstwami w architekturze trójwarstwowej pokazano na rysunku 1.2.

Na samym dole, w **warstwie dostępu do danych**, znajdują się dane, które chcemy zaprezentować użytkownikom, oraz sposoby ich uzyskiwania. Warstwa dostępu do danych może się składać z następujących elementów:



**Rysunek 1.2.** Trójwarstwowa architektura aplikacji

- bazy danych (MySQL, PostgreSQL, MSSQL itp.) oraz języka SQL służącego do wydobywania danych z bazy danych,
- plików, w których są zapisane dane i funkcje w PHP (lub innych językach), które przetwarzają te pliki,
- oprogramowania do zbierania danych (na przykład programu obsługi termometru podłączonego do portu równoległego).

Teraz, kiedy pobraliśmy dane, możemy je przetworzyć w celu uzyskania wyników do wyświetlenia. Przetwarzanie danych i sprawdzanie ich poprawności odbywa się w **warstwie reguł biznesu**, która może zawierać następujące elementy:

- mechanizmy sprawdzania poprawności danych na podstawie obowiązujących reguł (na przykład wyświetlanie tylko tych towarów, które są w magazynie),
- przetwarzanie danych zgodnie z obowiązującymi regułami (na przykład rabaty, wyprzedaże itp.),
- funkcje i formuły służące do obliczania takich danych, jak koszty wysyłki itp.

**Warstwa prezentacji** to miejsce, w którym definiuje się układ strony WWW — sposób wyświetlania danych z warstwy reguł biznesu na stronie WWW. Wykonuje się to za pomocą następujących mechanizmów:

- szablonów stron WWW,
- arkuszy stylów CSS,
- ilustracji, banerów i menu.

Bez mechanizmu obsługi szablonów w warstwie prezentacji znalazłby się kod HTML i PHP tworzący układ strony — byłby to zarówno czysty HTML, jak HTML generowany za pomocą kodu PHP. W takim przypadku nie można mówić o dwóch oddzielnych warstwach: prezentacji i reguł biznesu. W związku z tym, w przypadku złożonych projektów programistycznych współpraca projektantów z programistami byłaby bardzo trudna. W tej sytuacji z pomocą przychodzi technologia Smarty.

# Smarty — podstawowy system obsługi szablonów dla języka PHP

Teraz, kiedy zdecydowaliśmy, że ułatwimy sobie życie dzięki zastosowaniu wielowarstwowej architektury naszej witryny WWW, przyjrzymy się temu, co można osiągnąć za pomocą technologii Smarty, a także temu, czego osiągnąć nie można.

Smarty nie służy do oddzielania kodu HTML od PHP. Podstawowym celem tej technologii jest natomiast oddzielenie logiki aplikacji od logiki jej prezentacji. Gdyby celem technologii Smarty było rozdzielenie kodu PHP od HTML, warstwa prezentacji nie zawierałaby żadnej logiki. Szablony Smarty mogą jednak zawierać logikę, o ile wykorzystuje się ją wyłącznie do celów prezentacji.

Choć może to wyglądać na złamanie sztywnych reguł separacji pomiędzy warstwami i zadaniami, są ku temu ważne powody praktyczne. Przeanalizujemy prosty przykład witryny e-commerce z produktami wyświetlanymi w czterech kolumnach:

- **Bez technologii Smarty:** jeśli w warstwie prezentacji nie ma zaszytej logiki, wyświetlenie produktów w czterech tablicach wymaga modyfikacji kodu reguł biznesu.
- **Z technologią Smarty:** dzięki logice zaszytej w szablonach programista jedynie przekazuje produkty do szablonów w postaci pojedynczej tablicy, a projektant rozmieszcza produkty na stronie tak, jak chce.

Szablony Smarty zapewniają interfejs niemal do wszystkich elementów PHP, a zatem można włączać kod PHP w szablonach. Zaleca się jednak, aby kod PHP był umieszczany przede wszystkim w warstwie reguł biznesu. Na szczęście logika stosowana w szablonach Smarty jest, ogólnie rzecz biorąc, znacznie prostsza niż w skryptach PHP. Dzięki temu projektanci nie muszą wcielać się w rolę programistów tylko po to, by stworzyć logikę prezentacji w projektach Smarty.

