Wdrażanie funkcji AJAX

Ulepszymy nasz w pełni funkcjonalny koszyk zakupów, korzystając z technologii, która pojawiła się na pierwszych stronach gazet w 2005 roku. Ta technologia nazywa się AJAX i pozwala na uczynienie aplikacji internetowych łatwiejszymi i przyjemniejszymi w użyciu dla odwiedzających. Chociaż AJAX nie jest wielkim tematem, przynajmniej w porównaniu z innymi dziedzinami programowania, nie ma też trywialnej krzywej uczenia się. Przeprowadzimy Cię przez bardzo zwięzły samouczek AJAX, a następnie zaktualizujemy TShirtShop aby

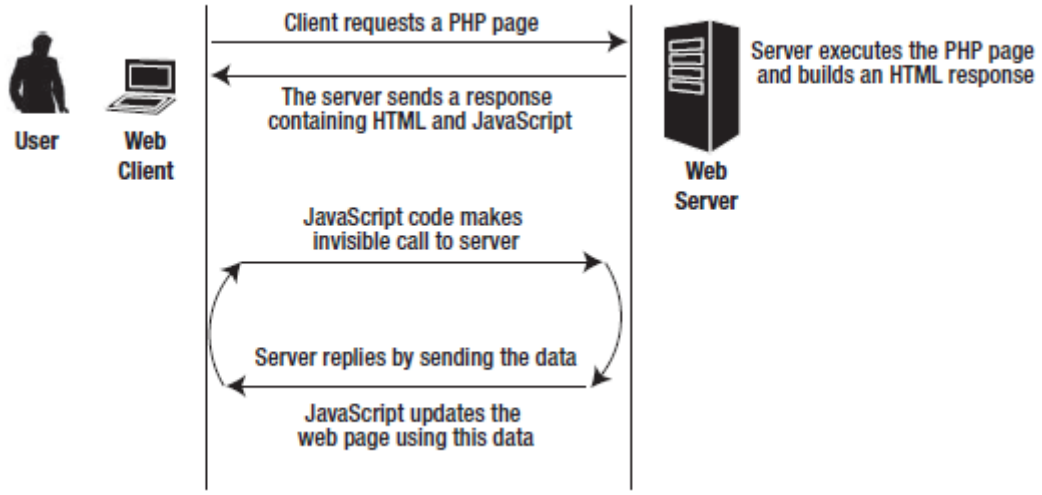
• Zezwalał na dodawanie nowych produktów do koszyka bez całkowitego przeładowania strony internetowej po kliknięciu przycisków Dodaj do koszyka. W ten sposób kliknięcie przycisku Dodaj do koszyka znacznie szybciej zaktualizuje przeskakujący koszyk, bez blokowania strony internetowej podczas wykonywania akcji.

• Poprawićł responsywność działań koszyka, takich jak zmiana ilości produktów, usuwanie produktów lub zapisywanie ich w sekcji „Zapisz na później".

Trzeba przyznać, że może to nie wydawać się dużymi ulepszeniami funkcji. Są to jednak miłe akcenty, które poprawiają wrażenia użytkownika w Twojej witrynie i mogą odróżnić nowoczesną witrynę internetową od tej, która wydaje się, że została stworzona w ubiegłym stuleciu.

Szybki start AJAX

Wielką zaletą, jaką AJAX może wnieść do aplikacji internetowej, jest to, że poprawia responsywność aplikacji. Responsywność to termin, którego używamy, gdy mówimy o tym, jak szybko aplikacja odpowiada na żądanie użytkownika. W świecie tworzenia stron internetowych obejmuje to czas potrzebny stronie internetowej (serwerowi) na odpowiedź na kliknięty link lub na zaktualizowanie strony o nowe dane. Termin AJAX został ukuty przez Jesse Jamesa Garretta w 2005 roku jako akronim Asynchronous JavaScript i XML. Co jest w nazwie? Jeśli mówisz „niewiele", zgadzamy się, przynajmniej w tym przypadku. AJAX to technika programowania, która wykorzystuje JavaScript po stronie klienta do wykonywania wywołań serwera w tle i pobierania dodatkowych danych w razie potrzeby, aktualizując niektóre części strony internetowej bez powodowania pełnego przeładowania strony. AJAX jest często przytaczany w kontekście Web 2.0 jako jedna z technologii pozwalających na tworzenie witryn internetowych nowej generacji, które zapewniają przyjazne środowisko ułatwiające dostęp do informacji online dzielenie się i współpraca. Technicznie jedyną „nową" rzeczą w AJAX jest nazwa. Technologie wymagane do tworzenia aplikacji internetowych AJAX istnieją już od jakiegoś czasu. Aby wykonać asynchroniczne wywołania serwera (wysyłać żądania serwera w tle), używamy obiektu XMLHttpRequest JavaScript (wraz z kilkoma innymi mniej znanymi technikami). Aby zaktualizować stronę odwiedzającego w odpowiedzi na dane pobrane z serwera, używamy Document Object Model (DOM) ostatecznie wraz z Cascading Style Sheets (CSS) dla nowoczesnej warstwy prezentacji. Aby zakodować informacje przekazywane tam i z powrotem z serwera, można użyć XML (format implikowany przez akronim AJAX) lub inne formaty danych, przy czym popularną alternatywą dla XML jest JavaScript Object Model (JSON). Omówimy każdy z nich krótko, ale najpierw pokażemy wizualną reprezentację działania strony internetowej obsługującej technologię AJAX.



JavaScript

JavaScript to język programowania obsługiwany przez wszystkie nowoczesne przeglądarki internetowe, co czyni go idealnym do implementacji różnych funkcji na poziomie klienta (przeglądarki internetowej). Kod JavaScript może być zawarty na stronach HTML (lub odwoływać się do niego) i może być wykonywany podczas wczytywania strony lub w przypadku wystąpienia określonych zdarzeń na stronie (takich jak kliknięcie przycisku). Jednym z pierwszych zastosowań JavaScript było dodawanie (mniej lub bardziej) przydatnych efektów graficznych do stron internetowych (pamiętasz padający śnieg i latające nietoperze?), ale w międzyczasie zarówno język, jak i umiejętności (i gusta) ludzi korzystanie z niego poprawiło się.

DOM

DOM to interfejs programowania aplikacji (API), który umożliwia odczytywanie i manipulowanie strukturami hierarchicznymi, takimi jak dokumenty HTML i XML. Większość języków zawiera komponent DOM; w kontekście programowania AJAX, interfejs DOM JavaScript pozwala czytać, parsować, zmieniać i tworzyć elementy HTML strony internetowej. Najczęściej po otrzymaniu odpowiedzi serwera na asynchroniczne połączenie, które wykonałeś, będziesz chciał odpowiednio zmodyfikować stronę, aby poinformować odwiedzającego o wynikach. JavaScript dostarcza domyślny obiekt dokumentu, który jest reprezentacją DOM bieżącej strony. Na przykład można użyć document.write(), aby dodać zawartość do elementu <body> strony, a document.getElementById(), aby pobrać element strony. Weźmy na przykład następujący fragment kodu, który edytuje zawartość elementu o nazwie koszyk-podsumowanie:

document.getElementById("podsumowanie koszyka").innerHTML = odpowiedź;

* **Przestroga** : Jest jeden oczywisty szczegół, o którym należy zawsze pamiętać podczas modyfikowania zawartości strony za pomocą JavaScript: klienci, którzy nie obsługują JavaScript, nie będą korzystać z nowych funkcji. Ważne funkcje strony internetowej nie mogą opierać się na włączonej obsłudze JavaScript; innymi słowy, implementacje muszą być degradowalne. Szczególnym przypadkiem odwiedzających, którzy nie wykonują kodu JavaScript, są roboty-pająki internetowe, takie jak te z Google, Yahoo lub MSN, indeksujące Twoją witrynę internetową. Nie możesz wyświetlać żadnych ważnych treści za pomocą JavaScript, ponieważ byłyby one całkowicie niewidoczne dla pająków internetowych, na co zwykle nie możesz sobie pozwolić, gdy prowadzisz biznes online.

