AJAX i PHP. Tworzenie interaktywnych aplikacji internetowych

Autorzy: Cristian Darie, Bogdan Brinzarea, Filip Cherecheş-Toşa, Mihai Bucica
Tłumaczenie: Julia Malinowska
ISBN: 83-246-0644-0
Tytuł oryginału: Ajax And PHP: Building Responsive Web Application
Format: B5, stron: 328

Poznaj możliwości technologii AJAX

- Utwórz aplikacje działające po stronie klienta i serwera.
- Wykorzystaj pliki w formacie XML.
- Zaimplementuj mechanizmy „przeciągnij i upuść” na stronach WWW.

Technologia AJAX powstała w wyniku połączenia kilku technik, dzięki którym możliwe było ograniczenie do minimum procesu „przeładowywania” stron WWW podczas ich przeglądania przez użytkowników. AJAX, łączący w sobie możliwości języków JavaScript i XML, jest świetnym narzędziem do tworzenia interaktywnych witryn internetowych, które pozwala na zaimplementowanie w nich mechanizmów dotychczas kojarzonych z aplikacjami „tradycyjnymi”. Za pomocą technologii AJAX możemy weryfikować dane wprowadzane do formularzy, tworzyć wykresy w czasie rzeczywistym i dodawać do aplikacji internetowych wiele innych, przydatnych funkcji.

Książka „AJAX i PHP. Tworzenie interaktywnych aplikacji internetowych” to przewodnik po technologii AJAX i jej możliwościach. Czytając ją, nauczysz się pisać wydajny i łatwy w konserwacji kod AJAX-a, łączyć tę technikę z językiem PHP i bazą danych MySQL oraz tworzyć systemy wielowątkowe. Dowiesz się, jak budować aplikacje WWW, których obsługa nie będzie różniła się od obsługi aplikacji dla systemu Windows. Wykorzystasz możliwości języków SVG i XML do kreowania interaktywnych i dynamicznych elementów witryn WWW, a także poznasz najlepsze praktyki programistyczne dla technologii AJAX.

- Korzystanie z obiektowego modelu dokumentu (DOM)
- Aplikacje AJAX działające po stronie przeglądarki
- Połączenie techniki AJAX z PHP i MySQL
- Weryfikacja danych z formularzy
- Tworzenie komunikatora internetowego
- Dynamiczne tworzenie tabel i wykresów
- Przetwarzanie kanałów RSS
- Mechanizmy „przeciągnij i upuść”

Poznaj najnowocześniejszą technologię tworzenia aplikacji WWW

Wydawnictwo Helion
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl

Spis treści

O autorach 7

O recenzentach 9

Przedmowa 11

Rozdział 1. Technologia AJAX a przyszłość aplikacji internetowych 17

Dostarczanie nowych funkcjonalności przez internet 19
Zalety aplikacji internetowych 20
Tworzenie stron WWW od roku 1990 20
HTTP i HTML 21
PHP i inne technologie strony serwera 22
JavaScript i inne technologie po stronie klienta 23
Czego brakuje? 24
Rozumienie technologii AJAX 25
Tworzenie prostej aplikacji w AJAX i PHP 29
Podsumowanie 41

Rozdział 2. Techniki po stronie klienta wykorzystujące ulepszony JavaScript 43

JavaScript i obiektowy model dokumentu 44
Zdarzenia w JavaScript i interfejs DOM 48
Jeszcze więcej o interfejsie DOM 52
JavaScript, DOM i CSS 55
Korzystanie z obiektu XMLHttpRequest 58
Utworzenie obiektu XMLHttpRequest 59
Inicjalizacja żądania za pomocą obiektu XMLHttpRequest 64
Obsługa odpowiedzi serwera 67
Praca ze strukturami XML 74
Więcej o obsłudze błędów i zwracaniu wyjątków 79
Tworzenie struktur XML 83
Podsumowanie 84

