Przetwarzanie transakcji kartą kredytową

Ostatnią rzeczą, jaką musimy zrobić przed uruchomieniem witryny e-commerce, jest umożliwienie przetwarzania kart kredytowych. W tej Części zbadamy, jak możemy wbudować to w potok, który stworzyliśmy w poprzedniej Części. Zaczniemy od przyjrzenia się teorii stojącej za transakcjami kartami kredytowymi, rodzajem organizacji, które pomagają w przetwarzaniu kart kredytowych oraz rodzajem transakcji, które są możliwe. Idąc dalej, weźmiemy dwie przykładowe organizacje, DataCash i Authorize.net, i omówimy specyfikę ich interfejsów programu aplikacji transakcyjnych (API), w jaki sposób wykorzystujemy funkcje transakcji kartą kredytową. Następnie zbudujemy nową bibliotekę klas, która pomoże korzystać z jednego z tych interfejsów API transakcji za pomocą prostego kodu testowego. Na koniec zintegrujemy API z aplikacją e-commerce TShirtShop i potokiem przetwarzania zamówień.

Podstawy transakcji kartą kredytową

W przypadku transakcji banki i inne instytucje finansowe korzystają z bezpiecznych sieci opartych na protokole X.25, a nie na protokole kontroli transmisji/protokołu internetowego (TCP/IP), podstawowym sposobie przesyłania danych przez Internet. Nie musisz dużo wiedzieć o X.25 poza tym, że jest to inny protokół sieciowy, który nie jest kompatybilny z TCP/IP. W związku z tym sieci X.25 są całkowicie oddzielone od Internetu i chociaż można uzyskać do nich bezpośredni dostęp, nie jest to prawdopodobnie rozsądna opcja. Aby to zrobić, być może będziemy musieli przeprowadzić poważne negocjacje z właścicielami sieci, z której chcemy korzystać. Właściciele będą chcieli mieć całkowitą pewność, że jesteśmy wiarygodnymi klientami, którzy są w stanie wyegzekwować niezbędne zabezpieczenia, aby zapobiec atakowi na ich system. W związku z tym właściciel sieci nie będzie rozdawał tych licencji nikomu – większości ludzi nie stać na wymagane środki bezpieczeństwa (w tym zamykanie serwerów w klatce, wysyłanie codziennych taśm z kopiami zapasowymi przez bezpieczną rynnę, posiadanie trzech osób z oddzielne klucze, aby uzyskać dostęp do tych taśm i tak dalej). Alternatywą jest dostęp do tych sieci za pośrednictwem dostawcy bramy. Dzięki temu możemy wykonywać naszą stronę protokołu transakcji kartą kredytową przez Internet (przy użyciu bezpiecznego protokołu), jednocześnie polegając na wybranej bramie do komunikacji z sieciami X.25. Chociaż prawdopodobnie wiąże się to z kosztami, dostawca powinien mieć umowę z instytucjami finansowymi, aby pomóc utrzymać koszty na niskim poziomie i przekazać nam oszczędności (po przejęciu przez bramę udziału), więc prawdopodobnie 623 będzie znacznie taniej niż posiadanie własnego połączenia X.25. Ta metoda może być również tańsza niż korzystanie z zewnętrznego procesora płatności, ponieważ potrzebujemy tylko minimalnej funkcjonalności, ponieważ obsługujemy własny potok zamówień. Nie ma potrzeby np. korzystania z całej funkcjonalności audytu zamówień oferowanej przez firmę taką jak PayPal, ponieważ wszystkie te funkcjonalności mamy już w naszym wdrożeniu.

Praca z bramkami płatności kartą kredytową

Aby współpracować z organizacją bramy, najpierw musimy otworzyć konto bankowe sprzedawcy. Można to zrobić w większości banków; bank dostarczy Ci identyfikator sprzedawcy, którego możesz użyć podczas rejestracji w bramce. Następnym krokiem jest znalezienie odpowiedniej bramy. To może być bardzo ciężka praca! Chociaż znalezienie bramki nie jest trudne, wyzwaniem jest znalezienie kompetentnej, która oferuje usługi w akceptowalnej cenie i jakości. Dosłownie setki firm są chętne do zmniejszenia Twojej sprzedaży. Szybkie wyszukiwanie w Internecie hasła „brama kart kredytowych" spowoduje powstanie długiej listy. Strony internetowe tych firm są w większości czystymi broszurami - znajdziesz się na stronach z tekstem o tym, że są najlepsze i najbezpieczniejsze w tym, co robią, tylko po to, aby wypełnić formularz, aby przedstawiciel obsługi klienta może zadzwonić do Ciebie, aby

„omówić Twoje potrzeby”. Na dłuższą metę możesz mieć pewność, że przynajmniej będziesz musiał przejść tę procedurę tylko raz. Prawdopodobnie przekonasz się, że większość organizacji oferujących tę usługę oferuje podobne pakiety. Jednak niektóre kluczowe punkty, na które należy zwrócić uwagę, obejmują banki, z którymi prowadzą interesy (twoje konto bankowe sprzedawcy będzie musiało znajdować się w jednym z nich), waluty, w których mają do czynienia, i oczywiście koszty.

DataCash i Authorize.net

W tym rozdziale zademonstrujemy wdrażanie transakcji kartą kredytową za pomocą dwóch usług online:

DataCash i Authorize.net.

DataCash to organizacja obsługująca karty kredytowe z siedzibą w Wielkiej Brytanii. Będziesz potrzebować konta bankowego sprzedawcy w Wielkiej Brytanii, jeśli chcesz go użyć w ostatecznej aplikacji. Jednak na razie nie musisz się o to martwić: bardzo łatwo jest uzyskać dostęp do dość przydatnego konta testowego - nie potrzebujesz nawet konta bankowego sprzedawcy. Authorize.net, jak wspomniano na swojej oficjalnej stronie internetowej pod adresem http://www.authorize.net, „zapewnia usługi bramek płatniczych protokołu internetowego (IP), które umożliwiają sprzedawcom autoryzowanie, rozliczanie i zarządzanie transakcjami kart kredytowych lub czeków elektronicznych w dowolnym momencie, gdziekolwiek." Innymi słowy, Authorize.net oferuje również usługę, której potrzebujemy do samodzielnego przetwarzania kart kredytowych, gdy ktoś kupi jedną z naszych koszulek. Ważnym punktem do zapamiętania jest to, że techniki opisane w tej Części mają zastosowanie do każdej bramki kart kredytowych. Szczegóły mogą się nieznacznie zmienić, jeśli przejdziesz do innej organizacji, ale większość ciężkiej pracy już wykonałeś. Jak zobaczysz w dalszej części, zarówno Authorize.net, jak i DataCash umożliwiają nam przeprowadzanie transakcji testowych przy użyciu tak zwanych „magicznych” numerów kart kredytowych (dostarczanych oddzielnie przez Authorize.net i DataCash), które będą akceptować lub odrzucać transakcje bez wykonywania wszelkie rzeczywiste transakcje finansowe. Jest to fantastyczne rozwiązanie do celów programistycznych, ponieważ z pewnością nie chcemy używać do testowania własnych kart kredytowych!

Zrozumienie transakcji kartą kredytową

Bez względu na to, z której bramki korzystasz, podstawowe zasady transakcji kartą kredytową są takie same. Po pierwsze, rodzaje transakcji, z którymi będziemy mieć do czynienia w witrynie e-commerce, są znane jako transakcje bez karty kredytowej (CNP), co oznacza, że nie mamy przed sobą karty kredytowej i nie możemy zweryfikować podpis klienta. To nie jest problem; w końcu prawdopodobnie od jakiegoś czasu wykonujesz transakcje CNP online, przez telefon, pocztą i tak dalej. To po prostu coś, o czym należy pamiętać, jeśli zobaczysz akronim CNP. Różne bramy oferują kilka zaawansowanych usług, w tym weryfikację adresu posiadacza karty, sprawdzanie kodu bezpieczeństwa, wykrywanie oszustw i tak dalej. Każdy z nich dodaje dodatkową warstwę złożoności do przetwarzania kart kredytowych i nie omawiamy tutaj tych szczegółów. Ta Część przedstawia raczej punkt wyjścia, od którego można w razie potrzeby dodać te usługi. To, czy wybierzesz te opcjonalne usługi dodatkowe, czy nie, zależy od tego, ile pieniędzy przepływa przez Twój system oraz od kompromisu między kosztami wdrożenia usług a potencjalnymi kosztami, którym można by zapobiec dzięki tym dodatkowym usługom, jeśli coś pójdzie nie tak. Jeśli jesteś zainteresowany tymi usługami, wspomniany wcześniej przedstawiciel obsługi klienta chętnie wszystko wyjaśni. W obecnej formie możemy wykonać kilka rodzajów transakcji:

Autoryzacja: Sprawdź kartę pod kątem odpowiednich środków i dokonaj potrącenia.