Żądanie XMLHttp

XMLHttpRequest to obiekt, którego możesz użyć z kodu JavaScript do tworzenia asynchronicznych żądań serwera HTTP. Umożliwia to inicjowanie żądań HTTP i odbieranie odpowiedzi z serwera w tle, bez konieczności przesyłania strony do serwera przez użytkownika. Zazwyczaj po otrzymaniu odpowiedzi z serwera DOM jest używany do odpowiedniej zmiany strony. Pozwala to na zaimplementowanie responsywnej funkcjonalności i efektów wizualnych popartych danymi na żywo z serwera, bez znacznych zakłóceń wizualnych użytkownika. To jest AJAX. XMLHttpRequest został zaimplementowany przez Microsoft w 1999 roku jako obiekt ActiveX w Internet Explorerze i ostatecznie stał się de facto standardem dla wszystkich przeglądarek, a jako obiekt natywny jest obsługiwany przez wszystkie nowoczesne przeglądarki internetowe z wyjątkiem Internet Explorera 6. Zobaczmy, jak działa XMLHttpRequest . Musisz nauczyć się jak stworzyć obiekt, jak go zainicjować ustawiając odpowiednie parametry, jak uruchomić żądanie serwera i jak odczytywać wyniki.

Tworzenie obiektu XMLHttpRequest

W Internet Explorer 6 i starszych, XMLHttpRequest jest zaimplementowany jako formant ActiveX i tworzysz go w następujący sposób:

xmlHttp = new ActiveXObject("Microsoft.XMLHttp");

W Internet Explorerze 7 i innych przeglądarkach internetowych - w tym Firefox, Opera i Safari - XMLHttpRequest jest obiektem natywnym, więc tworzysz jego instancje w następujący sposób:

xmlHttp = nowe żądanie XMLHttp();

Aby utworzyć obiekty XMLHttpRequest dla wszystkich przeglądarek, użyjemy następującej funkcji JavaScript, która tworzy instancję XMLHttpRequest przy użyciu obiektu natywnego, jeśli jest dostępny, lub formantu Microsoft.XMLHttp ActiveX dla użytkowników korzystających z przeglądarki Internet Explorer 6 lub starszej:

```
// Creates an XMLHttpRequest instance

function createXmlHttpRequestObject()

{

// xmlHttp will store the reference to the XMLHttpRequest object

var xmlHttp;

// Try to instantiate the native XMLHttpRequest object

try

{

// Create an XMLHttpRequest object

xmlHttp = new XMLHttpRequest();

}

catch(e)

{

// Assume IE6 or older
```

```
try

{

xmlHttp = new ActiveXObject("Microsoft.XMLHttp");

}

catch(e) { }

}

// Return the created object or display an error message

if (!xmlHttp)

alert("Error creating the XMLHttpRequest object.");

else

return xmlHttp;

}
```

* **Uwaga**: Microsoft.XMLHttp jest najstarszą wersją biblioteki ActiveX XMLHttp, ale biblioteka ma o wiele więcej odmian i wersji, niż można sobie wyobrazić. Każde oprogramowanie firmy Microsoft, w tym Internet Explorer i MDAC, jest dostarczane z nowymi wersjami tego formantu ActiveX, z których każda ma własną nazwę. W naszym kodzie zaimplementujemy technikę, która automatycznie wykrywa najnowszą wersję XMLHttp zainstalowaną na komputerze użytkownika, ale tylko wtedy, gdy korzysta z Internet Explorera 6 lub starszego. W przeciwnym razie nadal używany jest natywny obiekt XMLHttpRequest.

createXmlHttpRequestObject() używa konstrukcji JavaScript try-catch, która jest potężną techniką obsługi wyjątków, która została początkowo zaimplementowana w językach programowania obiektowego (OOP), a teraz jest również obsługiwana przez PHP 5. Wyjątek to obiekt, który reprezentuje błąd co zdarzyło się w kodzie w czasie wykonywania. Wyjątki mogą być przechwytywane i rozwiązywane , w którym to przypadku nie blokują wykonania kodu, gdy zostaną zgłoszone.

* **Uwaga** :Wyjątek propaguje przez stos wywołań programu, od miejsca, w którym został wywołany do środowiska, w którym działa. Stos wywołań to lista wykonywanych metod. Więc jeśli funkcja A() wywołuje funkcję B(), która z kolei wywołuje funkcję C(), to stos wywołań zostanie utworzony ze wszystkich trzech metod. Jeśli w C() wystąpi wyjątek, możesz go obsłużyć za pomocą bloku try-catch w funkcji C(), zanim wyjątek zostanie przekazany z funkcji. Jeśli wyjątek nie zostanie przechwycony i obsłużony w C(), propaguje się do B() i tak dalej. Ostatnią warstwą jest przeglądarka internetowa, która może wyświetlać nieprzyjemny komunikat o błędzie odwiedzającemu, w zależności od tego, jak jest skonfigurowana.

Korzystając ze składni try-catch, możesz przechwycić wyjątek i obsłużyć go lokalnie, aby błąd nie został rozpowszechniony w przeglądarce użytkownika. Składnia try-catch jest następująca:

```
try

{

// Code that might generate an exception
```

```
}

catch (e)

{

// Execution is passed to this block when an exception is thrown in the try block

// (exception details are available through the e parameter)

}
```

Jeśli wystąpi błąd podczas wykonywania kodu w bloku try, wykonanie jest przekazywane natychmiast do bloku catch. Jeśli w bloku try nie wystąpi błąd, kod w bloku catch nigdy nie zostanie wykonany. Sposób obsługi każdego wyjątku zależy w dużej mierze od zaistniałej sytuacji. Czasami po prostu zignorujesz błąd; innym razem oznaczysz to jakoś w kodzie lub wyświetlisz komunikat o błędzie odwiedzającemu. W naszym konkretnym przypadku, gdy chcemy utworzyć obiekt XMLHttpRequest, najpierw spróbujemy utworzyć obiekt tak, jakby był natywnym obiektem przeglądarki, w następujący sposób:

```
// Try to instantiate the native XMLHttpRequest object

try

{

// Create an XMLHttpRequest object

xmlHttp = new XMLHttpRequest();

}
```

Internet Explorer 7, Firefox, Opera, Safari i inne przeglądarki wykonają ten fragment kodu dobrze i nie zostanie wygenerowany żaden błąd, ponieważ XMLHttpRequest jest obsługiwany natywnie. Jednak Internet Explorer 6 i jego starsze wersje nie rozpoznają obiektu XMLHttpRequest, zostanie wygenerowany wyjątek, a wykonanie zostanie przekazane do bloku catch. W przypadku przeglądarki Internet Explorer 6 i starszych wersji obiekt XMLHttpRequest należy utworzyć jako kontrolkę ActiveX:

```
catch(e)

{

// Assume IE6 or older

try

{

xmlHttp = new ActiveXObject("Microsoft.XMLHttp");

}

catch(e) { }

}
```

Istnieje więcej technik tworzenia obiektu XMLHttpRequest. Na przykład inną techniką jest użycie funkcji JavaScript zwanej wykrywaniem obiektów. Ta funkcja pozwala sprawdzić, czy dany obiekt jest obsługiwany przez przeglądarkę i działa tak:

```
if (window.XMLHttpRequest)

{

xmlHttp = new XMLHttpRequest();

}
```

Na końcu createXmlHttpRequestObject sprawdzamy, czy xmlHttp zawiera poprawną instancję XMLHttpRequest:

```
// Return the created object or display an error message

if (!xmlHttp)

alert("Error creating the XMLHttpRequest object.");

else

return xmlHttp;
```

Tutaj użyliśmy odwrotnego efektu funkcji wykrywania obiektów JavaScript, która mówi, że JavaScript oceni poprawną instancję obiektu, taką jak xmlHttp, jako true. Negacja tego wyrażenia--!xmlHttp-zwraca wartość true, jeśli xmlHttp ma wartość false, null lub jest niezdefiniowana. (W JavaScript, undefined to specjalna „wartość", która jest automatycznie przypisywana do zmiennej lub obiektu, któremu nie przypisano wartości.)

Korzystanie z XMLHttpRequest

Po utworzeniu obiektu XMLHttpRequest można zacząć używać go do inicjowania synchronicznych lub asynchronicznych żądań serwera. Tabela opisuje publiczne metody i właściwości tego obiektu.