## Czy aplikacje z szablonami Smarty są szybkie?

Systemy obsługi szablonów wprowadzają dodatkowy poziom przetwarzania do aplikacji, a zatem można przypuszczać, że aplikacje, w których zastosowano technologię Smarty, będą działały wolniej od ich odpowiedników napisanych w czystym kodzie PHP. Ogólnie rzecz biorąc, wewnątrz języka skryptowego (PHP) stosuje się nowy język pseudoskryptowy (znaczniki mechanizmu obsługi szablonów).

Technologia Smarty jest niezwykle szybka dzięki tzw. kompilacji szablonów. Oznacza to, że szablony są odczytywane, na ich podstawie są tworzone skrypty PHP i dołączane do aplikacji. W ten sposób powstaje jeden skrypt PHP, który po skompilowaniu przez mechanizm obsługi PHP działa dość szybko. Siła tego procesu polega na tym, że szablony są przetwarzane tylko



raz przez mechanizm Smarty, a ponownej kompilacji podlegają jedynie te szablony, które zostaną zmodyfikowane. System Smarty wykonuje to automatycznie. W efekcie mechanizm PHP tworzy szybko działającą aplikację, a dodatkowe koszty związane z zastosowaniem technologii Smarty są nikłe.

Dodatkowo dobrą wiadomością dla osób szczególnie zatroskanych wydajnością witryny jest fakt, iż system Smarty posiada wbudowaną obsługę buforowania, która jeszcze bardziej przyspiesza aplikację. W szczególności dotyczy to witryn WWW z zawartością, która nie zmienia się zbyt często. W buforze można umieścić całą zawartość witryny WWW bądź tylko jej część i określić czas, przez jaki system Smarty będzie utrzymywał zawartość strony w buforze. Proces ten omówimy bardziej szczegółowo w rozdziale 9.

Ponieważ w wyniku kompilacji szablonów Smarty powstają skrypty PHP, nie ma powodu, by zaniechać stosowania technologii buforowania skryptów PHP takich jak PHP Accelerator lub Zend Cache.

Mechanizmy PHP Accelerator oraz Zend Cache bez problemu obsługują wyniki działania systemu Smarty i bardzo dobrze buforują skrypty PHP generowane przez Smarty. Z tego powodu, jeśli wydajność aplikacji jest dla nas priorytetowa, powinniśmy zastosować jedno z wymienionych rozwiązań buforowania w połączeniu z wbudowanym mechanizmem buforowania systemu Smarty.

## Czy szablony Smarty są bezpieczne?

A zatem doszliśmy do wniosku, że technologia Smarty jest szybka. Warto się jeszcze przekonać, czy jest bezpieczna. Moim zdaniem bezpieczeństwo witryny WWW jest najważniejszą cechą każdej witryny internetowej.

Ogólnie rzecz biorąc, zastosowanie technologii Smarty nie zmienia poziomu bezpieczeństwa aplikacji PHP. Oznacza to, że zastosowanie szablonów Smarty nie zmniejsza poziomu bezpieczeństwa aplikacji. Co więcej niektóre własności szablonów Smarty zwiększają poziom bezpieczeństwa aplikacji.

Kiedy zespół składający się z projektantów i programistów tworzy aplikację bez użycia technologii Smarty, projektanci mają dostęp do aplikacji i mogą modyfikować wszystkie skrypty PHP. Nie jest to sytuacja dobra z punktu widzenia bezpieczeństwa, ponieważ projektant o złych zamiarach, mając do dyspozycji wszystkie możliwości języka PHP, łatwo może naruszyć bezpieczeństwo witryny.

Szablony Smarty mają wbudowane własności zabezpieczeń obsługujące sytuacje, w których szablony edytują osoby, którym nie można ufać. W przypadku zastosowania tych mechanizmów projektanci, którzy modyfikują szablony za pośrednictwem niezabezpieczonych połączeń (na przykład FTP), nie mogą uzyskać dostępu do systemu.

Wykorzystując własności zabezpieczeń systemu Smarty, można:

- zezwalać bądź zabraniać używania kodu PHP w szablonach,
- zezwalać na wykorzystanie tylko podzbioru funkcji PHP jako modyfikatorów zmiennych,
- definiować foldery, z których można włączać szablony,
- definiować foldery, z których szablony mogą pobierać lokalne pliki.

Gorąco zachęcam do tego, aby podczas projektowania zawsze pamiętać o bezpieczeństwie. Zastanówmy się, co by się stało, gdyby aplikacja wykorzystywała bazę danych informacji dotyczących kart kredytowych, a projektant włączyłby na stronie WWW niezabezpieczony kod PHP? Dla naszej firmy oznaczałoby to katastrofę.