Rozdział 3. Techniki po stronie serwera z wykorzystaniem PHP i MySQL 85

PHP i DOM 86
Przekazywanie parametrów i obsługa błędów PHP 93
Łączenie się z serwerami zdalnymi a bezpieczeństwo w JavaScript 102
Korzystanie ze skryptu serwera proxy 109
Szkielet aplikacji dla powtarzających się żądań asynchronicznych 115
Praca z MySQL 127
Tworzenie tabel bazy danych 127
Manipulacja danymi 130
Łączenie się z bazą i wykonywanie zapytań 131
Pakowanie i porządkowanie struktury 136
Podsumowanie 148

Rozdział 4. Weryfikacja danych formularza AJAX 149

Implementacja weryfikacji danych formularza w AJAX 150
Wielowątkowy AJAX 153
Podsumowanie 175

Rozdział 5. Czat AJAX 177

Wstęp do AJAX czat 177
Rozwiązania AJAX czat 178
Implementacja AJAX czat 179
Podsumowanie 199

Rozdział 6. Podpowiadanie i autouzupełnianie w AJAX 201

Wstęp do autouzupełniania i podpowiadania w AJAX 202
Google Suggest 202
Implementacja autouzupełniania i podpowiadania w AJAX 203
Podsumowanie 227

Rozdział 7. Tworzenie wykresów w czasie rzeczywistym z wykorzystaniem AJAX SVG 229

Tworzenie kodu wykresu powstającego w czasie rzeczywistym przy użyciu technologii AJAX i SVG 230
Podsumowanie 245

Rozdział 8. Tabela AJAX 247

Implementacja tabeli danych AJAX przy użyciu transformacji XSLT po stronie klienta 248
Podsumowanie 270

Rozdział 9. Czytnik RSS AJAX 271

Praca z RSS 271
Struktura dokumentu RSS 272
Google Reader 273
Implementacja czytnika RSS w AJAX 274
Podsumowanie 286

Rozdział 10. „Przeciągnij i upuść" w wykonaniu AJAX **287**

Funkcja „Przeciągnij i upuść" w sieci **287**
Koszyki zakupów 288
Listy do sortowania 288
Tworzenie listy z możliwością sortowania przy użyciu funkcji „Przeciągnij i upuść" w technologii AJAX **289**
Podsumowanie **305**

Dodatek A Przygotowanie środowiska pracy **307**

Przygotowanie środowiska Windows **308**
Instalacja serwera Apache 308
Instalacja MySQL 310
Instalacja PHP 311
Przygotowanie środowiska *nix **313**
Instalacja serwera Apache 313
Instalacja MySQL 313
Instalacja PHP 314
Instalacja phpMyAdmin **315**
Przygotowanie bazy danych AJAX **316**

Skorowidz **319**

ROZDZIAŁ 1

Technologia AJAX a przyszłość aplikacji internetowych

Mój mały kuzyn zwrócił się do pierwszego widzianego w swoim życiu komputera z takimi słowami: „Komputerze, narysuj robota!". (Komputer nie przyjął tej komendy, ponieważ dostał ode mnie ścisłe instrukcje, by nie słuchać nieznajomych). Jeśli myślisz podobnie do mnie, to pierwszą Twoją myślą będzie „jakie to głupie" lub „jakie to zabawne" — ale to błąd. Nasze wyedukowane i wymodelowane mózgi opanowały do pewnego stopnia sztukę porozumiewania się z komputerem. Ludzi szkoli się, aby dostosowywali się do komputerów, aby kompensowali nieumiejętność maszyny do pojmowania ludzkiego sposobu myślenia. (Z drugiej strony, ludzie nie przystosowują się zbyt dobrze, ale to już inna historia).

Ta anegdotka bardzo dobrze pokazuje podejście większości ludzi do pracy na komputerze. W idealnym świecie taka ustna komenda wystarczyłaby, aby komputer podjął pracę, co z kolei zadowoliłoby mojego małego kuzyna. Chociaż rozwijająca się technologia jest coraz bardziej przyjazna użytkownikowi, to nadal prawdziwie inteligentne maszyny znajdują się daleko poza naszym zasięgiem. Do czasu, gdy uda się nam takie skonstruować, ludzie, więc również Ty, są skazani na żmudną naukę pracy z komputerem. Niektórzy użytkownicy do tego stopnia poświęcają się tej pasji, że kończą jako fanatycy czarnego ekranu z migającym w linii komend maleńkim znakiem zachęty.