Metoda/właściwość: Opis

abort : Zatrzymuje bieżące żądanie

getAllResponseHeaders() : Zwraca nagłówki odpowiedzi jako ciąg

getResponseHeader("headerLabel") : Zwraca pojedynczy nagłówek odpowiedzi jako ciąg

open("method", "URL"[, asyncFlag[, "userName"[, "password"]]]) : Inicjuje parametry żądania

send(content) : Wykonuje żądanie HTTP

setRequestHeader("label", "value") : Ustawia nagłówek żądania HTTP

onreadystatechange : Ustawia funkcję zwrotną, która obsługuje zmiany stanu żądania

readyState :Zwraca status żądania:

0 = niezainicjowany

1 = ładowanie

2 = załadowany

3 = interaktywny

4 = kompletny

responseText : Zwraca odpowiedź serwera jako ciąg

responseXml : Zwraca odpowiedź serwera jako dokument XML, którym można manipulować za pomocą funkcji DOM JavaScript

status : Zwraca kod statusu żądania

statusText : Zwraca komunikat o stanie żądania

Metody, których będziesz używać z każdym żądaniem serwera, to open() i send(). Metoda open() konfiguruje żądanie bez jego wysyłania. Posiada dwa wymagane parametry i kilka opcjonalnych. Pierwszy parametr określa metodę używaną do wysyłania danych do strony serwera; zwykłe wartości to GET i POST. Drugim parametrem jest adres URL, który określa, gdzie chcesz wysłać żądanie. Adres URL może być kompletny lub względny. Jeśli adres URL nie określa zasobu dostępnego przez HTTP, pierwszy parametr jest ignorowany. Trzeci parametr funkcji open(), zwany async, określa, czy żądanie powinno być obsługiwane asynchronicznie. prawda oznacza, że przetwarzanie kodu jest kontynuowane po zwróceniu funkcji send() bez oczekiwania na odpowiedź z serwera; false oznacza, że skrypt czeka na odpowiedź przed kontynuowaniem przetwarzania. Aby włączyć przetwarzanie asynchroniczne (które jest sercem mechanizmu AJAX), musisz ustawić async na true i obsłużyć zdarzenie onreadystatechange w celu przetworzenia odpowiedzi z serwera. Po skonfigurowaniu połączenia za pomocą open(), wykonujesz je za pomocą send(). Wykonując żądanie GET, wysyłasz parametry za pomocą ciągu zapytania URL, jak w http://localhost/ajax/test.php?param1=x&param2=y. To żądanie serwera przekazuje do serwera dwa parametry - parametr o nazwie param1 o wartości x oraz parametr o nazwie param2 o wartości y:

```
// Call the server to execute the server-side operation

xmlHttp.open("GET", "http://localhost/ajax/test.php?param1=x&param2=y", true);

xmlHttp.onreadystatechange = handleRequestStateChange;

xmlHttp.send(null);
```

When using POST, you send the query string as a parameter of the send() method, instead

of joining it on to the base URL, like this:

```
// Call the server page to execute the server-side operation

xmlHttp.open("POST", "http://localhost/ajax/test.php", true);

xmlHttp.onreadystatechange = handleRequestStateChange;

xmlHttp.send("param1=x&param2=y");
```

Dwie próbki kodu powinny mieć ten sam efekt. W praktyce istnieje kilka różnic między POST i GET, o których powinieneś wiedzieć:

• Korzystanie z GET może pomóc w debugowaniu, ponieważ można symulować żądania GET za pomocą przeglądarki internetowej, dzięki czemu można łatwo zobaczyć na własne oczy, co generuje skrypt serwera.

• POST jest wymagany podczas wysyłania dużych bloków danych, których GET nie może obsłużyć.

• GET służy do pobierania danych z serwera, natomiast POST służy do wprowadzania zmian. W prawdziwym świecie dobrze jest przestrzegać tych zasad; w przeciwnym razie mogą się zdarzyć dziwne rzeczy. Na przykład wyszukiwarki wysyłają żądania GET, aby odczytać dane z sieci, ale nigdy nie publikują żadnych danych. Jeśli użyjesz GET do przesyłania zmian, a wyszukiwarka dowie się o adresie skryptu serwera, wyszukiwarka może zacząć modyfikować Twoje dane - a na pewno tego nie chcesz!

Minimalna implementacja funkcji o nazwie process(), która wykonuje asynchroniczne wywołania serwera za pomocą GET, wygląda tak:

```
function process()

{

// Call the server to execute the server-side operation

xmlHttp.open("GET", "server_script.php", true);

xmlHttp.onreadystatechange = handleRequestStateChange;

xmlHttp.send(null);

}
```

Ta implementacja ma następujące potencjalne problemy:

• process() może zostać wykonany nawet jeśli xmlHttp nie zawiera poprawnej instancji XMLHttpRequest. Może się tak zdarzyć, jeśli na przykład przeglądarka użytkownika nie obsługuje XMLHttpRequest. Spowodowałoby to wystąpienie nieobsłużonego wyjątku. Nasze inne wysiłki w radzeniu sobie z błędami nie pomagają zbytnio, jeśli nie jesteśmy konsekwentni i robimy coś również z funkcją procesu.

• process() nie jest chroniony przed innymi rodzajami błędów, które mogą się zdarzyć. Na przykład, jak zobaczysz w dalszej części tego rozdziału, niektóre przeglądarki wygenerują wyjątek bezpieczeństwa, jeśli nie spodoba im się serwer, do którego chcesz uzyskać dostęp za pomocą obiektu XMLHttpRequest.

Lepsza wersja process() wygląda tak:

```
// Performs a server request and assigns a callback function

function process()

{

// Continue only if xmlHttp isn't void

if (xmlHttp)

{

// Try to connect to the server

try
```

```
{
// Initiate server request
xmlHttp.open("GET", "server_script.php", true);
xmlHttp.onreadystatechange = handleRequestStateChange;
xmlHttp.send(null);
}
// Display an error in case of failure
catch (e)
{
alert("Can't connect to server:\n" + e.toString());
}
}
}
```

Jeśli xmlHttp ma wartość null (lub false), nie wyświetlamy kolejnego komunikatu o błędzie, ponieważ zakładamy, że taki komunikat został już wyświetlony przez funkcję createXmlHttpRequestObject. Dbamy jednak o sygnalizowanie wszelkich innych problemów z połączeniem.

Obsługa odpowiedzi serwera

Podczas wykonywania żądania asynchronicznego, takiego jak we fragmentach kodu przedstawionych wcześniej, wykonanie xmlHttp.send() nie zawiesza się w oczekiwaniu na odpowiedź serwera; zamiast tego kod JavaScript kontynuuje wykonywanie. Pokazana wcześniej funkcja process() definiuje funkcję o nazwie handleRequestStateChange() jako metodę wywołania zwrotnego, która powinna obsługiwać zmiany stanu żądania. Właściwość readyState może mieć jedną z następujących wartości reprezentujących możliwe stany żądania:

0 = niezainicjowany

1 = ładowanie

2 = załadowany

3 = interaktywny

4 = kompletny

Zwykle metoda handleRequestStateChange() jest wywoływana wiele razy, gdy żądanie wchodzi w nowy etap, ale nie można polegać na przeglądarce internetowej, która podniesie zdarzenie readystatechange dla wszystkich wymienionych stanów z wyjątkiem stanu 4, który sygnalizuje zakończenie żądania. Nazwy stanów nie wymagają wyjaśnień, z wyjątkiem stanu 3. Stan interaktywny jest stanem pośrednim, gdy odpowiedź została częściowo odebrana. W naszych aplikacjach AJAX wykorzystamy tylko stan kompletny, co oznacza, że odpowiedź została w całości odebrana z serwera. Poniższy fragment kodu pokazuje typową implementację handleRequestStateChange i podkreśla fragment, w którym faktycznie można odczytać odpowiedź z serwera:

```
// Function executed when the state of the request changes
function handleRequestStateChange()
{
// Continue if the process is completed
if (xmlHttp.readyState == 4)
{
// Continue only if HTTP status is "OK"
if (xmlHttp.status == 200)
{
// Retrieve the response
response = xmlHttp.responseText;
// Do something with the response
// ...
}
}
}
```

Przed próbą odczytania odebranych danych weryfikujemy również, czy kod stanu odpowiedzi to 200. Wysłanie takiego kodu wskazuje, że stan żądania jest częścią protokołu HTTP, a 200 to kod stanu, który określa, że żądanie zostało zakończone pomyślnie. Inne popularne kody stanu HTTP to 404, co oznacza, że nie można znaleźć żądanego zasobu, oraz 500, co oznacza błąd serwera. Po raz kolejny możemy użyć bloków try-catch do obsługi błędów, które mogą wystąpić podczas nawiązywania połączenia z serwerem lub podczas odczytywania odpowiedzi z serwera. Ulepszona wersja funkcji handleRequestStateChange() wygląda tak:

```
// Function executed when the state of the request changes
function handleRequestStateChange()
{
// Continue if the process is completed
if (xmlHttp.readyState == 4)
{
// Continue only if HTTP status is "OK"
if (xmlHttp.status == 200)
{
try
```

```
{
// Retrieve the response
response = xmlHttp.responseText;
// Do something with the response
// ...
}
catch(e)
{
// Display error message
alert("Error reading the response: " + e.toString());
}
}
else
{
// Display status message
alert("There was a problem retrieving the data:\n" + xmlHttp.statusText);
}
}
}
```

W praktyce będziemy wyświetlać bardziej przyjazne dla użytkownika komunikaty o błędach. Co więcej, gdy tylko jest to możliwe, sprawimy, że nasz kod będzie degradowalny, aby w przypadku wystąpienia błędu w kodzie JavaScript funkcjonalność była wykonywana przy użyciu typowego formularza przesyłania. Zobaczmy jak.

