Przechowywanie zamówień klientów

Aplikacja e-commerce TShirtShop ładnie się układa. Dodaliśmy możliwości zarządzania kontem klienta i śledzimy adresy klientów i informacje o kartach kredytowych, które są przechowywane w bezpieczny sposób. Jednak obecnie nie używamy tych informacji w naszym systemie śledzenia zamówień, który został utworzony w fazie II rozwoju. Obecnie nie łączymy zamówienia z kontem klienta, który złożył to zamówienie. W tym rozdziale dokonamy modyfikacji wymaganych od klientów do składania zamówień powiązanych z ich profilami użytkowników. Ta funkcja pozwoli nam śledzić w naszej bazie danych zamówienia złożone przez konkretnego klienta i położyć podwaliny pod wdrożenie potoku zamówień i transakcji kartą kredytową w kolejnych rozdziałach. Również tu przyjrzymy się radzeniu sobie z podatkami i opłatami za wysyłkę. Dostępnych jest wiele opcji implementacji funkcjonalności, aby sobie z nimi poradzić, ale po prostu przeanalizujemy prosty sposób działania i położymy podwaliny pod Twój dalszy rozwój. Ta Część jest podzielona na trzy części w następujący sposób:

• Umożliwienie klientom składania zamówień za pośrednictwem ich kont.

• Zmodyfikuj sekcję administrowania zamówieniami, aby zintegrować nowe funkcje.

• Dodaj podatek i koszty wysyłki.

W następnej zaczniemy wdrażać bardziej wyrafinowany system zamówień, a kod, który napiszemy w tym rozdziale, to ułatwi. Dlatego wprowadzimy pewne modyfikacje, które na tym etapie nie będą wydawać się konieczne, ale ułatwią ci później życie.

Dodawanie zamówień do kont klientów

Aby umożliwić klientom składanie zamówień, musimy wprowadzić kilka modyfikacji do naszego obecnego mechanizmu składania zamówień. Obecnie zamówienia, które przechowujemy w naszej bazie danych, nie są powiązane z naszymi obecnymi klientami. W tej sekcji zaktualizujemy TShirtShop, aby umożliwić naszym klientom składanie zamówień za pośrednictwem ich kont (które mogą teraz tworzyć). Najpierw zmodyfikujemy bazę danych, aby była gotowa do przechowywania informacji o zamówieniach klientów. Najpierw zmodyfikujemy tabelę zamówień, a następnie procedurę składowaną shopping_cart_create_order.

Uwaga : Nowa tabela zamówień nie będzie zgodna z danymi, które aktualnie masz w tej tabeli i będziesz musiał usunąć wszystkie istniejące dane. Jeśli dane o zamówieniach, które aktualnie masz w swojej bazie danych, są dla Ciebie ważne, przed kontynuowaniem utwórz kopię zapasową bazy danych. Dokładniej, są to zmiany, które wprowadzimy w tabeli zamówień:

• Usuń wszystkie istniejące dane.

• Usuń pola customer_name, shipping_address i customer_email.

• Dodaj pola customer_id, auth_code i referencje. Pole customer_id odwołuje się do tabeli klientów, określając klienta, który złożył zamówienie.

Zmodyfikujemy również procedurę składowaną shopping_cart_create_order, aby odzwierciedlić zmiany w tabeli zamówień. Wykonaj kroki w ćwiczeniu, aby zmienić tabelę zamówień i procedurę składowaną shopping_cart_create_order.

Ćwiczenie: Dodawanie Zamówień do Kont Klienta

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

Uwaga : Pamiętaj, aby wykonać kopię zapasową danych, ponieważ zamierzasz usunąć dane z tabel order_detail i orders.

2. Następnie użyj strony zapytania, aby wykonać ten kod, który usunie dane przechowywane w tabelach order_detail i zamówieniach z bazy danych tshirtshop.

```
-- Delete all records from order_detail table

TRUNCATE TABLE order_detail;

-- Delete all records from orders table

TRUNCATE TABLE orders;
```

3. Usuń pola customer_name, shipping_address i customer_email z tabeli zamówień; nie są już potrzebne. Te dane są teraz przechowywane w tabeli klientów.

```
-- Drop customer_name, shipping_address and customer_email fields

-- from the orders table

ALTER TABLE `orders` DROP COLUMN `customer_name`,

DROP COLUMN `shipping_address`,

DROP COLUMN `customer_email`;
```

4. Dodaj nowe pola (customer_id, auth_code i reference) i dodaj nowy indeks w polu customer_id, który odwołuje się do istniejącego klienta.

```
-- Adding the three new fields: customer_id, auth_code and reference.

ALTER TABLE `orders` ADD COLUMN `customer_id` INT,

ADD COLUMN `auth_code` VARCHAR(50),

ADD COLUMN `reference` VARCHAR(50);

-- Adding a new index to orders table

CREATE INDEX `idx_orders_customer_id` ON `orders` (`customer_id`);
```

5. Usuń starą procedurę składowaną shopping_cart_create_order i utwórz nową, wykonując następujący kod (nie zapomnij ustawić ogranicznika na $$):

```
-- Drop shopping_cart_create_order stored procedure

DROP PROCEDURE shopping_cart_create_order$$

-- Create shopping_cart_create_order stored procedure

CREATE PROCEDURE shopping_cart_create_order(IN inCartId CHAR(32),

IN inCustomerId INT)

BEGIN

DECLARE orderId INT;

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

```
INSERT INTO orders (created_on, customer_id) VALUES (NOW(), inCustomerId);
-- 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$$
```

6. Zmodyfikuj metodę CreateOrder() klasy ShoppingCart w business/shopping_cart.php w następujący sposób:

```
// Create a new order
public static function CreateOrder($customerId)
{
// Build SQL query
$sql = 'CALL shopping_cart_create_order(:cart_id, :customer_id)';
// Build the parameters array
```

```
$params = array (':cart_id' => self::GetCartId(),
':customer_id' => $customerId);
// Execute the query and return the results
return DatabaseHandler::GetOne($sql, $params);
}
```

7. Zmodyfikuj metodę init() w Presentation/checkout_info.php tak jak zaznaczono:

```
public function init()
{
// Set members for use in the Smarty template
$this->mCartItems = ShoppingCart::GetCartProducts(GET_CART_PRODUCTS);
$this->mTotalAmount = ShoppingCart::GetTotalAmount();
$this->mCustomerData = Customer::Get();
// If the Place Order button was clicked, save the order to database ...
if(isset ($_POST['place_order']))
{
// Create the order and get the order ID
$order_id = ShoppingCart::CreateOrder(Customer::GetCurrentCustomerId());
// 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;
```

8. Złóż zamówienie lub dwa za pomocą nowego systemu, aby sprawdzić, czy kod działa. Aby to zrobić, musisz się zalogować i podać wystarczającą ilość danych, aby przejść weryfikację na stronie kasy.

Uwaga : Na tym etapie strona administrowania zamówieniami nie działa. Musi zostać zmodyfikowany, aby działał z naszą nową funkcjonalnością.

Jak to działa: dodawanie zamówień klientów do TShirtShop

Kod dodany w tym ćwiczeniu jest bardzo prosty i nie zasługuje na wiele uwagi. Funkcje obsługi zamówień w warstwach danych i biznesowych przyjmują teraz jako parametr identyfikator klienta, który jest przypisany do zamówienia. Po wdrożeniu większej ilości nowego kodu zamówienia będziemy mogli dostarczać klientom więcej informacji, na przykład wysyłając im e-maile z potwierdzeniem. Na razie jednak to wszystko, co możemy zrobić

Administrowanie zamówieniami klientów

OK, w tej chwili nasza baza danych kojarzy zamówienia z ich klientami. Następnie musimy zaktualizować strony zarządzania zamówieniami, ponieważ stare już nie działają. Wiąże się to z różnymi modyfikacjami danych i warstw biznesowych w celu zapewnienia nowych struktur danych i kodu dostępu w systemie administracyjnym dla zamówień, które opracowaliśmy w rozdziale 14. Ponieważ zmiany są rozległe, zajmiemy się nimi osobno dla danych, biznesu i warstwy prezentacji.

Modyfikowanie warstwy danych

W tym miejscu musimy wprowadzić kilka zmian. Zaktualizujemy te procedury składowane z bazy danych:

• orders_get_most_recent_orders

• orders_get_orders_between_dates

• orders_get_orders_by_status

• orders_get_order_info

• orders_update_order

Stworzymy trzy nowe procedury składowane:

• orders_get_by_customer_id

• orders_get_order_short_details

• customer_get_customers_list

Ćwiczenie: Aktualizacja warstwy danych

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

2. Usuń starą procedurę składowaną orders_get_most_recent_orders i utwórz nową, wykonując ten kod:

```
-- Drop orders_get_most_recent_orders stored procedure

DROP PROCEDURE orders_get_most_recent_orders$$

-- Create orders_get_most_recent_orders stored procedure

CREATE PROCEDURE orders_get_most_recent_orders(IN inHowMany INT)

BEGIN

PREPARE statement FROM

"SELECT o.order_id, o.total_amount, o.created_on,
```

```
o.shipped_on, o.status, c.name

FROM orders o

INNER JOIN customer c

ON o.customer_id = c.customer_id

ORDER BY o.created_on DESC

LIMIT ?";

SET @p1 = inHowMany;

EXECUTE statement USING @p1;

END$$
```

3. Usuń starą procedurę składowaną orders_get_orders_between_dates i utwórz nową, wykonując ten kod:

```
-- Drop orders_get_orders_between_dates stored procedure

DROP PROCEDURE orders_get_orders_between_dates$$

-- Create orders_get_orders_between_dates stored procedure

CREATE PROCEDURE orders_get_orders_between_dates(

IN inStartDate DATETIME, IN inEndDate DATETIME)

BEGIN

SELECT o.order_id, o.total_amount, o.created_on,

o.shipped_on, o.status, c.name

FROM orders o

INNER JOIN customer c

ON o.customer_id = c.customer_id

WHERE o.created_on >= inStartDate AND o.created_on <= inEndDate

ORDER BY o.created_on DESC;

END$$
```

4. Usuń starą procedurę składowaną orders_get_orders_by_status i utwórz nową, wykonując ten kod:

```
-- Drop orders_get_orders_by_status stored procedure

DROP PROCEDURE orders_get_orders_by_status$$

-- Create orders_get_orders_by_status stored procedure

CREATE PROCEDURE orders_get_orders_by_status(IN inStatus INT)

BEGIN
```

```
SELECT o.order_id, o.total_amount, o.created_on,
o.shipped_on, o.status, c.name
FROM orders o
INNER JOIN customer c
ON o.customer_id = c.customer_id
WHERE o.status = inStatus
ORDER BY o.created_on DESC;
END$$
```

5. Usuń starą procedurę składowaną orders_get_order_info i utwórz nową, wykonując ten kod:

```
-- Drop orders_get_order_info stored procedure
DROP PROCEDURE orders_get_order_info$$
-- Create orders_get_order_info stored procedure
CREATE PROCEDURE orders_get_order_info(IN inOrderId INT)
BEGIN
SELECT order_id, total_amount, created_on, shipped_on, status,
comments, customer_id, auth_code, reference
FROM orders
WHERE order_id = inOrderId;
END$$
```

6. Usuń starą procedurę składowaną orders_update_order i utwórz nową, wykonując ten kod:

```
-- Drop orders_update_order stored procedure
DROP PROCEDURE orders_update_order$$
-- Create orders_update_order stored procedure
CREATE PROCEDURE orders_update_order(IN inOrderId INT, IN inStatus INT,
IN inComments VARCHAR(255), IN inAuthCode VARCHAR(50),
IN inReference 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,
auth_code = inAuthCode, reference = inReference
WHERE order_id = inOrderId;
END$$
```

7. Wykonaj ten kod, który tworzy procedurę składowaną orders_get_orders_by_customer_id:

```
-- Create orders_get_by_customer_id stored procedure
CREATE PROCEDURE orders_get_by_customer_id(IN inCustomerId INT)
BEGIN
SELECT o.order_id, o.total_amount, o.created_on,
o.shipped_on, o.status, c.name
FROM orders o
INNER JOIN customer c
ON o.customer_id = c.customer_id
WHERE o.customer_id = inCustomerId
ORDER BY o.created_on DESC;
END$$
```

8. Wykonaj następujący kod, który utworzy procedurę składowaną orders_get_order_short_details:

```
-- Create orders_get_order_short_details stored procedure
CREATE PROCEDURE orders_get_order_short_details(IN inOrderId INT)
BEGIN
SELECT o.order_id, o.total_amount, o.created_on,
o.shipped_on, o.status, c.name
FROM orders o
INNER JOIN customer c
```

```
ON o.customer_id = c.customer_id
WHERE o.order_id = inOrderId;
END$$
```

9. Wykonaj ten kod, który tworzy procedurę składowaną customer_get_customers_list:

```
-- Create customer_get_customers_list stored procedure
CREATE PROCEDURE customer_get_customers_list()
BEGIN
SELECT customer_id, name FROM customer ORDER BY name ASC;
END$$
```

Modyfikowanie warstwy biznesowej

Musimy również wprowadzić kilka zmian w warstwie biznesowej. Musimy zmodyfikować metodę UpdateOrder() klasy Orders i dodać trzy nowe metody do klas Orders i Customers:

- PobierzIdentyfikatorKlienta()
- GetOrderShortDetails()
- PobierzListęKlientów()

Te nowe metody obsługują nowe funkcje administracyjne, których będziemy potrzebować w szablonie warstwy prezentacji admin_orders.tpl. Utwórz je, wykonując kroki ćwiczenia.

Ćwiczenie: Aktualizacja warstwy biznesowej

1. Dodaj nową metodę o nazwie GetByCustomerId() do klasy Orders w business/orders.php:

```
// Gets all orders placed by a specified customer
public static function GetByCustomerId($customerId)
{
// Build the SQL query
$sql = 'CALL orders_get_by_customer_id(:customer_id)';
// Build the parameters array
$params = array (':customer_id' => $customerId);
// Execute the query and return the results
return DatabaseHandler::GetAll($sql, $params);
}
```

2. Dodaj nową metodę o nazwie GetOrderShortDetails() do klasy Orders w business/orders.php:

```
// Get short details for an order
public static function GetOrderShortDetails($orderId)
```

```
{
// Build the SQL query
$sql = 'CALL orders_get_order_short_details(:order_id)';
// Build the parameters array
$params = array (':order_id' => $orderId);
// Execute the query and return the results
return DatabaseHandler::GetAll($sql, $params);
}
```

3. Zmodyfikuj metodę UpdateOrder() klasy Orders w następujący sposób:

```
// Updates order details
public static function UpdateOrder($orderId, $status, $comments,
$authCode, $reference)
{
// Build the SQL query
$sql = 'CALL orders_update_order(:order_id, :status, :comments,
:auth_code, :reference)';
// Build the parameters array
$params = array (':order_id' => $orderId,
':status' => $status,
':comments' => $comments,
':auth_code' => $authCode,
':reference' => $reference);
// Execute the query
DatabaseHandler::Execute($sql, $params);
}
```

4. Dodaj nową metodę o nazwie GetCustomersList() do klasy Customer w business/customer.php:

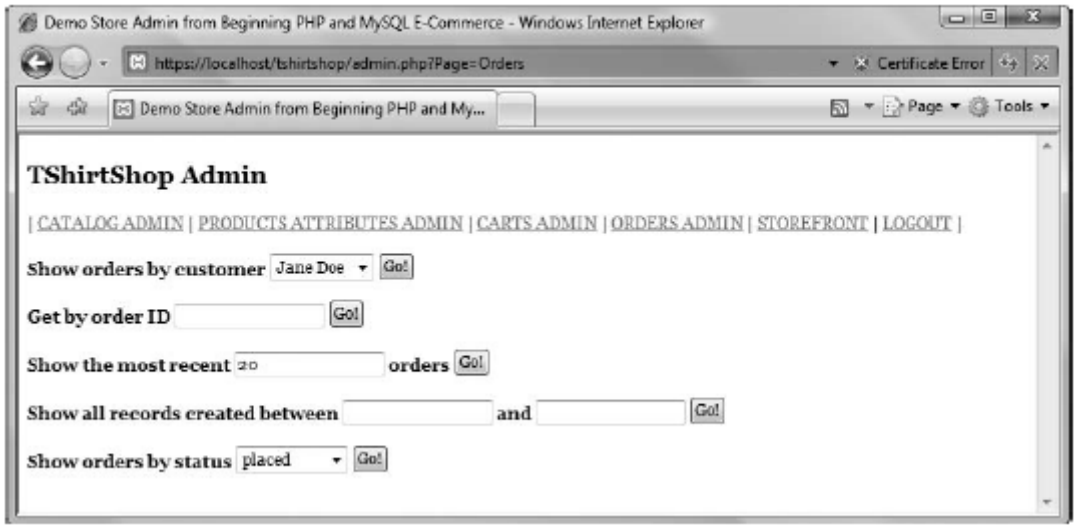
```
// Gets all customers names with their associated id
public static function GetCustomersList()
{
// Build the SQL query
$sql = 'CALL customer_get_customers_list()';
```

