Przyjmowanie zamówień klientów

Twój nowy, błyszczący koszyk AJAXified jest w pełni funkcjonalny, z wyjątkiem tego, że nie pozwala odwiedzającym na składanie zamówień, co jest dość kłopotliwe, ponieważ o to w tym wszystkim chodzi! Zajmiemy się tym problemem w dwóch oddzielnych etapach:

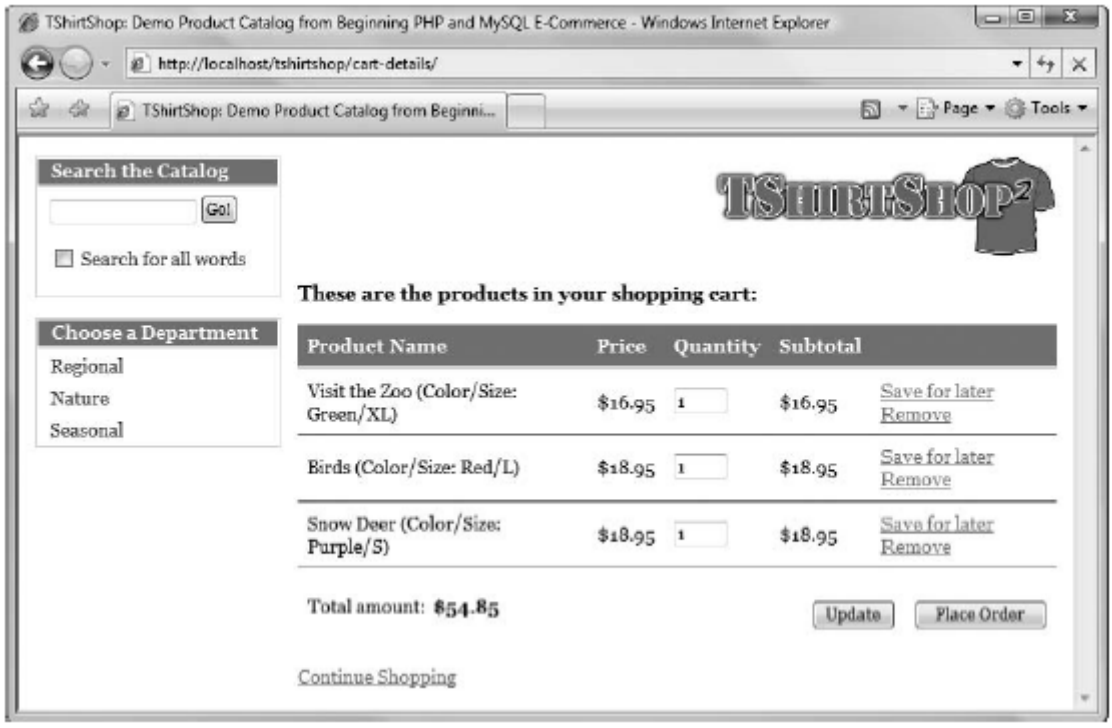
• Najpierw wdrożymy mechanizm składania zamówień po stronie odwiedzających. Dokładniej, dodamy przycisk Złóż zamówienie do strony koszyka, który tworzy zamówienie PayPal zawierające produkty w koszyku (pamiętaj, że na tym etapie nadal nie obsługujemy transakcji finansowych).

• Następnie wdrożymy prostą stronę administrowania zamówieniami, aby administrator witryny mógł przeglądać i obsługiwać oczekujące zamówienia.

Kod dla każdej części witryny zostanie zaprezentowany w zwykły sposób, zaczynając od warstwy bazy danych, kontynuując warstwę biznesową i kończąc na warstwie prezentacji (interfejsie użytkownika).

Wdrożenie systemu składania zamówień

Cały system składania zamówień związany jest ze wspomnianym wcześniej przyciskiem Złóż zamówienie. Rysunek pokazuje, jak ten przycisk będzie wyglądał po zaktualizowaniu szablonu składowego cart_details.



Tak, ten przycisk wygląda dość nudno jak na coś, co możemy szczerze powiedzieć, jest centrum wszechświata tej części. Jednak kryje się za tym sporo logiki, więc porozmawiajmy o tym, co powinno się stać, gdy klient kliknie ten przycisk. Pamiętaj, że na tym etapie nie obchodzi nas kto składa zamówienie, ale chcemy przechowywać szczegóły zamówienia w naszej bazie danych. Umożliwi nam to wdrożenie funkcji cross-sellingu („klienci, którzy to kupili, kupili również"). Zasadniczo po kliknięciu przycisku Złóż zamówienie muszą się wydarzyć trzy rzeczy:

• Po pierwsze, musisz przechowywać zamówienie gdzieś w bazie danych. Stworzysz kilka nowych tabel (orders i order_detail) i napiszesz kod, który zapisze zamówione produkty w tych tabelach.

• Następnie musimy wyczyścić koszyk. Po złożeniu zamówienia koszyk powinien być pusty. Istnieje szansa, że klienci anulują swoje zamówienia na etapie kasy – nie chcemy, aby ich koszyki kręciły się w pobliżu i zapełniały naszą bazę danych anulowanymi zamówieniami. PayPal i inne podmioty przetwarzające płatności oferują mechanizmy programowego powiadamiania o opłaceniu zamówienia.

• Na koniec wysyłamy odwiedzającego na stronę płatności PayPal, aby zapłacić za zamówienie.

* **Uwaga**: Ponieważ jesteśmy w fazie rozwoju,i nadal nie przetwarzamy płatności sami, ale korzystamy z zewnętrznego procesora płatności. Teraz nie potrzebujemy już koszyka na zakupy PayPal, ponieważ wprowadziliśmy własny w poprzednich kilku rozdziałach. Zamiast tego użyjemy opcji Zakupy pojedynczych przedmiotów w systemie PayPal, która przenosi użytkownika z naszego koszyka na stronę płatności PayPal.

Problem, który pojawia się podczas korzystania z zewnętrznego procesora płatności, polega na tym, że klient może anulować zamówienie na stronie kasy, która nadal znajduje się w systemie PayPal. Może to spowodować, że zamówienia zostaną zapisane w bazie danych, za które nie dokonano płatności. Oczywiście potrzebujemy systemu potwierdzania płatności wraz ze strukturą bazy danych, która jest w stanie przechowywać informacje o statusie każdego zamówienia.

System potwierdzeń, który wdrożymy, jest prosty. Każdy procesor płatności, w tym PayPal, można skonfigurować tak, aby wysyłał wiadomość potwierdzającą po przetworzeniu płatności. Pozwolimy administratorowi witryny ręcznie sprawdzić na stronie administracyjnej, za które zamówienia zostały opłacone i podjąć odpowiednie kroki.

* **Uwaga**: PayPal i jego konkurenci oferują zautomatyzowane systemy, które informują Twoją witrynę internetową o zakończeniu lub anulowaniu płatności. Jednak ta książka nie bada szczegółowych szczegółów żadnego z tych systemów płatności — musisz odrobić pracę domową i przestudiować dokumentację wybranej usługi. Dokumentacja natychmiastowego powiadomienia o płatnościach PayPal jest zawarta w Podręczniku integracji zarządzania zamówieniami.

Przechowywanie szczegółów zamówienia

Jak wspomniano wcześniej, implementację nowej funkcji rozpoczynamy od stworzenia niezbędnych struktur danych. W tym momencie nie powinno cię to dziwić. Wiesz, że decydowanie o tym, jakich informacji użyć i jak je przechowywać, bardzo pomaga podczas analizowania nowej funkcji i stanowi techniczną podstawę implementacji tej funkcji. Istnieją dwa rodzaje informacji, które chcemy przechowywać podczas składania zamówienia:

• Szczegóły dotyczące zamówienia jako całości: Jaką datę utworzono zamówienie? Czy produkty zostały wysłane, a jeśli tak, to kiedy zostały wysłane? A jaki jest teraz status zamówienia? Będziemy przechowywać te dane w tabeli o nazwie zamówienia, gdzie każdy rekord reprezentuje zamówienie.

• Szczegóły produktu dla zamówienia: Jakie produkty zostały zamówione w jakiej kolejności? Będziemy przechowywać te dane w tabeli o nazwie order_detail, gdzie każdy rekord reprezentuje zamówiony produkt. Wiele rekordów z tej tabeli będzie powiązanych z jednym rekordem w tabeli zamówień, tworząc relację jeden-do-wielu między tabelami .

* **Wskazówka**: Do tej pory konsekwentnie nazywaliśmy nasze tabele w liczbie pojedynczej (koszyk na zakupy, dział itd.). Jednak tutaj robimy wyjątek dla tabeli zamówień, ponieważ ORDER jest słowem kluczowym SQL. Na potrzeby tej książki wolimy złamać konwencję nazewnictwa, aby uniknąć

zamieszania podczas pisania kodu SQL, a ogólnie rzecz biorąc, używanie słów kluczowych SQL jako nazw obiektów nie jest dobrą praktyką.