Pisanie kodu degradowalnego

Idealnie, nowoczesna aplikacja internetowa powinna używać AJAX do ulepszania istniejących funkcji, gdy tylko jest to możliwe, przy jednoczesnym utrzymaniu funkcjonalności witryny nawet wtedy, gdy JavaScript nie jest dostępny. W naszym konkretnym przypadku chcemy umożliwić naszym odwiedzającym dodawanie produktów do swoich koszyków i edycję ich koszyków, nawet jeśli mają wyłączoną obsługę JavaScript lub ich przeglądarka internetowa nie obsługuje JavaScript. Gdy odwiedzający kliknie przycisk Dodaj do koszyka, najpierw próbujemy wykonać wywołanie AJAX w tle, aby dodać produkt do koszyka odwiedzającego i odpowiednio zaktualizować pole podsumowania koszyka. Jeśli wystąpi błąd podczas próby nawiązania połączenia lub jeśli JavaScript jest wyłączony, przesyłamy formularz na serwer i dodajemy produkt do koszyka odwiedzającego w sposób inny niż AJAX. W ten sposób zaoferujemy lepsze wrażenia z korzystania z Internetu większości naszych użytkowników, jednocześnie umożliwiając innym korzystanie z naszego sklepu. Jest to łatwiejsze do

wdrożenia niż się wydaje. Spójrz na poniższy kod, który reprezentuje linki AJAXified „Zapisz na później” i Usuń w koszyku.

mCartProducts[i].save}"

onclick="return executeCartAction(this);">Save for later

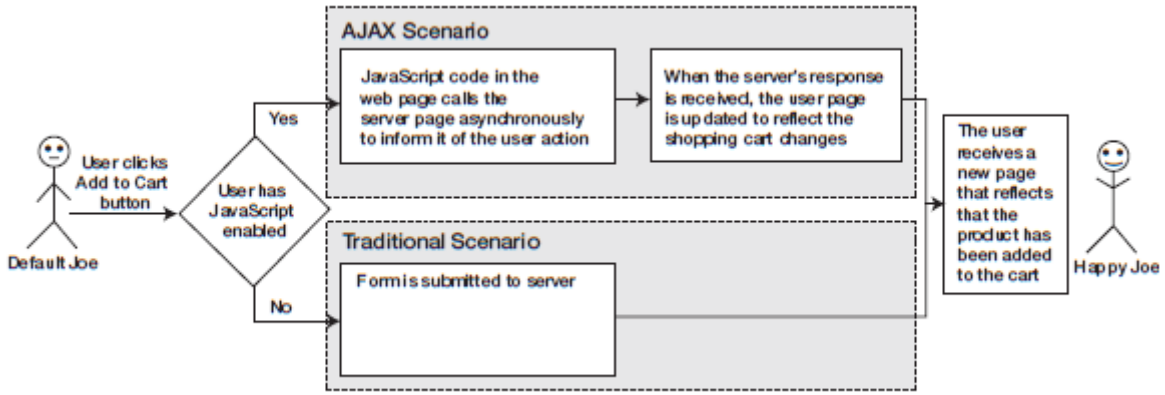
mCartProducts[i].remove}"

onclick="return executeCartAction(this);">Remove

W obu przypadkach w wyniku zdarzenia click wykonywana jest funkcja JavaScript. Jeśli ta funkcja zwróci wartość false, link nie jest już używany. Jeśli funkcja zwróci wartość true lub w ogóle nie zostanie wykonana, następuje kliknięcie linku i modyfikacja koszyka zakupów w sposób inny niż AJAX. Przyciski Dodaj do koszyka są obsługiwane w podobny sposób, z wyjątkiem tego, że przy wysłaniu formularza wywoływana jest funkcja JavaScript:

onsubmit="return addProductToCart(this);">

Rysunek opisuje, co się dzieje, gdy odwiedzający kliknie przycisk Dodaj do koszyka.



Proces jest podobny do aktualizacji elementów koszyka, którą również zaimplementujemy w tym rozdziale. Jak widać, bez względu na to, czy odwiedzający ma JavaScript, czy nie, osiągany jest ten sam efekt końcowy. Różnica polega na responsywności; w scenariuszu AJAX użytkownik nie musi czekać kilku sekund na ponowne załadowanie strony.

Czy AJAX jest zawsze odpowiedni?

Część teoretyczną rozdziału kończymy ostrzeżeniem: AJAX może poprawić wrażenia odwiedzających Twoją witrynę, ale może też je pogorszyć, gdy zostanie użyty w niewłaściwy sposób. Zazwyczaj AJAX najlepiej jest stosować w uzupełnieniu do tradycyjnych paradygmatów tworzenia stron internetowych, zamiast ich zmieniania lub zastępowania. Na przykład, o ile Twoja aplikacja nie ma naprawdę specjalnych wymagań, mądrze jest pozwolić użytkownikom na poruszanie się po treści za pomocą starych dobrych hiperłączy. Przeglądarki internetowe mają długą historię radzenia sobie z nawigacją treści, a użytkownicy sieci mają długą historię korzystania z funkcji nawigacyjnych przeglądarek. AJAX może przynieść Twoim projektom następujące korzyści:

• Umożliwia tworzenie lepszych i bardziej responsywnych aplikacji internetowych.

• Zachęca do opracowywania wzorców i struktur, które pomagają programistom uniknąć ponownego wymyślania koła podczas wykonywania typowych zadań.

• Wykorzystuje istniejące technologie i funkcje obsługiwane przez wszystkie nowoczesne przeglądarki internetowe.

• Wykorzystuje wiele istniejących umiejętności programistycznych.

Poniżej przedstawiono potencjalne problemy z AJAX:

• Jest niedostępny dla niektórych użytkowników: JavaScript może być wyłączony po stronie klienta, co sprawia, że funkcje AJAX nie działają.