Preautoryzacja: Sprawdź kartę pod kątem środków i przydziel je, jeśli są dostępne; to nie odlicza środków od razu.

Realizacja: Dokończ wstępnie autoryzowaną transakcję, odejmując już przydzielone środki.

Zwrot: Zwróć ukończoną transakcję lub po prostu przelej pieniądze na kartę kredytową.

Znowu specyfika jest różna, ale to są podstawowe typy. Użyjemy modelu preautoryzacji/realizacji, co oznacza, że nie przyjmujemy płatności, dopóki nie zlecimy naszemu dostawcy wysyłki towarów. Ta struktura została wskazana przez strukturę potoku, którą stworzyliśmy w poprzedniej Części.

Praca z DataCash

Teraz, gdy omówiliśmy już podstawy, zastanówmy się, jak sprawić, by rzeczy działały w aplikacji TShirtShop przy użyciu systemu DataCash. Pierwszą rzeczą do zrobienia jest założenie konta testowego w DataCash, wykonując następujące kroki:

1. Wejdź na http://www.datacash.com/.

2. Przejdź do obszaru programistów ➤Pobierz sekcję konta testowego na stronie internetowej.

3. Wprowadź swoje dane i wyślij formularz.

4. W otrzymanej wiadomości e-mail zanotuj nazwę użytkownika i hasło do konta, a także dodatkowe informacje wymagane do uzyskania dostępu do systemu raportowania DataCash.

Uwaga : Po uzyskaniu konta testowego należy pobrać Przewodnik programisty DataCash ze strony https://testserver.datacash.com/software/DevelopersGuide.pdf. Jest to oficjalny dokument, na którym możesz polegać, aby mieć dokładne i aktualne informacje o wszelkich usługach DataCash.

DataCash oferuje API PHP składające się z kilku klas PHP, których możesz użyć do połączenia się z jej usługami. Te klasy tworzą komunikaty XML, które współdziałają z usługą sieci Web DataCash. W tym rozdziale, aby jeszcze lepiej zrozumieć, jak system działa, tworzymy struktury XML i bezpośrednio wchodzimy w interakcję z usługą internetową DataCash. Będziemy dużo manipulować XML podczas komunikacji z DataCash, ponieważ będziemy musieli tworzyć dokumenty XML do wysłania do DataCash i wyodrębniania danych z odpowiedzi XML. Na następnych kilku stronach przyjrzymy się pokrótce plikowi XML wymaganemu do operacji, które będziemy wykonywać, oraz odpowiedziom, których możemy się spodziewać. Omówimy protokół komunikacyjny, co oznacza omówienie plików XML, które odzwierciedlają następujące etapy transakcji:

• Żądanie wstępnego uwierzytelnienia

• Odpowiedź na żądanie wstępnego uwierzytelnienia

• Prośba o realizację

• Odpowiedź dotycząca realizacji

Zaimplementujemy te etapy w potoku zamówień TShirtShop. Czytając ten rozdział, zawsze pamiętaj o korzystaniu z oficjalnej dokumentacji, która zawiera najbardziej aktualne i istotne aktualizacje, na https://testserver.datacash.com/software/DevelopersGuide.pdf.

Prośba o wstępne uwierzytelnienie

Wysyłając żądanie wstępnego uwierzytelnienia do DataCash, musimy podać następujące informacje:

• Nazwa użytkownika DataCash (znana jako klient DataCash)

• Hasło DataCash

• Unikalny numer referencyjny transakcji

• Kwota pieniędzy do obciążenia

• Waluta użyta w transakcji (dolary amerykańskie, funty brytyjskie itd.)

• Rodzaj transakcji (kod przed wstępnym uwierzytelnieniem i kod spełniający do realizacji)

• Numer karty kredytowej

• Data ważności karty kredytowej

• Data wydania karty kredytowej (jeśli dotyczy rodzaju używanej karty kredytowej)

• Numer wydania karty kredytowej (jeśli dotyczy typu używanej karty kredytowej)

Unikalny numer referencyjny transakcji musi być liczbą o długości od 6 do 12 cyfr, którą wybieramy, aby jednoznacznie identyfikować transakcję z zamówieniem. Ponieważ nie możemy użyć krótkiego numeru, nie możemy po prostu użyć wartości identyfikatora zamówienia, których do tej pory używaliśmy dla zamówień. Możemy jednak użyć tego identyfikatora zamówienia jako punktu wyjścia do utworzenia numeru referencyjnego, po prostu dodając do niego wysoką liczbę, na przykład 1 000 000. Nie możemy powielić numeru referencyjnego w żadnych przyszłych transakcjach; musimy mieć pewność, że po zakończeniu transakcji nie zostanie ona wykonana ponownie, co może skutkować dwukrotnym obciążeniem klienta. Proces ten oznacza jednak, że jeśli karta kredytowa zostanie odrzucona, może być konieczne utworzenie zupełnie nowego zamówienia, aby klient wygenerował nowy i unikalny numer referencyjny do wysłania do bramki, ale to nie powinno stanowić problemu. Żądanie XML jest sformatowane w następujący sposób, z wartościami wyszczególnionymi wcześniej pogrubioną czcionką:

<?xml version="1.0" encoding="UTF-8"?>

<Request>

<Authentication>

<password>**DataCash password**</password>

<client>**DataCash client**</client>

</Authentication>

<Transaction>

<TxnDetails>

<merchantreference>**Unique reference number**</merchantreference>

<amount currency='Currency Type'>**Cash amount**</amount>

</TxnDetails>

<CardTxn>

<method>**pre**</method>

<Card>

<pan>Credit card number</pan>

<expirydate>Credit card expiry date</expirydate>

</Card>

</CardTxn>

</Transaction>

</Request>

Odpowiedź na żądanie wstępnego uwierzytelnienia

Odpowiedź na żądanie wstępnego uwierzytelnienia zawiera następujące informacje:

• Numer kodu statusu wskazuje, co się stało; kod będzie 1, jeśli transakcja się powiodła lub inny z kilku innych kodów, jeśli wydarzy się coś innego. Pełną listę kodów zwrotnych dla serwera DataCash można znaleźć pod adresem https://testserver.datacash.com/software/returncodes.shtml.

• Powodem statusu jest w zasadzie napis wyjaśniający status w języku angielskim. Dla stanu 1 ten ciąg jest ZAAKCEPTOWANY.

• Podaje się kod uwierzytelniający i numer referencyjny do wykorzystania podczas finalizowania transakcji na etapie żądania realizacji (omówione dalej).

• Podany jest czas przetworzenia transakcji.

• Podany jest również tryb transakcji, którym jest TEST przy korzystaniu z konta testowego.

• Dostarczane jest potwierdzenie typu używanej karty kredytowej.

• Dołączone jest również potwierdzenie kraju, w którym karta kredytowa została wydana.

• Dostarczany jest również kod autoryzacyjny używany przez bank (tylko w celach informacyjnych).