## Główne własności technologii Smarty

System Smarty oferuje narzędzia do optymalizacji pracy zarówno projektantom, jak i programistom. Podczas lektury tej książki poznamy wszystkie te doskonałe własności, co pozwoli nam się przekonać, jak doskonałym narzędziem jest Smarty. Spróbujmy dokonać przeglądu niektórych spośród tych własności i przeanalizować powody, dla których należy je wykorzystywać, oraz te, dla których lepiej z nich zrezygnować.

### Modyfikatory zmiennych

Podczas wyświetlania zawartości w witrynie WWW niektóre jej fragmenty mogą się zmieniać w zależności od czasu przeglądania witryny oraz tego, skąd pochodzi odwiedzający. Na przykład, w niektórych sytuacjach chcemy wyświetlać daty w różnych formatach. Jeśli wykorzystamy technologię Smarty, zastosowanie różnych formatów wyświetlania daty nie wymaga interwencji programisty. Wystarczy, jeśli programista przekaże do szablonu datę za pomocą odpowiedniej zmiennej, a projektant będzie mógł ją sformatować tak, jak chce.

Projektant może również wyświetlić zmienną wielkimi bądź małymi literami, obciąć blok tekstu, wprowadzić spacje pomiędzy znakami itp. Stosując system Smarty, wszystko to można uzyskać na etapie wyświetlania strony.

Rozważmy sytuację, w której chcielibyśmy wyświetlić nazwę kategorii produktów w postaci niewielkiego tytułu pisanego wielkimi literami ze spacjami pomiędzy znakami (na przykład *P O J A Z D Y*). W bazie danych we właściwej kolumnie jest słowo *pojazd*y. W przypadku użycia technologii Smarty nie trzeba prosić programistę o zmianę tej nazwy na *P O J A Z D Y*. Po przekazaniu zmiennej do szablonu można ją sformatować za pomocą modyfikatorów zmiennych w fazie wyświetlania. Co więcej, późniejsza modyfikacja sposobu wyświetlania kategorii produktów (na przykład do postaci *Pojazdy*), również nie wymaga interwencji programisty.

Więcej informacji na temat modyfikatorów zmiennych można znaleźć w rozdziale 5.

## Funkcje szablonów

Analizując składnię szablonów Smarty, odkryjemy inny doskonały element — **funkcje szablonów**. Rozważmy przypadek projektowania rozbudowanego formularza z wieloma rozwijanymi menu HTML. Stary sposób polega na napisaniu znaczników dla każdego menu rozwijanego. W przypadku zastosowania technologii Smarty wystarczy napisać funkcję dla rozwijanych menu i wywołać ją za każdym razem, kiedy chcemy wyświetlić menu. Jest to sposób prostszy i szybszy. To był tylko nieskomplikowany przykład, jednak dzięki pisaniu funkcji wyświetlających wartość w sposób, który często powtarza się na stronie, można zaoszczędzić dużo czasu.

Więcej informacji na temat funkcji szablonów można znaleźć w rozdziale 6.

## Debugowanie

Na każdym etapie tworzenia aplikacji, programiści i projektanci muszą wykonać debugowanie po to, by w łatwy sposób poprawić swoją pracę.

Zastanówmy się, ile czasu możemy stracić, jeśli popełnimy literówkę w nazwie zmiennej i nie dysponujemy narzędziami do debugowania. Z mojego doświadczenia w pracy w charakterze programisty wynika, że jest to najgorszy scenariusz. Jeszcze gorszy od sytuacji, w której popełnia się błędy w algorytmie.

Chociaż funkcja debugowania w czystym PHP jest dostępna, system Smarty jest wyposażony w konsolę debugowania umożliwiającą poprawianie błędów związanych z szablonami Smarty. Jest to bardzo rozbudowane narzędzie, które dostarcza informacji o wszystkich włączonych szablonach, przypisanych zmiennych oraz zmiennych pliku konfiguracyjnego dla bieżącego szablonu.

Dzięki zastosowaniu technologii Smarty można zmodyfikować format konsoli debugowania. Pozwala to na wyróżnienie elementów, które w debugowaniu są ważne. Co więcej, korzystając z funkcji Smarty w szablonie, można rzucić zawartość konsoli debugowania na stronę. Ponieważ jednak funkcje Smarty wykonują się w fazie działania aplikacji, można zobaczyć jedynie użyte zmienne, a nie włączone szablony.