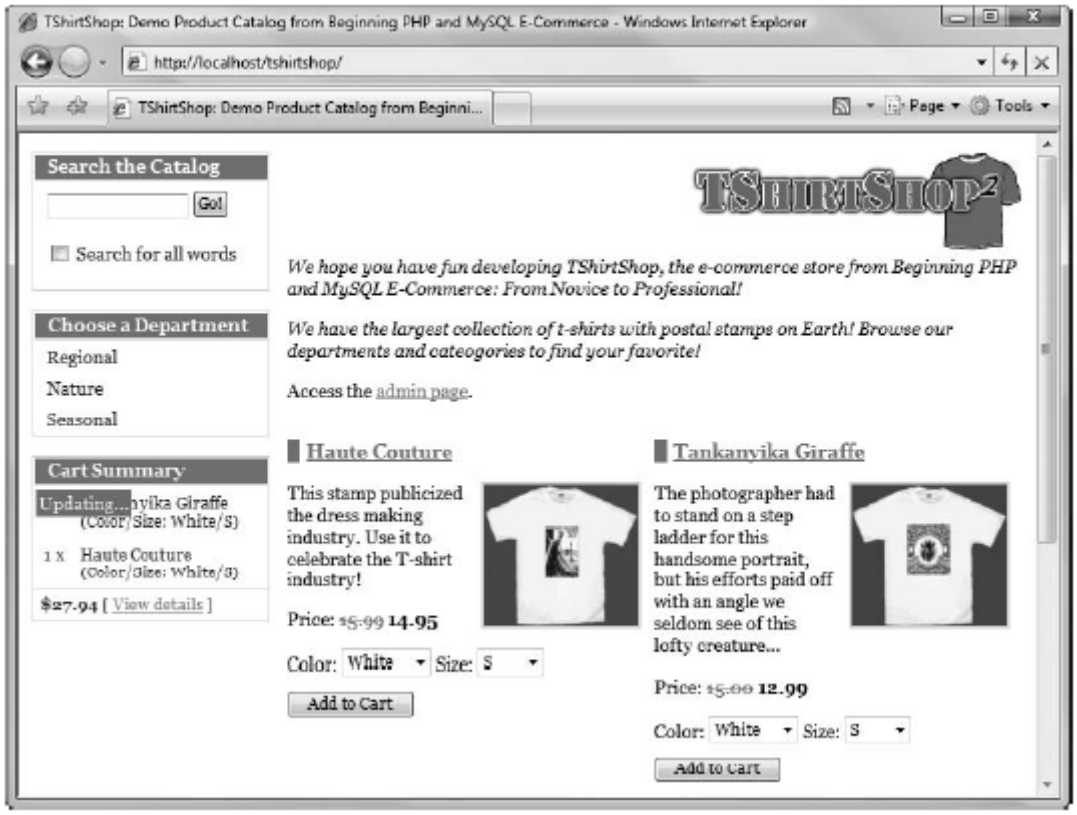
• Łatwo jest używać AJAX do niewłaściwych celów: zwiększona świadomość użyteczności, dostępności, standardów internetowych i optymalizacji pod kątem wyszukiwarek pomoże Ci podejmować lepsze decyzje podczas projektowania i wdrażania witryn internetowych.

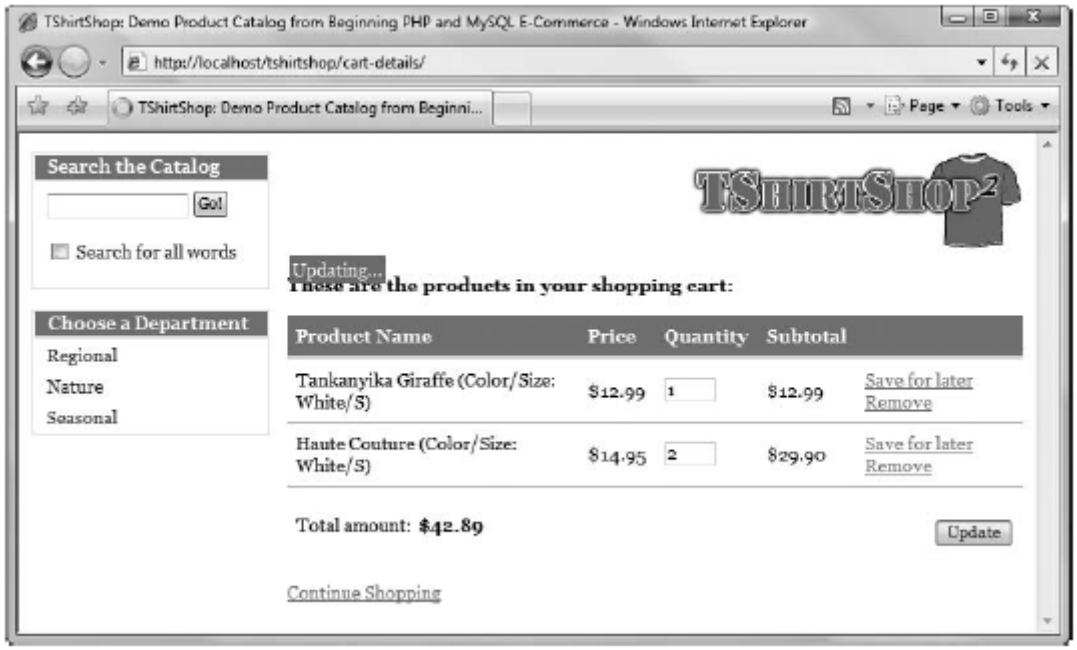
• Może sprawić, że treść będzie niewidoczna dla wyszukiwarek: Wyszukiwarki nie mogą indeksować treści dynamicznie generowanych przez JavaScript w aplikacji AJAX, ponieważ nie wykonują żadnego kodu JavaScript podczas indeksowania witryny internetowej. Jeśli optymalizacja pod kątem wyszukiwarek jest ważna dla Twojej witryny internetowej, nie powinieneś używać AJAX do dostarczania treści i nawigacji. W naszym przypadku nie chcemy, aby wyszukiwarki indeksowały stronę koszyka lub pole podsumowania koszyka, więc to ograniczenie nas nie dotyczy.

• Może to powodować problemy z tworzeniem zakładek w przeglądarce i nawigacją po stronach: Zazwyczaj aplikacje AJAX działają na stronie internetowej, której adres URL nie zmienia się w odpowiedzi na działania użytkownika, w którym to przypadku można dodać do zakładek tylko stronę wejściową. Przyciski Wstecz i Dalej w przeglądarkach nie dają takich samych wyników, jak w klasycznych witrynach internetowych, chyba że Twoja aplikacja AJAX jest specjalnie do tego zaprogramowana. Aby włączyć zakładki do stron AJAX oraz przyciski przeglądarki Wstecz i Dalej, możesz użyć frameworków, takich jak Really Simple. Ta struktura umożliwia tworzenie zakładek poprzez dynamiczne dodawanie kotwic strony przy użyciu kodu JavaScript, na przykład w http://www.example.com/my-ajax-app.html#Page2. Musisz również utworzyć kod pomocniczy, który ładuje i zapisuje stan Twojej aplikacji za pomocą parametru kotwicy. Jednak wspieranie tworzenia zakładów i nawigacji w ten sposób znacznie zwiększa złożoność kodu, co zwiększa ryzyko wystąpienia błędów i problemów ze zgodnością przeglądarek. Po usunięciu tych ostrzeżeń nadszedł czas, aby rozpocząć aktualizację TShirtShop o nowe funkcje AJAX.

Tworzenie koszyka zakupów AJAX

AJAXify koszyk zakupów wykonamy w dwóch krokach. Najpierw zaimplementujemy funkcje AJAX do przycisków Dodaj do koszyka, które aktualizują koszyk i pole podsumowania koszyka bez ponownego ładowania strony. Następnie ulepszymy stronę koszyka na zakupy. Rysunek 13-3 przedstawia okno podsumowania koszyka zakupów podczas aktualizacji po kliknięciu przycisku Dodaj do koszyka. Zwróć uwagę na tekst „Aktualizacja...”, który pojawia się w polu podsumowania koszyka, aby wskazać, że koszyk jest aktualizowany. Rysunek 13-4 przedstawia koszyk podczas wyjmowania produktu.





Ulepszenie funkcji Dodaj do koszyka za pomocą AJAX

Na rysunku powyżej widzieliście, co zamierzamy tutaj stworzyć. Implementacja jest stosunkowo prosta. Zaktualizujesz szablony product.tpl i products_list.tpl, aby wywołać funkcję JavaScript o nazwie addProductToCart() po kliknięciu przycisku Dodaj do koszyka w celu przesłania formularza. Następnie utworzysz ajax.js, dość duży plik JavaScript, który zawiera cały kod wymagany do obsługi nowej funkcjonalności klienta. Zawiera następujące funkcje:

• createXmlHttpRequestObject() tworzy obiekt XMLHttpRequest.

• handleError(message) wyświetla komunikaty o błędach generowane przez inne funkcje lub degraduje do klasycznego zachowania innego niż AJAX, w zależności od wartości zmiennej o nazwie showErrors.

• addProductToCart() jest wywoływana ze strony HTML w celu dodania produktu do koszyka.

• handleAddProductToCart() to funkcja zwrotna dla wywołań serwera Dodaj do koszyka.

• postAddProductToCartProcess() odczytuje odpowiedź serwera Dodaj do koszyka i aktualizuje pole podsumowania koszyka.

Zaktualizujesz również swój projekt w różnych miejscach, aby wykorzystać lub wspierać nowe funkcje. Przejdźmy przez poniższe ćwiczenie, a później omówimy ten proces bardziej szczegółowo.

