Administracja katalogiem: działy i kategorie

W poprzednich częściach pracowałeś z informacjami katalogowymi, które już istniały w bazie danych. Prawdopodobnie sam wstawiłeś jakieś rekordy, a może pobrałeś informacje o dziale, kategorii i produkcie. Oczywiście oba sposoby są nie do zaakceptowania w przypadku prawdziwej witryny internetowej, więc musisz napisać trochę kodu, aby umożliwić łatwe zarządzanie danymi. To powiedziawszy, ostatni szczegół, o który należy zadbać przed uruchomieniem strony internetowej to stworzyć swój interfejs administracyjny. Chociaż odwiedzający nigdy nie zobaczą tej części, jest to klucz do dostarczenia wysokiej jakości witryny internetowej Twojemu klientowi. W tej i następnej części zaimplementujesz stronę administracyjną katalogu. Dzięki tej funkcji ukończysz pierwszy etap rozwoju swojej strony internetowej! Ponieważ tę stronę można zaimplementować na wiele sposobów, wymagana jest poważna dyskusja z klientem, aby uzyskać konkretną listę wymaganych funkcji. W naszym przypadku wdrożymy panel kontrolny pozwalający na zarządzanie działami serwisu, kategoriami, produktami oraz atrybutami produktów. Zajmujemy się administrowaniem działami i kategoriami, resztę zostawiając do Części 11. Dokładniej, stworzymy funkcje, które pozwalają na

• Dodawanie i usuwanie działów

• Modyfikowanie informacji o istniejących działach (nazwa i opis)

• Przeglądanie listy kategorii należących do działu

• Dodawanie i usuwanie kategorii

• Edycja informacji o istniejących kategoriach (nazwa i opis)

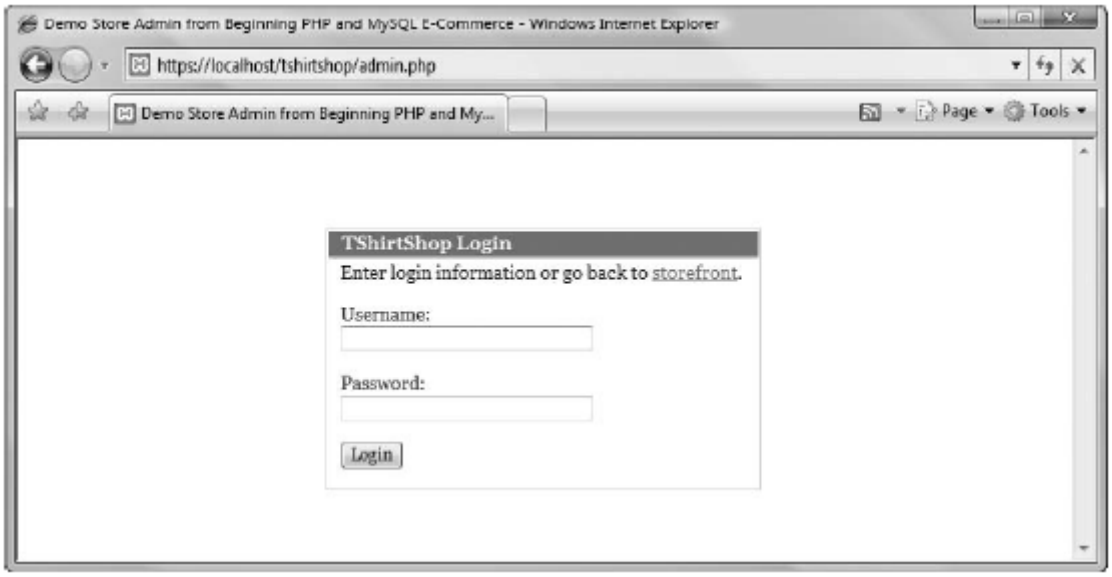
Aby zabezpieczyć poufne strony witryny, takie jak sekcja administracyjna, wykonaj również następujące czynności:

• Zaimplementuj formularz logowania, w którym administrator musi podać nazwę użytkownika i hasło.

• Dowiedz się, jak zabezpieczyć formularz logowania i strony administracyjne za pomocą protokołu SSL.

Podgląd strony administracyjnej katalogu

Chociaż długa lista celów może początkowo wydawać się onieśmielająca, będą one łatwe do wdrożenia. Omówiliśmy już większość teorii w poprzednich częściach, ale tu dowiesz się całkiem sporo. Pierwszym krokiem do stworzenia strony administracyjnej katalogu jest stworzenie mechanizmu logowania, który zostanie zaimplementowany jako prosta strona logowania, którą można zobaczyć na rysunku.

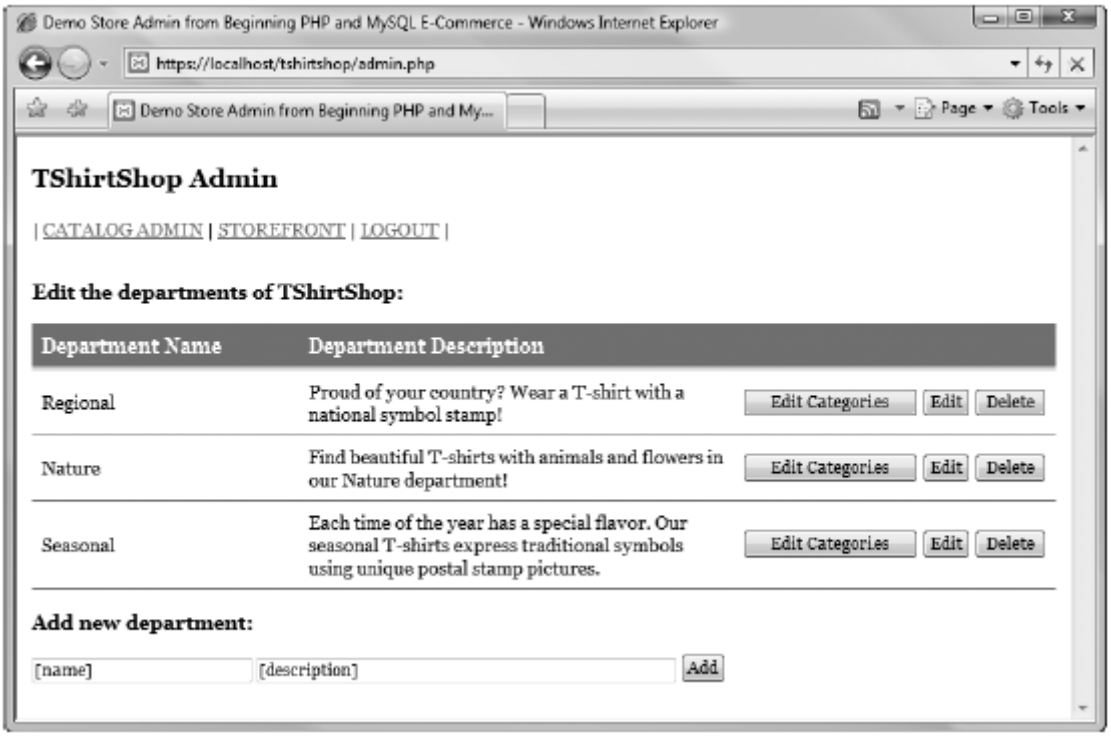


Następnie budujesz część zarządzającą witryny (powszechnie nazywaną panelem sterowania), tworząc jej stronę główną (admin.php), powiązany z nią szablon (store_admin.tpl), szablon menu głównego (admin_menu.tpl) używany do poruszaj się po różnych sekcjach administracyjnych, które rozszerzymy w następnych rozdziałach, szablon złożony do zarządzania uwierzytelnianiem (admin_login) oraz dwa szablony złożone do administrowania katalogiem (admin_departments i admin_categories). Po zalogowaniu administratorowi prezentowana jest lista działów (generowana przez szablon admin_departments Smarty, który jest ładowany ze strony głównej administracji admin.php), jak pokazano na rysunku. Tutaj administrator może:

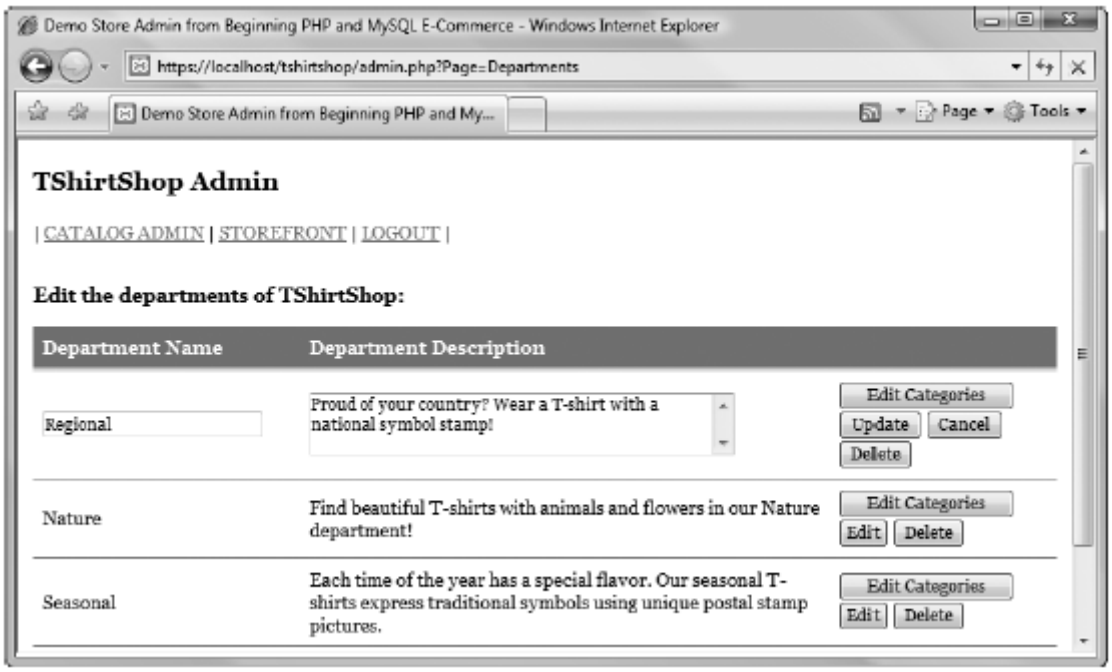
• Edytować nazwę lub opis działu, klikając przycisk Edytuj.