Więcej informacji na temat debugowania w systemie Smarty można znaleźć w rozdziale 7.

## Wtyczki

Jedną z najistotniejszych własności dla firmy zajmującej się tworzeniem oprogramowania jest możliwość wielokrotnego wykorzystywania kodu. W ten sposób można zaoszczędzić czas i pieniądze. Tworząc nowe projekty zawierające własności funkcjonalne związane z projektami tworzonymi wcześniej, można wykorzystać kod z poprzednich projektów. Własność ta jest jednym z czynników, które powodują, że przedsiębiorstwa programistyczne przynoszą zyski.

Wykorzystywanie kodu z innych aplikacji za pomocą techniki kopiowania i wklejania wymaga modyfikowania zmiennych, nazw funkcji, testowania oraz uważnej integracji z nową aplikacją, ale jest to sposób szybszy w porównaniu z pisanem kodu od początku.

Jest jednak lepsza metoda wielokrotnego wykorzystywania kodu — **wtyczki** (ang. *plug-ins*). Podczas tworzenia witryny warto wyodrębnić własności, które można wykorzystać wielokrotnie nawet w tym samym projekcie i utworzyć z nich wtyczki. W ten sposób, po utworzeniu kilku witryn, będziemy dysponowali zbiorem dobrze działających wtyczek, które można włączać do nowych projektów bez żadnych modyfikacji. To oczywiście pozwala na zaoszczędzenie mnóstwa czasu i pracy. Aby można było skorzystać z tego sposobu, oprogramowanie wykorzystywane do tworzenia projektu musi obsługiwać wtyczki.

System Smarty posiada architekturę wtyczek, którą można wykorzystać w celu dostosowania systemu do indywidualnych potrzeb. W systemie Smarty można napisać wtyczki dla funkcji, funkcji kompilatora, funkcji blokowych, modyfikatorów, zasobów, operacji wstawiania, a także filtrów wstępnych, końcowych i wynikowych.

Wiele programów wykorzystujących architekturę wtyczek ładuje wszystkie wtyczki przed kompilacją. Nie zawsze jest to złe, jednak projektanci systemu Smarty pamiętali o wydajności i zaprojektowali system w ten sposób, że wtyczki ładują się dopiero wtedy, gdy są wywoływane w szablonie. Co więcej, wtyczka raz wywołana w szablonie ładuje się w nim tylko raz. Nawet wtedy, gdy większa liczba egzemplarzy szablonu Smarty żąda tej samej wtyczki.

Na przykład w aplikacji e-commerce można utworzyć wtyczkę dla własności koszyka na zakupy lub filtrów konwersji waluty. Można je później wykorzystać w innych projektach e-commerce. Choć nie musimy pisać kodu jeszcze raz, możemy uzyskać za niego podobną zapłatę.

Więcej informacji na temat wtyczek można znaleźć w rozdziale 10.

## Filtry

Zdyscyplinowani projektanci piszą wiele komentarzy w swoich szablonach, co powoduje, że pliki wynikowe po kompilacji mają dużą objętość. Twórcy systemu Smarty pomyśleli o tym i stworzyli zbiór filtrów pozwalających na rozwiązanie tego problemu. W systemie Smarty są dostępne **filtry wstępne** (ang. *prefilters*) — są to funkcje PHP, które przetwarzają szablony Smarty przed ich skompilowaniem. Dzięki filtrom wstępnym można zmodyfikować w szablonach dowolne elementy jeszcze przed *kompilacją szablonu*. Na przykład, stosując filtry wstępne, można z łatwością usunąć z szablonu wszystkie komentarze.

Jeśli jednak chcemy wprowadzić jakieś komentarze do skompilowanych szablonów, możemy to zrobić, stosując **filtry końcowe** (ang. *postfilters*). Filtry końcowe w systemie Smarty to funkcje PHP, które przetwarzają szablony Smarty po ich skompilowaniu.

Zawartość szablonów można również filtrować za pomocą **filtrów wynikowych** (ang. *output filters*) — funkcji PHP służących do filtrowania wyniku szablonu w czasie jego działania. Dzięki temu mechanizmowi można na przykład, zastąpić zakazane słowa znakiem \*, ukryć adresy e-mail itp. Dzięki rozbudowanym filtrom programista ma pełną kontrolę nad szablonami.

Więcej informacji na temat filtrów wstępnych, końcowych i wynikowych można znaleźć w rozdziale 11.