Tabela orders zawiera informacje dotyczące zamówienia jako całości, natomiast order_detail zawiera produkty, które należą do każdego zamówienia. W poniższym ćwiczeniu stworzymy tabele

Ćwiczenie: Tworzenie zamówień i tablic order_detail

1. Załaduj phpMyAdmin, wybierz bazę danych tshirtshop i otwórz nową stronę zapytań SQL.

2. Wykonaj ten kod, który utworzy tabelę zamówień w Twojej bazie danych tshirtshop:

```
-- Create orders table
CREATE TABLE `orders` (
`order_id` INT NOT NULL AUTO_INCREMENT,
`total_amount` NUMERIC(10, 2) NOT NULL DEFAULT 0.00,
`created_on` DATETIME NOT NULL,
`shipped_on` DATETIME,
`status` INT NOT NULL DEFAULT 0,
`comments` VARCHAR(255),
`customer_name` VARCHAR(100),
`shipping_address` VARCHAR(255),
`customer_email` VARCHAR(50),
PRIMARY KEY (`order_id`)
);
```

3. Wykonaj następujący kod, który utworzy tabelę order_detail w Twojej bazie danych tshirtshop:

```
-- Create order_detail table
CREATE TABLE `order_detail` (
`item_id` INT NOT NULL AUTO_INCREMENT,
`order_id` INT NOT NULL,
`product_id` INT NOT NULL,
`attributes` VARCHAR(1000) NOT NULL,
`product_name` VARCHAR(100) NOT NULL,
`quantity` INT NOT NULL,
`unit_cost` NUMERIC(10, 2) NOT NULL,
PRIMARY KEY (`item_id`),
KEY `idx_order_detail_order_id` (`order_id`)
```

);

Teraz, gdy już stworzyłeś tabele, przyjrzyjmy się bliżej ich strukturze i relacjom

Jak to działa: tabela zamówień

Tabela zamówień zawiera dwie kategorie informacji: dane o samym zamówieniu (pierwsze sześć pól) oraz dane o kliencie, który złożył zamówienie (ostatnie trzy pola). Przechowywanie danych klienta w tabeli zamówień (w polach customer_name, shipping_address i customer_email) jest opcjonalne. Administrator witryny może użyć tej funkcji, jeśli pomaga w zadaniach związanych z zarządzaniem zamówieniami, ale na tym etapie nie ma znaczenia, kto złożył zamówienie, tylko jakie produkty zostały sprzedane. W III fazie rozwoju przejdziemy do w pełni profesjonalnego wdrożenia i będziemy przechowywać informacje o klientach we własnej tabeli danych. Nazwy pól nie wymagają wyjaśnień. order_id to klucz podstawowy tabeli. total_amount przechowuje całkowitą wartość zamówienia. created_on i send_on określają, kiedy zamówienie zostało utworzone i wysłane (ten ostatni obsługuje wartości NULL, co oznacza po prostu, że zamówienie nie zostało jeszcze wysłane). Pole statusu zawiera liczbę całkowitą, która może mieć następujące wartości:

• 0: Zamówienie zostało złożone. Jest to początkowy status zamówienia po kliknięciu przycisku Złóż zamówienie w koszyku.

• 1: Zamówienie zostało zweryfikowane. Administrator oznacza zamówienie jako zweryfikowane po potwierdzeniu płatności.

• 2: Zamówienie zakończone. Administrator oznacza zamówienie jako zrealizowane po wysłaniu produktów. Jednocześnie wypełniane jest również pole seand_on.

• 3: Zamówienie zostało anulowane. Zazwyczaj administrator oznacza zamówienie jako anulowane, jeśli zamówienie zostało złożone (klikając przycisk Złóż zamówienie), ale płatność nie została przetworzona, lub w innych sytuacjach wymagających anulowania zamówienia.

Rysunek przedstawia kilka przykładowych rekordów z tabeli zamówień.

order_id	total_amount	created_on	shipped_on	status	comments	customer_name	shipping_address	customer_email
1	30.94	2007-12-09 17:50:04	2007-12-09 20:31:00	2				
2	37.00	2007-12-09 19:56:17	*NULL*	1				
3	54.85	2007-12-09 20:03:03	*NULL*	0				

Jak to działa: tabela order_detail

Rzućmy okiem na rysunek, aby zobaczyć kilka przykładów rekordów w tabeli order_detail.

item_id	order_id	product_id	attributes	product_name	quantity	unit_cost
1	1	87	Color/Size: White/S	Christmas Tree	1	17.95
2	1	51	Color/Size: Yellow/L	Tankanyika Giraffe	1	12.99
3	2	94	Color/Size: Orange/M	Swede Santa	1	18.50
4	2	85	Color/Size: Red/L	Altar Piece	1	18.50
5	3	36	Color/Size: Green/XL	Visit the Zoo	1	16.95
6	3	97	Color/Size: Red/L	Birds	1	18.95
7	3	80	Color/Size: Purple/S	Snow Deer	1	18.95

Każdy rekord w order_detail reprezentuje zamówiony produkt, który należy do zamówienia określonego przez order_id. Tworząc klucz podstawowy tej tabeli, napotykamy te same ograniczenia, które mieliśmy przy tworzeniu tabeli shopping_cart. Zwykle klucz podstawowy może być utworzony z

(identyfikator_zamówienia, identyfikator_produktu, atrybuty), ale ponieważ nie jest to możliwe, utworzyliśmy dodatkowe pole o nazwie item_id, które pełni rolę klucza podstawowego. Przechowujemy również nazwę produktu, atrybuty, ilość i cenę (koszt_jednostka). Być może zastanawiasz się, dlaczego rejestrujemy te dane, skoro mamy już pole product_id, które może prowadzić do tych danych. Ważne jest, aby zrozumieć, że dane, które rejestrujemy dla zamówienia, są przechowywane w celach historycznych. Identyfikatory produktów, nazwy i ceny mogą ulec zmianie, ale aby te zmiany nie wpłynęły na szczegóły zamówienia złożonego w przeszłości, musimy mieć pewność, że zapiszemy je w tabeli, na którą nie mają wpływu zmiany produktów. Przechowujemy product_id, ponieważ jest to jedyny zautomatyzowany sposób linkowania do oryginalnych informacji o produkcie (jeśli produkt nadal istnieje).

Implementacja warstwy danych

Na tym etapie musisz dodać dwie dodatkowe procedury składowane w warstwie danych w bazie danych tshirtshop. Pierwszym i najważniejszym jest shopping_cart_create_order, który pobiera produkty z koszyka i tworzy z nimi zamówienie. Druga procedura składowana to shopping_cart_empty, która opróżnia koszyk odwiedzającego po złożeniu zamówienia. W poniższym ćwiczeniu zaimplementujemy te procedury składowane zaczynając od shopping_cart_empty, ponieważ jest on wywoływany z shopping_cart_create_order.

Ćwiczenie: Tworzenie procedur składowanych

1. Użyj phpMyAdmin, aby utworzyć procedury składowane opisane w poniższych krokach. Nie zapomnij ustawić ogranicznika $$ przed wykonaniem kodu każdego kroku.

2. Wykonaj następujący kod, który utworzy procedurę składowaną shopping_cart_empty w bazie danych sklepu tshirtshop. Gdy klient złoży zamówienie, shopping_cart_create_order wywoła shopping_cart_empty, aby usunąć produkty z koszyka klienta.

```
-- Create shopping_cart_empty stored procedure

CREATE PROCEDURE shopping_cart_empty(IN inCartId CHAR(32))

BEGIN

DELETE FROM shopping_cart WHERE cart_id = inCartId;

END$$
```

3. Wykonaj następujący kod, który utworzy procedurę składowaną shopping_cart_create_order w bazie danych sklepu tshirtshop. Ta procedura składowana jest wywoływana, gdy klient zdecyduje się kupić produkty w koszyku i kliknie przycisk Złóż zamówienie. Rolą shopping_cart_create_order jest utworzenie nowego zamówienia na podstawie produktów w koszyku klienta. Oznacza to dodanie nowego rekordu do tabeli zamówień i pewnej liczby rekordów (jeden rekord dla każdego produktu) w tabeli order_detail.

```
-- Create shopping_cart_create_order stored procedure

CREATE PROCEDURE shopping_cart_create_order(IN inCartId CHAR(32))

BEGIN

DECLARE orderId INT;

-- Insert a new record into orders and obtain the new order ID
```

```
INSERT INTO orders (created_on) VALUES (NOW());

-- Obtain the new Order ID

SELECT LAST_INSERT_ID() INTO orderId;

-- Insert order details in order_detail table

INSERT INTO order_detail (order_id, product_id, attributes,

product_name, quantity, unit_cost)

SELECT orderId, p.product_id, sc.attributes, p.name, sc.quantity,

COALESCE(NULLIF(p.discounted_price, 0), p.price) AS unit_cost

FROM shopping_cart sc

INNER JOIN product p

ON sc.product_id = p.product_id

WHERE sc.cart_id = inCartId AND sc.buy_now;

-- Save the order's total amount

UPDATE orders

SET total_amount = (SELECT SUM(unit_cost * quantity)

FROM order_detail

WHERE order_id = orderId)

WHERE order_id = orderId;

-- Clear the shopping cart

CALL shopping_cart_empty(inCartId);

-- Return the Order ID

SELECT orderId;

END$$
```