```
// Execute the query and return the results
return DatabaseHandler::GetAll($sql);
}
```

Modyfikowanie poziomu prezentacji

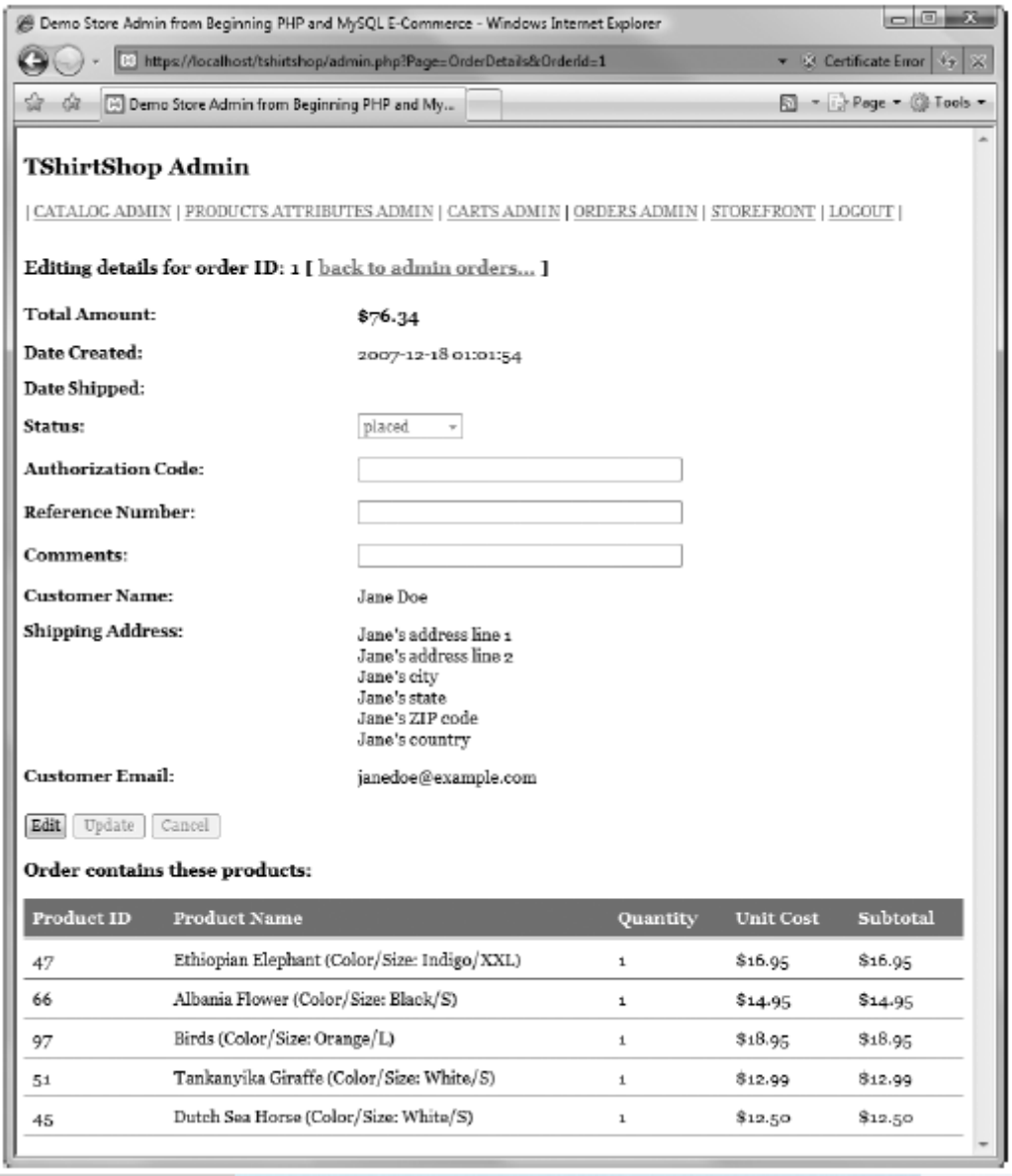
Teraz musimy zaktualizować warstwę prezentacji, aby korzystać z nowych funkcji warstwy danych i warstwy biznesowej. Na tym etapie nie zamierzamy wprowadzać ogromnych zmian w kodzie zarządzania zamówieniami, ponieważ po prostu zmodyfikujemy go później, po zakończeniu nowego systemu przetwarzania zamówień.

Rysunek przedstawia szablon admin_orders. Ta strona daje administratorom różne możliwości filtrowania bieżących zamówień.



Bez względu na to, jakiej metody selekcji użyjemy, otrzymamy listę z zamówieniami spełniającymi kryteria. Na rysunku widzimy zamówienie, które właśnie złożyła Jane.

Szablon admin_order_details wygląda jak ten pokazany na rysunku. Zwróć uwagę na dane dotyczące podatku i wysyłki, które dodamy w dalszej części tego rozdziału.



Ćwiczenie: Aktualizacja poziomu prezentacji

1. Dodaj podświetlony kod do prezentacji/templates/admin_orders.tpl:

```
{* 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 orders by customer</font>
```

```
<select name="customer_id">
{section name=i loop=$obj->mCustomers}
<option value="{$obj->mCustomers[i].customer_id}"
{if $obj->mCustomers[i].customer_id == $obj->mCustomerId}
selected="selected"
{/if}>
{$obj->mCustomers[i].name}
</option>
{/section}
</select>
<input type="submit" name="submitByCustomer" value="Go!" />
</p>
<p>
<font class="bold-text">Get by order ID</font>
<input name="orderId" type="text" value="{$obj->mOrderId}" />
<input type="submit" name="submitByOrderId" value="Go!" />
</p>
<p>
<font class="bold-text">Show the most recent</font>
<input name="recordCount" type="text" value="{$obj->mRecordCount}" />
...
{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].name}</td>
<td align="right">
<a href="{$obj->mOrders[i].link_to_order_details_admin}">View Details</a>
```

```
</td>
</tr>
{/section}
</table>
{/if}
```

2. Dodaj podświetlonych członków do klasy AdminOrders w Presentation/admin_orders.php:

```
public $mLinkToAdmin;
public $mCustomers;
public $mCustomerId;
public $mOrderId;
```

3. Dodaj podświetlony kod do metody init() klasy AdminOrders w Presentation/admin_orders.php:

```
// If "Show orders by status" filter is in action ...
if (isset ($_GET['submitOrdersByStatus']))
{
$this->mSelectedStatus = $_GET['status'];
$this->mOrders = Orders::GetOrdersByStatus($this->mSelectedStatus);
}
// If the "Show orders by customer ID" filter is in action ...
if (isset ($_GET['submitByCustomer']))
{
if (empty ($_GET['customer_id']))
$this->mErrorMessage = 'No customer has been selected';
else
{
$this->mCustomerId = $_GET['customer_id'];
$this->mOrders = Orders::GetByCustomerId($this->mCustomerId);
}
}
// If the "Get order by ID" filter is in action ...
if (isset ($_GET['submitByOrderId']))
{
```

```
if (empty ($_GET['orderId']))
$this->mErrorMessage = 'You must enter an order ID.';
else
{
$this->mOrderId = $_GET['orderId'];
$this->mOrders = Orders::GetOrderShortDetails($this->mOrderId);
}
}
$this->mCustomers = Customer::GetCustomersList();
if (is_array($this->mOrders) && count($this->mOrders) == 0)
$this->mErrorMessage =
'No orders found matching your searching criteria!';
```

4. Dodaj nowego członka do klasy AdminOrderDetails w Presentation/admin_order_details.php:

```
public $mLinkToAdmin;
public $mLinkToOrdersAdmin;
public $mCustomerInfo;
```

5. Zmodyfikuj wiersz, który aktualizuje zamówienie w metodzie init() AdminOrderDetails, tak jak zaznaczono:

```
// Initializes class members
public function init()
{
if (isset ($_GET['submitUpdate']))
{
Orders::UpdateOrder($this->mOrderId, $_GET['status'],
$_GET['comments'], $_GET['authCode'], $_GET['reference']);
}
```

6. Również w metodzie init() klasy AdminOrderDetails dodaj wiersz odczytujący dane klienta, który złożył zamówienie:

```
$this->mOrderInfo = Orders::GetOrderInfo($this->mOrderId);
$this->mOrderDetails = Orders::GetOrderDetails($this->mOrderId);
$this->mCustomerInfo = Customer::Get($this->mOrderInfo['customer_id']);
// Value which specifies whether to enable or disable edit mode
```

if (isset ($_GET['submitEdit']))

7. Zmodyfikuj prezentację/templates/admin_order_details.tpl zgodnie z zaznaczeniem:

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">Authorization Code: </td>

<td>

<input name="authCode" type="text" size="50"

value="{$obj->mOrderInfo.auth_code}"

{if ! $obj->mEditEnabled} disabled="disabled" {/if} />

<td>

</tr>

<tr>

<td class="bold-text">Reference Number: </td>

<td>

<input name="reference" type="text" size="50"

value="{$obj->mOrderInfo.reference}"

{if ! $obj->mEditEnabled} disabled="disabled" {/if} />

<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>{$obj->mCustomerInfo.name}</td>

</tr>

<tr>

<td class="bold-text" valign="top">Shipping Address: </td>

<td>

**{$obj->mCustomerInfo.address_1}
**

{if $obj->mCustomerInfo.address_2}

**{$obj->mCustomerInfo.address_2}
**

{/if}

**{$obj->mCustomerInfo.city}
**

**{$obj->mCustomerInfo.region}
**

**{$obj->mCustomerInfo.postal_code}
**

**{$obj->mCustomerInfo.country}
**

</td>

</tr>

<tr>

<td class="bold-text">Customer Email: </td>

<td>{$obj->mCustomerInfo.email}</td>

</tr>

</table>

8. Zaktualizuj admin.php, dodając odniesienie do symetrycznej krypty, bezpiecznej karty i klas warstwy biznesowej klienta:

require_once BUSINESS_DIR . 'shopping_cart.php';

require_once BUSINESS_DIR . 'orders.php';

require_once BUSINESS_DIR . 'symmetric_crypt.php';

require_once BUSINESS_DIR . 'secure_card.php';

require_once BUSINESS_DIR . 'customer.php';

9. Załaduj stronę internetową, aby upewnić się, że nasz nowo dodany kod działa tak, jak pokazano na rysunkach powyżej.

Jak to działa: zmiany poziomu prezentacji

To było długie ćwiczenie, a jednak, aby jak najlepiej je wykorzystać, nadal musimy przejść przez kilka dodatkowych ćwiczeń, aby wprowadzić opłaty podatkowe i koszty wysyłki. W tym momencie cała nasza funkcjonalność obsługi klienta zostanie zakończona, a my pozostaniemy z dodawaniem tylko potoku zamówień i obsługi kart kredytowych. Przejdźmy teraz do dodania obsługi podatków i opłat za wysyłkę.

Obsługa podatków i opłat za wysyłkę

Jedną z cech wspólnych dla wielu witryn e-commerce jest konieczność naliczania opłat związanych z podatkiem i wysyłką. Oczywiście nie zawsze tak jest – cyfrowe strony pobierania nie muszą na przykład pobierać opłat za wysyłkę, ponieważ nie jest to związane z fizyczną wysyłką. Jednak prawdopodobnie będziemy chcieli uwzględnić w naszych zamówieniach dodatkowe opłaty tego czy innego rodzaju. W rzeczywistości dodanie tej funkcjonalności może być bardzo proste – lub nie – w zależności od tego, jak skomplikowane chcemy zrobić. W tym rozdziale utrzymamy prostotę i zapewnimy podstawową, ale rozszerzalną funkcjonalność do obliczania zarówno podatków, jak i opłat za wysyłkę. Najpierw omówmy problemy.

Kwestie podatkowe

Tematyka serwisów podatkowych i e-commerce ma skomplikowaną historię. Na początku sprzedawcom internetowym zwykle ujdzie na sucho wszystko. Opodatkowanie było słabo egzekwowane, a wiele witryn po prostu całkowicie je ignorowało. Dotyczyło to zwłaszcza zamówień międzynarodowych, gdzie klienci często mogli uniknąć płacenia podatku przez większość czasu – chyba że zamówienia zostały przechwycone przez celników! Kiedy coraz więcej osób zaczęło dowiadywać się o witrynach e-commerce, organy podatkowe, takie jak IRS, zdały sobie sprawę, że tracą dużo pieniędzy – a przynajmniej nie otrzymują wszystkiego, co mogli – i to ich denerwuje. Nastąpiła lawina działań, gdy różne organizacje na całym świecie próbowały podłączyć się do tego strumienia przychodów. Zaproponowano i wdrożono szereg rozwiązań z mieszanymi rezultatami. Teraz sprawy stają się nieco bardziej uregulowane. Kluczową koncepcją, o której należy pamiętać, myśląc o podatkach, jest związek, czyli biznes, który ma wystarczająco dużą obecność w jurysdykcji podatkowej, aby zagwarantować pobór podatków. W praktyce oznacza to, że w przypadku wysyłki międzynarodowej w większości sytuacji możesz nie ponosić odpowiedzialności za to, co dzieje się pod względem podatkowym, chyba że Twoja firma ma znaczącą obecność w kraju docelowym. W przypadku wysyłki wewnętrznej w kraju (lub powiedzmy w Unii Europejskiej), prawdopodobnie będziesz odpowiedzialny. Ustawodawstwo jest trochę niejasne i na pewno nie zbadaliśmy przepisów dla każdego kraju na świecie, ale ta ogólna zasada jest słuszna. Pozostałe kluczowe kwestie można podsumować następująco:

• Opodatkowanie zależy od tego, skąd i dokąd wysyłasz przesyłkę.

• Obowiązują przepisy krajowe.

• Ważny jest rodzaj sprzedawanego produktu.

W niektórych krajach jest łatwiej niż w innych. Na przykład w Wielkiej Brytanii możemy naliczać aktualną stawkę podatku od wartości dodanej (VAT) od wszystkich zakupów, w których ma ona zastosowanie (niektóre rodzaje produktów są zwolnione lub objęte obniżoną stawką) i być względnie zadowolonym, że zrobiliśmy wszystko możemy. Jeśli chcemy pójść o krok dalej, możemy rozważyć wysyłkę naszych towarów za granicą (Amazon to robi, więc dlaczego nie mielibyśmy tego zrobić?). Stany Zjednoczone (i inne kraje) mają do czynienia ze znacznie bardziej złożonym systemem. W Stanach Zjednoczonych podatki od sprzedaży różnią się nie tylko w zależności od stanu, ale często również w obrębie stanów. W rzeczywistości prawie tylko wtedy, gdy większość sprzedawców dokładnie wie, co robić, jest wysyłanie towarów do klienta w tym samym obszarze podatkowym, w którym znajduje się ich firma. Innym razem, cóż, szczerze mówiąc, twoje przypuszczenia są równie dobre jak nasze.

Dodany przez nas schemat opodatkowania jest tak prosty, jak to tylko możliwe. Tabela bazy danych będzie zawierała informacje dotyczące różnych stawek podatkowych, które można zastosować, a ich wybór będzie na razie zależał od regionu wysyłki klienta. Wszystkie produkty są uważane za podlegające opodatkowaniu według tej samej stawki. To pozostawia wiele do życzenia, ale przynajmniej podatek zostanie obliczony i zastosowany. Możesz go później zastąpić własnym, konkretnym systemem.

Problemy z wysyłką

Dostawa jest nieco prostsza niż podatek, chociaż znowu możemy wszystko tak skomplikować, jak tylko chcemy. Ponieważ wysyłanie zamówień od firmy, która handluje za pośrednictwem interfejsu e-commerce, jest bardzo podobne do wysyłania zamówień, powiedzmy, od firmy zajmującej się sprzedażą wysyłkową, praktyki są wdrożone i stosunkowo łatwe do zrozumienia. Do naszej dyspozycji mogą być nowe sposoby robienia rzeczy, ale ogólne zasady są dobrze znane. Możesz mieć związek z usługą pocztową od czasów handlu przed Internetem, w takim przypadku prawdopodobnie najłatwiej jest trzymać rzeczy tak blisko starego sposobu robienia rzeczy, jak to tylko możliwe. Jeśli jednak dopiero zaczynasz lub zmieniasz sposób, w jaki robisz rzeczy, masz wiele opcji do rozważenia. Najprostszą opcją jest w ogóle nie martwić się o koszty wysyłki, co ma sens, jeśli nie ma kosztów, na przykład w przypadku pobrań cyfrowych. Alternatywnie moglibyśmy po prostu uwzględnić koszt wysyłki w kosztach naszych produktów. Albo możemy nałożyć zryczałtowaną opłatę niezależnie od zamówionych przedmiotów lub miejsca przeznaczenia. Jednak niektóre z tych opcji mogą wiązać się z przepłacaniem lub niedopłacaniem przez klientów, co nie jest idealne. Drugą skrajnością jest uwzględnienie wagi i wymiarów wszystkich zamówionych produktów oraz obliczenie dokładnego kosztu. Można to nieco uprościć, ponieważ niektóre firmy przewozowe (m.in. FedEx) udostępniają przydatne interfejsy API, które mogą nam pomóc. W niektórych przypadkach możemy użyć dynamicznego systemu do obliczenia dostępnych opcji wysyłki (na noc, od trzech do czterech dni itd.) na podstawie wielu czynników, w tym wagi paczki i miejsca dostawy. Dokładne metody realizacji tego zadania mogą się jednak bardzo różnić w zależności od firm spedycyjnych, a wdrożenie takiego rozwiązania pozostawiamy Tobie, jeśli tego potrzebujesz. W tej książce ponownie przyjrzymy się prostej linii. Dla każdego regionu wysyłki w bazie danych udostępnimy użytkownikowi szereg opcji wysyłki, z których każdy będzie miał powiązany koszt. Koszt ten jest po prostu dodawany do kosztu zamówienia. To jest powód, dla którego w rozdziale 16 zamieściliśmy tabelę shipping_region - jej użycie wkrótce stanie się oczywiste.

Wdrażanie podatków i opłat za wysyłkę

Zgodnie z oczekiwaniami musimy wprowadzić kilka modyfikacji w TShirtShop, aby umożliwić opisane wcześniej schematy podatkowe i wysyłkowe. Mamy jeszcze dwie tabele bazy danych do dodania,

podatki i wysyłkę, a także modyfikacje do wprowadzenia w tabeli zamówień. Będziemy musieli dodać nowe funkcje bazy danych i dokonać pewnych modyfikacji w już istniejących. Niektóre klasy warstwy biznesowej wymagają modyfikacji w celu uwzględnienia tych zmian, a warstwa prezentacji musi zawierać metodę wyboru metody wysyłki przez użytkowników (schemat opodatkowania jest wybierany automatycznie). W idealnym przypadku wyczyścilibyśmy zawartość tabel zamówień i order_detail lub uwzględnili fakt, że stare zamówienia nie zawierają podatku i kosztów wysyłki, dla których łączna kwota zostanie obliczona na 0,00 zł. Więc zacznijmy.

Modyfikowanie warstwy danych

W tej sekcji dodamy nowe tabele i zmodyfikujemy tabelę zamówień oraz dodamy lub zmodyfikujemy stare procedury składowane w bazie danych.

Ćwiczenie: Tworzenie struktur bazy danych

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

2. Wykonaj ten kod, który doda tabelę wysyłkową do bazy danych tshirtshop:

```
-- Create shipping table

CREATE TABLE `shipping` (

`shipping_id` INT NOT NULL AUTO_INCREMENT,

`shipping_type` VARCHAR(100) NOT NULL,

`shipping_cost` NUMERIC(10, 2) NOT NULL,

`shipping_region_id` INT NOT NULL,

PRIMARY KEY (`shipping_id`),

KEY `idx_shipping_shipping_region_id` (`shipping_region_id`)

);
```

3. Wykonaj następujący kod, który wypełni tabelę wysyłki z bazy danych tshirtshop:

```
-- Populate shipping table

INSERT INTO `shipping` (`shipping_id`, `shipping_type`,

`shipping_cost`, `shipping_region_id`) VALUES

(1, 'Next Day Delivery ($20)', 20.00, 2),

(2, '3-4 Days ($10)', 10.00, 2),

(3, '7 Days ($5)', 5.00, 2),

(4, 'By air (7 days, $25)', 25.00, 3),

(5, 'By sea (28 days, $10)', 10.00, 3),

(6, 'By air (10 days, $35)', 35.00, 4),

(7, 'By sea (28 days, $30)', 30.00, 4);
```

4. Wykonaj ten kod, który doda tabelę podatkową do bazy danych tshirtshop:

```
-- Create tax table

CREATE TABLE `tax` (

`tax_id` INT NOT NULL AUTO_INCREMENT,

`tax_type` VARCHAR(100) NOT NULL,

`tax_percentage` NUMERIC(10, 2) NOT NULL,

PRIMARY KEY (`tax_id`)

);
```

5. Wykonaj następujący kod, który wypełni tabelę podatkową z bazy danych tshirtshop:

```
-- Populate tax table

INSERT INTO `tax` (`tax_id`, `tax_type`, `tax_percentage`) VALUES

(1, 'Sales Tax at 8.5%', 8.50),

(2, 'No Tax', 0.00);
```

6. Wykonaj ten kod, który doda kolumnę shipping_id i nowy indeks do tabeli zamówień z bazy danych tshirtshop:

```
-- Adding a new field named shipping_id to orders table

ALTER TABLE `orders` ADD COLUMN `shipping_id` INT;

-- Adding a new index to orders table

CREATE INDEX `idx_orders_shipping_id` ON `orders` (`shipping_id`);
```

7. Wykonaj poniższy kod, który doda kolumnę tax_id oraz nowy indeks do tabeli orders z bazy danych tshirtshop:

```
-- Adding a new field named tax_id to orders table

ALTER TABLE orders ADD COLUMN tax_id INT;

-- Adding a new index to orders table

CREATE INDEX `idx_orders_tax_id` ON `orders` (`tax_id`);
```

8. Usuń obecną procedurę składowaną shopping_cart_create_order i utwórz nową uwzględniającą nowe zmiany wprowadzone w tabeli zamówień (nie zapomnij ustawić ogranicznika na $$):

```
-- Drop shopping_cart_create_order stored procedure

DROP PROCEDURE shopping_cart_create_order$$

-- Create shopping_cart_create_order stored procedure

CREATE PROCEDURE shopping_cart_create_order(IN inCartId CHAR(32),

IN inCustomerId INT, IN inShippingId INT, IN inTaxId INT)

BEGIN
```

```
DECLARE orderId INT;
-- Insert a new record into orders and obtain the new order ID
INSERT INTO orders (created_on, customer_id, shipping_id, tax_id) VALUES
(NOW(), inCustomerId, inShippingId, inTaxId);
-- 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$$
```

9. Zmodyfikuj procedurę składowaną orders_get_order_info usuwając starą wersję i tworząc nową (nie zapomnij ustawić ogranicznika na $$):

```
-- Drop orders_get_order_info stored procedure
DROP PROCEDURE orders_get_order_info$$
-- Create orders_get_order_info stored procedure
```

```
CREATE PROCEDURE orders_get_order_info(IN inOrderId INT)
BEGIN
SELECT o.order_id, o.total_amount, o.created_on, o.shipped_on,
o.status, o.comments, o.customer_id, o.auth_code,
o.reference, o.shipping_id, s.shipping_type, s.shipping_cost,
o.tax_id, t.tax_type, t.tax_percentage
FROM orders o
INNER JOIN tax t
ON t.tax_id = o.tax_id
INNER JOIN shipping s
ON s.shipping_id = o.shipping_id
WHERE o.order_id = inOrderId;
END$$
```

10. Wykonaj ten kod, który dodaje procedurę składowaną orders_get_shipping_info do bazy danych tshirtshop:

```
-- Create orders_get_shipping_info stored procedure
CREATE PROCEDURE orders_get_shipping_info(IN inShippingRegionId INT)
BEGIN
SELECT shipping_id, shipping_type, shipping_cost, shipping_region_id
FROM shipping
WHERE shipping_region_id = inShippingRegionId;
END$$
```

Modyfikowanie warstwy biznesowej

Aby pracować z nowymi tabelami bazy danych i procedurami składowanymi, musimy wprowadzić kilka zmian w business/shopping_cart.php. Musimy zmodyfikować CreateOrder() w ShoppingCart, aby skonfigurować podatek i wysyłkę również dla nowych zamówień.

Ćwiczenie: Aktualizacja warstwy biznesowej

1. Zmodyfikuj metodę CreateOrder() w business/shopping_cart.php w następujący sposób:

```
// Create a new order
public static function CreateOrder($customerId, $shippingId, $taxId)
{
// Build SQL query
```

```
$sql = 'CALL shopping_cart_create_order(:cart_id, :customer_id,
:shipping_id, :tax_id)';
// Build the parameters array
$params = array (':cart_id' => self::GetCartId(),
':customer_id' => $customerId,
':shipping_id' => $shippingId,
':tax_id' => $taxId);
// Execute the query and return the results
return DatabaseHandler::GetOne($sql, $params);
}
```

2. Dodaj metodę GetShippingInfo() do klasy Orders w business/orders.php:

```
// Retrieves the shipping details for a given $shippingRegionId
public static function GetShippingInfo($shippingRegionId)
{
// Build the SQL query
$sql = 'CALL orders_get_shipping_info(:shipping_region_id)';
// Build the parameters array
$params = array (':shipping_region_id' => $shippingRegionId);
// Execute the query and return the results
return DatabaseHandler::GetAll($sql, $params);
}
```

Modyfikowanie poziomu prezentacji

Wreszcie dochodzimy do warstwy prezentacji. W rzeczywistości, ze względu na wprowadzone przez nas zmiany, jedyne zmiany, które należy wprowadzić, dotyczą strony kasy i zarządzania zamówieniami.

Ćwiczenie: Aktualizacja poziomu prezentacji

1. Zmodyfikuj prezentację/szablony/checkout_info.tpl zgodnie z zaznaczeniem:

```
Shipping region: {$obj->mShippingRegion}
</p>
{/if}
```

```
{if $obj->mNoCreditCard!= 'yes' && $obj->mNoShippingAddress != 'yes'}
<p>
```

Shipping type:

<select name="shipping">

{section name=i loop=$obj->mShippingInfo}

<option value="{$obj->mShippingInfo[i].shipping_id}">

{$obj->mShippingInfo[i].shipping_type}

</option>

{/section}

</select>

</p>

{/if}

<input type="submit" name="place_order" value="Place Order"

{$obj->mOrderButtonVisible} /> |

mLinkToCart}">Edit Shopping Cart |

2. Dodaj nowego członka do klasy CheckoutInfo w Presentation/checkout_info.php w następujący sposób:

public $mLinkToCart;

public $mLinkToContinueShopping;

public $mShippingInfo;

3. Zmodyfikuj metodę init() w klasie CheckoutInfo w pliku Presentation/checkout_info.php:

// If the Place Order button was clicked, save the order to database ...

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

{

$this->mCustomerData = Customer::Get();

$tax_id = '';

switch ($this->mCustomerData['shipping_region_id'])

{

case 2:

$tax_id = 1;

break;

default:

$tax_id = 2;

```
}
// Create the order and get the order ID
$order_id = ShoppingCart::CreateOrder(
$this->mCustomerData['customer_id'],
(int)$_POST['shipping'], $tax_id);
// 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;
```

4. W ten sam sposób dodaj następujący kod:

```
foreach ($shipping_regions as $item)
if ($item['shipping_region_id'] ==
$this->mCustomerData['shipping_region_id'])
$this->mShippingRegion = $item['shipping_region'];
if ($this->mNoCreditCard == 'no' && $this->mNoShippingAddress == 'no')
{
$this->mShippingInfo = Orders::GetShippingInfo(
$this->mCustomerData['shipping_region_id']);
}
}
}
}
?>
```

5. Zaktualizuj index.php, dodając odniesienie do klasy biznesowej zamówień, jak pokazano tutaj:

```
require_once BUSINESS_DIR . 'secure_card.php';
require_once BUSINESS_DIR . 'customer.php';
```

require_once BUSINESS_DIR . 'orders.php';

6. Kontynuuj modyfikowanie klasy AdminOrderDetails z pliku Presentation/admin_order_details.php, dodając dwa elementy:

public $mLinkToOrdersAdmin;

public $mCustomerInfo;

public $mTotalCost;

public $mTax = 0.0;

7. Dodaj te wiersze do klasy AdminOrderDetails w metodzie init():

$this->mOrderInfo = Orders::GetOrderInfo($this->mOrderId);

$this->mOrderDetails = Orders::GetOrderDetails($this->mOrderId);

$this->mCustomerInfo = Customer::Get($this->mOrderInfo['customer_id']);

$this->mTotalCost = $this->mOrderInfo['total_amount'];

if ($this->mOrderInfo['tax_percentage'] !== 0.0)

$this->mTax = round((float)$this->mTotalCost *

(float)$this->mOrderInfo['tax_percentage'], 2)

/ 100.00;

$this->mTotalCost += $this->mOrderInfo['shipping_cost'];

$this->mTotalCost += $this->mTax;

// Format the values

$this->mTotalCost = number_format($this->mTotalCost, 2, '.', '');

$this->mTax = number_format($this->mTax, 2, '.', '');

// Value which specifies whether to enable or disable edit mode

if (isset ($_GET['submitEdit']))

$this->mEditEnabled = true;

8. Zmodyfikuj szablon prezentacji/templates/admin_order_details.tpl zgodnie z podświetleniem:

<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">Tax: </td>
<td class="price">{$obj->mOrderInfo.tax_type} ${$obj->mTax}</td>
</tr>
<tr>
<td class="bold-text">Shipping: </td>
<td class="price">{$obj->mOrderInfo.shipping_type}</td>
</tr>
<tr>
<td class="bold-text">Date Created: </td>
<td>
{$obj->mOrderInfo.created_on|date_format:"%Y-%m-%d %T"}
</td>
</tr>
```

Jak to działa: rozwiązywanie problemów związanych z podatkami i wysyłką

Zauważ, że jest to jeden z najważniejszych fragmentów kodu. Jest to kod, w którym najprawdopodobniej dokonasz modyfikacji systemu podatkowego i wysyłkowego, jeśli zdecydujesz się wdrożyć własny system. Zmiany w bazie danych i warstwie biznesowej są znacznie bardziej ogólne – choć nie oznacza to, że takie modyfikacje nie byłyby konieczne. Przed sprawdzeniem, czy nowy system działa pod kątem podatków i opłat za wysyłkę, skorzystaj ze strony zarządzania zamówieniami, aby sprawdzić, czy nie ma to wpływu na stare zamówienia. Informacje pobrane dla starego zamówienia powinny pozostać nienaruszone, ponieważ dane pozostają niezmienione. Złóż nowe zamówienie, najlepiej dla klienta w regionie wysyłki w Stanach Zjednoczonych/Kanadzie (ponieważ jest to obecnie jedyny region, w którym naliczany jest podatek). Zauważ, że na stronie kasy musisz wybrać opcję wysyłki. Po złożeniu zamówienia sprawdź nowe zamówienie w bazie danych. Wynik powinien wyglądać jak strona pokazana na rysunku 3. W tej Części, prowadząc do tego przykładu, właściwie zbadaliśmy, jak działają podatki i opłaty za wysyłkę, ale podsumujmy. Najpierw klient musi wybrać region wysyłki dla swojego adresu. Bez wybrania tego regionu wysyłki odwiedzający nie mogą składać zamówień, ponieważ nie mogą wybrać opcji wysyłki. Gdy gość składa zamówienie, wybrany region wysyłki jest dołączany do zamówienia w tabeli zamówień. Dołączony jest również wymóg podatkowy dla zamówienia, chociaż nie wymaga to wprowadzania danych przez użytkownika i jest obecnie wybierany za pomocą bardzo prostego algorytmu.

Dalszy rozwój

Z tego miejsca można przejść na kilka sposobów. Być może pierwszym może być dodanie systemu administracyjnego dla opcji podatkowych i wysyłki. Nie zostało to tutaj zaimplementowane, częściowo dlatego, że byłoby to trywialne, biorąc pod uwagę doświadczenie, które miałeś do tej pory w tej książce, a częściowo dlatego, że przedstawione tutaj techniki są bardziej szablonem dla rozwoju niż w pełni rozwiniętym sposobem robienia rzeczy – tam jest tak wiele opcji do wyboru zarówno dla kalkulacji podatku, jak i kosztów wysyłki, że tutaj omówione są tylko podstawy. Atrakcyjną opcją jest podłączenie się do usług online w celu obliczenia podatku i kosztów wysyłki; w przypadku usług spedycyjnych jest to bardzo możliwe. W rzeczywistości usługi oferowane przez firmy wysyłkowe, takie jak FedEx, wykorzystują proces podobny do firm obsługujących karty kredytowe, które omówimy w dalszej części tej książki. Większość kodu, który musiałbyś napisać, aby uzyskać dostęp do usług wysyłkowych, będzie bardzo podobna do kodu do przetwarzania kart kredytowych, chociaż oczywiście będziesz musiał go dostosować, aby uzyskać właściwe szczegóły. W Twoim przypadku mogą być wymagane większe zmiany, takie jak dodanie wagi i wymiarów do produktów, ale to bardzo zależy od tego, jakie produkty sprzedajesz.

Podsumowanie

Rozszerzyliśmy witrynę TShirtShop, aby umożliwić klientom składanie zamówień przy użyciu wszystkich nowych danych i technik wprowadzonych w Części poprzedniej. Wiele modyfikacji wprowadzonych w tym rozdziale stanowi podstawę do wykorzystania potoku zamówień w pozostałych tej książki. Dodaliśmy również szybki sposób sprawdzania zamówień klientów, chociaż w żadnym wypadku nie jest to w pełni rozwinięte narzędzie administracyjne – które pojawi się później. Wdrożyliśmy również prosty system dodawania podatku i kosztów wysyłki do zamówień. Ten system nie jest uniwersalnym rozwiązaniem, ale działa i jest prosty. Co ważniejsze, techniki te można łatwo rozbudować, aby wprowadzić bardziej złożone algorytmy i interakcję użytkownika w celu odpowiedniego wyboru opcji podatku i wysyłki oraz zamówień cenowych. Od następnego rozdziału będziemy jeszcze bardziej rozwijać system zamówień klientów, rozpoczynając opracowywanie profesjonalnego potoku zamówień do przetwarzania zamówień.