• Wyświetlić kategorie należące do działu, klikając przycisk Edytuj kategorie.

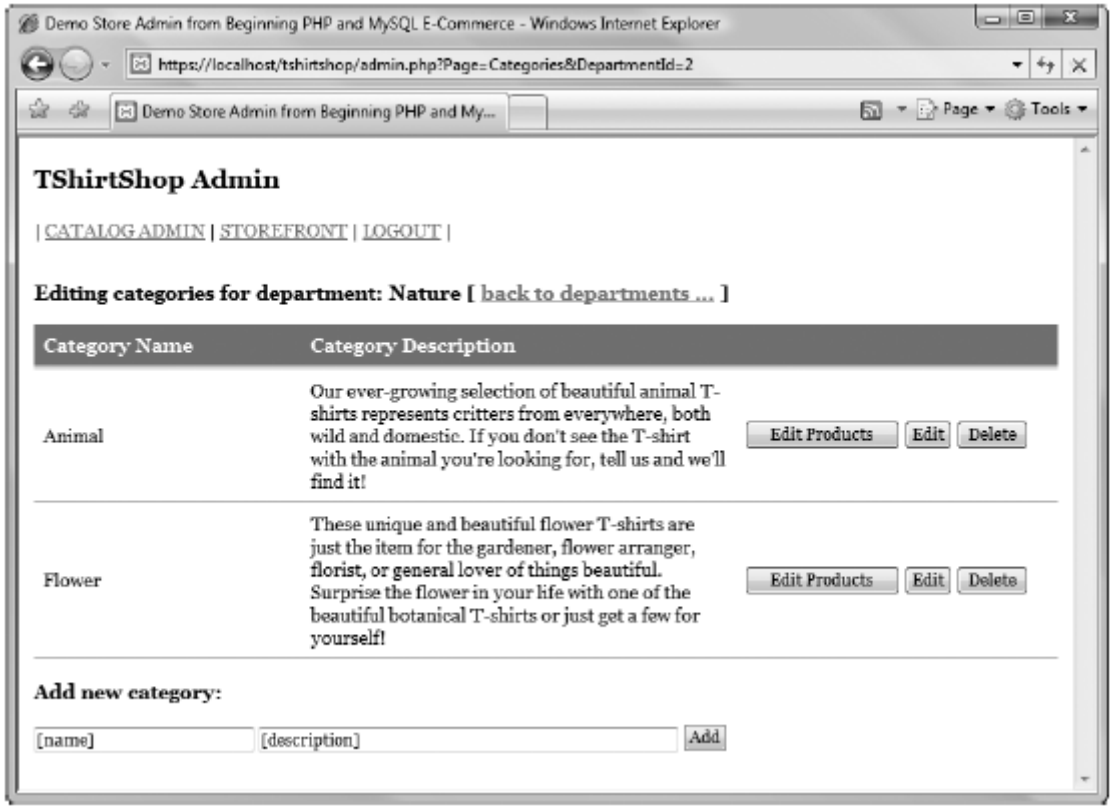
• Całkowicie usunąć dział z bazy danych, klikając przycisk Usuń (działa to tylko wtedy, gdy dział nie ma powiązanych kategorii).



Po kliknięciu przycisku Edytuj odpowiedni wiersz z tabeli przechodzi w tryb edycji, a jego pola stają się edytowalne, jak pokazano na rysunku. Ponadto, jak widać, zamiast przycisku Edytuj dostępne są przyciski Aktualizuj i Anuluj. Kliknięcie Aktualizuj przesyła zmiany do bazy danych, natomiast kliknięcie Anuluj po prostu wychodzi z trybu edycji i przywraca pierwotny stan tabeli danych.



Administrator może dodawać nowe działy, wprowadzając nazwę i opis nowego działu w pola tekstowe pod tabelą i klikając przycisk Dodaj. Gdy administrator kliknie przycisk Edytuj kategorie, strona admin.php jest ponownie ładowana z dodatkowym parametrem w ciągu zapytania: DepartmentId. Ten parametr mówi admin.php, aby załadował szablon admin_categories Smarty, który pozwala administratorowi edytować kategorie należące do wybranego działu



Ta strona działa podobnie do tej dla działów redakcyjnych. Otrzymasz również link („powrót do działów”), który przeniesie Cię z powrotem do strony administracyjnej działu. Logika nawigacji między stronami administracyjnymi działu, kategorii i produktu odbywa się za pomocą parametrów ciągu zapytania. Jak widać na rysunku , po wybraniu działu jego identyfikator jest dołączany do ciągu zapytania. Użyłeś tej techniki również podczas tworzenia strony index.php. Tam zdecydowałeś, który skomponowany szablon ma zostać załadowany (w czasie wykonywania) przez analizowanie parametrów ciągu zapytania. Część administracyjna katalogu strony składa się z admin.php i szeregu innych plików PHP i szablonów Smarty. Będziesz budować te komponenty pojedynczo. Dla każdego komponentu najpierw zaimplementujesz warstwę prezentacji, następnie napiszesz kod warstwy biznesowej, a na końcu napiszesz metody warstwy danych. W kolejnych rozdziałach rozszerzysz sekcję administracyjną witryny. W Części 11 dodasz funkcje zarządzania produktami, a w dalszych częściach zaimplementujesz funkcje zarządzania zamówieniami i koszykiem zakupów, a także będziesz obsługiwał poufne dane klientów, takie jak dane kart kredytowych, numery telefonów itd.

Konfigurowanie strony administracyjnej katalogu

Przed zbudowaniem jakichkolwiek stron administracyjnych musimy wprowadzić mechanizm bezpieczeństwa ograniczający dostęp do tych stron. Tylko upoważniony personel powinien mieć możliwość modyfikowania katalogu produktów! Bezpieczeństwo to oczywiście duży temat, a jego złożoność w dużej mierze zależy od wartości danych, które chronisz. Chociaż nie mamy środków, aby stworzyć tak bezpieczne środowisko, jakie wdrażają banki, na przykład tworząc sklep internetowy, nadal ponosimy wielką odpowiedzialność za zapewnienie bezpieczeństwa naszych danych i danych naszych klientów. Nasza implementacja zabezpieczeń dotyczy tych ważnych koncepcji:

• Uwierzytelnianie: jest to proces, w którym użytkownicy są jednoznacznie identyfikowani. Typowym sposobem identyfikacji użytkowników, który zaimplementujemy również w TShirtShop, jest poproszenie o nazwę użytkownika i hasło.

• Autoryzacja: ta koncepcja odnosi się do procesu identyfikowania zasobów, do których uwierzytelniony użytkownik może uzyskać dostęp, i odpowiedniego ograniczania jego dostępu. Na przykład możesz mieć administratorów, którzy mogą tylko edytować nazwy i opisy produktów, oraz administratorów, którzy mogą również przeglądać dane osobowe klientów. Administratorzy naszego małego sklepu będą mieli dostęp do wszystkich zastrzeżonych obszarów, ale w miarę powiększania się witryny możesz chcieć delegować zadania administracyjne większej liczbie pracowników, zarówno ze względów zarządzania, jak i bezpieczeństwa.

• Bezpieczny kanał komunikacji: Oczywiście wszystkie nasze wysiłki związane z uwierzytelnianiem i autoryzacją są daremne, jeśli hakerowi łatwo jest przeprowadzić atak typu „man-in-the-middle", który odnosi się do scenariusza, w którym dana osoba nasłuchuje ruchu na sieć do przechwytywania wrażliwych danych. Taki atak może nastąpić, gdy administrator zaloguje się, podczas gdy atakujący nasłuchuje ruchu sieciowego, aby przechwycić nazwę użytkownika i hasło administratora. Aby ustrzec się przed tym potencjalnym problemem, stosujemy protokół HTTPS, który szyfruje przesyłane dane i zapewnia pewien stopień poufności transmisji.

Korzystanie z bezpiecznych połączeń

HTTP nie jest bezpiecznym protokołem, a nawet jeśli Twoja witryna chroni wrażliwe obszary za pomocą haseł (lub innych form uwierzytelniania), przesyłane dane mogą zostać przechwycone i skradzione. Aby tego uniknąć, należy skonfigurować aplikację do pracy z połączeniami Secure Socket Layer (SSL) przy użyciu protokołu Hypertext Transport Protocol, Secure (HTTPS). Aby móc akceptować przychodzące połączenia HTTPS, serwer WWW musi być skonfigurowany z certyfikatem bezpieczeństwa. Certyfikaty bezpieczeństwa to zasadniczo pary kluczy publiczno-prywatnych, podobne do tych używanych w asynchronicznych algorytmach szyfrowania. Możesz je wygenerować samodzielnie, ale jeśli nie jesteś zaufanym urzędem certyfikacji (takim jak VeriSign lub Thawte), ta metoda może być problematyczna. Cyfrowo podpisane certyfikaty SSL, które nie są wydawane przez zaufane urzędy certyfikacji, spowodują, że przeglądarki będą wątpić w Twoje bezpieczeństwo. Gdy użytkownik uzyskuje dostęp do bezpiecznych stron, których certyfikat nie jest wystawiony przez zaufany urząd certyfikacji, przeglądarka wyświetli komunikat ostrzegawczy. Nie jest to katastrofalne w przypadku zabezpieczania stron, które mają odwiedzać pracownicy Twojej firmy, ale z pewnością wpłynęłoby na zaufanie klientów, gdyby takie ostrzeżenie pojawiło się np. podczas płacenia za zamówienie. Jeśli skonfigurowałeś swój system za pomocą XAMPP, twój serwer sieciowy Apache jest już skonfigurowany z certyfikatem. W przypadku scenariusza produkcyjnego musisz kupić zaufany certyfikat za pośrednictwem swojej firmy hostingowej lub, jeśli samodzielnie zarządzasz serwerem internetowym, uzyskać certyfikat SSL od znanej i szanowanej organizacji, która specjalizuje się w bezpieczeństwie sieciowym, na przykład:

• VeriSign (http://www.verisign.com/)

• Thawte (http://www.thawte.com/)

• InstantSSL (http://www.instantssl.com/)

Przeglądarki internetowe mają wbudowane certyfikaty główne organizacji takich jak te i są w stanie uwierzytelnić podpis cyfrowy dostarczonych przez nie certyfikatów SSL. Oznacza to, że nie pojawi się żaden komunikat ostrzegawczy, a połączenie zabezpieczone SSL będzie dostępne przy minimalnym zamieszaniu. Na przykład podczas ładowania takiego adresu URL w Operze, obok paska adresu pojawia się mała złota kłódka. Kliknięcie tego symbolu pokazuje nazwę firmy, która zarejestrowała certyfikat SSL.