Nieprzypadkowo nawyki, które większość z nas wykształca w czasie pracy na komputerze, powstają podczas pracy z oprogramowaniem posiadającym intuicyjny (oczywiście dla człowieka) interfejs. Stąd zapewne bierze się zawrotna kariera prawego przycisku myszy, fenomen funkcji *przeciągnij i upuść* czy maleńkiego okienka tekstowego, które potrafi przeszukać dla

Ciebie całą zawartość internetu w poszukiwaniu podanego przez Ciebie hasła. I wszystko to w ciągu 0,1 sekundy (a przynajmniej tak twierdzą jego projektanci)! Przemysł związany z projektowaniem oprogramowania (czy też raczej ta jego część, która przynosi zyski) obserwował, analizował i uczył się. W rezultacie rynek oprogramowania jest zalewany programami, których interfejs obfituje w błyszczące przyciski, ikonki, okna i kreatory, a **ludzie płacą ciężkie pieniądze za takie produkty**.

Przemysł oprogramowania wyciągnął ze swoich obserwacji wniosek, że odpowiednikiem potężnego silnika w sportowym samochodzie jest program łączący w sobie **użyteczność i przystępność**. I naprawdę wspaniała jest sytuacja, kiedy dobro producenta idzie w parze z dobrem klienta. Oznacza to, że zyski firm są mniej więcej proporcjonalne do stopnia zadowolenia klienta.

Mamy zamiar precyzyjnie i zwięźle przekazywać naszą wiedzę, lecz zanim przejdziemy do Twojego ulubionego zajęcia (pisania kodu), chcemy cofnąć się troszkę i przypomnieć cel i motywy naszych działań. Jesteśmy pasjonatami współczesnych technologii, a dźwięk uderzanego klawisza przyprawia nas o dreszcze. Dlatego jest nam bardzo łatwo zapomnieć o tym, że technologia została stworzona, by służyć ludziom, dawać im rozrywkę w domu i wspierać w pracy.

Kluczem do stworzenia ostatecznych aplikacji jest poznanie sposobu działania ludzkiego mózgu. I choć od tego jesteśmy dalecy, to rozumiemy, że użytkownik końcowy potrzebuje aplikacji z intuicyjnym interfejsem. Tak naprawdę użytkownika końcowego nie obchodzi, na jakim systemie operacyjnym pracuje, dopóki ten system działa zgodnie z jego **oczekiwaniami**. Jest to rzecz, o której musisz koniecznie pamiętać. Wielu programistów popełnia błąd, próbując pracować z użytkownikiem końcowym i używać technicznej terminologii (chociaż w typowym zespole projektanckim programista nie ma bezpośrednio do czynienia z użytkownikiem końcowym). Jeśli się z tym nie zgadzasz, postaraj się przypomnieć sobie, ile razy użyłeś słów **baza danych** w rozmowie z laikiem.

Obserwacje przyzwyczajeń i potrzeb ludzi pracujących z systemami komputerowymi zrodziły określenie ***użyteczność oprogramowania***. Odnosi się ono do sztuki wychodzenia naprzeciw potrzebom użytkownika, zrozumienia dla charakteru jego pracy i budowania aplikacji zgodnie z tymi wytycznymi.

Historycznie rzecz biorąc, techniki zwiększania użyteczności były wykorzystywane do tworzenia ***aplikacji biurowych***, głównie dlatego, że twórcy ***aplikacji internetowych*** nie dysponowali odpowiednimi narzędziami. Jednak wraz z rozwojem internetu, technologie dostępne dla niego stają się coraz bardziej zaawansowane.