Jak to działa: Implementacja shopping_cart_empty i shopping_cart_create_order

Pierwszym krokiem we wdrożeniu shopping_cart_create_order jest utworzenie nowego rekordu w tabeli zamówień. Musisz to zrobić na początku, aby dowiedzieć się, jaki identyfikator zamówienia został wygenerowany dla nowego zamówienia. Pamiętaj, że pole order_id jest kolumną AUTO_INCREMENT i jest automatycznie generowane przez bazę danych, więc po wstawieniu rekordu do zamówień musisz pobrać jego wartość:

-- Insert a new record into orders and obtain the new order ID INSERT INTO orders (created_on) VALUES (NOW());

-- Obtain the new Order ID

SELECT LAST_INSERT_ID() INTO orderId;

Jest to podstawowy mechanizm wydobywania nowo wygenerowanego identyfikatora. Po instrukcji INSERT można uzyskać ostatnią wartość wygenerowaną dla AUTO_INCREMENT, czytając LAST_INSERT_ID(). Funkcjonalność jest dość prosta. Odczytujesz wartość LAST_INSERT_ID() i zapisujesz ją w zmiennej o nazwie orderId. Korzystając z wartości orderId, dodajesz rekordy order_detail, zbierając informacje z tabel product i shopping_cart. Otrzymasz listę produktów i ich ilości z koszyka_zakupów, ich nazwy i ceny z produktu i zapisujesz te rekordy jeden po drugim w tabeli order_detail.

```
-- Insert order details in order_detail table

INSERT INTO order_detail (order_id, product_id, attributes,

product_name, quantity, unit_cost)

SELECT orderId, p.product_id, sc.attributes, p.name, sc.quantity,

COALESCE(NULLIF(p.discounted_price, 0), p.price) AS unit_cost

FROM shopping_cart sc

INNER JOIN product p

ON sc.product_id = p.product_id

WHERE sc.cart_id = inCartId AND sc.buy_now;
```

* **Wskazówka** : Łącząc koszyk i produkt, otrzymujesz identyfikator produktu z produktu, ale możesz go również uzyskać z koszyka; wynik byłby taki sam, ponieważ sprzężenie tabeli odbywa się w kolumnie product_id.

Procedura składowana oblicza również całkowitą kwotę zamówienia, mnożąc cenę każdego produktu przez jego ilość. Ta wartość jest następnie zapisywana jako całkowita_kwota zamówienia:

```
-- Save the order's total amount

UPDATE orders

SET total_amount = (SELECT SUM(unit_cost * quantity)

FROM order_detail

WHERE order_id = orderId)

WHERE order_id = orderId;
```

Na koniec funkcja opróżnia koszyk odwiedzającego, wywołując procedurę składowaną shopping_cart_empty i zwraca identyfikator zamówienia:

```
-- Clear the shopping cart

CALL shopping_cart_empty(inCartId);

-- Return the Order ID

SELECT orderId;
```

Wdrażanie poziomu biznesowego

Warstwa biznesowa funkcji składania zamówień składa się z jednej metody, CreateOrder. Dodaj tę metodę do klasy ShoppingCart wewnątrz business/shopping_cart.php:

```
// Create a new order

public static function CreateOrder()

{

// Build SQL query

$sql = 'CALL shopping_cart_create_order(:cart_id)';

// Build the parameters array

$params = array (':cart_id' => self::GetCartId());

// Execute the query and return the results

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

}
```

Metoda wywołuje procedurę składowaną warstwy danych shopping_cart create_order, która tworzy nowe zamówienie na podstawie otrzymanego identyfikatora koszyka na zakupy i zwraca order_id nowo utworzonego zamówienia.

Wdrażanie poziomu prezentacji

Na koniec zobaczysz, że napisany przez Ciebie kod został wprowadzony w życie. Przycisk Złóż zamówienie jest jedynym dodatkiem po stronie odwiedzającego w interfejsie niestandardowej kasy. Najpierw umieśćmy przycisk na pliku szablonu cart_details, a następnie zaimplementujmy jego funkcjonalność.

Ćwiczenie: Składanie zamówień

1. Zmodyfikuj prezentację/templates/cart_details.tpl, dodając nowy przycisk tuż za przyciskiem Aktualizuj, jak zaznaczono w poniższym fragmencie kodu:

```
<table class="cart-subtotal">

<tr>

<td>

<p>

Total amount: 

<font class="price">${$obj->mTotalAmount}</font>

</p>

</td>

<td align="right">

<input type="submit" name="update" value="Update" />

</td>
```

```
<td align="right">
<input type="submit" name="place_order" value="Place Order"
onclick="placingOrder=true;" />
</td>
</tr>
</table>
```

2. Teraz musimy również dokonać małej zmiany w ajax.js, aby upewnić się, że nowy przycisk nie jest obsługiwany przez kod JavaScript. Jeśli pamiętasz, w rozdziale 13 dodaliśmy procedurę obsługi zdarzenia onsubmit do formularza koszyka zakupów, dzięki czemu akcje koszyka są obsługiwane asynchronicznie, w miarę możliwości przy użyciu AJAX. Nie chcemy, aby tak się działo w przypadku przycisku Złóż zamówienie. Ten przycisk musi przesłać formularz na serwer, który przekierowuje żądanie do strony kasy. Zmodyfikuj ajax.js tak, jak zaznaczono:

```
// Holds an instance of XMLHttpRequest
var xmlHttp = createXmlHttpRequestObject();
// Display error messages (true) or degrade to non-AJAX behavior (false)
var showErrors = true;
// Contains the link or form clicked or submitted by the visitor
var actionObject = '';
// This is true when the Place Order button is clicked, false otherwise
var placingOrder = false;
// Creates an XMLHttpRequest instance
function createXmlHttpRequestObject()
```

3. W momencie składania zamówienia odczytujemy wartość złożenia Zamówienia. Dzieje się tak po kliknięciu przycisku Złóż zamówienie; dlatego zwracamy true, aby formularz został przesłany w zwykły sposób. Upewniamy się również, że nie wysyłamy elementu formularza place_order podczas żądań AJAX dotyczących aktualizacji koszyka. Zmodyfikuj funkcję executeCartAction() w ajax.js, tak jak zaznaczono:

```
// Called on shopping cart update actions
function executeCartAction(obj)
{
// Degrade to classical form submit for Place Order action
if (placingOrder) return true;
// Display "Updating..." message
document.getElementById('updating').style.visibility = 'visible';
```

```
// Degrade to classical form submit if XMLHttpRequest is not available
if (!xmlHttp) return true;
// Save object reference
actionObject = obj;
// Initialize response and parameters
response = '';
params = '';
// If a link was clicked we get its href attribute
if (obj.tagName == 'A')
{
url = obj.href + '&AjaxRequest';
}
// If the form was submitted we get its elements
else
{
url = obj.action + '&AjaxRequest';
formElements = obj.getElementsByTagName('INPUT');
if (formElements)
{
for (i = 0; i < formElements.length; i++)
{
if (formElements[i].name != 'place_order')
{
params += '&' + formElements[i].name + '=';
params += encodeURIComponent(formElements[i].value);
}
}
}
}
```

4. Sprawmy, aby przycisk Złóż zamówienie działał teraz. Ponieważ ta funkcja zależy od firmy, która przetwarza Twoje płatności, może być konieczne dostosowanie jej do zachowania Twojej firmy obsługującej płatności. Tutaj używamy PayPala. Zacznij od zmodyfikowania stałych związanych z PayPal

w config.php w następujący sposób. Nie zapomnij zastąpić adresu e-mail@example.com adresem e-mail zarejestrowanym w systemie PayPal.

```
// PayPal configuration
define('PAYPAL_URL',
'https://www.paypal.com/xclick/business=youremail@example.com');
define('PAYPAL_CURRENCY_CODE', 'USD');
define('PAYPAL_RETURN_URL', 'http://www.example.com');
define('PAYPAL_CANCEL_RETURN_URL', 'http://www.example.com');
```

5. Dodaj następujący podświetlony kod w metodzie init() klasy CartDetails w pliku Presentation/cart_details.php:

```
/* Calculate the total amount for the shopping cart
before applicable taxes and/or shipping */
$this->mTotalAmount = ShoppingCart::GetTotalAmount();
// If the Place Order button was clicked ...
if(isset ($_POST['place_order']))
{
// Create the order and get the order ID
$order_id = ShoppingCart::CreateOrder();
// This will contain the PayPal link
$redirect =
PAYPAL_URL . '&item_name=TShirtShop Order ' . urlencode('#') . $order_id .
'&item_number=' . $order_id .
'&amount=' . $this->mTotalAmount .
'&currency_code=' . PAYPAL_CURRENCY_CODE .
'&return=' . PAYPAL_RETURN_URL .
'&cancel_return=' . PAYPAL_CANCEL_RETURN_URL;
// Redirection to the payment page
header('Location: ' . $redirect);
exit();
}
// Get shopping cart products
$this->mCartProducts =
```