Certyfikat, który mamy z XAMPP, wystawiony przez lokalną maszynę, nie znajduje się na liście zaufanych dostawców certyfikatów (oczywiście). Przy tej konfiguracji przeglądarki internetowe będą wyświetlać komunikat ostrzegawczy, taki jak ten wyświetlany przez Safari pokazany na rysunku.

Jeśli klikniesz Pokaż certyfikat, zobaczysz, że certyfikat został wystawiony przez localhost dla Apache Friends. Apache Friends (http://www.apachefriends.org) jest twórcą pakietu XAMPP. Komunikat ostrzegawczy, który pojawia się podczas korzystania z niezaufanego certyfikatu, różni się w zależności od przeglądarki. W przeglądarce Internet Explorer 7 wiadomość jest jeszcze bardziej oczywista

Konfiguracja TShirtShop dla SSL

Jeśli zdecydujesz się korzystać z SSL, musisz zainstalować certyfikat SSL, jak pokazano dalej. Podczas korzystania z SSL wskazane jest również wymuszenie dostępu do każdej wrażliwej strony przez SSL; oznacza to, że jeśli ktoś spróbuje uzyskać dostęp do wrażliwej strony (takiej jak strona logowania) za pośrednictwem http://, żądanie zostanie automatycznie przekierowane na adres URL https://. Jeśli jednak chcesz odłożyć obsługę SSL i na razie skupić się na budowaniu stron administracyjnych, możesz. Aby rozwiązanie było konfigurowalne, dodamy stałą o nazwie USE_SSL do pliku include/config.php. Jeśli jego wartość to tak, bezpieczne obszary zostaną wymuszone do załadowania przez HTTPS; w przeciwnym razie będą działać przez HTTP.

Uzyskanie certyfikatu SSL

Uzyskanie certyfikatu to stosunkowo bezbolesne doświadczenie. Omówimy tutaj kroki wymagane do uzyskania certyfikatu od VeriSign, ale proces jest podobny w przypadku innych dostawców. Pełne instrukcje są dostępne na stronie internetowej VeriSign (http://www.verisign.com/). Możesz również uzyskać certyfikaty testowe od VeriSign, z których możesz korzystać bezpłatnie przez okres próbny. Oto podstawowe kroki:

1. Zarejestruj się, aby otrzymać próbny certyfikat na stronie internetowej VeriSign.

2. Wygeneruj żądanie podpisania certyfikatu (CSR) na swoim serwerze sieciowym. Wiąże się to z wypełnieniem różnych danych osobowych, w tym nazwy witryny internetowej i tak dalej. Aby to zadziałało, musisz zainstalować moduł SSL na swoim serwerze sieciowym.

3. Skopiuj zawartość wygenerowanego CSR do systemu zgłoszeń VeriSign.

4. Wkrótce otrzymasz certyfikat od VeriSign, który należy skopiować na serwer sieciowy w celu zainstalowania certyfikatu.

Jest w tym trochę więcej, ale jak wspomniano wcześniej, szczegółowe instrukcje są dostępne na stronie internetowej VeriSign i nie powinieneś napotkać żadnych trudności.

Wymuszanie połączeń SSL

Po zainstalowaniu certyfikatu możesz uzyskać dostęp do dowolnych stron internetowych na swoim serwerze internetowym za pomocą połączenia SSL, po prostu zastępując http:// część adresu URL używanego do uzyskiwania dostępu do strony na https:// (zakładając, że zapora jest skonfigurowany tak, aby zezwalał na połączenie SSL, które domyślnie używa portu 443). Oczywiście nie potrzebujesz połączeń SSL dla wszystkich obszarów witryny. Jeśli dostęp do strony można uzyskać tylko za pośrednictwem protokołu HTTPS, należy pamiętać o dwóch szczegółach:

• Wyszukiwarki nie indeksują lokalizacji HTTPS.

• Dostarczanie stron przez HTTPS zużywa zasoby serwera WWW, który musi szyfrować przesyłane dane.

Masz więc dwa solidne powody, dla których powinieneś wymusić połączenia HTTPS tylko dla wrażliwych obszarów swojej witryny. W tym rozdziale wymusimy SSL dla strony logowania administratora i dla stron administracyjnych Twojej witryny (w późniejszych rozdziałach, gdy sami będziemy obsługiwać płatności, będziemy również chcieli wymusić SSL dla kasy, logowania klienta, rejestracja klienta i inne strony administracyjne). Jeśli chcesz mieć pewność, że wszystkie żądania do skryptu administracyjnego (admin.php) są wykonywane przez HTTPS, wystarczy dodać ten kod na początku prezentacji/store_admin.php:

```
// Class constructor

public function __construct()

{

$this->mSiteUrl = Link::Build('', 'https');

// Enforce page to be accessed through HTTPS if USE_SSL is on

if (USE_SSL == 'yes' && getenv('HTTPS') != 'on')

{

header ('Location: https://' . getenv('SERVER_NAME') .

getenv('REQUEST_URI'));

exit();

}

}
```

Zauważ, że bezpieczne połączenie nie jest wymuszane, jeśli stała USE_SSL zdefiniowana w include/config.php jest ustawiona na no. Ustawienie stałej na no może być przydatne podczas tworzenia strony internetowej, jeśli nie masz dostępu do prawdziwego serwera obsługującego SSL.

Administratorzy uwierzytelniania

Ponieważ chcesz, aby tylko niektórzy użytkownicy mieli dostęp do strony administracyjnej katalogu, musisz zaimplementować mechanizm uwierzytelniania i autoryzacji, który kontroluje dostęp do poufnych stron w witrynie. Użytkownicy, którzy chcą uzyskać dostęp do strony administracyjnej katalogu, powinni najpierw dokonać uwierzytelnienia. Gdy już wiesz, kim jest użytkownik, decydujesz, czy ma on uprawnienia dostępu do strony administracyjnej. Na tym etapie będziemy mieć tylko dwa rodzaje użytkowników: anonimowych użytkowników, którzy regularnie odwiedzają Twoją witrynę,

oraz administratorów, którzy mają dostęp do części administracyjnych witryny (w dalszej części książki umożliwisz odwiedzającym tworzenie kont w Twojej witrynie, ale jeszcze nas tam nie ma). W TShirtShop użyjesz metody uwierzytelniania zwanej uwierzytelnianiem HTTP, która pozwala kontrolować proces logowania za pomocą formularza HTML. Po uwierzytelnieniu klienta zapisujemy plik cookie na kliencie i używamy go do uwierzytelniania wszystkich kolejnych żądań. Jeśli plik cookie nie zostanie znaleziony, klientowi zostanie wyświetlony formularz logowania HTML.

Uwaga: Zakładamy, że administrator uzyskuje dostęp do stron administracyjnych z klienta, który ma włączoną obsługę plików cookie.

Kombinacje nazwy użytkownika i hasła mogą być fizycznie przechowywane na różne sposoby. Na przykład w rozdziale 16 zobaczysz, jak przechowywać zaszyfrowane (zaszyfrowane) hasła klientów w bazie danych.

Wskazówka : haszowanie to powszechna metoda przechowywania haseł. Wartość skrótu hasła jest obliczana przez zastosowanie do niego funkcji matematycznej (algorytmu skrótu). Gdy użytkownik próbuje się uwierzytelnić, hasło jest szyfrowane, a wynikowa wartość skrótu jest porównywana z wartością skrótu oryginalnego (poprawnego) hasła. Jeżeli obie wartości są identyczne, to wprowadzone hasło jest poprawne. Istotną właściwością algorytmu skrótu jest to, że teoretycznie nie można uzyskać oryginalnego hasła z jego wartości skrótu (algorytm jest jednokierunkowy). W praktyce naukowcy odkryli niedawno luki w popularnych algorytmach haszujących MD5, SHA-0 i SHA-1.

Prostszą metodą jest przechowywanie kombinacji nazwy użytkownika i hasła w pliku PHP. Ta metoda nie jest tak elastyczna jak korzystanie z bazy danych, ale jest szybka i łatwa do wdrożenia. Podczas przechowywania danych dotyczących nazwy użytkownika i hasła można wybrać przechowywanie hasła w postaci zwykłego tekstu lub zaszyfrowanego tekstu za pomocą algorytmu mieszającego, takiego jak MD5 lub SHA-1. W poniższym ćwiczeniu po prostu zapiszesz hasło w postaci zwykłego tekstu, ale dobrze jest wiedzieć, że masz również inne opcje.