Kod XML do tego jest sformatowany w następujący sposób:

<?xml version="1.0" encoding="utf-8"?>

<Response>

<status>Status code</status>

<reason>Reason</reason>

<merchantreference>Authentication code</merchantreference>

<datacash_reference>Reference number</datacash_reference>

<time>Time</time>

<mode>TEST</mode>

<CardTxn>

<card_scheme>Card Type</card_scheme>

<country>Country</country>

<issuer>Card issuing bank</issuer>

<authcode>Bank authorization code</authcode>

</CardTxn>

</Response>

Prośba o realizację

W przypadku żądania realizacji musimy przesłać następujące informacje:

• Nazwa użytkownika DataCash (klient DataCash)

• Hasło DataCash

• Rodzaj transakcji (do realizacji kod spełnia)

• Otrzymany wcześniej kod uwierzytelniający

• Numer referencyjny otrzymany wcześniej

Opcjonalnie możemy dołączyć dodatkowe informacje, takie jak potwierdzenie kwoty do pobrania z karty kredytowej, choć nie jest to konieczne. Komunikat XML żądania realizacji ma następujący format:

<?xml version="1.0" encoding="UTF-8"?>

<Request>

<Authentication>

<password>DataCash password</password>

<client>DataCash client</client>

</Authentication>

<Transaction>

<HistoricTxn>

<reference>Reference Number</reference>

<authcode>Authentication code</authcode>

<method>fulfill</method>

</HistoricTxn>

</Transaction>

</Request>

Odpowiedź dotycząca realizacji

Odpowiedź na żądanie realizacji zawiera następujące informacje:

• Numer kodu statusu wskazuje, co się stało; kod będzie 1, jeśli transakcja się powiodła lub inny z kilku innych kodów, jeśli wydarzy się coś innego. Ponownie, aby uzyskać pełną listę kodów, zobacz https://testserver.datacash.com/software/returncodes.shtml.

• Powodem statusu jest w zasadzie napis wyjaśniający status w języku angielskim. Dla stanu 1 ten ciąg jest WYPEŁNIONY OK.

• DataCash udostępnia dwie kopie kodu referencyjnego.

• Podano czas przetworzenia transakcji.

• Dostępny jest również tryb transakcji, którym jest TEST przy korzystaniu z konta testowego.

Wiadomość XML dla odpowiedzi realizacji jest sformatowana w następujący sposób:

<?xml version="1.0" encoding="utf-8"?>

<Response>

<status>Status code</status>

<reason>Reason</reason>

<merchantreference>Reference Code</merchantreference>

<datacash_reference>Reference Code</datacash_reference>

<time>Time</time>

<mode>TEST</mode>

</Response>

Wymiana danych XML z DataCash

Ponieważ dane XML, które musimy wysłać do DataCash mają prostą i standardową strukturę, zbudujemy je ręcznie w ciągu, bez korzystania z obsługi XML oferowanej przez PHP . Skorzystamy jednak z rozszerzenia SimpleXML PHP , które sprawia, że czytanie prostych danych XML to bułka z masłem. Chociaż jest mniej złożone i wydajne niż DOMDocument, rozszerzenie SimpleXML ułatwia parsowanie danych XML, przekształcając je w strukturę danych, którą możemy po prostu iterować.

Uwaga : Dla kodu, który komunikuje się z DataCash, używamy biblioteki cURL (http://curl.haxx.se/). Odkomentuj następujący wiersz w php.ini, usuwając początkowy średnik: extension=php_curl.dll.

Następnie uruchom ponownie Apache. Plik php.ini znajdziesz w folderze C:\xampp\apache\bin\ w systemie Windows lub w folderze /opt/lampp/etc w systemie Linux.

Ćwiczenie: Komunikacja z DataCash

1. Utwórz nowy plik o nazwie datacash_request.php w folderze biznesowym i dodaj do niego następujący kod:

<?php

class DataCashRequest

{

// DataCash Server URL

```
private $_mUrl;
// Will hold the current XML document to be sent to DataCash
private $_mXml;
// Constructor initializes the class with URL of DataCash
public function __construct($url)
{
// Datacash URL
$this->_mUrl = $url;
}
/* Compose the XML structure for the pre-authentication
request to DataCash */
public function MakeXmlPre($dataCashClient, $dataCashPassword,
$merchantReference, $amount, $currency,
$cardNumber, $expiryDate,
$startDate = '', $issueNumber = '')
{
$this->_mXml =
"<?xml version=\"1.0\" encoding=\"UTF-8\"\x3F>
<Request>
<Authentication>
<password>$dataCashPassword</password>
<client>$dataCashClient</client>
</Authentication>
<Transaction>
<TxnDetails>
<merchantreference>$merchantReference</merchantreference>
<amount currency=\"$currency\">$amount</amount>
</TxnDetails>
<CardTxn>
<method>pre</method>
<Card>
```

```
<pan>$cardNumber</pan>
<expirydate>$expiryDate</expirydate>
<startdate>$startDate</startdate>
<issuenumber>$issueNumber</issuenumber>
</Card>
</CardTxn>
</Transaction>
</Request>";
}
// Compose the XML structure for the fulfillment request to DataCash
public function MakeXmlFulfill($dataCashClient, $dataCashPassword,
$authCode, $reference)
{
$this->_mXml =
"<?xml version=\"1.0\" encoding=\"UTF-8\"\x3F>
<Request>
<Authentication>
<password>$dataCashPassword</password>
<client>$dataCashClient</client>
</Authentication>
<Transaction>
<HistoricTxn>
<reference>$reference</reference>
<authcode>$authCode</authcode>
<method>fulfill</method>
</HistoricTxn>
</Transaction>
</Request>";
}
// Get the current XML
public function GetRequest()
```

```
{
return $this->_mXml;
}
// Send an HTTP POST request to DataCash using cURL
public function GetResponse()
{
// Initialize a cURL session
$ch = curl_init();
// Prepare for an HTTP POST request
curl_setopt($ch, CURLOPT_POST, 1);
// Prepare the XML document to be POSTed
curl_setopt($ch, CURLOPT_POSTFIELDS, $this->_mXml);
// Set the URL where we want to POST our XML structure
curl_setopt($ch, CURLOPT_URL, $this->_mUrl);
/* Do not verify the Common name of the peer certificate in the SSL
handshake */
curl_setopt($ch, CURLOPT_SSL_VERIFYHOST, 0);
// Prevent cURL from verifying the peer's certificate
curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, 0);
/* We want cURL to directly return the transfer instead of
printing it */
curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
// Perform a cURL session
$result = curl_exec($ch);
// Close a cURL session
curl_close ($ch);
// Return the response
return $result;
}
}
?>
```

2. Zdefiniuj adres URL DataCash i dane logowania na końcu pliku include/config.php:

```
// Constant definitions for datacash
define('DATACASH_URL', 'https://testserver.datacash.com/Transaction');
define('DATACASH_CLIENT', 'your account client number');
define('DATACASH_PASSWORD', 'your account password');
```