Ćwiczenie: Szybsze dodawanie produktów

1. Otwórz Presentation\templates\product.tpl, dodaj podświetlony kod i zmień nazwę elementu input na add_to_cart:

```
{* The Add to Cart form *}
<form class="add-product-form" target="_self" method="post"
action="{$obj->mProduct.link_to_add_product}"
onsubmit="return addProductToCart(this);">
{* Generate the list of attribute values *}
<p class="attributes">
{* Parse the list of attributes and attribute values *}
{section name=k loop=$obj->mProduct.attributes}
...
{/section}
</p>
{* Add the submit button and close the form *}
<p>
<input type="submit" name="add_to_cart" value="Add to Cart" />
</p>
</form>
```

2. Otwórz prezentację\templates\products_list.tpl i zmodyfikuj ją, jak pokazano tutaj:

```
{* The Add to Cart form *}
<form class="add-product-form" target="_self" method="post"
action="{$obj->mProducts[k].link_to_add_product}"
```

```
onsubmit="return addProductToCart(this);">
{* Generate the list of attribute values *}
<p class="attributes">
{* Parse the list of attributes and attribute values *}
{section name=l loop=$obj->mProducts[k].attributes}
...
{/section}
</p>
{* Add the submit button and close the form *}
<p>
<input type="submit" name="add_to_cart" value="Add to Cart" />
</p>
</form>
```

3. Utwórz nowy folder w folderze głównym projektu (tshirtshop) o nazwie scripts, dodaj do niego nowy plik o nazwie ajax.js i wpisz następujący kod:

```
// Holds an instance of XMLHttpRequest
var xmlHttp = createXmlHttpRequestObject();
// Display error messages (true) or degrade to non-AJAX behavior (false)
var showErrors = true;
// Contains the link or form clicked or submitted by the visitor
var actionObject = '';
// Creates an XMLHttpRequest instance
function createXmlHttpRequestObject()
{
// Will store the XMLHttpRequest object
var xmlHttp;
// Create the XMLHttpRequest object
try
{
// Try to create native XMLHttpRequest object
xmlHttp = new XMLHttpRequest();
```

```
}
catch(e)
{
// Assume IE6 or older
var XmlHttpVersions = new Array(
"MSXML2.XMLHTTP.6.0", "MSXML2.XMLHTTP.5.0", "MSXML2.XMLHTTP.4.0",
"MSXML2.XMLHTTP.3.0", "MSXML2.XMLHTTP", "Microsoft.XMLHTTP");
// Try every id until one works
for (i = 0; i < XmlHttpVersions.length && !xmlHttp; i++)
{
try
{
// Try to create XMLHttpRequest object
xmlHttp = new ActiveXObject(XmlHttpVersions[i]);
}
catch (e) {} // Ignore potential error
}
}
// If the XMLHttpRequest object was created successfully, return it
if (xmlHttp)
{
return xmlHttp;
}
// If an error happened, pass it to handleError
else
{
handleError("Error creating the XMLHttpRequest object.");
}
}
// Displays an the error message or degrades to non-AJAX behavior
function handleError($message)
```

```
{
// Ignore errors if showErrors is false
if (showErrors)
{
// Display error message
alert("Error encountered: \n" + $message);
return false;
}
// Fall back to non-AJAX behavior
else if (!actionObject.tagName)
{
return true;
}
// Fall back to non-AJAX behavior by following the link
else if (actionObject.tagName == 'A')
{
window.location = actionObject.href;
}
// Fall back to non-AJAX behavior by submitting the form
else if (actionObject.tagName == 'FORM')
{
actionObject.submit();
}
}
// Adds a product to the shopping cart
function addProductToCart(form)
{
// Display "Updating" message
document.getElementById('updating').style.visibility = 'visible';
// Degrade to classical form submit if XMLHttpRequest is not available
if (!xmlHttp) return true;
```

```
// Create the URL we open asynchronously
request = form.action + '&AjaxRequest';
params = '';
// obtain selected attributes
formSelects = form.getElementsByTagName('SELECT');
if (formSelects)
{
for (i = 0; i < formSelects.length; i++)
{
params += '&' + formSelects[i].name + '=';
selected_index = formSelects[i].selectedIndex;
params += encodeURIComponent(formSelects[i][selected_index].text);
}
}
// Try to connect to the server
try
{
// Continue only if the XMLHttpRequest object isn't busy
if (xmlHttp.readyState == 4 || xmlHttp.readyState == 0)
{
// Make a server request to validate the extracted data
xmlHttp.open("POST", request, true);
xmlHttp.setRequestHeader("Content-Type",
"application/x-www-form-urlencoded");
xmlHttp.onreadystatechange = addToCartStateChange;
xmlHttp.send(params);
}
}
catch (e)
{
// Handle error
```

```
handleError(e.toString());
}
// Stop classical form submit if AJAX action succeeded
return false;
}
// Function that retrieves the HTTP response
function addToCartStateChange()
{
// When readyState is 4, we also read the server response
if (xmlHttp.readyState == 4)
{
// Continue only if HTTP status is "OK"
if (xmlHttp.status == 200)
{
try
{
updateCartSummary();
}
catch (e)
{
handleError(e.toString());
}
}
else
{
handleError(xmlHttp.statusText);
}
}
}
// Process server's response
function updateCartSummary()
```

```
{
// Read the response
response = xmlHttp.responseText;
// Server error?
if (response.indexOf("ERRNO") >= 0 || response.indexOf("error") >= 0)
{
handleError(response);
}
else
{
// Extract the contents of the cart_summary div element
var cartSummaryRegEx = /^<div class="box" id="cart-summary">➥
([\s\S]*)<\/div>$/m;
matches = cartSummaryRegEx.exec(response);
response = matches[1];
// Update the cart summary box and hide the Loading message
document.getElementById("cart-summary").innerHTML = response;
// Hide the "Updating..." message
document.getElementById('updating').style.visibility = 'hidden';
}
}
```

4. Open presentation\templates\store_front.tpl, and add a reference to your JavaScript file, ajax.js:

```
<html>
<head>
<title>{$obj->mPageTitle}</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<link href="{$obj->mSiteUrl}tshirtshop.css" type="text/css"
rel="stylesheet" />
<script type="text/javascript"
src="{$obj->mSiteUrl}scripts/ajax.js"></script>
```

</head>

5. Zmodyfikuj index.php tak jak zaznaczono:

```
// Load Business Tier
require_once BUSINESS_DIR . 'catalog.php';
require_once BUSINESS_DIR . 'shopping_cart.php';
// URL correction
Link::CheckRequest();
// Load Smarty template file
$application = new Application();
// Handle AJAX requests
if (isset ($_GET['AjaxRequest']))
{
// Headers are sent to prevent browsers from caching
header('Expires: Fri, 25 Dec 1980 00:00:00 GMT'); // Time in the past
header('Last-Modified: ' . gmdate('D, d M Y H:i:s') . ' GMT');
header('Cache-Control: no-cache, must-revalidate');
header('Pragma: no-cache');
header('Content-Type: text/html');
if (isset ($_GET['CartAction']))
{
$cart_action = $_GET['CartAction'];
if ($cart_action == ADD_PRODUCT)
{
require_once PRESENTATION_DIR . 'cart_details.php';
$cart_details = new CartDetails();
$cart_details->init();
$application->display('cart_summary.tpl');
}
}
else
trigger_error('CartAction not set', E_USER_ERROR);
```

else

{

// Display the page

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

}

// Close database connection

DatabaseHandler::Close();

6. Otwórz Presentation\cart_details.php i dokonaj następujących zmian w bloku switch w metodzie init():

if (count($selected_attributes) > 0)

$attributes = implode('/', $selected_attributes) . ': ' .

implode('/', $selected_attribute_values);

ShoppingCart::AddProduct($this->_mItemId, $attributes);

if (!isset ($_GET['AjaxRequest']))

header('Location: ' . $this->mLinkToContinueShopping);

else

return;

break;

case REMOVE_PRODUCT:

ShoppingCart::RemoveProduct($this->_mItemId);

if (!isset ($_GET['AjaxRequest']))

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

break;

case UPDATE_PRODUCTS_QUANTITIES:

for($i = 0; $i < count($_POST['itemId']); $i++)

ShoppingCart::Update($_POST['itemId'][$i], $_POST['quantity'][$i]);

if (!isset ($_GET['AjaxRequest']))