## Wnętrze systemu Smarty

Z punktu widzenia projektantów Smarty jest nowym, prostym systemem skryptowym o rozbudowanych możliwościach, który można używać razem z HTML w celu usprawnienia pracy z programistami. Nowy język skryptowy, który system Smarty udostępnia projektantom, zapewnia sposoby ułatwienia ich pracy.

Dobłą ilustracją do wcześniejszego zdania jest następujący fragment kodu:

```
<select name="Employee">
    {HTML_options options=$names}
</select>
```

Pierwszy i ostatni wiersz jest napisany w języku HTML, natomiast wiersz środkowy to Smarty. Jego działanie polega na wybraniu wierszy z wartościami z tablicy \$names otrzymanej od programisty. Dzięki temu projektant musi jedynie poprosić programistę, aby ten przekazał nazwiska pracowników w tablicy \$names (minimum interakcji). Zamiast wielokrotnego pisania znaczników <option></option> projektant wykorzystuje funkcję Smarty, która upraszcza jego pracę.

Zamiast umieszczania kodu pokazanego w powyższym przykładzie w pliku HTML projektant umieszcza go w pliku z rozszerzeniem *.tpl* i przekazuje programiście pełną nazwę pliku z szablonem (*.tpl*).

Jedną z części tej książki zatytułowano „Smarty dla projektantów”. Projektanci nauczą się z niej języka skryptowego wraz z przydatnymi przykładami, wskazówkami i sztuczkami zaczerpniętymi z doświadczeń autora.

Z punktu widzenia programisty Smarty jest rozbudowaną klasą PHP zawierającą zmienne i metody. Nazwa klasy to Smarty, a zatem w celu przypisania zmiennej obiektu klasy Smarty programista musi napisać następujący kod:

```
$smarty = new Smarty;
```

Klasa Smarty zapewnia programistom lepszy sposób interakcji z projektantami. Załóżmy, na przykład, że projektant poprosił o nazwiska wszystkich pracowników w zmiennej \$names. Po wydobyciu nazwisk z bazy danych w postaci tablicy, programista przekazuje ją do projektanta. Można to zrobić, wpisując następujący kod:

```
$smarty->assign('names', $names_array);
```

Dla programistów jedną z największych zalet technologii Smarty jest brak konieczności umieszczania kodu HTML w skryptach PHP. Kod HTML znajduje się w plikach *.tpl*, które są wyświetlane za pomocą metod klasy Smarty:

```
$smarty->display("index.tpl");
```

Drugą część tej książki zatytułowano *Smarty dla programistów*. Po jej przestudiowaniu programiści zapoznają się z klasą Smarty oraz najlepszymi sposobami jej wykorzystywania.

## Instalacja i konfiguracja systemu Smarty

System Smarty jest rozprowadzany zgodnie z licencją *GNU Lesser General Public License*, która stanowi kontynuację licencji *GNU Library General Public License*. Różnica pomiędzy nową licencją *Lesser GPL* i starą *Library GPL* polega na tym, że programy z licencją *Lesser GPL* są w mniejszym stopniu chronione. Oznacza to, że biblioteki rozprowadzane zgodnie z tą licencją są darmowe i można je wykorzystać do tworzenia oprogramowania komercyjnego. Stara licencja *Library GPL* pozwala na wykorzystywanie oprogramowania rozprowadzanego na jej prawach wyłącznie do tworzenia darmowego oprogramowania.

Istotne jest to, że Smarty jest narzędziem darmowym, które można wykorzystać do tworzenia oprogramowania komercyjnego bez konieczności wnoszenia opłat.

Czytelnicy, którzy chcą się przekonać, czy postępują w 100% zgodnie z prawem, powinni przeczytać całą licencję *Lesser GPL* dostępną pod adresem <http://www.gnu.org/copyleft/lesser.html>.

### Krok 1. Zdobycie systemu Smarty

Najnowszą stabilną wersję kodu źródłowego systemu Smarty można pobrać z witryny WWW systemu Smarty pod adresem <http://smarty.php.net/download.php>. Zawsze należy pamiętać, aby pobrać najnowszą wersję systemu, ponieważ zawiera ona poprawki błędów wszystkich wersji wcześniejszych.