Ćwiczenie: Implementacja szkieletu strony administratora

1. Zmodyfikuj plik Presentation/templates/first_page_contents.tpl, aby dodać link do strony administracyjnej. Pamiętaj, że dodanie tego linku jest opcjonalne, ponieważ pomaga tylko w łatwiejszym dostępie do strony podczas tworzenia witryny. Jeśli zdecydujesz się nie dodawać linku, pomiń ten i następny krok.

```
{* first_page_contents.tpl *}

{load_presentation_object filename="first_page_contents" assign="obj"}

<p class="description">

We hope you have fun developing TShirtShop, the e-commerce store from

Beginning PHP and MySQL E-Commerce: From Novice to Professional!

</p>

<p class="description">

We have the largest collection of t-shirts with postal stamps on Earth!

Browse our departments and categories to find your favorite!

</p>
```

```
<p>Access the <a href="{$obj->mLinkToAdmin}">admin page</a>.</p>

{include file="products_list.tpl"}
```

2. Utwórz nowy plik o nazwie first_page_contents.php w folderze prezentacji i dodaj do niego następujący kod. Jest to niezbędne do dodania linku na stronie głównej do strony administracyjnej.

```
<?php

class FirstPageContents

{

public $mLinkToAdmin;

public function __construct()

{

$this->mLinkToAdmin = Link::ToAdmin();

}

}

?>
```

3. Utwórz nowy plik o nazwie admin.php w folderze głównym witryny (tshirtshop) i napisz w nim następujący kod. Zauważysz, że admin.php jest dość podobny do index.php, z tą różnicą, że w admin.php nie sprawdzamy linku przychodzącego za pomocą Link::CheckRequest() i ładujemy inny plik szablonu.

```
<?php

// Activate session

session_start();

// Start output buffer

ob_start();

// Include utility files

require_once 'include/config.php';

require_once BUSINESS_DIR . 'error_handler.php';

// Set the error handler

ErrorHandler::SetHandler();

// Load the application page template

require_once PRESENTATION_DIR . 'application.php';

require_once PRESENTATION_DIR . 'link.php';

// Load the database handler
```

```
require_once BUSINESS_DIR . 'database_handler.php';

// Load Business Tier

require_once BUSINESS_DIR . 'catalog.php';

// Load Smarty template file

$application = new Application();

// Display the page

$application->display('store_admin.tpl');

// Close database connection

DatabaseHandler::Close();

// Output content from the buffer

flush();

ob_flush();

ob_end_clean();

?>
```

4. Utwórz plik szablonu Presentation/templates/store_admin.tpl, który jest ładowany z właśnie utworzonego pliku admin.php i dodaj w nim następujący kod:

```
{load_presentation_object filename="store_admin" assign="obj"}

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html>

<head>

<title>Demo Store Admin from Beginning PHP and MySQL E-Commerce</title>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />

<link href="{$obj->mSiteUrl}styles/tshirtshop.css" type="text/css"

rel="stylesheet" />

</head>

<body>

<div id="doc" class="yui-t7">

<div id="bd">

<div class="yui-g">

{include file=$obj->mMenuCell}
```

```
</div>
<div class="yui-g">
{include file=$obj->mContentsCell}
</div>
</div>
</div>
</body>
</html>
```

5. Stwórz prezentację/store_admin.php i dodaj w niej następujący kod:

```
<?php
class StoreAdmin
{
public $mSiteUrl;
// Define the template file for the page menu
public $mMenuCell = 'blank.tpl';
// Define the template file for the page contents
public $mContentsCell = 'blank.tpl';
// Class constructor
public function __construct()
{
$this->mSiteUrl = Link::Build('', 'https');
// Enforce page to be accessed through HTTPS if USE_SSL is on
if (USE_SSL == 'yes' && getenv('HTTPS') != 'on')
{
header ('Location: https://' . getenv('SERVER_NAME') .
getenv('REQUEST_URI'));
exit();
}
}
public function init()
{
```

```
// If admin is not logged in, load the admin_login template
if (!(isset ($_SESSION['admin_logged'])) ||
$_SESSION['admin_logged'] != true)
$this->mContentsCell = 'admin_login.tpl';
else
{
// If admin is logged in, load the admin menu page
$this->mMenuCell = 'admin_menu.tpl';
// If logging out ...
if (isset ($_GET['Page']) && ($_GET['Page'] == 'Logout'))
{
unset($_SESSION['admin_logged']);
header('Location: ' . Link::ToAdmin());
exit();
}
}
}
}
?>
```

6. Dodaj stałą USE_SSL i dane logowania administratora na końcu include/config.php. Jeśli wolisz na razie nie używać SSL, po prostu ustaw stałą USE_SSL na no. Jak widać, konto administratora nosi domyślną nazwę tshirtshopadmin, a jego hasło to również tshirtshopadmin. W scenariuszu produkcyjnym będziesz chciał zmienić te wartości na coś mniej oczywistego dla potencjalnego hakera.

```
// We enable and enforce SSL when this is set to anything else than 'no'
define('USE_SSL', 'yes');
// Administrator login information
define('ADMIN_USERNAME', 'tshirtshopadmin');
define('ADMIN_PASSWORD', 'tshirtshopadmin');
```

Uwaga: Jak wspomniano wcześniej, później dowiesz się o haszowaniu i pracy z haszowanymi hasłami przechowywanymi w bazie danych. Jeśli chcesz teraz używać hashowania, musisz zapisać wartość hash hasła w pliku konfiguracyjnym zamiast przechowywać hasło w postaci zwykłego tekstu (w tym przypadku tshirtshopadmin). W czasie logowania porównujesz wartość hash ciągu wpisanego przez

użytkownika z wartością hash zapisaną w config.php. Wartość skrótu ciągu można obliczyć, stosując do niego funkcję sha1 (funkcja sha1 oblicza wartość skrótu za pomocą algorytmu SHA1).

7. Teraz utworzymy skomponowany szablon admin_login, który wyświetla okno logowania. Zacznij od utworzenia pliku Presentation/templates/admin_login.tpl, a następnie dodaj do niego następujący kod:

```
{* admin_login.tpl *}

{load_presentation_object filename="admin_login" assign="obj"}

<div class="login">

<p class="login-title">TShirtShop Login</p>

<form method="post" action="{$obj->mLinkToAdmin}">

<p>

Enter login information or go back to

<a href="{$obj->mLinkToIndex}">storefront</a>.

</p>

{if $obj->mLoginMessage neq ""}

<p class="error">{$obj->mLoginMessage}</p>

{/if}

<p>

<label for="username">Username:</label>

<input type="text" name="username" size="35" value="{$obj->mUsername}" />

</p>

<p>

<label for="password">Password:</label>

<input type="password" name="password" size="35" value="" />

</p>

<p>

<input type="submit" name="submit" value="Login" />

</p>

</form>

</div>
```

8. Utwórz nowy plik obiektu prezentacji o nazwie admin_login.php w folderze prezentacji i wpisz następujący kod:

```
<?php
// Class that deals with authenticating administrators
class AdminLogin
{
// Public variables available in smarty templates
public $mUsername;
public $mLoginMessage = '';
public $mLinkToAdmin;
public $mLinkToIndex;
// Class constructor
public function __construct()
{
// Verify if the correct username and password have been supplied
if (isset ($_POST['submit']))
{
if ($_POST['username'] == ADMIN_USERNAME
&& $_POST['password'] == ADMIN_PASSWORD)
{
$_SESSION['admin_logged'] = true;
header('Location: ' . Link::ToAdmin());
exit();
}
else
$this->mLoginMessage = 'Login failed. Please try again:';
}
$this->mLinkToAdmin = Link::ToAdmin();
$this->mLinkToIndex = Link::ToIndex();
}
}
?>
```

9. Utwórz prezentację/templates/admin_menu.tpl i dodaj następujący kod:

```
{* admin_menu.tpl *}
{load_presentation_object filename="admin_menu" assign="obj"}
<h1>TShirtShop Admin</h1>
<p> |
<a href="{$obj->mLinkToStoreAdmin}">CATALOG ADMIN</a> |
<a href="{$obj->mLinkToStoreFront}">STOREFRONT</a> |
<a href="{$obj->mLinkToLogout}">LOGOUT</a> |
</p>
```

10. Teraz utwórz nowy plik o nazwie admin_menu.php w folderze prezentacji i dodaj następujący kod:

```
<?php
class AdminMenu
{
public $mLinkToStoreAdmin;
public $mLinkToStoreFront;
public $mLinkToLogout;
public function __construct()
{
$this->mLinkToStoreAdmin = Link::ToAdmin();
$this->mLinkToStoreFront = Link::ToIndex();
$this->mLinkToLogout = Link::ToLogout();
}
}
?>
```

11. Otwórz plik Presentation/link.php i zmodyfikuj metodę Build() klasy Link, jak pokazano w poniższym fragmencie kodu. Dodaje obsługę bezpiecznych łączy (HTTPS):

```
public static function Build($link, $type = 'http')
{
$base = (($type == 'http' || USE_SSL == 'no') ? 'http://' : 'https://') .
getenv('SERVER_NAME');
// If HTTP_SERVER_PORT is defined and different than default
if (defined('HTTP_SERVER_PORT') && HTTP_SERVER_PORT != '80' &&
```

```
strpos($base, 'https') === false)
{
// Append server port
$base .= ':' . HTTP_SERVER_PORT;
}
$link = $base . VIRTUAL_LOCATION . $link;
// Escape html
return htmlspecialchars($link, ENT_QUOTES);
}
```

12. Również w Presentation/link.php dodaj następujące metody na końcu klasy Link:

```
// Create link to admin page
public static function ToAdmin($params = '')
{
$link = 'admin.php';
if ($params != '')
$link .= '?' . $params;
return self::Build($link, 'https');
}
// Create logout link
public static function ToLogout()
{
return self::ToAdmin('Page=Logout');
}
```

13. Dodaj następujące style do pliku styles/tshirtshop.css:

```
.login {
color: #333333;
display: block;
border: 1px solid #c6e1ec;
margin: 50px auto;
width: 325px;
}
```

```
.login form p {
padding: 0 10px;
}
label {
display: block;
}
.error {
color: #ff0000;
font-size: 85%;
font-weight: bold;
}
.login-title {
background: #0590c7;
border-bottom: 2px solid #c6e1ec;
color: #ffffff;
display: block;
font-size: 93%;
font-weight: bold;
margin: 1px;
padding: 2px 10px;
}
.no-items-found {
color: #ff0000;
}
table.tss-table {
width: 100%;
}
table.tss-table th {
background: #0590c7;;
border: none;
border-bottom: 3px solid #c6e1ec;
```

color: #ffffff;

text-align: left;