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

break;

case SAVE_PRODUCT_FOR_LATER:

```
ShoppingCart::SaveProductForLater($this->_mItemId);
if (!isset ($_GET['AjaxRequest']))
header('Location: ' . Link::ToCart());
break;
case MOVE_PRODUCT_TO_CART:
ShoppingCart::MoveProductToCart($this->_mItemId);
if (!isset ($_GET['AjaxRequest']))
header('Location: ' . Link::ToCart());
break;
default:
// Do nothing
break;
}
```

7. Zmodyfikuj metodę CheckRequest() klasy Link w link.php, tak jak zaznaczono. Nie sprawdzamy poprawności adresu URL podczas odpowiadania na żądanie AJAX.

```
// Redirects to proper URL if not already there
public static function CheckRequest()
{
$proper_url = '';
if (isset ($_GET['Search']) || isset($_GET['SearchResults']) ||
isset ($_GET['CartAction']) || isset ($_GET['AjaxRequest']))
{
return ;
}
```

8. Otwórz Presentation\templates\cart_summary.tpl i wprowadź zmiany pokazane tutaj:

```
{* cart_summary.tpl *}
{load_presentation_object filename="cart_summary" assign="obj"}
{* Start cart summary *}
<div class="box" id="cart-summary">
<p class="box-title">Cart Summary</p>
<div id="updating">Updating...</div>
```

{if $obj->mEmptyCart}

Your shopping cart is empty!

{else}

9. Dodaj następującą definicję stylu do pliku tshirtshop.css z folderu stylów:

#updating {

background-color: #ff0000;

border: none;

color: #ffffff;

margin: 2px;

padding: 2px;

visibility: hidden;

position: absolute;

width: 70px;

}

10. Załaduj swój katalog i upewnij się, że przyciski Dodaj do koszyka działają poprawnie przy użyciu nowego kodu AJAX.

Jak to działa

Zmieniliśmy nazwę przesyłanego elementu HTML na add_to_cart, dzięki czemu możemy przesłać go z JavaScript

{/section}

{* Add the submit button and close the form *}

<input type="submit" name="add_to_cart" value="Add to Cart" />

</form>

Mamy zmienną o nazwie showErrors, której wartość wpływa na sposób obsługi błędów. Jeśli ta zmienna ma wartość true, komunikaty błędów debugowania są wyświetlane za pomocą alert(). Nie jest to bardzo przyjazna dla użytkownika metoda obsługi błędów, ale pomaga w debugowaniu. Jeśli showErrors ma wartość false, nie są wyświetlane żadne błędy. Zamiast tego, gdy wystąpi błąd JavaScript, zachowanie jest degradowane do klasycznego formularza submit.

// Display error messages (true) or degrade to non-AJAX behavior (false)

var showErrors = true;

Ta zmienna jest używana w funkcji handleError(), która jest wywoływana za każdym razem, gdy wystąpi błąd w kodzie JavaScript. Jeśli showErrors ma wartość true, ta funkcja wyświetla po prostu szczegóły błędu:

```
// Displays an error message or degrades to non-AJAX behavior

function handleError($message)

{

// Ignore errors if showErrors is false

if (showErrors)

{

// Display error message

alert("Error encountered: \n" + $message);

return false;

}
```

Jeśli showErrors ma wartość false, nie wyświetlamy szczegółów błędu. Zamiast tego próbujemy wykonać akcję żądaną przez odwiedzającego w sposób inny niż AJAX. Tutaj używamy actionObject, który reprezentuje formularz przesłany przez odwiedzającego lub link akcji kliknięty przez odwiedzającego. Jeśli actionObject nie jest ustawione, oznacza to, że wystąpił błąd w funkcji JavaScript, do której odwołuje się atrybut onclick lub onsubmit. W takim przypadku musimy po prostu zwrócić true, co spowoduje, że zostanie wykonana pierwotna żądana akcja:

```
// Fall back to non-AJAX behavior

else if (!actionObject.tagName)

{

return true;

}
```

Jeśli actionObject jest linkiem, oznacza to, że błąd wystąpił po kliknięciu linku przez użytkownika, w którym to przypadku przekierowujemy żądanie na ten adres URL:

```
// Fall back to non-AJAX behavior by following the link

else if (actionObject.tagName == 'A')

{

window.location = actionObject.href;

}
```

Jeśli actionObject jest formularzem, oznacza to, że błąd wystąpił po przesłaniu formularza przez użytkownika, w którym to przypadku przekierowujemy żądanie na ten adres URL:

```
// Fall back to non-AJAX behavior by submitting the form
```

```
else if (actionObject.tagName == 'FORM')

{

actionObject.submit();

}

}
```

Skrypt index.php został zaktualizowany, aby odpowiadał również na żądania AJAX. Podczas tworzenia asynchronicznego żądania do index.php, parametr AjaxRequest jest dodawany do ciągu zapytania, aby index.php wiedział, że ma odpowiednio zareagować. Najpierw ustawia odpowiednie wartości nagłówka, aby upewnić się, że odpowiedź nie jest buforowana:

```
if (isset ($_GET['AjaxRequest']))

{

// Headers are sent to prevent browsers from caching

header('Expires: Fri, 25 Dec 1980 00:00:00 GMT'); // Time in the past

header('Last-Modified: ' . gmdate('D, d M Y H:i:s') . ' GMT');

header('Cache-Control: no-cache, must-revalidate');

header('Pragma: no-cache');

header('Content-Type: text/html');
```

Następnie do wykonania żądanej akcji wykorzystywany jest istniejący kod, który dodaje nowy produkt do koszyka. Szablon cart_summary.tpl jest wysyłany do danych wyjściowych, a JavaScript użyje go do zaktualizowania pola podsumowania koszyka za pomocą funkcji JavaScript DOM:

```
if (isset ($_GET['CartAction']))

{

$cart_action = $_GET['CartAction'];

if ($cart_action == ADD_PRODUCT)

{

require_once PRESENTATION_DIR . 'cart_details.php';

$cart_details = new CartDetails();

$cart_details->init();

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

}

}

else

trigger_error('CartAction not set', E_USER_ERROR);
```

```
}
```

Funkcja JavaScript updateCartSummary() w ajax.js odpowiada za odczytanie kodu HTML wygenerowanego przez cart_summary.tpl i wstrzyknięcie go do pola podsumowania koszyka. Funkcja najpierw sprawdza, czy żądanie nie spowodowało błędu, w którym to przypadku funkcja handleError() jest wywoływana w celu wyświetlenia błędu lub powrotu do zachowania innego niż AJAX:

```
// Process server's response

function updateCartSummary()

{

// Read the response

response = xmlHttp.responseText;

// Server error?

if (response.indexOf("ERRNO") >= 0 || response.indexOf("error") >= 0)

{

handleError(response);

}
```

Jeśli odpowiedź nie zawiera błędu, zakłada się, że zawiera zawartość HTML nowego pola podsumowania koszyka. Ta odpowiedź będzie mniej więcej taka:

```
{* Start cart summary *}

<div class="box" id="cart-summary">

<p class="box-title">Cart Summary</p>

<div id="updating">Updating...</div>

...

</div>
```

Nasz kod JavaScript musi pobrać zawartość głównego elementu i wstrzyknąć ją na stronę pod elementem koszyka podsumowania. W tym celu użyliśmy wyrażenia regularnego o nazwie cartSummaryRegEx:

```
else

{

// Extract the contents of the cart_summary div element

var cartSummaryRegEx = /^<div class="box" id="cart-summary">➥

([\s\S]*)<\/div>$/m;

matches = cartSummaryRegEx.exec(response);

response = matches[1];
```

Alternatywnie moglibyśmy użyć XML DOM lub funkcji podciągu łańcucha. Metoda podciągów jest najszybsza i najłatwiejsza do wdrożenia, ale ma tę wadę, że nie jest elastyczna. Jeśli format danych wyjściowych lub nazwa elementu zmieni się, to przestanie działać. Oto możliwa implementacja:

response = response.substring(25, response.length - 7);

Po uzyskaniu ciągu, updateCartSummary() używa go do zastąpienia zawartości elementu koszyka-podsumowania strony, skutecznie aktualizując podsumowanie koszyka. Tekst „Ładowanie...” jest również ukryty:

// Update the cart summary box and hide the Loading message

document.getElementById("cart-summary").innerHTML = response;

// Hide the "Updating..." message

document.getElementById('updating').style.visibility = 'hidden';

}

}

Ulepszanie koszyka zakupowego za pomocą AJAX

Ulepszony koszyk na zakupy będzie używać AJAX do aktualizacji koszyka. Podczas usuwania produktów z koszyka, zapisywania produktów do późniejszego zakupu lub aktualizacji ilości produktów, akcja odbędzie się w tle. Podczas żądania asynchronicznego pojawia się etykieta „Aktualizacja...”, aby wizualnie wskazać, że koszyk jest aktualizowany. Podobnie jak w przypadku dodawania produktów do koszyka, koszyk jest aktualizowany w degradowalny sposób. Jeśli odwiedzający wyłączy JavaScript, funkcje będą nadal działać, ale będą korzystać z klasycznego formularza przesyłania zamiast AJAX. Wykonaj kroki tego ćwiczenia, aby zaimplementować koszyk zakupów AJAX.