ShoppingCart::GetCartProducts(GET_CART_PRODUCTS);

6. Twój przycisk Złóż zamówienie jest w pełni funkcjonalny! Przetestuj, dodając produkty do koszyka i klikając Złóż zamówienie. Twój koszyk powinien zostać wyczyszczony, a Ty powinieneś zostać przekierowany na stronę płatności PayPal, taką jak pokazana na rysunku

Jak to działa: składanie zamówień

Kiedy odwiedzający kliknie przycisk Złóż zamówienie, zachodzą dwie ważne czynności. Najpierw zamówienie jest tworzone w bazie danych poprzez wywołanie metody CreateOrder klasy ShoppingCart. Ta funkcja wywołuje procedurę składowaną bazy danych shopping_cart_create_order w celu utworzenia nowego zamówienia z produktami w koszyku i zwraca identyfikator nowego zamówienia:

// Create the order and get the order ID

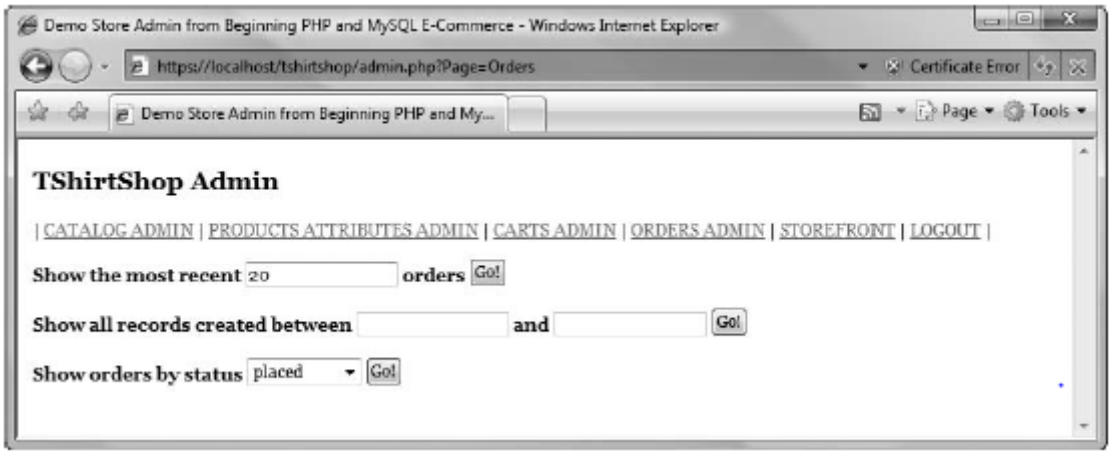
$order_id = ShoppingCart::CreateOrder();

Second, the visitor is redirected to the payment page, which requests payment for an item named “TShirtShop Order nnn” with a value that amounts to the total value of the order.

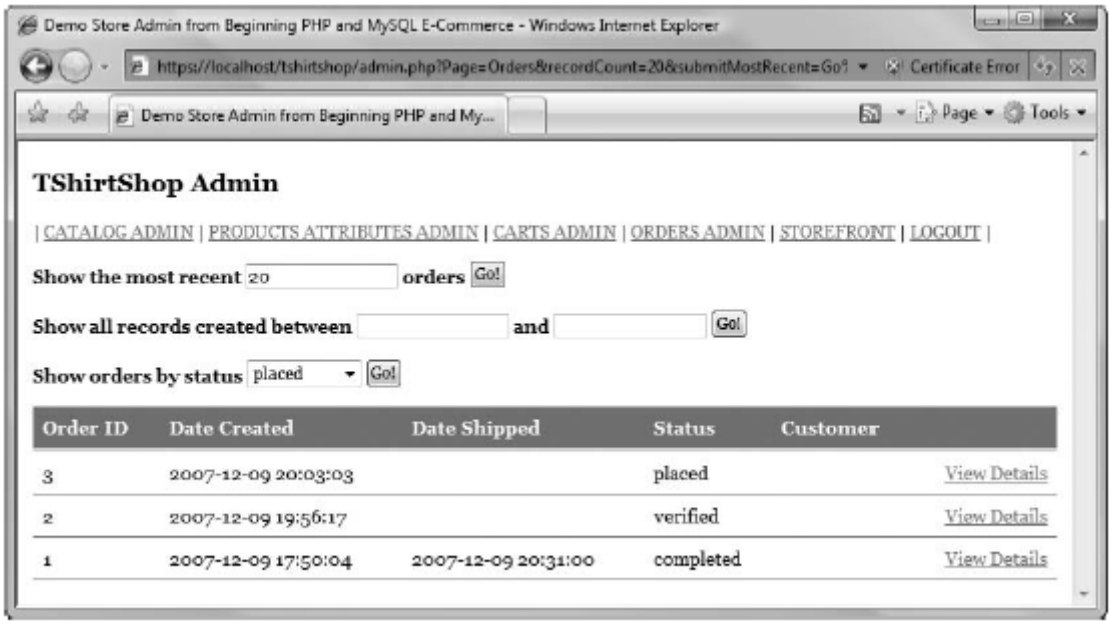
Administrowanie zamówieniami

Twój gość właśnie złożył zamówienie. Co teraz? Po udostępnieniu odwiedzającym opcji zapłaty za Twoje produkty musisz upewnić się, że faktycznie dostaną to, za co zapłacili. TShirtShop potrzebuje starannie zaprojektowanej strony zarządzania zamówieniami, na której administrator może szybko zobaczyć status oczekujących zamówień.

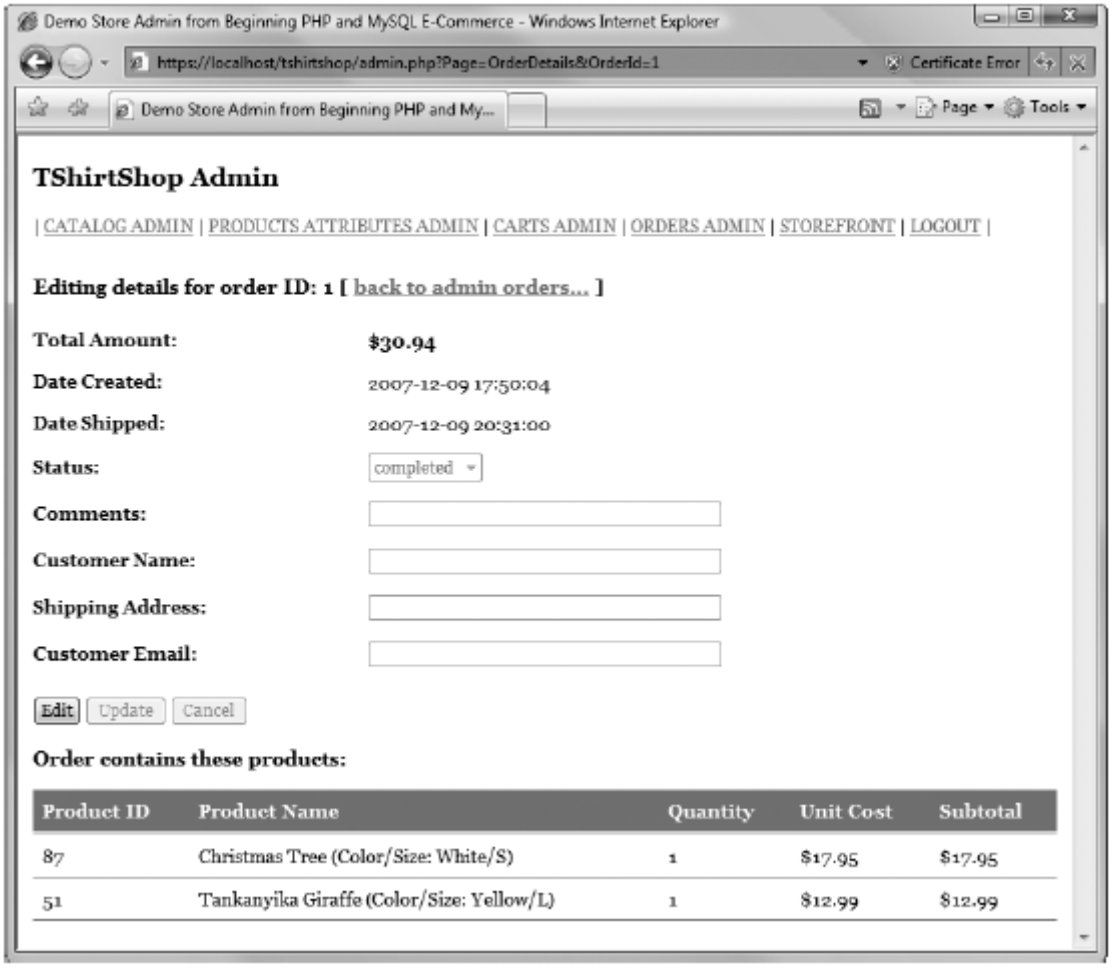
Część serwisu dotycząca administrowania zamówieniami będzie się składać z dwóch szablonów składowych o nazwach admin_orders i admin_order_details. Kiedy administrator kliknie link ORDERS ADMIN, strona admin.php ładuje skomponowany szablon admin_orders, który oferuje możliwość filtrowania zamówień. Po pierwszym załadowaniu oferuje różne sposoby wybierania zamówień, jak pokazano na rysunku