Współczesne technologie internetowe nie tylko pozwalają Ci na tworzenie lepszych stron WWW, pozwalających zaistnieć w sieci, ale dają narzędzia potrzebne do tworzenia aplikacji dedykowanych na potrzeby intranetu. Dla działalności handlowej online przyjazna strona WWW jest czymś nieodzownym, ponieważ **internet nigdy nie śpi**, a klienci często migrują do kolejnej nowinki, która wygląda lepiej lub **sprawia wrażenie** szybszej. Jednocześnie możliwości tworzenia przyjaznej strony WWW dają Ci alternatywne metody projektowania aplikacji dla intranetu. Poprzednio w tej dziedzinie królowały aplikacje biurowe.

Stworzenie przyjaznego programu było zawsze prostsze, jeśli chodziło o aplikację biurową, a nie internetową, głównie dlatego, że internet był pomyślany jako nośnik tekstu i grafiki, a nie kompleksowych rozwiązań. Ten problem dał się szczególnie odczuć w ciągu kilku minionych lat, kiedy to zaczęło się pojawiać coraz więcej rozwiązań programowych dostarczanych przez internet.

W rezultacie rozwinęło się wiele technologii (które nadal ewoluują), mających upiększyć, usprawnić i uprzystępnić aplikacje internetowe. Jako przykłady można tu podać ***aplety Java*** i ***aplikacje Macromedia Flash***. Wymagają one od użytkownika zainstalowania oddzielnych bibliotek do przeglądarek.

Dostarczanie nowych funkcjonalności przez internet

Aplikacje internetowe są szczególnym rodzajem oprogramowania. Ich zadania są przetwarzane na serwerze i dostarczane do użytkownika końcowego przez sieć — internet lub intranet. Do uruchomienia aplikacji użytkownik korzysta z usług ***minimalnego klienta*** (przeglądarki), który potrafi wyświetlić i wywołać dane otrzymane z serwera. Ich przeciwieństwem są aplikacje biurowe, korzystające z usług ***pełnego klienta***, którego zadaniem jest przetworzenie większości danych.

Aplikacje internetowe rozwijają się, marząc, że pewnego dnia będą działać i wyglądać jak ich dorośli (i potężni) krewni — aplikacje biurowe. Obecnie sposób działania aplikacji czynnie wykorzystywanej przez ludzi jest jeszcze ważniejszy niż w przeszłości. Dawniej z komputerami pracowali głównie ludzie obyci z techniką. Dziś krąg użytkowników komputerów jest znacznie szerszy. W dzisiejszych czasach musisz przygotować ładnie wyglądający raport dla Kasi, kierownika działu, i zapewnić Dawidowi z działu handlowego łatwy dostęp do danych.

Ponieważ klient, a w naszym przypadku użytkownik, ma zawsze rację, programy, które stworzysz, będą musiały zadowalać wszystkich, którzy się z nimi zetkną. W przypadku aplikacji internetowych proces dojrzewania zakończy się w momencie, gdy ich interfejs i zachowanie nie będą zdradzały, czy dostępne funkcjonalności są otrzymywane z sieci, czy pobierane z lokalnego komputera. Dostarczenie użytecznych elementów interfejsu w aplikacjach internetowych nie było dotąd możliwe ze względu na przywiązanie użytkowników do funkcji aplikacji biurowych, takich jak metoda *przeciągnij i upuść* czy wykonywanie wielu zadań w tym samym oknie w jednej chwili. Takie funkcjonalności były po prostu niedostępne dla aplikacji internetowych.

Kolejnym problemem pojawiającym się w trakcie tworzenia aplikacji internetowej jest ***standaryzacja***. Dziś wszystko, co ma związek z siecią, musi być testowane przynajmniej w dwóch przeglądarkach. Dopiero wtedy masz pewność, że goście Twojej witryny będą mogli w pełni cieszyć się jej możliwościami.

Zalety aplikacji internetowych

Tak, dostarczanie pewnych funkcjonalności przez internet jest źródłem niezliczonych bolączek. Dlaczego więc w ogóle miałbyś zaprzątać sobie tym głowę, zamiast napisać zwykłą aplikację biurową? Ponieważ nawet teraz, mając tyle problemów z przystępnością interfejsu, aplikacje internetowe mają przewagę nad aplikacjami biurowymi. Stąd też ich ogromna popularność.