}

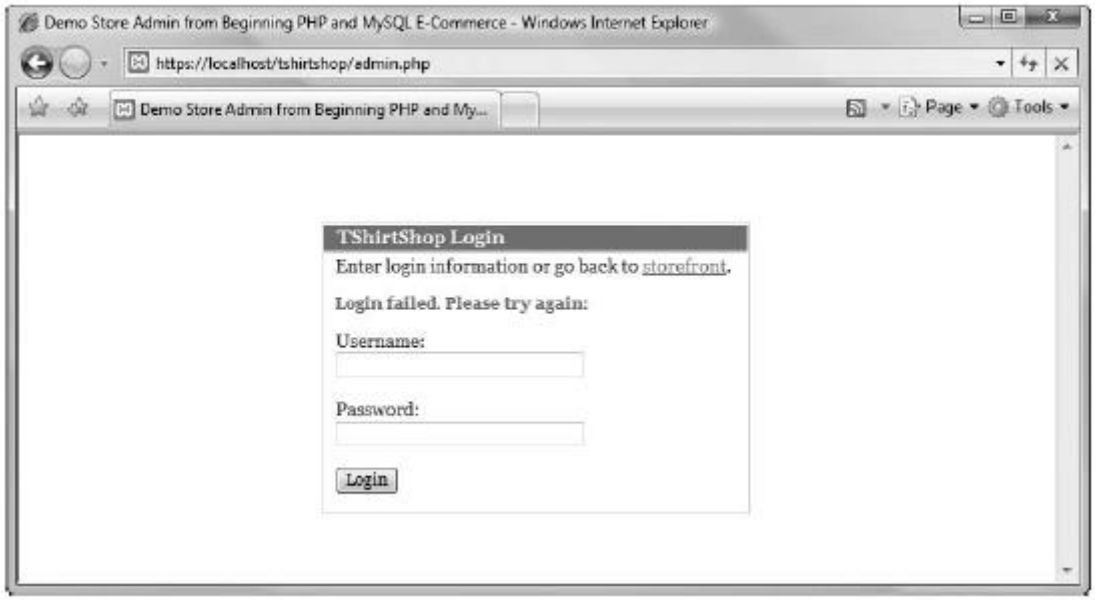
table.tss-table td {

border: none;

border-bottom: 1px solid #0590c7;

}

14. Załaduj TShirtShop w swojej ulubionej przeglądarce, a zobaczysz link do strony administratora w wiadomości powitalnej na stronie głównej. Kliknij go, a zostanie wyświetlony formularz logowania HTML; Rysunek przedstawia komunikat, który otrzymasz, jeśli wpiszesz nieprawidłowe hasło. Po podaniu poprawnych danych logowania (nazwa użytkownika: tshirtshopadmin, hasło: tshirtshopadmin) zostaniesz przekierowany na stronę administracyjną katalogu, która obecnie zawiera tylko menu główne - wkrótce to zmienimy.



Jak to działa: uwierzytelnianie administratorów

Główne komponenty utworzone w tym ćwiczeniu to:

• admin.php to punkt wejścia do części administracyjnej serwisu. Będziesz go dalej rozwijać w dalszej części tego rozdziału. Studiując ten plik, możesz zobaczyć, że nie ma on żadnych efektów wizualnych. Po prostu przygotowuje środowisko i ładuje skomponowany szablon store_admin Smarty.

• store_admin to skomponowany szablon (złożony z store_admin.php i store_admin.tpl), który jest ładowany przez admin.php w celu wygenerowania szkieletu strony administracyjnej katalogu. Podczas wdrażania różnych funkcji administracyjnych zostaną podłączone różne szablony. Na przykład w dalszej części tego rozdziału dodasz funkcje administracyjne działów i kategorii.

• admin_login to złożony szablon, który implementuje funkcję administracyjnego uwierzytelniania i autoryzacji. Szablon wyświetla formularz logowania i w razie potrzeby uwierzytelnia odwiedzającego.

Ze wszystkich plików, które utworzyłeś, store_admin.php jest prawdopodobnie najbardziej interesujący. Najpierw sprawdza, czy odwiedzający został uwierzytelniony jako administrator (sprawdzając, czy zmienna sesji admin_logged ma wartość true). Jeśli odwiedzający nie jest zalogowany jako administrator, ładowany jest skomponowany szablon admin_login:

```
public function init()

{

// If admin is not logged in, load admin_login template

if (!(isset ($_SESSION['admin_logged'])) ||

$_SESSION['admin_logged'] != true)

$this->mContentsCell = 'admin_login.tpl';
```

Mechanizm logowania w obiekcie prezentacji AdminLogin przechowuje aktualny stan uwierzytelniania w sesji odwiedzającego pod zmienną o nazwie admin_logged. W funkcji __construct testujemy, czy podana nazwa użytkownika i hasło odpowiadają wartościom przechowywanym w config.php jako ADMIN_USERNAME i ADMIN_PASSWORD; jeśli się zgadzają, ustawiamy wartość dmin_logged na true i przekierowujemy do admin.php:

```
// Verify if the correct username and password have been supplied

if (isset ($_POST['submit']))

{

if ($_POST['username'] == ADMIN_USERNAME

&& $_POST['password'] == ADMIN_PASSWORD)

{

$_SESSION['admin_logged'] = true;

header('Location: ' . Link::ToAdmin());

exit();

}

else

$this->mLoginMessage = 'Login failed. Please try again:';

}
```

Łącze wylogowania w admin_menu.tpl po prostu usuwa ustawienie zmiennej sesji admin_logged w store_admin.php i przekierowuje administratora do admin.php. W ten sposób przy kolejnej próbie wejścia na stronę administracyjną administrator zostanie przekierowany na stronę logowania.

```
// If logging out ...

if (isset ($_GET['Page']) && ($_GET['Page'] == 'Logout'))

{
```

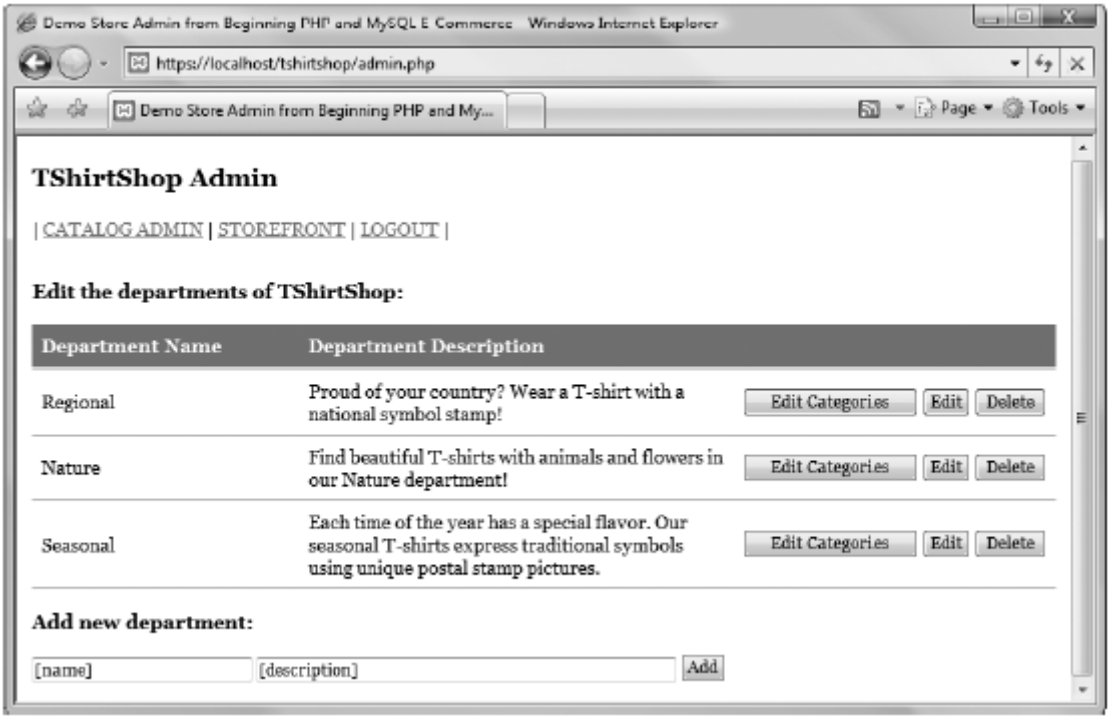
```
unset($_SESSION['admin_logged']);

header('Location: ' . Link::ToAdmin());

exit();

}
```

Działy administracyjne

Sekcja administracji działem pozwala użytkownikowi dodawać, usuwać i zmieniać informacje o działach. Aby zaimplementować tę funkcjonalność, musisz napisać niezbędny kod dla warstwy prezentacji, biznesowej i danych. Jedną z podstawowych prawd dotyczących aplikacji wielowarstwowych (która dotyczy tego konkretnego przypadku) jest to, że warstwy biznesowe i warstwy danych są ostatecznie tworzone w celu obsługi warstwy prezentacji. Rysowanie na papierze i ustalanie dokładnego wyglądu witryny jest dobrym wskaźnikiem tego, co będzie zawierać baza danych i warstwa biznesowa (innymi słowy, jaką funkcjonalność musi obsługiwać interfejs użytkownika lub interfejs użytkownika). Po przygotowaniu specyfikacji funkcjonalnych i technicznych będziesz dokładnie wiedział, co umieścić na każdym poziomie, więc kolejność, w jakiej piszesz kod, nie ma znaczenia. Jednak poza największymi projektami, które naprawdę wymagają bardzo starannego projektowania i planowania, w praktyce taka elastyczność rzadko się zdarza. W tej książce zwykle rozpoczyna się implementację funkcji od niższych poziomów (bazy danych i obiekt danych). Jednak w tym rozdziale zawsze zaczynamy programowanie od warstwy prezentacji, co możesz teraz zrobić, ponieważ masz już dobry przegląd architektury i możesz przewidzieć szczegóły implementacji warstw biznesowych i danych. Ta wiedza jest niezbędna, ponieważ podczas implementacji w warstwie prezentacji będziesz wywoływać metody z obu pozostałych warstw, z których żadna nie została utworzona. Jeśli nie masz jasnego pomysłu na wdrożenie wszystkich trzech warstw, rozpoczęcie od warstwy prezentacji byłoby złym pomysłem.

Wdrażanie poziomu prezentacji

Przyjrzyj się jeszcze raz, jak wygląda skomponentowany szablon admin_departments w działaniu.



Ten złożony szablon generuje listę wypełnioną informacjami o każdym dziale, dwoma polami tekstowymi i przyciskiem służącym do dodawania nowego działu do listy. Po kliknięciu przycisku Edytuj w dziale, nazwa i opis tego działu stają się dostępne do edycji, a przyciski Aktualizuj i Anuluj pojawiają się zamiast przycisku Edytuj, jak widzieliśmy wcześniej.