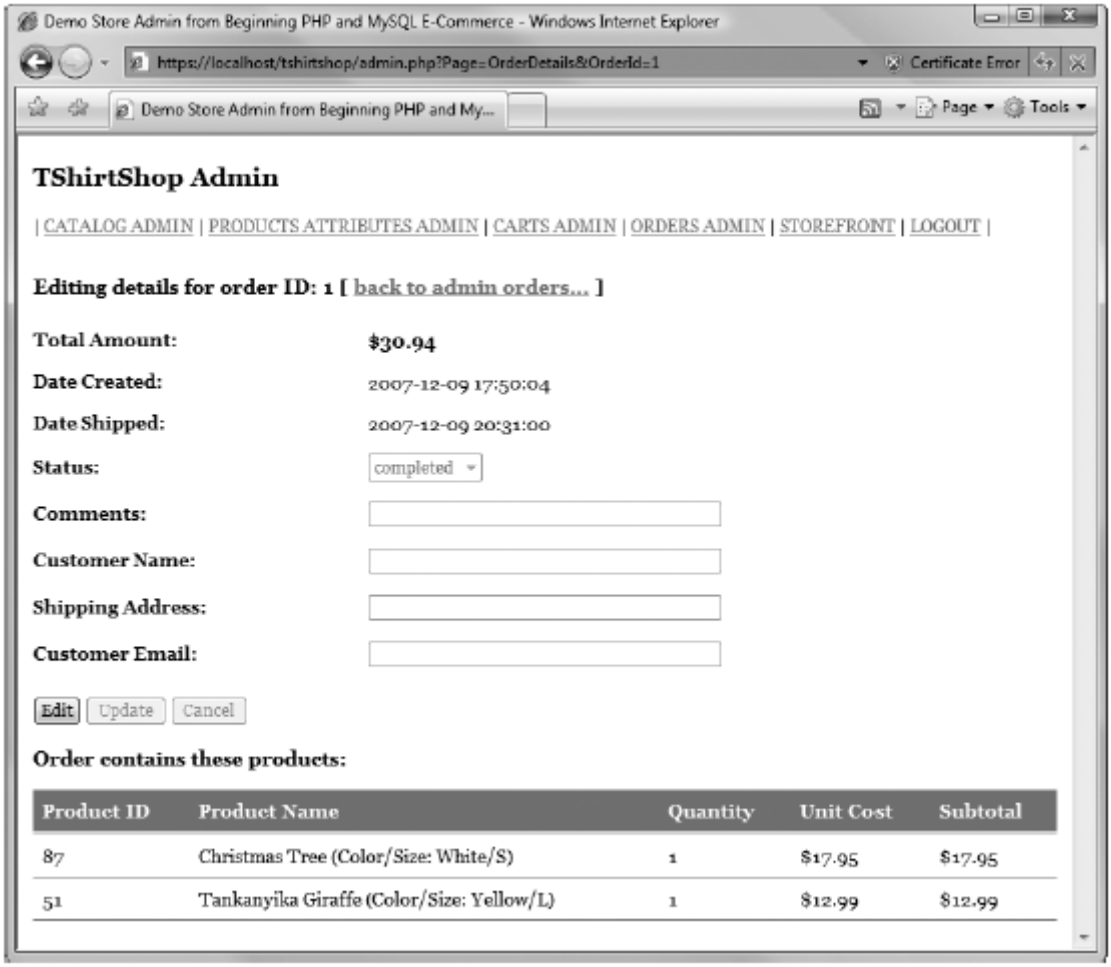
Po kliknięciu jednego z przycisków Idź, pasujące zamówienia pojawiają się w tabeli



Po kliknięciu przycisku Wyświetl szczegóły zamówienia zostaniesz przekierowany na stronę, na której możesz wyświetlić i zaktualizować informacje o zamówieniu, jak pokazano na rysunku



Po kliknięciu przycisku Wyświetl szczegóły zamówienia zostaniesz przekierowany na stronę, na której możesz wyświetlić i zaktualizować informacje o zamówieniu, jak pokazano na rysunku



Przed utworzeniem skomponentowych szablonów admin_orders i admin_order_details musimy zmodyfikować admin.php, aby załadować te skomponentowane szablony, a także zmodyfikować admin_menu.tpl, aby wyświetlał link ORDERS ADMIN. Wykonaj następujące ćwiczenie, aby przygotować grunt pod funkcje administracyjne, które stworzymy w dalszej części rozdziału. Zwróć uwagę, że po wykonaniu ćwiczenia admin.php nie będzie działał, ponieważ będzie odwoływał się do nowego skryptu warstwy biznesowej, który stworzymy nieco później.

Ćwiczenie: Konfigurowanie strony ADMIN ORDERS

1. Zmodyfikuj admin.php tak, aby zawierał odniesienie do business/orders.php, który utworzymy później:

```
// Load Business Tier

require_once BUSINESS_DIR . 'catalog.php';

require_once BUSINESS_DIR . 'shopping_cart.php';

require_once BUSINESS_DIR . 'orders.php';
```

2. W Presentation/store_admin.php zmodyfikuj klasę StoreAdmin, dodając podświetlony kod na końcu metody init(). Ten kod ładuje admin_orders.tpl i admin_order_details.tpl:

```
elseif ($admin_page == 'ProductDetails')
```

```
$this->mContentsCell = 'admin_product_details.tpl';
elseif ($admin_page == 'Carts')
$this->mContentsCell = 'admin_carts.tpl';
elseif ($admin_page == 'Orders')
$this->mContentsCell = 'admin_orders.tpl';
elseif ($admin_page == 'OrderDetails')
$this->mContentsCell = 'admin_order_details.tpl';
}
}
}
?>
```

3. Dodaj następujące dwie metody w klasie Link, którą znajdziesz w Presentation/link.php:

```
// Create link to orders administration page
public static function ToOrdersAdmin()
{
return self::ToAdmin('Page=Orders');
}
// Create link to the order details administration page
public static function ToOrderDetailsAdmin($orderId)
{
$link = 'Page=OrderDetails&OrderId=' . $orderId;
return self::ToAdmin($link);
}
```

4. Otwórz Presentation/admin_menu.php i zmodyfikuj klasę AdminMenu dodając podświetlony kod tworzący link ORDERS ADMIN w menu administracyjnym:

```
<?php
class AdminMenu
{
public $mLinkToStoreAdmin;
public $mLinkToAttributesAdmin;
public $mLinkToCartsAdmin;
```

```
public $mLinkToOrdersAdmin;
public $mLinkToStoreFront;
public $mLinkToLogout;
public function __construct()
{
$this->mLinkToStoreAdmin = Link::ToAdmin();
$this->mLinkToAttributesAdmin = Link::ToAttributesAdmin();
$this->mLinkToCartsAdmin = Link::ToCartsAdmin();
$this->mLinkToOrdersAdmin = Link::ToOrdersAdmin();
if (isset ($_SESSION['link_to_store_front']))
$this->mLinkToStoreFront = $_SESSION['link_to_store_front'];
else
$this->mLinkToStoreFront = Link::ToIndex();
$this->mLinkToLogout = Link::ToLogout();
}
}
?>
```

5. Zmodyfikuj prezentację/templates/admin_menu.tpl, dodając podświetlony kod linku do strony zarządzania koszykiem. Używamy również stylu menu, aby elementy menu ładnie pasowały do układu.

```
<p class="menu"> |
<a href="{$obj->mLinkToStoreAdmin}">CATALOG ADMIN</a> |
<a href="{$obj->mLinkToAttributesAdmin}">PRODUCTS ATTRIBUTES ADMIN</a> |
<a href="{$obj->mLinkToCartsAdmin}">CARTS ADMIN</a> |
<a href="{$obj->mLinkToOrdersAdmin}">ORDERS ADMIN</a> |
<a href="{$obj->mLinkToStoreFront}">STOREFRONT</a> |
<a href="{$obj->mLinkToLogout}">LOGOUT</a> |
```

6. Otwórz plik tshirtshop.css z folderu stylów i dodaj następującą definicję stylu:

```
.menu {
font-size: 93%;
}
```

Jak to działa: konfiguracja strony zarządzania zamówieniami

W tym momencie nie ma co testować, ponieważ nie stworzyliśmy żadnej nowej znaczącej funkcjonalności. Przygotowaliśmy jedynie grunt pod wdrożenie funkcji zarządzania zamówieniami i szczegółami zamówień. Dodaliśmy również styl do tshirtshop.css, który sprawi, że menu administracyjne ładnie mieści się na jego stronie. Zauważ, że w tej chwili pojawia się błąd, jeśli spróbujesz załadować admin.php, ponieważ odwołujemy się do pliku, który jeszcze nie istnieje - business/orders.php. Ten plik utworzysz w następnym ćwiczeniu.