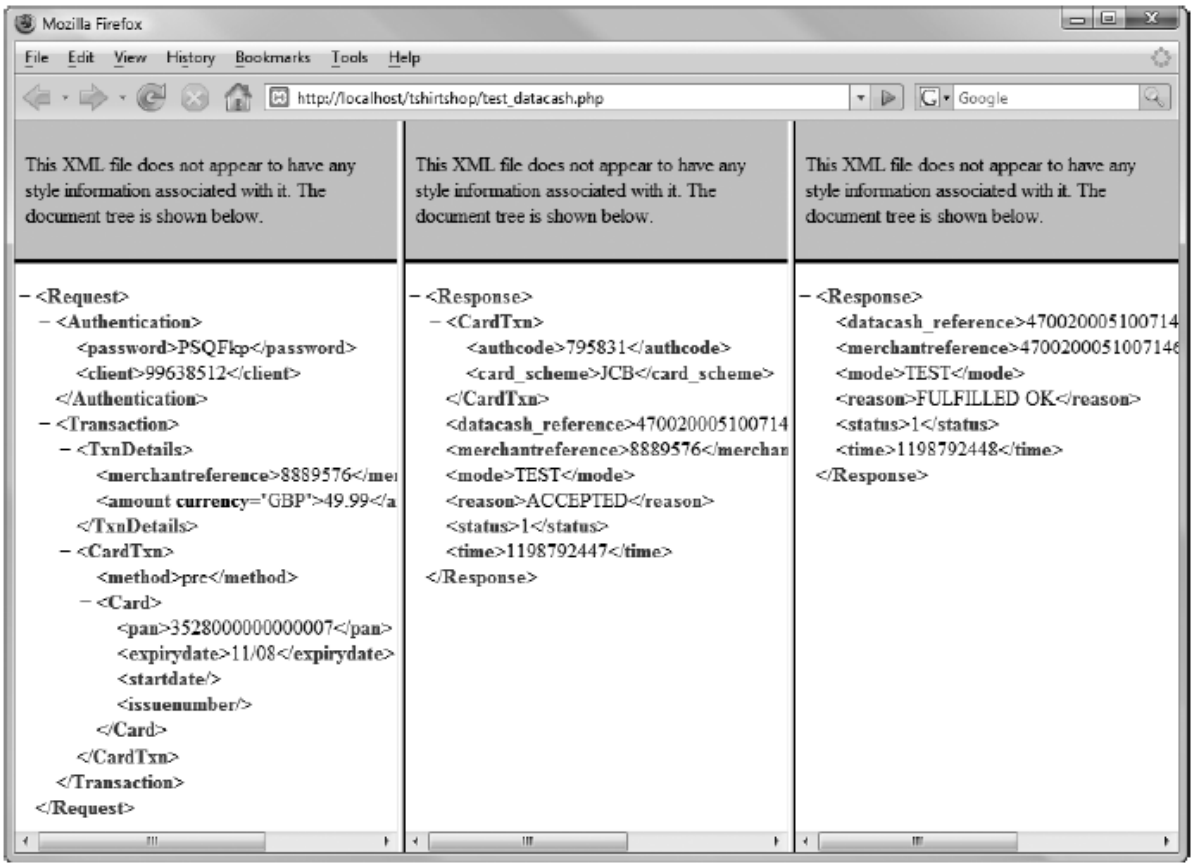
Uwaga : Nie zapomnij użyć danych z konta DataCash!

3. Utwórz plik test_datacash.php w domu swojego projektu (folder tshirtshop) i dodaj w nim:

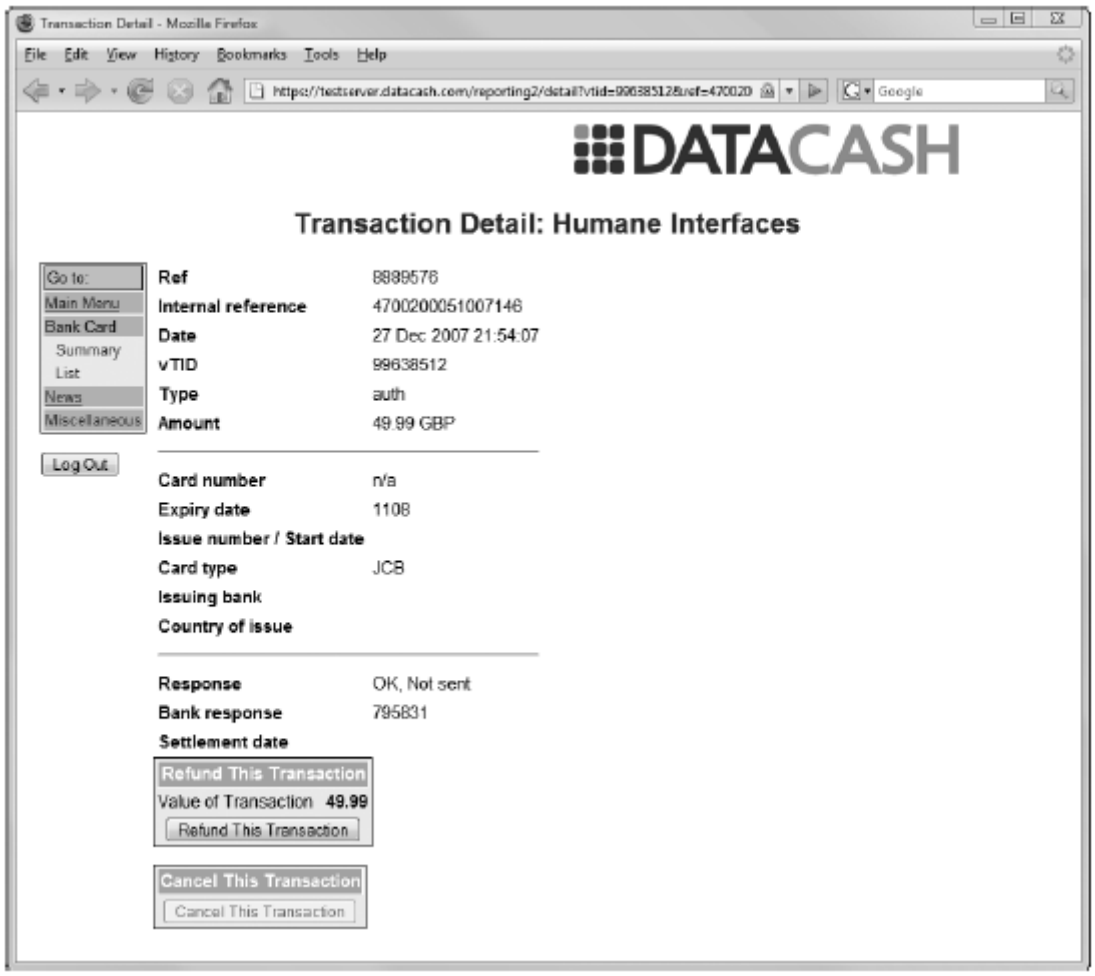
```
<?php
session_start();
if (empty ($_GET['step']))
{
require_once 'include/config.php';
require_once BUSINESS_DIR . 'datacash_request.php';
$request = new DataCashRequest(DATACASH_URL);
$request->MakeXmlPre(DATACASH_CLIENT, DATACASH_PASSWORD,
8880000 + rand(0, 10000), 49.99, 'GBP',
'3528000000000007', '11/09');
$request_xml = $request->GetRequest();
$_SESSION['pre_request'] = $request_xml;
$response_xml = $request->GetResponse();
$_SESSION['pre_response'] = $response_xml;
$xml = simplexml_load_string($response_xml);
$request->MakeXmlFulfill(DATACASH_CLIENT, DATACASH_PASSWORD,
$xml->merchantreference,
$xml->datacash_reference);
$response_xml = $request->GetResponse();
$_SESSION['fulfill_response'] = $response_xml;
}
else
{
header('Content-type: text/xml');
```

```
switch ($_GET['step'])

{

case 1:

print $_SESSION['pre_request'];

break;

case 2:

print $_SESSION['pre_response'];

break;

case 3:

print $_SESSION['fulfill_response'];

break;

}

exit();

}

?>

<frameset cols="33%, 33%, 33%">

<frame src="test_datacash.php?step=1">

<frame src="test_datacash.php?step=2">

<frame src="test_datacash.php?step=3">

</frameset>
```

4. Załaduj plik test_datacash.php w przeglądarce, aby zobaczyć wyniki. Jeśli używasz Opery, wynik powinien wyglądać jak na rysunku, ponieważ Opera pokazuje tylko zawartość elementów XML. Jeśli używasz innej przeglądarki internetowej, powinieneś zobaczyć poprawnie sformatowane dokumenty XML.



5. Zaloguj się na https://testserver.datacash.com/reporting2, aby zobaczyć dziennik transakcji dla swojego konta DataCash (pamiętaj, że aktualizacja tego widoku zajmuje trochę czasu, więc możesz nie widzieć transakcji od razu). Ten raport pokazano na rysunku.