Ćwiczenie: Implementacja szablonu złożonego admin_departments

1. Utwórz nowy plik szablonu o nazwie admin_departments.tpl w folderze prezentacji/templates i dodaj do niego następujący kod:

```
{* admin_departments.tpl *}

{load_presentation_object filename="admin_departments" assign="obj"}

<form method="post"

action="{$obj->mLinkToDepartmentsAdmin}">

<h3>Edit the departments of TShirtShop:</h3>

{if $obj->mErrorMessage}<p class="error">{$obj->mErrorMessage}</p>{/if}

{if $obj->mDepartmentsCount eq 0}

<p class="no-items-found">There are no departments in your database!</p>

{else}

<table class="tss-table">

<tr>

<th width="200">Department Name</th>

<th>Department Description</th>
```

```
<th width="240"> </th>
</tr>
{section name=i loop=$obj->mDepartments}
{if $obj->mEditItem == $obj->mDepartments[i].department_id}
<tr>
<td>
<input type="text" name="name"
value="{$obj->mDepartments[i].name}" size="30" />
</td>
<td>
{strip}
<textarea name="description" rows="3" cols="60">
{$obj->mDepartments[i].description}
</textarea>
{/strip}
</td>
<td>
<input type="submit"
name="submit_edit_cat_{$obj->mDepartments[i].department_id}"
value="Edit Categories" />
<input type="submit"
name="submit_update_dept_{$obj->mDepartments[i].department_id}"
value="Update" />
<input type="submit" name="cancel" value="Cancel" />
<input type="submit"
name="submit_delete_dept_{$obj->mDepartments[i].department_id}"
value="Delete" />
</td>
</tr>
{else}
<tr>
```

```
<td>{$obj->mDepartments[i].name}</td>
<td>{$obj->mDepartments[i].description}</td>
<td>
<input type="submit"
name="submit_edit_cat_{$obj->mDepartments[i].department_id}"
value="Edit Categories" />
<input type="submit"
name="submit_edit_dept_{$obj->mDepartments[i].department_id}"
value="Edit" />
<input type="submit"
name="submit_delete_dept_{$obj->mDepartments[i].department_id}"
value="Delete" />
</td>
</tr>
{/if}
{/section}
</table>
{/if}
<h3>Add new department:</h3>
<p>
<input type="text" name="department_name" value="[name]" size="30" />
<input type="text" name="department_description" value="[description]"
size="60" />
<input type="submit" name="submit_add_dept_0" value="Add" />
</p>
</form>
```

2. Utwórz nowy plik obiektu prezentacji o nazwie admin_departments.php w folderze prezentacji i dodaj do niego następujący kod:

```
<?php
// Class that supports departments admin functionality
class AdminDepartments
```

```
{
// Public variables available in smarty template
public $mDepartmentsCount;
public $mDepartments;
public $mErrorMessage;
public $mEditItem;
public $mLinkToDepartmentsAdmin;
// Private members
private $_mAction;
private $_mActionedDepartmentId;
// Class constructor
public function __construct()
{
// Parse the list with posted variables
foreach ($_POST as $key => $value)
// If a submit button was clicked ...
if (substr($key, 0, 6) == 'submit')
{
/* Get the position of the last '_' underscore from submit
button name e.g strtpos('submit_edit_dept_1', '_') is 17 */
$last_underscore = strrpos($key, '_');
/* Get the scope of submit button
(e.g 'edit_dep' from 'submit_edit_dept_1') */
$this->_mAction = substr($key, strlen('submit_'),
$last_underscore - strlen('submit_'));
/* Get the department id targeted by submit button
(the number at the end of submit button name)
e.g '1' from 'submit_edit_dept_1' */
$this->_mActionedDepartmentId = substr($key, $last_underscore + 1);
break;
}
```

```
$this->mLinkToDepartmentsAdmin = Link::ToDepartmentsAdmin();
}
public function init()
{
// If adding a new department ...
if ($this->_mAction == 'add_dept')
{
$department_name = $_POST['department_name'];
$department_description = $_POST['department_description'];
if ($department_name == null)
$this->mErrorMessage = 'Department name required';
if ($this->mErrorMessage == null)
{
Catalog::AddDepartment($department_name, $department_description);
header('Location: ' . $this->mLinkToDepartmentsAdmin);
}
}
// If editing an existing department ...
if ($this->_mAction == 'edit_dept')
$this->mEditItem = $this->_mActionedDepartmentId;
// If updating a department ...
if ($this->_mAction == 'update_dept')
{
$department_name = $_POST['name'];
$department_description = $_POST['description'];
if ($department_name == null)
$this->mErrorMessage = 'Department name required';
if ($this->mErrorMessage == null)
{
Catalog::UpdateDepartment($this->_mActionedDepartmentId,
$department_name, $department_description);
```

```
header('Location: ' . $this->mLinkToDepartmentsAdmin);
}
}
// If deleting a department ...
if ($this->_mAction == 'delete_dept')
{
$status = Catalog::DeleteDepartment($this->_mActionedDepartmentId);
if ($status < 0)
$this->mErrorMessage = 'Department not empty';
else
header('Location: ' . $this->mLinkToDepartmentsAdmin);
}
// If editing department's categories ...
if ($this->_mAction == 'edit_cat')
{
header('Location: ' .
htmlspecialchars_decode(
Link::ToDepartmentCategoriesAdmin(
$this->_mActionedDepartmentId)));
exit();
}
// Load the list of departments
$this->mDepartments = Catalog::GetDepartmentsWithDescriptions();
$this->mDepartmentsCount = count($this->mDepartments);
}
}
?>
```

3. Otwórz Presentation/link.php i dodaj następującą metodę na końcu klasy Link:

```
// Create link to the departments administration page
public static function ToDepartmentsAdmin()
{
```

```
return self::ToAdmin('Page=Departments');

}
```

4. Zmodyfikuj metodę init() klasy StoreAdmin znajdującą się w pliku Presentation/store_admin.php, aby załadować nowo utworzony skomponowany szablon admin_departments:

```
// If logging out ...

if (isset ($_GET['Page']) && ($_GET['Page'] == 'Logout'))

{

unset($_SESSION['admin_logged']);

header('Location: ' . Link::ToAdmin());

exit();

}

// If Page is not explicitly set, assume the Departments page

$admin_page = isset ($_GET['Page']) ? $_GET['Page'] : 'Departments';

// Choose what admin page to load ...

if ($admin_page == 'Departments')

$this->mContentsCell = 'admin_departments.tpl';

}

}

}

?>
```

Jak to działa: skomponowany szablon admin_departments

W tym ćwiczeniu napisałeś dużo kodu i nadal nie możesz niczego testować! Jest to jeden z najtrudniejszych etapów tworzenia pierwszego interfejsu użytkownika. Zobaczmy, jak działa szablon admin_departments.tpl. Oto schemat konstrukcji {sekcja} użytej do zbudowania wierszy tabeli działów:

```
{section name=i loop=$obj->mDepartments}

{if $obj->mEditItem == $obj->mDepartments[i].department_id}

<!--

Here goes a form where the administrator can edit the department name

and description with Update/Cancel, Edit Categories, and Delete buttons.

//-->

{else}
```

<!--

Here goes a form that displays the department name and description, and

also Edit, Edit Categories, and Delete buttons.

//-->

{/if}

{/section}

Domyślnie nazwa i opis działu nie są edytowalne, ale po kliknięciu przycisku Edytuj jednego z działów, $obj->mEditItem jest ustawiana na wartość id_działu kliknięty działu, a logika prezentacji Smarty generuje edytowalne pola tekstowe zamiast etykiety. Umożliwi to administratorowi edycję danych wybranego działu (w trybie edycji przyciski Aktualizuj/Anuluj pojawiają się zamiast przycisku Edytuj, jak widzieliśmy we wcześniejszych danych liczbowych). Obiekt prezentacji admin_departments - AdminDepartments - zawiera logikę, dzięki której strona administracyjna działów działa. Wczytany w wyniku kliknięcia przez użytkownika jednego z przycisków (np. Edytuj lub Usuń), sprawdza, który przycisk został kliknięty i odpowiednio reaguje. Przyciski są zgodne ze specjalną konwencją nazewnictwa, dzięki czemu można je zidentyfikować na podstawie kodu. Wszystkie nazwy przycisków zaczynają się od „prześlij", a kończą się identyfikatorem działu. W środku nazwy znajduje się kod typu przycisku, który określa jaką operację należy wykonać we wspomnianym dziale. Ten kod może być jednym z następujących:

• add_dept dla przycisków Dodaj

• edit_dept dla przycisków edycji

• update_dept dla przycisków aktualizacji

• delete_dept dla przycisków Usuń

• edit_cat dla przycisków Edytuj kategorie

Obiekt rozpoznaje, który przycisk został kliknięty i wie, co zrobić po przeanalizowaniu listy przesłanych zmiennych i odczytaniu nazwy kliknięty przycisku. Przycisk o nazwie submit_edit_dept_1 nakazuje obiektowi prezentacji przejście w tryb edycji dla działu z wartością id_działu wynoszącą 1. Należy zauważyć, że w przypadku przycisku Dodaj dział identyfikator działu określony w nazwie przycisku staje się nieistotny, ponieważ jego wartość jest automatycznie generowana przez bazę danych (department_id to kolumna AUTO_INCREMENT). W zależności od typu kliknięty przycisku wywoływana jest jedna z odpowiednich metod warstwy biznesowej. Rozważmy teraz te metody.

Wdrażanie poziomu biznesowego

Wywołałeś cztery metody warstwy środkowej z klasy AdminDepartments. Teraz nadszedł czas, aby dodać ich odpowiedniki na poziomie biznesowym:

• GetDepartmentsWithDescriptions zwraca listę działów do wyświetlenia na stronie administracji działu.

• AddDepartment dodaje nowy dział używając podanej nazwy i opisu. Jego identyfikator jest automatycznie generowany przez bazę danych (kolumna id_działu w tabeli działów jest kolumną AUTO_INCREMENT).