AJAXyfikowanie koszyka

1. Zacznijmy od modyfikacji pliku szablonu,

Presentation\templates\cart_details.tpl, aby wywoływać funkcje JavaScript po kliknięciu przycisku. Otwórz plik i zastosuj podświetlone zmiany:

{* cart_details.tpl *}

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

Updating...

{if $obj->mIsCartNowEmpty eq 1}

<h3>Your shopping cart is empty!</h3>

{else}

<h3>These are the products in your shopping cart:</h3>

<form class="cart-form" method="post" action="{$obj->mUpdateCartTarget}"

onsubmit="return executeCartAction(this);">

<table class="tss-table">

```
<tr>
<th>Product Name</th>
<th>Price</th>
<th>Quantity</th>
<th>Subtotal</th>
<th> </th>
</tr>
{section name=i loop=$obj->mCartProducts}
<tr>
<td>
<input name="itemId[]" type="hidden"
value="{$obj->mCartProducts[i].item_id}" />
{$obj->mCartProducts[i].name}
({$obj->mCartProducts[i].attributes})
</td>
<td>${$obj->mCartProducts[i].price}</td>
<td>
<input type="text" name="quantity[]" size="5"
value="{$obj->mCartProducts[i].quantity}" />
</td>
<td>${$obj->mCartProducts[i].subtotal}</td>
<td>
<a href="{$obj->mCartProducts[i].save}"
onclick="return executeCartAction(this);">Save for later</a>
<a href="{$obj->mCartProducts[i].remove}"
onclick="return executeCartAction(this);">Remove</a>
</td>
</tr>
{/section}
</table>
<table class="cart-subtotal">
```

```
<tr>
<td>
<p>
Total amount: 
<font class="price">${$obj->mTotalAmount}</font>
</p>
</td>
<td align="right">
<input type="submit" name="update" value="Update" />
</td>
</tr>
</table>
</form>
{/if}
{if ($obj->mIsCartLaterEmpty eq 0)}
<h3>Saved products to buy later:</h3>
<table class="tss-table">
<tr>
<th>Product Name</th>
<th>Price</th>
<th> </th>
</tr>
{section name=j loop=$obj->mSavedCartProducts}
<tr>
<td>
{$obj->mSavedCartProducts[j].name}
({$obj->mSavedCartProducts[j].attributes})
</td>
<td>
${$obj->mSavedCartProducts[j].price}
</td>
```

```
<td>
<a href="{$obj->mSavedCartProducts[j].move}"
onclick="return executeCartAction(this);">Move to cart</a>
<a href="{$obj->mSavedCartProducts[j].remove}"
onclick="return executeCartAction(this);">Remove</a>
</td>
</tr>
{/section}
</table>
{/if}
{if $obj->mLinkToContinueShopping}
<p><a href="{$obj->mLinkToContinueShopping}">Continue Shopping </a></p>
{/if}
```

2. Otwórz ajax.js i dodaj te funkcje:

```
// Called on shopping cart update actions
function executeCartAction(obj)
{
// Display "Updating..." message
document.getElementById('updating').style.visibility = 'visible';
// Degrade to classical form submit if XMLHttpRequest is not available
if (!xmlHttp) return true;
// Save object reference
actionObject = obj;
// Initialize response and parameters
response = '';
params = '';
// If a link was clicked we get its href attribute
if (obj.tagName == 'A')
{
url = obj.href + '&AjaxRequest';
}
```

```
// If the form was submitted we get its elements
else
{
url = obj.action + '&AjaxRequest';
formElements = obj.getElementsByTagName('INPUT');
if (formElements)
{
for (i = 0; i < formElements.length; i++)
{
params += '&' + formElements[i].name + '=';
params += encodeURIComponent(formElements[i].value);
}
}
}
// Try to connect to the server
try
{
// Make server request only if the XMLHttpRequest object isn't busy
if (xmlHttp.readyState == 4 || xmlHttp.readyState == 0)
{
xmlHttp.open("POST", url, true);
xmlHttp.setRequestHeader("Content-Type",
"application/x-www-form-urlencoded");
xmlHttp.onreadystatechange = cartActionStateChange;
xmlHttp.send(params);
}
}
catch (e)
{
// Handle error
handleError(e.toString());
```

```
}
// Stop classical form submit if AJAX action succeeded
return false;
}
// Function that retrieves the HTTP response
function cartActionStateChange()
{
// When readyState is 4, we also read the server response
if (xmlHttp.readyState == 4)
{
// Continue only if HTTP status is "OK"
if (xmlHttp.status == 200)
{
try
{
// Read the response
response = xmlHttp.responseText;
// Server error?
if (response.indexOf("ERRNO") >= 0 || response.indexOf("error") >= 0)
{
handleError(response);
}
else
{
// Update the cart
document.getElementById("contents").innerHTML = response;
// Hide the "Updating..." message
document.getElementById('updating').style.visibility = 'hidden';
}
}
catch (e)
```

```
{
// Handle error
handleError(e.toString());
}
}
else
{
// Handle error
handleError(xmlHttp.statusText);
}
}
}
```

3. Zmień index.php na podświetlony

```
if ($cart_action == ADD_PRODUCT)
{
require_once PRESENTATION_DIR . 'cart_details.php';
$cart_details = new CartDetails();
$cart_details->init();
$application->display('cart_summary.tpl');
}
else
{
$application->display('cart_details.tpl');
}
}
else
trigger_error('CartAction not set', E_USER_ERROR);
```

4. Załaduj koszyk i upewnij się, że działa zgodnie z oczekiwaniami.

Jak to działa: koszyk AJAX

Ponieważ kod AJAX napisałeś wcześniej w tym rozdziale, zadanie w tym ćwiczeniu było dość łatwe. Aby ulepszyć koszyk zakupowy, musiałeś wykonać trzy proste kroki. Najpierw zmodyfikowałeś szablon koszyka zakupów (cart_details.tpl), definiując procedurę obsługi zdarzenia kliknięcia linków „Zapisz na

później” i przycisków Usuń oraz procedurę obsługi zdarzenia przesyłania formularza. W ten sposób, gdy JavaScript jest dostępny i te przyciski lub linki są klikane, programy obsługi zdarzeń JavaScript są wykonywane, zanim przeglądarka będzie miała możliwość przesłania formularza lub skorzystania z kliknięty linku. Następnie utworzyłeś niezbędny kod JavaScript, który implementuje obsługę zdarzeń:

• executeCartAction() jest wywoływana ze strony koszyka zakupów w celu wykonania akcji koszyka.

• handleExecuteCartAction() to funkcja zwrotna dla wywołań serwera akcji koszyka.

• postExecuteCartActionProcess() odczytuje odpowiedź serwera akcji koszyka i odpowiednio aktualizuje stronę.

Na koniec zaktualizowałeś index.php, aby zwracał kod HTML koszyka na żądanie AJAX i voilà! Twój koszyk jest teraz szybszy i bardziej przyjazny dla użytkownika.

Podsumowanie

AJAX jest fajny, podobnie jak nasza nowa witryna obsługująca technologię AJAX! Nie była to krótka i łatwa część, ale to, co osiągnąłeś (i czego się nauczyłeś!) okaże się bardzo przydatne w przyszłości. Oczywiście zakres funkcji AJAX, które możesz dodać do swojej strony internetowej, jest bardzo szeroki, ale w tej chwili masz zaimplementowane podstawy, a Twoi klienci z pewnością docenią akcent AJAX, który dodałeś do swojego sklepu.