Ściśle rzecz biorąc, Smarty jest biblioteką PHP, a jej wymagania są bardzo proste: serwer WWW z obsługą PHP w wersji 4.0.6 lub nowszej. Jako serwer WWW można wykorzystać Apache, ze względu na jego popularność, ale nie jest to jedyny możliwy wybór. Instrukcja instalacji serwera Apache jest dostępna pod adresem <http://www.apache.org>.

Po pobraniu systemu Smarty z serwera i rozpakowaniu archiwum tworzy się folder z dystrybucją systemu Smarty. W głównym folderze dystrybucji znajduje się folder *libs* zawierający następujące pliki i foldery:

- *smarty.class.php*,
- *smarty\_Compiler.class.php*,
- *config\_File.class.php*,
- *debug.tpl*,
- *\internals*,
- *\plugins*.

Wszystkie te pliki są wymagane do działania aplikacji Smarty. Skryptów PHP nie należy modyfikować.

## Krok 2. Konfigurowanie PHP w sposób umożliwiający odnalezienie bibliotek Smarty

Metoda instalacji systemu Smarty jest prosta — należy dodać folder *libs* do ścieżki include w pliku *php.ini*.

### W systemie Windows

Oto bardzo prosta, a jednocześnie skuteczna metoda konfiguracji bibliotek Smarty: najpierw należy utworzyć folder, na przykład *mycode* w głównym katalogu dokumentów instalacji serwera Apache. Następnie trzeba skopiować folder *libs* z dystrybucji Smarty do folderu *mycode* (na przykład *c:\apache\htdocs\mycode\libs\*).

Po wykonaniu tej czynności, należy wyedytować plik *php.ini* i wprowadzić do niego zapis `include_path = ".;c:\apache\htdocs\mycode\libs\"`, gdzie *c:\apache\htdocs\mycode\libs\* reprezentuje lokalizację folderu *libs*.

### W systemie Linux

Załóżmy, że dystrybucję Smarty pobraną z internetu rozpakowaliśmy w katalogu */usr/local/Smarty*.

Należy wyedytować plik *php.ini* i wprowadzić doń zapis `include_path = "./usr/local/Smarty/libs"`. To wszystko. Aby uniknąć edycji pliku *php.ini*, można również skopiować folder *libs* do dowolnej lokalizacji w ramach ścieżki `include_path` ustawionej w tym pliku.

Jest to najłatwiejszy sposób instalacji bibliotek Smarty. W każdym skrypcie PHP, który ma korzystać z szablonów Smarty, należy załadować skrypt *Smarty.class.php*. Zatem jeżeli plik ten znajduje się w folderze ścieżki include języka PHP, utworzenie egzemplarza klasy Smarty wymaga wprowadzenia następującego kodu:

```
<?php
require('Smarty.class.php');
$smarty = new Smarty;
?>
```

Po skonfigurowaniu środowiska należy zmienić uprawnienia do folderu *templates\_c* znajdującego się w ścieżce skryptów. Oznacza to, że folder, w którym znajduje się kod, oraz *templates\_c* muszą być zapisane w tym samym katalogu. Wykorzystując FTP, można zmienić uprawnienia do folderu *templates\_c* na 777 (rwxrwxrwx).

Operację tę można wykonać za pomocą następującego polecenia powłoki:

```
chmod -R 0777 templates_c
```

## Alternatywa kroku 2. Korzystanie z bibliotek Smarty w sytuacji, kiedy nie mamy pełnego dostępu do systemu

Jeśli nie mamy uprawnień do modyfikacji pliku *php.ini* lub skopiowania katalogu *libs* biblioteki Smarty do jednego z folderów w ramach ścieżki *include\_path*, w dalszym ciągu możemy używać biblioteki Smarty. W tym celu wystarczy umieścić katalog *libs* w katalogu, w którym znajdują się skrypty wykorzystujące szablony Smarty.

System Smarty korzysta ze stałej PHP o nazwie *SMARTY\_DIR*, której nie trzeba ustawiać w przypadku korzystania z pierwszego sposobu instalacji polegającego na edycji pliku *php.ini*. Można jednak ręcznie ustawić stałą *SMARTY\_DIR* w taki sposób, aby wskazywała na ten sam katalog, w którym znajduje się katalog *libs*.