• UpdateDepartment zmienia dane działu. Jego parametrami są wartość identyfikatora działu, jego nowa nazwa i nowy opis.

• DeleteDepartment usuwa dział określony przez parametr id_działu.

Ćwiczenie: Wdrażanie warstwy biznesowej

Teraz nadszedł czas na wdrożenie tych czterech nowych metod. Dodaj ten kod do klasy Catalog w business/catalog.php:

```
// Retrieves all departments with their descriptions
public static function GetDepartmentsWithDescriptions()
{
// Build the SQL query
$sql = 'CALL catalog_get_departments()';
// Execute the query and return the results
return DatabaseHandler::GetAll($sql);
}
// Add a department
public static function AddDepartment($departmentName, $departmentDescription)
{
// Build the SQL query
$sql = 'CALL catalog_add_department(:department_name,
:department_description)';
// Build the parameters array
$params = array (':department_name' => $departmentName,
':department_description' => $departmentDescription);
// Execute the query
DatabaseHandler::Execute($sql, $params);
}
// Updates department details
public static function UpdateDepartment($departmentId, $departmentName,
$departmentDescription)
{
// Build the SQL query
$sql = 'CALL catalog_update_department(:department_id, :department_name,
```

```
:department_description)';

// Build the parameters array

$params = array (':department_id' => $departmentId,

':department_name' => $departmentName,

':department_description' => $departmentDescription);

// Execute the query

DatabaseHandler::Execute($sql, $params);

}

// Deletes a department

public static function DeleteDepartment($departmentId)

{

// Build the SQL query

$sql = 'CALL catalog_delete_department(:department_id)';

// Build the parameters array

$params = array (':department_id' => $departmentId);

// Execute the query and return the results

return DatabaseHandler::GetOne($sql, $params);

}
```

Implementacja warstwy danych

Do warstwy danych zostaną dodane cztery procedury składowane, które odpowiadają czterem metodom warstwy biznesowej, które napisałeś wcześniej. Zobaczmy, o co w tym wszystkim chodzi.

Ćwiczenie: Dodawanie procedur składowanych w warstwie danych do bazy danych

1. Użyj phpMyAdmin, aby wykonać i utworzyć procedury składowane opisane w poniższych krokach. Nie zapomnij też ustawić $$ jako separatora przed wykonaniem kodu.

2. Wykonaj ten kod, który utworzy procedurę składowaną catalog_get_departments w Twojej bazie danych tshirtshop. Catalog_get_departments to najprostsza procedura składowana, jaką tutaj napiszesz. Zwraca listę działów wraz z ich identyfikatorami, nazwami i opisami. Jest podobny do katalogu_get_departments_list procedura składowana wywoływana w celu wypełnienia listy działów z witryny sklepowej, ale ta zwraca również opisy.

```
-- Create catalog_get_departments stored procedure

CREATE PROCEDURE catalog_get_departments()

BEGIN

SELECT department_id, name, description
```

```
FROM department

ORDER BY department_id;

END$$
```

3. Execute the following code, which creates the catalog_add_department stored procedure in your tshirtshop database. This procedure inserts a new department into the database.

```
-- Create catalog_add_department stored procedure

CREATE PROCEDURE catalog_add_department(

IN inName VARCHAR(100), IN inDescription VARCHAR(1000))

BEGIN

INSERT INTO department (name, description)

VALUES (inName, inDescription);

END$
```

4. Wykonaj ten kod, który utworzy procedurę składowaną catalog_update_department w Twojej bazie danych tshirtshop. Ta procedura składowana aktualizuje nazwę i opis istniejącego działu za pomocą instrukcji UPDATE SQL.

```
-- Create catalog_update_department stored procedure

CREATE PROCEDURE catalog_update_department(IN inDepartmentId INT,

IN inName VARCHAR(100), IN inDescription VARCHAR(1000))

BEGIN

UPDATE department

SET name = inName, description = inDescription

WHERE department_id = inDepartmentId;

END$$
```

5. Wykonaj następujący kod, który utworzy procedurę składowaną catalog_delete_department w bazie danych tshirtshop. Ta procedura usuwa istniejący dział z bazy danych, ale tylko wtedy, gdy nie są z nim powiązane żadne kategorie.

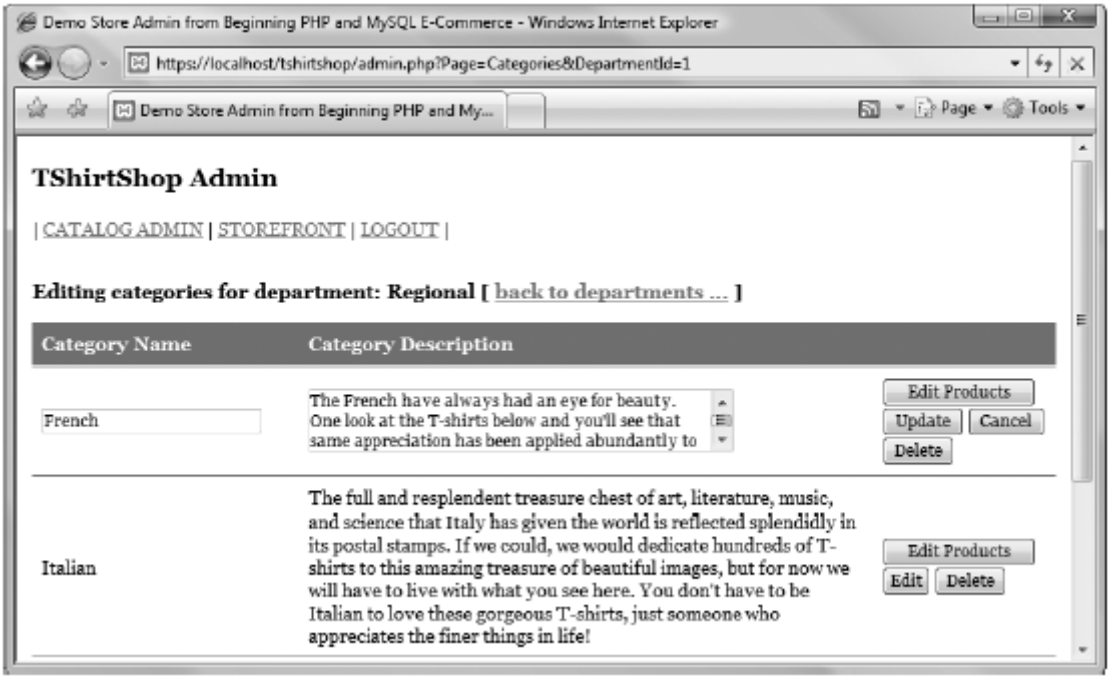
```
-- Create catalog_delete_department stored procedure

CREATE PROCEDURE catalog_delete_department(IN inDepartmentId INT)

BEGIN

DECLARE categoryRowsCount INT;

SELECT count(*)

FROM category
```

```
WHERE department_id = inDepartmentId

INTO categoryRowsCount;

IF categoryRowsCount = 0 THEN

DELETE FROM department WHERE department_id = inDepartmentId;

SELECT 1;

ELSE

SELECT -1;

END IF;

END$$
```

6. Na koniec załaduj stronę admin.php w przeglądarce i podziwiaj swoje wyniki. Powinieneś być w stanie dodawać, usuwać i edytować działy, jak pokazano na rysunkach 10-2, 10-3 i 10-9.

Administrowanie kategoriami

Ponieważ strona administracyjna kategorii opiera się na tych samych krokach i koncepcjach, co strona administracyjna działów, szybko wymienimy kroki, które musisz wykonać. Po wykonaniu poniższego ćwiczenia będziesz mógł wprowadzać zmiany i dodatki do kategorii produktów, klikając przycisk Edytuj kategorie na stronie działów. Kliknięcie przycisku Edytuj na stronie kategorii powoduje przejście do kategorii w trybie edycji.



Ćwiczenie: Administrowanie kategoriami

1. Utwórz nowy plik szablonu o nazwie admin_categories.tpl w pliku folderu prezentacji/templates i dodaj do niego następujący kod:

```
{* admin_categories.tpl *}
```

```
{load_presentation_object filename="admin_categories" assign="obj"}
<form method="post"
action="{$obj->mLinkToDepartmentCategoriesAdmin}">
<h3>
Editing categories for department: {$obj->mDepartmentName} [
<a href="{$obj->mLinkToDepartmentsAdmin}">back to departments ...</a> ]
</h3>
{if $obj->mErrorMessage}<p class="error">{$obj->mErrorMessage}</p>{/if}
{if $obj->mCategoriesCount eq 0}
<p class="no-items-found">There are no categories in this department!</p>
{else}
<table class="tss-table">
<tr>
<th width="200">Category Name</th>
<th>Category Description</th>
<th width="240"> </th>
</tr>
{section name=i loop=$obj->mCategories}
{if $obj->mEditItem == $obj->mCategories[i].category_id}
<tr>
<td>
<input type="text" name="name"
value="{$obj->mCategories[i].name}" size="30" />
</td>
<td>
{strip}
<textarea name="description" rows="3" cols="60">
{$obj->mCategories[i].description}
</textarea>
{/strip}
</td>
```

```
<td>
<input type="submit"
name="submit_edit_prod_{$obj->mCategories[i].category_id}"
value="Edit Products" />
<input type="submit"
name="submit_update_cat_{$obj->mCategories[i].category_id}"
value="Update" />
<input type="submit" name="cancel" value="Cancel" />
<input type="submit"
name="submit_delete_cat_{$obj->mCategories[i].category_id}"
value="Delete" />
</td>
</tr>
{else}
<tr>
<td>{$obj->mCategories[i].name}</td>
<td>{$obj->mCategories[i].description}</td>
<td>
<input type="submit"
name="submit_edit_prod_{$obj->mCategories[i].category_id}"
value="Edit Products" />
<input type="submit"
name="submit_edit_cat_{$obj->mCategories[i].category_id}"
value="Edit" />
<input type="submit"
name="submit_delete_cat_{$obj->mCategories[i].category_id}"
value="Delete" />
</td>
</tr>
{/if}
{/section}
```

```
</table>
{/if}
<h3>Add new category:</h3>
<input type="text" name="category_name" value="[name]" size="30" />
<input type="text" name="category_description" value="[description]"
size="60" />
<input type="submit" name="submit_add_cat_0" value="Add" />
</form>
```