Jak to działa: kod, który komunikuje się z DataCash

Klasa DataCashRequest jest dość prosta. Najpierw konstruktor ustawia adres HTTPS, na który wysyłamy Twoje żądania:

```
// Constructor initializes the class with URL of DataCash

public function __construct($url)

{

// Datacash URL

$this->_mUrl = $url;

}
```

Gdy chcemy wykonać żądanie wstępnego uwierzytelnienia, najpierw musimy wywołać metodę MakeXmlPre(), aby utworzyć wymagany kod XML dla tego rodzaju żądania. Niektóre elementy XML są opcjonalne (takie jak startdate lub issuenumber, które otrzymują wartości domyślne, jeśli nie dostarczamy własnych – patrz metoda MakeXmlPre()), ale inne elementy są obowiązkowe.

Notatka: Jeśli chcesz zobaczyć dokładnie, które elementy są obowiązkowe, a które opcjonalne dla każdego rodzaju żądania, sprawdź dokument z często zadawanymi pytaniami API XML z DataCash.

Kolejnym rodzajem żądania, które musimy być w stanie złożyć do systemu DataCash, jest żądanie spełnienia. XML dla tego rodzaju żądania jest przygotowywany w metodzie MakeXmlFulfill(). Następnie mamy metodę GetRequest(), która zwraca ostatni dokument XML zbudowany przez MakeXmlPre() lub MakeXmlFulfill():

```
// Get the current XML

public function GetRequest()

{

return $this->_mXml;

}
```

Wreszcie metoda GetResponse() faktycznie wysyła najnowszy plik żądania XML, zbudowany przez wywołanie MakeXmlPre() lub MakeXmlFulfill() i zwraca XML odpowiedzi. Przyjrzyjmy się bliżej tej metodzie. GetResponse() rozpoczyna się od zainicjowania sesji cURL i ustawienia metody POST do wysyłania danych:

```
// Send an HTTP POST request to DataCash using cURL

public function GetResponse()

{

// Initialize a cURL session

$ch = curl_init();

// Prepare for an HTTP POST request

curl_setopt($ch, CURLOPT_POST, 1);

// Prepare the XML document to be POSTed

curl_setopt($ch, CURLOPT_POSTFIELDS, $this->_mXml);

// Set the URL where we want to POST our XML structure

curl_setopt($ch, CURLOPT_URL, $this->_mUrl);

/* Do not verify the Common name of the peer certificate in the SSL

handshake */

curl_setopt($ch, CURLOPT_SSL_VERIFYHOST, 0);

// Prevent cURL from verifying the peer's certificate

curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, 0);
```

Aby zwrócić transfer do zmiennej PHP, ustawiamy parametr CURLOPT_RETURNTRANSFER na 1, wysyłamy żądanie i zamykamy sesję cURL:

```
/* We want cURL to directly return the transfer instead of

printing it */

curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
```

```
// Perform a cURL session

$result = curl_exec($ch);

// Close a cURL session

curl_close ($ch);

// Return the response

return $result;

}
```

Plik test_datacash.php działa w następujący sposób: Kiedy ładujemy go w przeglądarce, skrypt wysyła żądanie wstępnego uwierzytelnienia i żądanie spełnienia, a następnie zapisuje żądanie wstępnego uwierzytelnienia, odpowiedź i dane XML dotyczące spełnienia w sesji:

```
session_start();

if (empty ($_GET['step']))

{

require_once 'include/config.php';

require_once BUSINESS_DIR . 'datacash_request.php';

$request = new DataCashRequest(DATACASH_URL);

$request->MakeXmlPre(DATACASH_CLIENT, DATACASH_PASSWORD,

8880000 + rand(0, 10000), 49.99, 'GBP',

'3528000000000007', '11/09');

$request_xml = $request->GetRequest();

$_SESSION['pre_request'] = $request_xml;

$response_xml = $request->GetResponse();

$_SESSION['pre_response'] = $response_xml;

$xml = simplexml_load_string($response_xml);

$request->MakeXmlFulfill(DATACASH_CLIENT, DATACASH_PASSWORD,

$xml->merchantreference,

$xml->datacash_reference);

$response_xml = $request->GetResponse();

$_SESSION['fulfill_response'] = $response_xml;

}
```

Strona test_datacash.php załaduje się trzy razy więcej, ponieważ mamy trzy ramki, które chcemy wypełnić danymi:

```
<frameset cols="33%, 33%, 33%">

<frame src="test_datacash.php?step=1">

<frame src="test_datacash.php?step=2">

<frame src="test_datacash.php?step=3">

</frameset>
```

W zależności od wartości kroku decydujemy, które z zapisanych wcześniej w sesji danych XML zostaną wyświetlone w bieżącej ramce. Jeśli wartość kroku wynosi 1, wyświetlany jest kod XML żądania wstępnego. Jeśli wartość wynosi 2, wyświetlany jest kod XML odpowiedzi wstępnej. Jeśli wartość kroku wynosi 3, wyświetlany jest kod XML odpowiedzi spełniania.

```
else

{

header('Content-type: text/xml');

switch ($_GET['step'])

{

case 1:

print $_SESSION['pre_request'];

break;

case 2:

print $_SESSION['pre_response'];

break;

case 3:

print $_SESSION['fulfill_response'];

break;

}

exit();

}
```

Integracja DataCash z TShirtShop

Teraz, gdy mamy nową klasę, która wykonuje transakcje kartą kredytową, wszystko, co musimy zrobić, to zintegrować jej funkcjonalność z potokiem zamówień, który zbudowaliśmy w poprzednich rozdziałach. Aby w pełni zintegrować DataCash z TShirtShop, musimy zaktualizować istniejące klasy PsCheckFunds i PsTakePayments. Musimy zmodyfikować klasy sekcji potoku, które zajmują się transakcjami kartami kredytowymi. Dodaliśmy już infrastrukturę do przechowywania i pobierania kodów uwierzytelniających oraz informacji referencyjnych za pomocą metody zrderProcessor::SetOrderAuthCodeAndReference().

Ćwiczenie: Implementacja klas potoku zamówień

1. Najpierw zastąp kod w business/ps_check_funds.php następującym kodem, który działa z DataCash:

```
<?php
class PsCheckFunds implements IPipelineSection
{
public function Process($processor)
{
// Audit
$processor->CreateAudit('PsCheckFunds started.', 20100);
$order_total_cost = $processor->mOrderInfo['total_amount'];
$order_total_cost += $processor->mOrderInfo['shipping_cost'];
$order_total_cost +=
round((float)$order_total_cost *
(float)$processor->mOrderInfo['tax_percentage'], 2) / 100.00;
$request = new DataCashRequest(DATACASH_URL);
$request->MakeXmlPre(DATACASH_CLIENT, DATACASH_PASSWORD,
$processor->mOrderInfo['order_id'] + 1000006,
$order_total_cost, 'GBP',
$processor->mCustomerInfo['credit_card']->CardNumber,
$processor->mCustomerInfo['credit_card']->ExpiryDate,
$processor->mCustomerInfo['credit_card']->IssueDate,
$processor->mCustomerInfo['credit_card']->IssueNumber);
$responseXml = $request->GetResponse();
$xml = simplexml_load_string($responseXml);
if ($xml->status == 1)
{
$processor->SetAuthCodeAndReference(
$xml->merchantreference, $xml->datacash_reference);
// Audit
$processor->CreateAudit('Funds available for purchase.', 20102);
// Update order status
```

```
$processor->UpdateOrderStatus(2);
// Continue processing
$processor->mContinueNow = true;
}
else
{
// Audit
$processor->CreateAudit('Funds not available for purchase.', 20103);
throw new Exception('Credit card check funds failed for order ' .
$processor->mOrderInfo['order_id'] . "\n\n" .
'Data exchanged:' . "\n" .
$request->GetResponse() . "\n" . $responseXml);
}
// Audit
$processor->CreateAudit('PsCheckFunds finished.', 20101);
}
}
?>
```