### ■ W systemie Windows:

```
<?php
define('SMARTY_DIR', 'C:\apache\htdocs\mycode\libs\');
require(SMARTY_DIR . 'Smarty.class.php');
$smarty = new Smarty;
?>
```

### ■ W systemie Linux:

```
<?php
define('SMARTY_DIR', '/usr/local/smarty/libs');
require(SMARTY_DIR . 'Smarty.class.php');
$smarty = new Smarty;
?>
```

W związku z wieloma możliwościami korzystania z technologii Smarty w niniejszej książce przyjęto założenie, że folder *libs* skopiowano z głównego katalogu dystrybucji systemu Smarty do folderu *mycode* w obrębie głównego katalogu dokumentów systemu Apache i dodano tę lokalizację do pliku *php.ini*. W przypadku systemu Linux dodatkowo założono, że uprawnienia do folderu *templates\_c* ustawiono na 777 (rwxrwxrwx).



## Krok 3. Konfiguracja Smarty w aplikacji

Kiedy już mamy pewność, że system odnajdzie plik *Smarty.class.php*, powinniśmy skonfigurować foldery wykorzystywane w aplikacji. Każda aplikacja korzystająca z szablonów Smarty wymaga czterech folderów, które domyślnie mają nazwy *templates*, *templates\_c*, *configs* oraz *cache*.

Każdy z tych folderów definiuje się za pomocą właściwości klasy Smarty: `$template_dir`, `$compile_dir`, `$config_dir` i `$cache_dir`. Każda aplikacja Smarty powinna używać własnych folderów.

W systemie Linux biblioteka Smarty wymaga uprawnień zapisu do katalogów `$compile_dir` i `$cache_dir`, a zatem trzeba ustawić takie uprawnienia dla użytkowników serwera WWW. Należy otworzyć plik *httpd.conf* (plik konfiguracyjny serwera Apache) i odczytać z niego użytkownika i grupę wykorzystywaną do uruchomienia serwera. Zazwyczaj wykorzystuje się użytkownika i grupę *nobody*, zatem podobnie zrobię w moim przykładzie. Założmy, że główny katalog dokumentów serwera WWW to */www/htdocs*. Aby udzielić uprawnień do zapisu użytkownikowi *nobody*, trzeba ustawić uprawnienia do plików */www/htdocs/templates\_c* i */www/htdocs/cache*.

```
chown nobody:nobody /www/htdocs/templates_c
chown nobody:nobody /www/htdocs/cache
chmod 770 /www/htdocs/templates_c
chmod 770 /www/htdocs/cache
```

## Krok 4. Sprawdzenie poprawności instalacji

Aby się upewnić, że instalację wykonano prawidłowo, należy skopiować folder *demo* z dystrybucji Smarty do głównego katalogu dokumentów serwera WWW (na przykład *c:\Apache2\htdocs*), sprawdzić, czy w pliku *index.php* jest instrukcja `require('Smarty.class.php');`, i wpisać *http://localhost/demo/index.php* w polu adresu w przeglądarce. Powinna wyświetlić się strona demonstracyjna (rysunek 1.3) z informacjami o możliwościach biblioteki Smarty oraz konsolą debugowania. Należy pamiętać, aby w przeglądarce włączyć wyświetlanie wyskakujących okien.

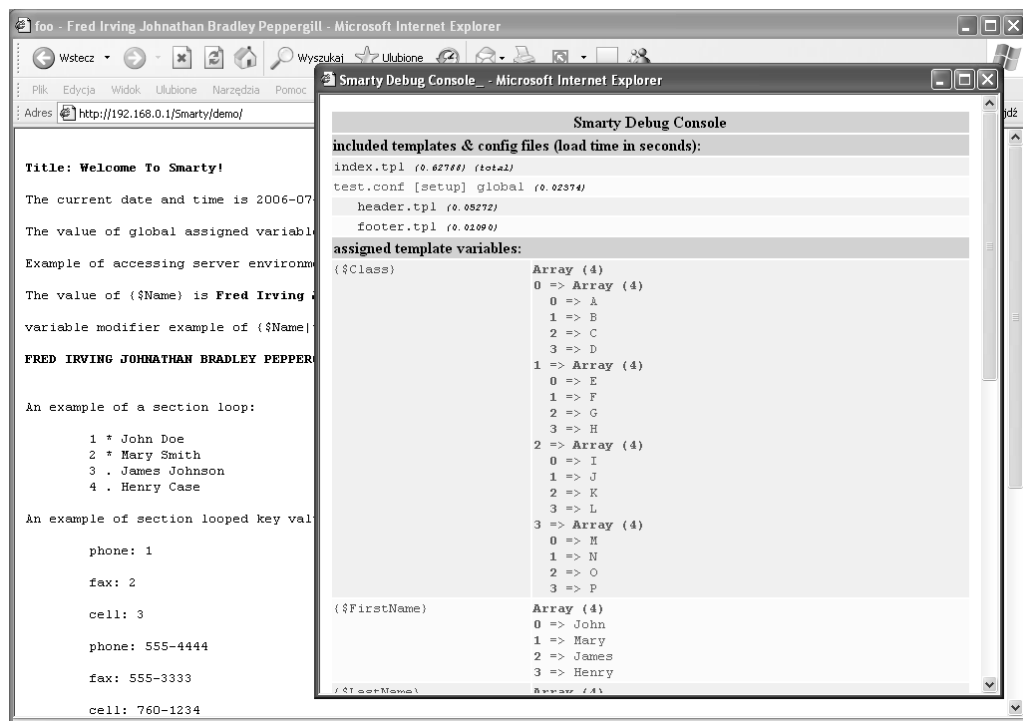
## Wersja rozwojowa systemu Smarty w repozytorium CVS