2. Utwórz nowy plik obiektu prezentacji o nazwie admin_categories.php w folderze prezentacji i dodaj do niego:

```
?php
// Class that deals with categories admin
class AdminCategories
{
// Public variables available in smarty template
public $mCategoriesCount;
public $mCategories;
public $mErrorMessage;
public $mEditItem;
public $mDepartmentId;
public $mDepartmentName;
public $mLinkToDepartmentsAdmin;
public $mLinkToDepartmentCategoriesAdmin;
// Private members
private $_mAction;
private $_mActionedCategoryId;
// Class constructor
public function __construct()
{
if (isset ($_GET['DepartmentId']))
$this->mDepartmentId = (int)$_GET['DepartmentId'];
```

```
else
trigger_error('DepartmentId not set');
$department_details = Catalog::GetDepartmentDetails($this->mDepartmentId);
$this->mDepartmentName = $department_details['name'];
foreach ($_POST as $key => $value)
// If a submit button was clicked ...
if (substr($key, 0, 6) == 'submit')
{
/* Get the position of the last '_' underscore from submit
button name e.g strtpos('submit_edit_cat_1', '_') is 16 */
$last_underscore = strrpos($key, '_');
/* Get the scope of submit button
(e.g 'edit_cat' from 'submit_edit_cat_1') */
$this->_mAction = substr($key, strlen('submit_'),
$last_underscore - strlen('submit_'));
/* Get the category id targeted by submit button
(the number at the end of submit button name)
e.g '1' from 'submit_edit_cat_1' */
$this->_mActionedCategoryId = (int)substr($key, $last_underscore + 1);
break;
}
$this->mLinkToDepartmentsAdmin = Link::ToDepartmentsAdmin();
$this->mLinkToDepartmentCategoriesAdmin =
Link::ToDepartmentCategoriesAdmin($this->mDepartmentId);
}
public function init()
{
// If adding a new category ...
if ($this->_mAction == 'add_cat')
{
$category_name = $_POST['category_name'];
```

```
$category_description = $_POST['category_description'];
if ($category_name == null)
$this->mErrorMessage = 'Category name is empty';
if ($this->mErrorMessage == null)
{
Catalog::AddCategory($this->mDepartmentId, $category_name,
$category_description);
header('Location: ' .
htmlspecialchars_decode(
$this->mLinkToDepartmentCategoriesAdmin));
}
}
// If editing an existing category ...
if ($this->_mAction == 'edit_cat')
{
$this->mEditItem = $this->_mActionedCategoryId;
}
// If updating a category ...
if ($this->_mAction == 'update_cat')
{
$category_name = $_POST['name'];
$category_description = $_POST['description'];
if ($category_name == null)
$this->mErrorMessage = 'Category name is empty';
if ($this->mErrorMessage == null)
{
Catalog::UpdateCategory($this->_mActionedCategoryId, $category_name,
$category_description);
header('Location: ' .
htmlspecialchars_decode(
$this->mLinkToDepartmentCategoriesAdmin));
```

```
}
}
// If deleting a category ...
if ($this->_mAction == 'delete_cat')
{
$status = Catalog::DeleteCategory($this->_mActionedCategoryId);
if ($status < 0)
$this->mErrorMessage = 'Category not empty';
else
header('Location: ' .
htmlspecialchars_decode(
$this->mLinkToDepartmentCategoriesAdmin));
}
// If editing category's products ...
if ($this->_mAction == 'edit_prod')
{
header('Location: ' .
htmlspecialchars_decode(
Link::ToCategoryProductsAdmin($this->mDepartmentId,
$this->_mActionedCategoryId)));
exit();
}
// Load the list of categories
$this->mCategories =
Catalog::GetDepartmentCategories($this->mDepartmentId);
$this->mCategoriesCount = count($this->mCategories);
}
}
?>
```

3. Teraz otwórz Presentation/link.php i dodaj następującą metodę na końcu klasy Link:

```
// Create link to the categories administration page
```

```
public static function ToDepartmentCategoriesAdmin($departmentId)
{
$link = 'Page=Categories&DepartmentId=' . $departmentId;
return self::ToAdmin($link);
}
```

4. Otwórz business/catalog.php, aby dodać następujące metody warstwy biznesowej do klasy Catalog:

```
// Gets categories in a department
public static function GetDepartmentCategories($departmentId)
{
// Build the SQL query
$sql = 'CALL catalog_get_department_categories(:department_id)';
// Build the parameters array
$params = array (':department_id' => $departmentId);
// Execute the query and return the results
return DatabaseHandler::GetAll($sql, $params);
}
// Adds a category
public static function AddCategory($departmentId, $categoryName,
$categoryDescription)
{
// Build the SQL query
$sql = 'CALL catalog_add_category(:department_id, :category_name,
:category_description)';
// Build the parameters array
$params = array (':department_id' => $departmentId,
':category_name' => $categoryName,
':category_description' => $categoryDescription);
// Execute the query
DatabaseHandler::Execute($sql, $params);
}
// Updates a category
```

```
public static function UpdateCategory($categoryId, $categoryName,
$categoryDescription)
{
// Build the SQL query
$sql = 'CALL catalog_update_category(:category_id, :category_name,
:category_description)';
// Build the parameters array
$params = array (':category_id' => $categoryId,
':category_name' => $categoryName,
':category_description' => $categoryDescription);
// Execute the query
DatabaseHandler::Execute($sql, $params);
}
// Deletes a category
public static function DeleteCategory($categoryId)
{
// Build the SQL query
$sql = 'CALL catalog_delete_category(:category_id)';
// Build the parameters array
$params = array (':category_id' => $categoryId);
// Execute the query and return the results
return DatabaseHandler::GetOne($sql, $params);
}
```

5. Zmodyfikuj metodę init() klasy StoreAdmin znajdującą się w Presentation/store_admin.php, aby załadować nowo dodane szablony składowe:

```
// Choose what admin page to load ...
if ($admin_page == 'Departments')
$this->mContentsCell = 'admin_departments.tpl';
elseif ($admin_page == 'Categories')
$this->mContentsCell = 'admin_categories.tpl';
```

6. Użyj phpMyAdmin, aby wykonać następujący kod, który tworzy procedury składowane warstwy danych w bazie danych tshirtshop. Nie zapomnij ustawić $$ jako separatora.

```
-- Create catalog_get_department_categories stored procedure
CREATE PROCEDURE catalog_get_department_categories(IN inDepartmentId INT)
BEGIN
SELECT category_id, name, description
FROM category
WHERE department_id = inDepartmentId
ORDER BY category_id;
END$$
-- Create catalog_add_category stored procedure
CREATE PROCEDURE catalog_add_category(IN inDepartmentId INT,
IN inName VARCHAR(100), IN inDescription VARCHAR(1000))
BEGIN
INSERT INTO category (department_id, name, description)
VALUES (inDepartmentId, inName, inDescription);
END$$
-- Create catalog_update_category stored procedure
CREATE PROCEDURE catalog_update_category(IN inCategoryId INT,
IN inName VARCHAR(100), IN inDescription VARCHAR(1000))
BEGIN
UPDATE category
SET name = inName, description = inDescription
WHERE category_id = inCategoryId;
END$$
-- Create catalog_delete_category stored procedure
CREATE PROCEDURE catalog_delete_category(IN inCategoryId INT)
BEGIN
DECLARE productCategoryRowsCount INT;
SELECT count(*)
FROM product p
INNER JOIN product_category pc
ON p.product_id = pc.product_id
```

```
WHERE pc.category_id = inCategoryId

INTO productCategoryRowsCount;

IF productCategoryRowsCount = 0 THEN

DELETE FROM category WHERE category_id = inCategoryId;

SELECT 1;

ELSE

SELECT -1;

END IF;

END$$
```

7. Załaduj admin.php w przeglądarce, wybierz dział i kliknij przycisk Edytuj kategorie. Powinien zostać załadowany szablon z komponentami kategorii. Edycja kategorii powinna działać tak, jak pokazano na rysunku powyżej.

Jak to działa: administrowanie kategoriami

Kod do zarządzania kategoriami jest zgodny z tymi samymi wzorcami, co kod do zarządzania działami. Krótko mówiąc, wykonałeś te zadania:

• Utworzyłeś składnikowy szablon admin_categories Smarty, który składa się z dwóch części: szablonu (admin_categories.tpl) i powiązanego z nim obiektu prezentacji (admin_categories.php). Te dwa pliki implementują warstwę prezentacji funkcji administrowania kategoriami.

• Zaktualizowano skomponowany szablon store_admin Smarty (dokładniej jego obiekt prezentacji), aby załadować szablon admin_categories po wybraniu kategorii. Jak wiecie od początku rozdziału, store_admin to skomponowany szablon ładowany przez admin.php; jego rolą jest decydowanie, jaki szablon administracyjny ma zostać załadowany.

• Utworzono metody warstwy biznesowej, które obsługują niezbędne funkcje administracyjne kategorii: dodawanie, edytowanie i usuwanie kategorii.

• Utworzono procedury składowane bazy danych, które są wywoływane przez ich odpowiedniki w warstwie biznesowej w celu efektywnego dodawania, edytowania lub usuwania kategorii.

Przyjrzyj się bliżej nowemu kodowi, który dodałeś, aby upewnić się, że dokładnie rozumiesz, jak on działa, zanim przejdziesz do implementacji funkcji zarządzania produktami i atrybutami w rozdziale 11.

Podsumowanie

Wykonałeś całkiem sporo kodowania. Zaimplementowano szereg szablonów składowych wraz z ich metodami warstwy środkowej i procedurami składowanymi dla warstwy danych. Nauczyłeś się, jak zaimplementować prosty schemat uwierzytelniania, dzięki czemu tylko administratorzy mają dostęp do strony zarządzania katalogiem, a także zaimplementowałeś funkcje zarządzania działami i kategoriami w TShirtShop. W Części 11 zakończysz stronę administracyjną, dodając funkcje zarządzania produktami i ich atrybutami.