Wyświetlanie zleceń oczekujących

Na następnych kilku stronach zaimplementujesz skomponentowany szablon admin_orders i jego obsługę warstwy danych i warstwy biznesowej. admin_orders to złożony szablon który umożliwia administratorowi przeglądanie zamówień, które zostały złożone w serwisie. Ponieważ lista zamówień będzie z czasem bardzo długa, ważne jest, aby mieć kilka dobrze dobranych opcji filtrowania.

Administrator będzie mógł wybierać zamówienia według następujących kryteriów:

• Pokaż najnowsze zamówienia.

• Pokaż zamówienia, które miały miejsce w określonym czasie.

• Pokaż zamówienia z określoną wartością statusu.

Stworzymy te funkcje w następnym ćwiczeniu.

Ćwiczenie: Implementacja strony Administracja zamówieniami

1. Zaczynamy od stworzenia niezbędnych procedur składowanych. Użyj phpMyAdmin, aby utworzyć procedury składowane opisane w poniższych krokach. Nie zapomnij ustawić ogranicznika $$ przed wykonaniem kodu każdego kroku.

2. Wykonaj następujący kod, który utworzy procedurę składowaną orders_get_most_recent_orders w bazie danych tshirtshop. Ta procedura zwraca najnowsze zamówienia. Instrukcja SELECT używa klauzuli LIMIT, aby ograniczyć liczbę zwracanych wierszy do wartości parametru wejściowego inHowMany. The

Klauzula ORDER BY DESC, używana do sortowania wyników w kolejności malejącej, jest ustawiona tak, że najnowsze zamówienia będą wyświetlane jako pierwsze.

```
-- Create orders_get_most_recent_orders stored procedure

CREATE PROCEDURE orders_get_most_recent_orders(IN inHowMany INT)

BEGIN

PREPARE statement FROM

"SELECT order_id, total_amount, created_on,

shipped_on, status, customer_name

FROM orders

ORDER BY created_on DESC

LIMIT ?";

SET @p1 = inHowMany;

EXECUTE statement USING @p1;
```

END$$

3. Wykonaj następujący kod, aby utworzyć procedurę składowaną orders_get_orders_between_dates. Ta procedura zwraca zamówienia, których data utworzenia przypada między inStartDate a inEndDate. Wyniki są sortowane malejąco według daty utworzenia.

```
-- Create orders_get_orders_between_dates stored procedure

CREATE PROCEDURE orders_get_orders_between_dates(

IN inStartDate DATETIME, IN inEndDate DATETIME)

BEGIN

SELECT order_id, total_amount, created_on,

shipped_on, status, customer_name

FROM orders

WHERE created_on >= inStartDate AND created_on <= inEndDate

ORDER BY created_on DESC;

END$$
```

4. Wykonaj ten kod, który tworzy procedurę składowaną orders_get_orders_by_status. Ta procedura zwraca zamówienia, które mają wartość statusu określoną przez parametr inStatus

```
-- Create orders_get_orders_by_status stored procedure

CREATE PROCEDURE orders_get_orders_by_status(IN inStatus INT)

BEGIN

SELECT order_id, total_amount, created_on,

shipped_on, status, customer_name

FROM orders

WHERE status = inStatus

ORDER BY created_on DESC;

END$$
```

5. Warstwa biznesowa składa się z nowej klasy o nazwie Orders, której metody wywołują ich odpowiedniki w warstwie danych. Ta klasa jest dość prosta bez szczególnie złożonej logiki, więc po prostu wymienimy kod. Utwórz plik business/orders.php i dodaj do niego następujący kod:

```
<?php

// Business tier class for the orders

class Orders

{

public static $mOrderStatusOptions = array ('placed', // 0
```

```
'verified', // 1
'completed', // 2
'canceled'); // 3
// Get the most recent $how_many orders
public static function GetMostRecentOrders($how_many)
{
// Build the SQL query
$sql = 'CALL orders_get_most_recent_orders(:how_many)';
// Build the parameters array
$params = array (':how_many' => $how_many);
// Execute the query and return the results
return DatabaseHandler::GetAll($sql, $params);
}
// Get orders between two dates
public static function GetOrdersBetweenDates($startDate, $endDate)
{
// Build the SQL query
$sql = 'CALL orders_get_orders_between_dates(:start_date, :end_date)';
// Build the parameters array
$params = array (':start_date' => $startDate, ':end_date' => $endDate);
// Execute the query and return the results
return DatabaseHandler::GetAll($sql, $params);
}
// Gets orders by status
public static function GetOrdersByStatus($status)
{
// Build the SQL query
$sql = 'CALL orders_get_orders_by_status(:status)';
// Build the parameters array
$params = array (':status' => $status);
// Execute the query and return the results
```

```
return DatabaseHandler::GetAll($sql, $params);

}

}

?>
```

6. Teraz nadszedł czas na wdrożenie skomponentowanego szablonu admin_orders. Utwórz nowy plik o nazwie admin_orders.tpl w folderze prezentacji/szablonów z następującym kodem

```
{* admin_orders.tpl *}

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

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

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

<input name="Page" type="hidden" value="Orders" />

<p>

<font class="bold-text">Show the most recent</font>

<input name="recordCount" type="text" value="{$obj->mRecordCount}" />

<font class="bold-text">orders</font>

<input type="submit" name="submitMostRecent" value="Go!" />

</p>

<p>

<font class="bold-text">Show all records created between</font>

<input name="startDate" type="text" value="{$obj->mStartDate}" />

<font class="bold-text">and</font>

<input name="endDate" type="text" value="{$obj->mEndDate}" />

<input type="submit" name="submitBetweenDates" value="Go!" />

</p>

<p>

<font class="bold-text">Show orders by status</font>

{html_options name="status" options=$obj->mOrderStatusOptions

selected=$obj->mSelectedStatus}

<input type="submit" name="submitOrdersByStatus" value="Go!" />

</p>

</form>
```

```
{if $obj->mOrders}
<table class="tss-table">
<tr>
<th>Order ID</th>
<th>Date Created</th>
<th>Date Shipped</th>
<th>Status</th>
<th>Customer</th>
<th>&amp;nbsp;</th>
</tr>
{section name=i loop=$obj->mOrders}
{assign var=status value=$obj->mOrders[i].status}
<tr>
<td>{$obj->mOrders[i].order_id}</td>
<td>{$obj->mOrders[i].created_on|date_format:"%Y-%m-%d %T"}</td>
<td>{$obj->mOrders[i].shipped_on|date_format:"%Y-%m-%d %T"}</td>
<td>{$obj->mOrderStatusOptions[$status]}</td>
<td>{$obj->mOrders[i].customer_name}</td>
<td align="right">
<a href="{$obj->mOrders[i].link_to_order_details_admin}">View Details</a>
</td>
</tr>
{/section}
</table>
{/if}
```

7. Utwórz nowy plik o nazwie Presentation/admin_orders.php i dodaj do niego następujący kod:

```
<?php
/* Presentation tier class that supports order administration
functionality */
class AdminOrders
{
```

```
// Public variables available in smarty template
public $mOrders;
public $mStartDate;
public $mEndDate;
public $mRecordCount = 20;
public $mOrderStatusOptions;
public $mSelectedStatus = 0;
public $mErrorMessage = '';
public $mLinkToAdmin;
// Class constructor
public function __construct()
{
/* Save the link to the current page in the link_to_orders_admin
session variable; it will be used to create the
"back to admin orders ..." link in admin order details pages */
$_SESSION['link_to_orders_admin'] =
Link::Build(str_replace(VIRTUAL_LOCATION, '', getenv('REQUEST_URI')));
$this->mLinkToAdmin = Link::ToAdmin();
$this->mOrderStatusOptions = Orders::$mOrderStatusOptions;
}
public function init()
{
// If the "Show the most recent x orders" filter is in action ...
if (isset ($_GET['submitMostRecent']))
{
// If the record count value is not a valid integer, display error
if ((string)(int)$_GET['recordCount'] == (string)$_GET['recordCount'])
{
$this->mRecordCount = (int)$_GET['recordCount'];
$this->mOrders = Orders::GetMostRecentOrders($this->mRecordCount);
}
```

```
else
$this->mErrorMessage = $_GET['recordCount'] . ' is not a number.';
}
/* If the "Show all records created between date_1 and date_2"
filter is in action ... */
if (isset ($_GET['submitBetweenDates']))
{
$this->mStartDate = $_GET['startDate'];
$this->mEndDate = $_GET['endDate'];
// Check if the start date is in accepted format
if (($this->mStartDate == '') ||
($timestamp = strtotime($this->mStartDate)) == -1)
$this->mErrorMessage = 'The start date is invalid. ';
else
// Transform date to YYYY/MM/DD HH:MM:SS format
$this->mStartDate =
strftime('%Y/%m/%d %H:%M:%S', strtotime($this->mStartDate));
// Check if the end date is in accepted format
if (($this->mEndDate == '') ||
($timestamp = strtotime($this->mEndDate)) == -1)
$this->mErrorMessage .= 'The end date is invalid.';
else
// Transform date to YYYY/MM/DD HH:MM:SS format
$this->mEndDate =
strftime('%Y/%m/%d %H:%M:%S', strtotime($this->mEndDate));
// Check if start date is more recent than the end date
if ((empty ($this->mErrorMessage)) &&
(strtotime($this->mStartDate) > strtotime($this->mEndDate)))
$this->mErrorMessage .=
'The start date should be more recent than the end date.';
// If there are no errors, get the orders between the two dates
```

```
if (empty($this->mErrorMessage))
$this->mOrders = Orders::GetOrdersBetweenDates(
$this->mStartDate, $this->mEndDate);
}
// If "Show orders by status" filter is in action ...
if (isset ($_GET['submitOrdersByStatus']))
{
$this->mSelectedStatus = $_GET['status'];
$this->mOrders = Orders::GetOrdersByStatus($this->mSelectedStatus);
}
if (is_array($this->mOrders) && count($this->mOrders) == 0)
$this->mErrorMessage =
'No orders found matching your searching criteria!';
// Build View Details link
for ($i = 0; $i < count($this->mOrders); $i++)
{
$this->mOrders[$i]['link_to_order_details_admin'] =
Link::ToOrderDetailsAdmin($this->mOrders[$i]['order_id']);
}
}
}
?>
```