Jeśli ktoś chce zapoznać się z własnościami systemu Smarty, które są przedmiotem pracy programistów i są planowane do wdrożenia w przyszłych wersjach systemu, może pobrać najnowszą wersję systemu (niestabilną) i większość poprzednich wydań z serwera CVS PHP. System CVS można pobrać pod adresem *http://cvs.php.net/projectDocumentList*.

Aby pobrać Smarty z serwera PHP CVS, najpierw należy zalogować się na serwerze za pomocą następującego polecenia:

```
cvs -d :pserver:cvsread@cvs.php.net:/repository login
```

wykorzystując ciąg *phpfi* jako hasło.



Rysunek 1.3. Strona demonstracyjna systemu Smarty

Aby pobrać bieżące drzewo CVS systemu Smarty, wystarczy wpisać:

```
cvs -d :pserver:cvsread@cvs.php.net:/repository co smarty
```

Można również pobrać wskazaną stabilną wersję systemu Smarty za pomocą następującego polecenia:

```
cvs -d :pserver:cvsread@cvs.php.net:/repository co -r smarty_X_Y_Z smarty
```

przy czym `X_Y_Z` oznacza numer wersji. Na przykład wpisanie `smarty_2_6_10` w wierszu polecenia powyżej spowoduje pobranie wersji 2.6.10 systemu Smarty do folderu `smarty`.

Użycie bieżącej dystrybucji systemu Smarty pobranej z serwera CVS PHP w aplikacjach „produkcyjnych” nie jest zalecane. Jest to wersja niestabilna, nad którą trwają prace. Do tworzenia witryn WWW wykorzystywanych w praktyce należy używać najnowszej stabilnej wersji systemu Smarty.

## Aktualizacja witryny korzystającej z szablonów Smarty

System Smarty jest zgodny wstecz, a zatem jeśli wystąpi potrzeba aktualizacji witryny Smarty, wystarczy wykonać następujące czynności:

- Pobrać najnowszą dystrybucję systemu Smarty zgodnie z opisem zamieszczonym w kroku 1.
- Zastąpić starą zawartość folderu *libs*, w którym zainstalowano system Smarty, zawartością folderu *libs* z nowego głównego katalogu dystrybucji.
- Upewnić się, że w żadnym z katalogów wchodzących w skład ścieżki *include* interpretera PHP nie ma starego pliku *Smarty.class.php*.

## Podsumowanie

Z rozdziału dowiedzieliśmy się, że stosowanie wielowarstwowej architektury oprogramowania ułatwia życie. Celem stosowania systemu Smarty jest oddzielenie logiki warstwy reguł biznesowych od logiki prezentacji. Smarty to system szybki i bezpieczny. Wynika to ze sposobu przetwarzania szablonów — kompilacji. W systemie Smarty występują filtry wstępne, końcowe i wynikowe, które gwarantują programiście pełną kontrolę nad wynikami generowanymi przez szablon i ich zawartością. Utworzenie uchwytów do zasobów umożliwia korzystanie z szablonów z dowolnego kodu źródłowego, do którego jest dostęp z PHP. System Smarty posiada architekturę wtyczek, którą można wykorzystać w celu dostosowania systemu do indywidualnych potrzeb. Jest również wyposażony w rozbudowane narzędzia debugowania. Z rozdziału dowiedzieliśmy się również, w jaki sposób instaluje się bibliotekę Smarty w systemach Windows i Linux.