2. Zastąp kod w business/ps_take_payment.php następującym kodem:

```
<?php
class PsTakePayment implements IPipelineSection
{
public function Process($processor)
{
// Audit
$processor->CreateAudit('PsTakePayment started.', 20400);
$request = new DataCashRequest(DATACASH_URL);
$request->MakeXmlFulFill(DATACASH_CLIENT, DATACASH_PASSWORD,
$processor->mOrderInfo['auth_code'],
$processor->mOrderInfo['reference']);
$responseXml = $request->GetResponse();
```

```
$xml = simplexml_load_string($responseXml);
if ($xml->status == 1)
{
// Audit
$processor->CreateAudit(
'Funds deducted from customer credit card account.', 20402);
// Update order status
$processor->UpdateOrderStatus(5);
// Continue processing
$processor->mContinueNow = true;
}
else
{
// Audit
$processor->CreateAudit('Could not deduct funds from credit card.',
20403);
throw new Exception('Credit card take payment failed for order ' .
$processor->mOrderInfo['order_id'] . "\n\n" .
'Data exchanged:' . "\n" .
$request->GetResponse() . "\n" . $responseXml);
}
// Audit
$processor->CreateAudit('PsTakePayment finished.', 20401);
}
}
?>
```

3. Dodaj odwołanie do pliku business/datacash_request.php w index.php, jak zaznaczono:

```
require_once BUSINESS_DIR . 'ps_check_funds.php';
require_once BUSINESS_DIR . 'ps_check_stock.php';
require_once BUSINESS_DIR . 'datacash_request.php';
```

4. Dodaj odwołanie do pliku business/datacash_request.php w admin.php, jak zaznaczono:

require_once BUSINESS_DIR . 'ps_ship_ok.php';

require_once BUSINESS_DIR . 'ps_final_notification.php';

require_once BUSINESS_DIR . **'datacash_request.php'**;

Testowanie integracji DataCash

Teraz, gdy mamy to wszystko na miejscu, ważne jest, aby przetestować to za pomocą kilku zamówień. Możemy to łatwo zrobić, tworząc klienta z tymi magicznymi danymi karty kredytowej. Jak wspomniano wcześniej w tym rozdziale, DataCash dostarcza te liczby do celów testowych i uzyskania określonych odpowiedzi od DataCash.

W tej chwili możemy eksperymentować z nową, w pełni funkcjonalną witryną e-commerce, składając zamówienia z testowymi numerami kart kredytowych, sprawdzając wiadomości e-mail wysyłane przez witrynę internetową i sprawdzając, jak witryna reaguje w określonych sytuacjach, takich jak: jak rejestruje błędy, jak administruje zamówieniami za pomocą strony zarządzania zamówieniami i tak dalej.

Na żywo

Przejście z konta testowego na konto rzeczywiste jest teraz po prostu kwestią zastąpienia danych logowania DataCash w pliku include/config.php wartościami ze świata rzeczywistego. Po założeniu konta bankowego sprzedawcy możesz użyć nowych danych, aby założyć nowe konto DataCash, zdobywając po drodze nowe dane klienta i hasła. Musisz również zmienić adres URL serwera DataCash, do którego wysyłasz dane, ponieważ musi to być serwer produkcyjny, a nie serwer testowy. Poza usunięciem testowych kont użytkowników z bazy danych i przeniesieniem witryny internetowej do lokalizacji internetowej, to wszystko, co musisz zrobić, zanim udostępnisz klientom nowo ukończoną aplikację e-commerce.

Praca z Authorize.net

Aby przetestować Authorize.net, musisz złożyć wniosek o konto testowe na http://developer.authorize.net/testaccount/. Strona główna, na której programiści mogą uzyskać informacje na temat integracji Authorize.net, to http://developer.authorize.net/.Komunikacja z Authorize.net różni się od komunikacji z DataCash. Zamiast wysyłać i odbierać pliki XML, wysyłamy ciągi składające się z par nazwa-wartość oddzielonych znakami ampersand (&). W rzeczywistości używamy podobnej składni do ciągów zapytań dołączanych do adresów URL. Authorize.net zwraca wynik transakcji w postaci ciągu znaków, który zawiera zwracane wartości (bez ich nazw) oddzielone znakiem, który określimy przy składaniu początkowego żądania. W naszych przykładach użyjemy znaku kreski pionowej (|). Zwracane wartości występują w ustalonej kolejności, a ich znaczenie jest określone przez ich pozycję w zwracanym ciągu.

Uwaga Pełną dokumentację interfejsu Authorize.net API można znaleźć w Przewodniku implementacji Advanced Integration Method (AIM): Card-Not-Present Transactions pod adresem http://www.authorize.net/support/AIM_guide.pdf. Jeszcze więcej dokumentów jest dostępnych w bibliotece dokumentów pod adresem http://www.authorize.net/resources/documentlibrary/.

Domyślnym typem transakcji jest AUTH_CAPTURE, gdzie żądamy i potrącamy środki z karty kredytowej za pomocą jednego żądania. W przypadku TShirtShop użyjemy dwóch innych typów transakcji: AUTH_ONLY, który sprawdza, czy dostępne są potrzebne środki (działa to na etapie potoku PsCheckFunds), oraz PRIOR_AUTH_CAPTURE, który odejmuje kwotę pieniędzy, która została wcześniej

sprawdzona przy użyciu AUTH_ONLY (tak się dzieje na etapie potoku PsTakePayment). Aby wykonać transakcję AUTH_ONLY, najpierw utworzymy tablicę zawierającą niezbędne dane transakcji:

```
// Auth

$transaction =

array ('x_invoice_num' => '99999', // Invoice number

'x_amount' => '45.99', // Amount

'x_card_num' => '4007000000027', // Credit card number

'x_exp_date' => '1209', // Expiration date

'x_method' => 'CC', // Payment method

'x_type' => 'AUTH_ONLY'); // Transaction type
```

W przypadku transakcji PRIOR_AUTH_CAPTURE nie musimy ponownie podawać wszystkich tych informacji; musimy tylko przekazać identyfikator transakcji, który został zwrócony w odpowiedzi na żądanie AUTH_ONLY.

```
// Capture

$transaction =

array ('x_ref_trans_id' => $ref_trans_id, // Transaction id

'x_method' => 'CC', // Payment method

'x_type' => 'PRIOR_AUTH_CAPTURE'); // Transaction type
```

Przekształcimy te tablice w ciąg par nazwa-wartość i prześlemy je na serwer Authorize.net. Odpowiedź ma postać ciągu znaków, którego wartości są oddzielone konfigurowalnym znakiem. Później, na rysunku 20-3, możesz zobaczyć przykładową odpowiedź na żądanie AUTH_ONLY (w lewej części okna) i przykładową odpowiedź na żądanie PRIOR_AUTH_CAPTURE (w prawej części okna). Napiszemy prosty test z tym typem transakcji przed wprowadzeniem jakichkolwiek modyfikacji w TShirtShop. Postępuj zgodnie z instrukcjami w ćwiczeniu, aby przetestować Authorize.net.

Ćwiczenie: Testowanie Authorize.net

1. Utwórz nowy plik o nazwie authorize_net_request.php w folderze biznesowym i dodaj do niego następujący kod:

```
<?php

class AuthorizeNetRequest

{

// Authorize Server URL

private $_mUrl;

// Will hold the current request to be sent to Authorize.net

private $_mRequest;
```

```
// Constructor initializes the class with URL of Authorize.net
public function __construct($url)
{
// Authorize.net URL
$this->_mUrl = $url;
}
public function SetRequest($request)
{
$this->_mRequest = '';
$request_init =
array ('x_login' => AUTHORIZE_NET_LOGIN_ID,
'x_tran_key' => AUTHORIZE_NET_TRANSACTION_KEY,
'x_version' => '3.1',
'x_test_request' => AUTHORIZE_NET_TEST_REQUEST,
'x_delim_data' => 'TRUE',
'x_delim_char' => '|',
'x_relay_response' => 'FALSE');
$request = array_merge($request_init, $request);
foreach($request as $key => $value )
$this->_mRequest .= $key . '=' . urlencode($value) . '&';
}
// Send an HTTP POST request to Authorize.net using cURL
public function GetResponse()
{
// Initialize a cURL session
$ch = curl_init();
// Prepare for an HTTP POST request
curl_setopt($ch, CURLOPT_POST, 1);
// Prepare the request to be POSTed
curl_setopt($ch, CURLOPT_POSTFIELDS, rtrim($this->_mRequest, '& '));
// Set the URL where we want to POST our data
```

```
curl_setopt($ch, CURLOPT_URL, $this->_mUrl);
/* Do not verify the Common name of the peer certificate in the SSL
handshake */
curl_setopt($ch, CURLOPT_SSL_VERIFYHOST, 0);
// Prevent cURL from verifying the peer's certificate
curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, 0);
/* We want cURL to directly return the transfer instead of
printing it */
curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
// Perform a cURL session
$result = curl_exec($ch);
// Close a cURL session
curl_close ($ch);
// Return the response
return $result;
}
}
?>
```