8. Załaduj admin.php do przeglądarki i podaj kombinację nazwy użytkownika/hasła, jeśli zostaniesz o to poproszony. Kliknij łącze menu ORDERS ADMIN, a następnie kliknij jeden z przycisków Przejdź. Powinieneś zobaczyć wyniki podobne do tych pokazanych na rysunku 14-5.

Jak to działa: pobieranie zamówień

Każdy z przycisków Przejdź wywołuje jedną z metod warstwy biznesowej (w klasie Orders) i wypełnia tabelę zwróconymi informacjami o zamówieniach. Podczas przetwarzania żądania testujemy dane wprowadzone przez odwiedzającego, aby upewnić się, że są one prawidłowe. Po kliknięciu pierwszego przycisku Go weryfikujemy, czy wprowadzona wartość jest liczbą (ile rekordów należy wyświetlić). Weryfikujemy również, czy daty wprowadzone w polach tekstowych Data rozpoczęcia i Data zakończenia są prawidłowe. Najpierw przetwarzamy daty za pomocą strtotime, które analizuje łańcuch i przekształca go w uniksowy znacznik czasu. Ta funkcja jest przydatna, ponieważ akceptuje również wpisy takie jak „teraz”, „jutro”, „ostatni piątek”, „następny wtorek” oraz jako wartości wejściowe.

Otrzymany znacznik czasu jest następnie przetwarzany za pomocą funkcji strftime, która przekształca go w format RRRR/MM/DD HH:MM:SS. Zobacz, jak analizowane są te wartości daty/godziny:

```
// Check if the start date is in accepted format

if (($this->mStartDate == '') ||

($timestamp = strtotime($this->mStartDate)) == -1)

$this->mErrorMessage = 'The start date is invalid. ';

else

// Transform date to YYYY/MM/DD HH:MM:SS format

$this->mStartDate =

strftime('%Y/%m/%d %H:%M:%S', strtotime($this->mStartDate));
```

Poza tym szczegółem plik szablonu admin_orders.tpl jest dość prosty i nie wprowadza dla Ciebie żadnych nowych elementów teoretycznych.

Wyświetlanie szczegółów zamówienia

W tej sekcji utworzysz skomponentowany szablon admin_order_details, który umożliwi administratorowi edycję szczegółów konkretnego zamówienia. Najczęstsze zadania to oznaczenie złożonego zamówienia jako zweryfikowanego lub anulowanego oraz oznaczenie zamówienia zweryfikowanego jako zrealizowanego, gdy przesyłka zostanie wysłana. Spójrz na rysunek 14-5, aby zobaczyć, jak działa szablon admin_order_details. Administrator witryny oznacza zamówienie jako zweryfikowane, gdy płatność za to zamówienie zostanie potwierdzona przez PayPal i oznacza zamówienie jako zrealizowane, gdy zamówienie jest składane, adresowane i wysyłane do kupującego. Administrator może oznaczyć zamówienie jako anulowane, jeśli na przykład PayPal nie potwierdzi płatności w rozsądnym czasie (dokładne znaczenie „rozsądne” zależy od administratora). Pozostałe przyciski — Edytuj, Aktualizuj i Anuluj — umożliwiają administratorowi ręczną edycję dowolnych szczegółów zamówienia. Po kliknięciu przycisku Edytuj pole wyboru i pola tekstowe stają się dostępne do edycji. Teraz, gdy masz już pomysł na to, co zrobi ten skomponentowany szablon, zaimplementujmy go w zwykłym stylu, zaczynając od warstwy danych.

Ćwiczenie: Administrowanie szczegółami zamówienia

1. Użyj phpMyAdmin, aby utworzyć trzy procedury składowane opisane w następujących krokach: orders_get_order_info, orders_get_order_details i orders_update_order. Nie zapomnij ustawić ogranicznika $$ przed wykonaniem kodu każdego kroku.

2. Wykonaj ten kod, który tworzy procedurę składowaną orders_get_order_info. Ta procedura składowana zwraca informacje niezbędne do wypełnienia formularza w złożonym szablonie admin_order_details, takie jak łączna kwota, data utworzenia, data wysłania itd. Możesz zobaczyć te dane na rysunku 14-6.

```
-- Create orders_get_order_info stored procedure

CREATE PROCEDURE orders_get_order_info(IN inOrderId INT)

BEGIN

SELECT order_id, total_amount, created_on, shipped_on, status,
```

comments, customer_name, shipping_address, customer_email

FROM orders

WHERE order_id = inOrderId;

END$$

3. Wykonaj następujący kod, który utworzy procedurę składowaną orders_get_order_details w Twojej bazie danych tshirtshop. Ta procedura zwraca produkty należące do określonego zamówienia. Dane te służą do wypełnienia tabeli zawierającej szczegóły zamówienia, znajdującej się na dole strony.

-- Create orders_get_order_details stored procedure

CREATE PROCEDURE orders_get_order_details(IN inOrderId INT)

BEGIN

SELECT order_id, product_id, attributes, product_name,

quantity, unit_cost, (quantity * unit_cost) AS subtotal

FROM order_detail

WHERE order_id = inOrderId;

END$$

4. Wykonaj ten kod, który tworzy procedurę składowaną orders_update_order procedura składowana. Jest to wywoływane, gdy administrator aktualizuje zamówienie w trybie edycji; aktualizuje szczegóły zamówienia.

-- Create orders_update_order stored procedure

CREATE PROCEDURE orders_update_order(IN inOrderId INT, IN inStatus INT,

IN inComments VARCHAR(255), IN inCustomerName VARCHAR(50),

IN inShippingAddress VARCHAR(255), IN inCustomerEmail VARCHAR(50))

BEGIN

DECLARE currentStatus INT;

SELECT status

FROM orders

WHERE order_id = inOrderId

INTO currentStatus;

IF inStatus != currentStatus AND (inStatus = 0 OR inStatus = 1) THEN

UPDATE orders SET shipped_on = NULL WHERE order_id = inOrderId;

ELSEIF inStatus != currentStatus AND inStatus = 2 THEN

UPDATE orders SET shipped_on = NOW() WHERE order_id = inOrderId;

```
END IF;
UPDATE orders
SET status = inStatus, comments = inComments,
customer_name = inCustomerName,
shipping_address = inShippingAddress,
customer_email = inCustomerEmail
WHERE order_id = inOrderId;
END$$
```

5. Część warstwy biznesowej dla złożonego szablonu admin_order_details jest bardzo prosta i składa się z następujących metod, które należy dodać do klasy Orders w pliku business/orders.php:

```
// Gets the details of a specific order
public static function GetOrderInfo($orderId)
{
// Build the SQL query
$sql = 'CALL orders_get_order_info(:order_id)';
// Build the parameters array
$params = array (':order_id' => $orderId);
// Execute the query and return the results
return DatabaseHandler::GetRow($sql, $params);
}
// Gets the products that belong to a specific order
public static function GetOrderDetails($orderId)
{
// Build the SQL query
$sql = 'CALL orders_get_order_details(:order_id)';
// Build the parameters array
$params = array (':order_id' => $orderId);
// Execute the query and return the results
return DatabaseHandler::GetAll($sql, $params);
}
// Updates order details
```

```
public static function UpdateOrder($orderId, $status, $comments,
$customerName, $shippingAddress, $customerEmail)
{
// Build the SQL query
$sql = 'CALL orders_update_order(:order_id, :status, :comments,
:customer_name, :shipping_address, :customer_email)';
// Build the parameters array
$params = array (':order_id' => $orderId,
':status' => $status,
':comments' => $comments,
':customer_name' => $customerName,
':shipping_address' => $shippingAddress,
':customer_email' => $customerEmail);
// Execute the query
DatabaseHandler::Execute($sql, $params);
}
```