2. Dodaj następujące informacje na końcu pliku include/config.php, modyfikując stałe dane szczegółami konta Authorize.net:

```
// Constant definitions for authorize.net
define('AUTHORIZE_NET_URL', 'https://test.authorize.net/gateway/transact.dll');
define('AUTHORIZE_NET_LOGIN_ID', '[Your Login ID]');
define('AUTHORIZE_NET_TRANSACTION_KEY', '[Your Transaction Key]');
define('AUTHORIZE_NET_TEST_REQUEST', 'FALSE');
```

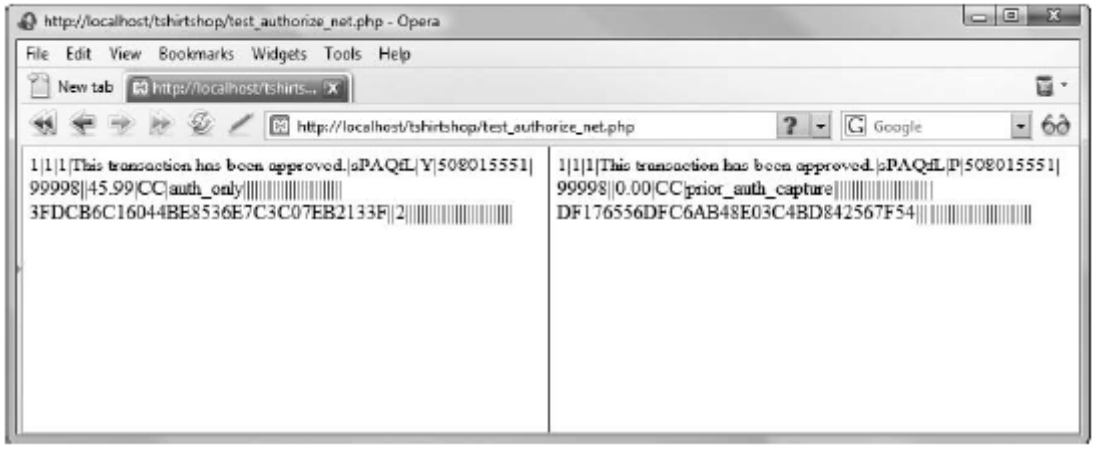
3. Dodaj następujący plik testowy test_authorize_net.php do katalogu głównego witryny:

```
<?php
session_start();
if (empty ($_GET['step']))
{
require_once 'include/config.php';
```

```
require_once BUSINESS_DIR . 'authorize_net_request.php';
$request = new AuthorizeNetRequest(AUTHORIZE_NET_URL);
// Auth
$transaction =
array ('x_invoice_num' => '99999', // Invoice number
'x_amount' => '45.99', // Amount
'x_card_num' => '4007000000027', // Credit card number
'x_exp_date' => '1209', // Expiration date
'x_method' => 'CC', // Payment method
'x_type' => 'AUTH_ONLY'); // Transaction type
$request->SetRequest($transaction);
$auth_only_response = $request->GetResponse();
$_SESSION['auth_only_response'] = $auth_only_response;
$auth_only_response = explode('|', $auth_only_response);
// Read the transaction ID, which will be necessary for taking the payment
$ref_trans_id = $auth_only_response[6];
// Capture
$transaction =
array ('x_ref_trans_id' => $ref_trans_id, // Transaction id
'x_method' => 'CC', // Payment method
'x_type' => 'PRIOR_AUTH_CAPTURE'); // Transaction type
$request->SetRequest($transaction);
$prior_auth_capture_response = $request->GetResponse();
$_SESSION['prior_auth_capture_response'] = $prior_auth_capture_response;
}
else
{
switch ($_GET['step'])
{
case 1:
print $_SESSION['auth_only_response'];
```

```
break;
case 2:
print $_SESSION['prior_auth_capture_response'];
break;
}
exit();
}
?>
<frameset cols="50%, 50%">
<frame src="test_authorize_net.php?step=1">
<frame src="test_authorize_net.php?step=2">
</frameset>
```

4. Załaduj stronę test_authorize_net.php w ulubionej przeglądarce, aby zobaczyć wynik



5. Przejdź do Authorize.net i zaloguj się do Merchant Interface (https://test.authorize.net/). Transakcję, którą właśnie wykonałeś, możesz zobaczyć w sekcji Transakcje nierozliczone w zakładce Szukaj. Ten raport pokazano na rysunku

Jak to działa: transakcje Authorize.net

Ciężka praca jest wykonywana przez klasę AuthorizeNetRequest, która ma dwie ważne metody: SetRequest(), używaną do ustawiania szczegółów transakcji, oraz GetResponse(), używaną do wysyłania żądania i pobierania odpowiedzi z Authorize.net. Poniższy fragment kodu pokazuje, jak są używane:

```
// Auth

$transaction =

array ('x_invoice_num' => '99999', // Invoice number

'x_amount' => '45.99', // Amount

'x_card_num' => '4007000000027', // Credit card number

'x_exp_date' => '1209', // Expiration date

'x_method' => 'CC', // Payment method

'x_type' => 'AUTH_ONLY'); // Transaction type

$request->SetRequest($transaction);
```

$response = $request->GetResponse();

Uwaga : Dane karty kredytowej wymienione w tej transakcji to jeden z magicznych numerów kart dostarczonych przez Authorize.net do celów testowych. Zapoznaj się z Przewodnikiem wdrażania AIM, aby uzyskać pełną listę takich numerów kart kredytowych

Jako parametr wysyłamy tablicę ze szczegółami transakcji do metody SetRequest(), która następnie łączy tę tablicę z inną tablicą zawierającą szczegóły konta Authorize.net:

```
public function SetRequest($request)

{

$this->_mRequest = '';

$request_init =

array ('x_login' => AUTHORIZE_NET_LOGIN_ID,

'x_tran_key' => AUTHORIZE_NET_TRANSACTION_KEY,

'x_version' => '3.1',

'x_test_request' => AUTHORIZE_NET_TEST_REQUEST,

'x_delim_data' => 'TRUE',

'x_delim_char' => '|',

'x_relay_response' => 'FALSE');

$request = array_merge($request_init, $request);
```

Dane tablicy są scalane w ciąg nazwa-wartość, który można wysłać do Authorize.net. Wartości są kodowane w celu włączenia do adresu URL za pomocą funkcji urlencode():

```
foreach($request as $key => $value )

$this->_mRequest .= $key . '=' . urlencode($value) . '&';

}
```

Metoda GetResponse() AuthorizeNetRequest wykonuje rzeczywiste żądanie przy użyciu biblioteki cURL.

```
// Send an HTTP POST request to Authorize.net using cURL

public function GetResponse()

{

...

// Perform a cURL session

$result = curl_exec($ch);

// Close a cURL session

curl_close ($ch);
```

```
// Return the response

return $result;

}

}

?>
```

Podczas wykonywania funkcji GetResponse() w celu wykonania transakcji AUTH_ONLY, odpowiedź będzie zawierać identyfikator transakcji. Jeśli autoryzacja się powiedzie, możemy wykorzystać ten identyfikator transakcji do wykonania transakcji PRIOR_AUTH_CAPTURE, która skutecznie pobiera pieniądze z konta klienta. Jak wyjaśniono wcześniej, odpowiedź z Authorize.net ma postać ciągu znaków, który zawiera wartości oddzielone konfigurowalnym znakiem, który w naszym przypadku jest znakiem potoku (|). Aby odczytać określoną wartość z ciągu, przekształcamy ciąg w tablicę za pomocą funkcji PHP explode() (http://www.php.net/manual/en/function.explode.php):

```
$auth_only_response = $request->GetResponse();

$_SESSION['auth_only_response'] = $auth_only_response;

$auth_only_response = explode('|', $auth_only_response);
```

Po wykonaniu tego fragmentu kodu $auth_only_response będzie zawierać tablicę, której elementami są wartości oddzielone znakiem potoku w oryginalnym ciągu. Z tej tablicy interesuje nas siódmy element, którym zgodnie z dokumentacją Authorize.net jest identyfikator transakcji (przeczytaj szczegóły Gateway Response API z http://www.authorize.net/support/AIM_guide.pdf aby uzyskać szczegółowe informacje na temat odpowiedzi Authorize.net). // Odczytaj identyfikator transakcji, który będzie niezbędny do pobrania płatności

```
$ref_trans_id = $auth_only_response[6];
```

Uwaga: Tablica $auth_only_response utworzona przez explode() jest liczona od zera, więc $auth_only_response[6] reprezentuje siódmy element tablicy.

Kod, który pobiera pieniądze przy użyciu tego identyfikatora transakcji, jest prosty. Ponieważ transakcja została już autoryzowana, wystarczy podać identyfikator transakcji otrzymany po autoryzacji, aby zakończyć transakcję:

```
// Capture

$transaction =

array ('x_ref_trans_id' => $ref_trans_id, // Transaction id

'x_method' => 'CC', // Payment method

'x_type' => 'PRIOR_AUTH_CAPTURE'); // Transaction type

$request->SetRequest($transaction);

$prior_auth_capture_response = $request->GetResponse();
```

Integracja Authorize.net z TShirtShop

Podobnie jak w przypadku DataCash, będziemy musieli zmodyfikować klasy PsCheckFunds i PsTakePayment, aby korzystać z nowej funkcjonalności Authorize.net. Pamiętaj, że możesz użyć plików z sekcji Kod źródłowy/Pobieranie na stronie internetowej Apress (http://www.apress.com/) zamiast wpisywać kod samodzielnie. Ostatnie modyfikacje polegają na zmianie klas sekcji rurociągu, które zajmują się transakcjami kartami kredytowymi (PsCheckFunds i PsTakePayment). Dodaliśmy już infrastrukturę do przechowywania i pobierania kodu uwierzytelniającego oraz informacji referencyjnych za pomocą metody OrderProcessor::SetOrderAuthCodeAndReference().

Ćwiczenie: Implementacja klas potoku zamówień

1. Najpierw zmodyfikuj business/ps_check_funds.php, aby działał z Authorize.net. Możesz wykonać kopię zapasową wersji DataCash tego pliku, jeśli wykonałeś ćwiczenie DataCash we wcześniejszej części działu.

```
<?php

class PsCheckFunds implements IPipelineSection

{

public function Process($processor)

{

// Audit

$processor->CreateAudit('PsCheckFunds started.', 20100);

$order_total_cost = $processor->mOrderInfo['total_amount'];

$order_total_cost += $processor->mOrderInfo['shipping_cost'];

$order_total_cost +=

round((float)$order_total_cost *

(float)$this->mOrderInfo['tax_percentage'], 2) / 100.00;

$exp_date = str_replace('/', '',

$processor->mCustomerInfo['credit_card']->ExpiryDate);

$transaction =

array (

'x_invoice_num' => $processor->mOrderInfo['order_id'],

'x_amount' => $order_total_cost, // Amount to charge

'x_card_num' => $processor->mCustomerInfo['credit_card']->CardNumber,

'x_exp_date' => $exp_date, // Expiry (MMYY)

'x_method' => 'CC',

'x_type' => 'AUTH_ONLY');

// Process Transaction
```

```
$request = new AuthorizeNetRequest(AUTHORIZE_NET_URL);
$request->SetRequest($transaction);
$response = $request->GetResponse();
$response = explode('|', $response);
if ($response[0] == 1)
{
$processor->SetAuthCodeAndReference($response[4], $response[6]);
// Audit
$processor->CreateAudit('Funds available for purchase.', 20102);
// Update order status
$processor->UpdateOrderStatus(2);
// Continue processing
$processor->mContinueNow = true;
}
else
{
// Audit
$processor->CreateAudit('Funds not available for purchase.', 20103);
throw new Exception('Credit card check funds failed for order ' .
$processor->mOrderInfo['order_id'] . ".\n\n" .
'Data exchanged:' . "\n" .
var_export($transaction, true) . "\n" .
var_export($response, true));
}
// Audit
$processor->CreateAudit('PsCheckFunds finished.', 20101);
}
}
?>
```

2. Zmodyfikuj business/ps_take_payment.php w następujący sposób:

```
<?php
```

```
class PsTakePayment implements IPipelineSection
{
public function Process($processor)
{
// Audit
$processor->CreateAudit('PsTakePayment started.', 20400);
$transaction =
array ('x_ref_trans_id' => $processor->mOrderInfo['reference'],
'x_method' => 'CC',
'x_type' => 'PRIOR_AUTH_CAPTURE');
// Process Transaction
$request = new AuthorizeNetRequest(AUTHORIZE_NET_URL);
$request->SetRequest($transaction);
$response = $request->GetResponse();
$response = explode('|', $response);
if ($response[0] == 1)
{
// Audit
$processor->CreateAudit(
'Funds deducted from customer credit card account.', 20402);
// Update order status
$processor->UpdateOrderStatus(5);
// Continue processing
$processor->mContinueNow = true;
// Audit
$processor->CreateAudit('PsTakePayment finished.', 20401);
}
else
{
// Audit
$processor->CreateAudit(
```

'Error taking funds from customer credit card.', 20403);

throw new Exception('Credit card take payment failed for order ' .

$processor->mOrderInfo['order_id'] . ".\n\n" .

'Data exchanged:' . "\n" .

var_export($transaction, true) . "\n" .

var_export($response, true));

}

}

}

?>

3. Dodaj odwołanie do pliku business/authorize_net_request.php w index.php, jak zaznaczono:

require_once BUSINESS_DIR . 'ps_check_funds.php';

require_once BUSINESS_DIR . 'ps_check_stock.php';

require_once BUSINESS_DIR . **'authorize_net_request.php'**;

4. Dodaj odwołanie do pliku business/authorize_net_request.php w admin.php, jak zaznaczono:

require_once BUSINESS_DIR . 'ps_ship_ok.php';

require_once BUSINESS_DIR . 'ps_final_notification.php';

require_once BUSINESS_DIR . **'authorize_net_request.php'**;

Testowanie integracji Authorize.net

Jedyne, co musimy teraz zrobić, to przeprowadzić testy na naszej nowej stronie internetowej. Pobierz listę magicznych numerów kart kredytowych Authorize.net z Przewodnika implementacji AIM i poeksperymentuj z wykonywaniem transakcji z ich użyciem.

Podsumowanie

W tej Części zakończyliśmy naszą aplikację e-commerce integrując ją z autoryzacją kart kredytowych. Gdy już umieścisz własne produkty w bazie danych, połączysz je z dostawcami, uzyskasz konto bankowe sprzedawcy i umieścisz je w sieci, jesteś gotowy do pracy! OK, więc to wciąż sporo pracy, ale żadna z nich nie jest szczególnie trudna. Ciężka praca już za Tobą! W szczególności tu przyjrzeliśmy się teorii leżącej u podstaw transakcji kartami kredytowymi w sieci oraz jednej pełnej implementacji - DataCash. Stworzyliśmy bibliotekę umożliwiającą dostęp do DataCash i zintegrowaliśmy ją z naszą aplikacją. Przyjrzeliśmy się również Authorize.net i stworzyliśmy kod, który testuje transakcje kartami kredytowymi przetwarzane przez Authorize.net