6. Podsumujmy teraz i stwórzmy interfejs użytkownika. Warstwa prezentacji składa się z podzielonego na komponenty szablonu admin_order_details. Utwórz nowy plik szablonu o nazwie admin_order_details.tpl w folderze Presentation/Templates i dodaj do niego następujący kod:

```
{* admin_order_details.tpl *}
{load_presentation_object filename="admin_order_details" assign="obj"}
<form method="get" action="{$obj->mLinkToAdmin}">
<h3>
Editing details for order ID:
{$obj->mOrderInfo.order_id} [
<a href="{$obj->mLinkToOrdersAdmin}">back to admin orders...</a> ]
</h3>
<input type="hidden" name="Page" value="OrderDetails" />
<input type="hidden" name="OrderId"
value="{$obj->mOrderInfo.order_id}" />
<table class="borderless-table">
<tr>
```

```
<td class="bold-text">Total Amount: </td>
<td class="price">
${$obj->mOrderInfo.total_amount}
</td>
</tr>
<tr>
<td class="bold-text">Date Created: </td>
<td>
{$obj->mOrderInfo.created_on|date_format:"%Y-%m-%d %T"}
</td>
</tr>
<tr>
<td class="bold-text">Date Shipped: </td>
<td>
{$obj->mOrderInfo.shipped_on|date_format:"%Y-%m-%d %T"}
</td>
</tr>
<tr>
<td class="bold-text">Status: </td>
<td>
<select name="status"
{if ! $obj->mEditEnabled} disabled="disabled" {/if} >
{html_options options=$obj->mOrderStatusOptions
selected=$obj->mOrderInfo.status}
</select>
</td>
</tr>
<tr>
<td class="bold-text">Comments: </td>
<td>
<input name="comments" type="text" size="50"
```

```
value="{$obj->mOrderInfo.comments}"
{if ! $obj->mEditEnabled} disabled="disabled" {/if} />
<td>
</tr>
<tr>
<td class="bold-text">Customer Name: </td>
<td>
<input name="customerName" type="text" size="50"
value="{$obj->mOrderInfo.customer_name}"
{if ! $obj->mEditEnabled} disabled="disabled" {/if} />
<td>
</tr>
<tr>
<td class="bold-text">Shipping Address: </td>
<td>
<input name="shippingAddress" type="text" size="50"
value="{$obj->mOrderInfo.shipping_address}"
{if ! $obj->mEditEnabled} disabled="disabled" {/if} />
</td>
</tr>
<tr>
<td class="bold-text">Customer Email: </td>
<td>
<input name="customerEmail" type="text" size="50"
value="{$obj->mOrderInfo.customer_email}"
{if ! $obj->mEditEnabled} disabled="disabled" {/if} />
</td>
</tr>
</table>
<p>
<input type="submit" name="submitEdit" value="Edit"
```

```
{if $obj->mEditEnabled} disabled="disabled" {/if} />
<input type="submit" name="submitUpdate" value="Update"
{if ! $obj->mEditEnabled} disabled="disabled" {/if} />
<input type="submit" name="submitCancel" value="Cancel"
{if ! $obj->mEditEnabled} disabled="disabled" {/if} />
</p>
<h3>Order contains these products:</h3>
<table class="tss-table">
<tr>
<th>Product ID</th>
<th>Product Name</th>
<th>Quantity</th>
<th>Unit Cost</th>
<th>Subtotal</th>
</tr>
{section name=i loop=$obj->mOrderDetails}
<tr>
<td>{$obj->mOrderDetails[i].product_id}</td>
<td>
{$obj->mOrderDetails[i].product_name}
({$obj->mOrderDetails[i].attributes})
</td>
<td>{$obj->mOrderDetails[i].quantity}</td>
<td>${$obj->mOrderDetails[i].unit_cost}</td>
<td>${$obj->mOrderDetails[i].subtotal}</td>
</tr>
{/section}
</table>
</form>
```

7. Utwórz nowy plik o nazwie admin_order_details.php w folderze prezentacji i zapisz w nim następujący kod:

```
<?php
// Presentation tier class that deals with administering order details
class AdminOrderDetails
{
// Public variables available in smarty template
public $mOrderId;
public $mOrderInfo;
public $mOrderDetails;
public $mEditEnabled;
public $mOrderStatusOptions;
public $mLinkToAdmin;
public $mLinkToOrdersAdmin;
// Class constructor
public function __construct()
{
// Get the back link from session
$this->mLinkToOrdersAdmin = $_SESSION['link_to_orders_admin'];
$this->mLinkToAdmin = Link::ToAdmin();
// We receive the order ID in the query string
if (isset ($_GET['OrderId']))
$this->mOrderId = (int) $_GET['OrderId'];
else
trigger_error('OrderId paramater is required');
$this->mOrderStatusOptions = Orders::$mOrderStatusOptions;
}
// Initializes class members
public function init()
{
if (isset ($_GET['submitUpdate']))
{
Orders::UpdateOrder($this->mOrderId, $_GET['status'],
```

```
$_GET['comments'], $_GET['customerName'], $_GET['shippingAddress'],
$_GET['customerEmail']);
}
$this->mOrderInfo = Orders::GetOrderInfo($this->mOrderId);
$this->mOrderDetails = Orders::GetOrderDetails($this->mOrderId);
// Value which specifies whether to enable or disable edit mode
if (isset ($_GET['submitEdit']))
$this->mEditEnabled = true;
else
$this->mEditEnabled = false;
}
}
?>
```

8. Dodaj fikcyjne zlecenia do bazy danych, a następnie załaduj plik admin.php w przeglądarce. Kliknij łącze menu ADMINISTRACJA ZAMÓWIEŃ, kliknij przycisk Przejdź, aby wyświetlić niektóre zamówienia, a następnie kliknij przycisk Wyświetl szczegóły jednego z zamówień. Pojawi się strona administrowania szczegółami zamówienia, umożliwiająca edycję szczegółów zamówienia, jak wspomniano wcześniej w tym rozdziale.

Jak to działa: administrowanie szczegółami zamówienia

Dwa pliki, które właśnie napisałeś, admin_order_details.tpl i admin_order_details.php, umożliwiają przeglądanie i aktualizowanie szczegółów konkretnego zamówienia. Konstruktor klasy AdminOrderDetails (metoda __construct) zapewnia, że w ciągu zapytania znajduje się parametr OrderId, ponieważ bez niego ten skomponentowany szablon nie ma sensu:

```
// Class constructor
public function __construct()
{
// Get the back link from session
$this->mLinkToOrdersAdmin = $_SESSION['link_to_orders_admin'];
$this->mLinkToAdmin = Link::ToAdmin();
// We receive the order ID in the query string
if (isset ($_GET['OrderId']))
$this->mOrderId = (int) $_GET['OrderId'];
else
trigger_error('OrderId paramater is required');
```

```
$this->mOrderStatusOptions = Orders::$mOrderStatusOptions;
}
```

Metoda init() reaguje na działania użytkownika i wywołuje różne metody warstwy biznesowej w celu realizacji żądań użytkownika. Wypełnia formularz danymi, które otrzymuje z Orders::GetOrderInfo() i Orders::GetOrderDetails() metody warstwy biznesowej. Element klasy $mEditEnabled wchodzi lub wychodzi z trybu edycji, w zależności od tego, czy ustawiono parametr submitEdit z ciągu zapytania. Po przejściu do trybu edycji wszystkie pola tekstowe oraz przyciski Aktualizuj i Anuluj stają się aktywne, ale przycisk Edytuj jest wyłączony. Odwrotna sytuacja ma miejsce po wyjściu z trybu edycji, co dzieje się po kliknięciu przycisku Anuluj lub Aktualizuj.

Podsumowanie

W tej części przebyliśmy sporo terenu. W dwóch osobnych etapach wdrożyłeś system przyjmowania zamówień i ręcznego ich administrowania. Dodałeś przycisk Złóż zamówienie do kontrolki koszyka, aby umożliwić odwiedzającym zamawianie produktów w koszyku. Zaimplementowano prostą stronę administrowania zamówieniami, w której administrator witryny może przeglądać i obsługiwać oczekujące zamówienia. Ponieważ dane o zamówieniach są teraz przechowywane w bazie danych, możesz wykonywać różne statystyki i obliczenia na podstawie sprzedanych przedmiotów. W kolejnym rozdziale dowiesz się, jak wdrożyć funkcję dynamicznych rekomendacji produktów, co nie byłoby możliwe bez danych o zamówieniach przechowywanych w bazie danych.