- **Aplikacje internetowe są niedrogie i łatwe do dostarczenia**. Korzystając z aplikacji internetowych, firma może w znacznym stopniu zredukować koszty utrzymania działu informatycznego, wynikające z potrzeby instalowania oprogramowania na komputerach użytkowników. Aplikacje internetowe potrzebują jedynie komputera z przeglądarką i dostępem do sieci.
- **Aplikacje internetowe są niedrogie i proste w aktualizacji**. Koszty utrzymania oprogramowania zawsze były wysokie. Zalety aplikacji internetowych, które podaliśmy powyżej, odnoszą się również do tego punktu, ponieważ aktualizacja posiadanego oprogramowania jest czynnością podobną do jego pierwszej instalacji. W przypadku aplikacji internetowej wystarczy dokonać aktualizacji na serwerze, a wszyscy użytkownicy automatycznie otrzymują jej nową wersję.
- **Aplikacje internetowe są bardzo elastyczne**. Kiedy tylko zainstalujesz aplikację internetową na serwerze z dowolnym systemem operacyjnym, będziesz mógł jej używać przez internet, intranet, na maszynach Mac, windowsowych czy linuksowych, itd. Każda właściwie napisana aplikacja będzie spisywać się równie dobrze w dowolnej przeglądarce, niezależnie czy będzie to Internet Explorer, Mozilla Firefox, Opera czy Safari.
- **Aplikacje internetowe pozwalają na łatwiejsze przechowywanie danych**. Kiedy trzeba uzyskać dostęp do tych samych danych z różnych lokalizacji, znacznie prościej osiąga się ten cel, gdy dane przechowywane są w jednym miejscu, a nie w oddzielnych bazach znajdujących się w tych lokalizacjach. W ten sposób unikasz potencjalnej synchronizacji danych i obniżasz ryzyko złamania zabezpieczeń.

Postaramy się zgłębić sposoby wykorzystania współczesnych technologii internetowych do tworzenia lepszych aplikacji. Naszym celem będzie wydobycie wszystkiego, co oferuje nam sieć. Ale zanim zaczniemy, chcemy Ci udzielić **krótkiej** lekcji historii.

Tworzenie stron WWW od roku 1990

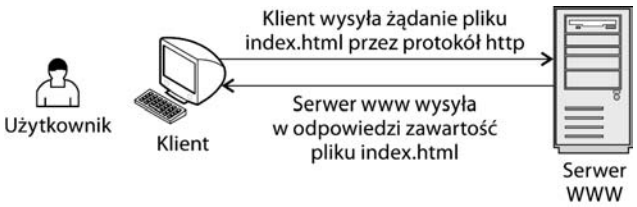
Choć historia internetu sięga nieco dalej wstecz, to rok 1991 był rokiem pojawienia się ***hipertekstowego protokołu transferu (HTTP — ang. HyperText Transfer Protocol)***. Jest on nadal używany do przesyłu danych w internecie. W kilku początkowych wersjach nie robił nic poza otwieraniem i zamykaniem połączenia. Późniejsze wersje (wersja 1.0 pojawiła się w roku 1996, a wersja 1.1 w 1999) to już ten protokół, który znamy i którego używamy.

HTTP i HTML

Protokół HTTP jest wspierany przez wszystkie przeglądarki i doskonale radzi sobie z zadaniem, do którego jest przeznaczony — pobieraniem prostej zawartości sieci. Za każdym razem gdy podajesz swojej ulubionej przeglądarce adres strony WWW, domyślnie używany jest protokół HTTP. Na przykład, gdy wpisujesz *www.mozilla.org* w pasek adresowy przeglądarki Firefox, domyślnie przyjmie ona, że chciałeś napisać *http://www.mozilla.org*.

Standardowym formatem używanym w internecie jest ***hipertekstowy język znaczników (HTML — ang. HyperText Markup Language)***, oparty o znaczniki, które przeglądarki **rozumieją, parsują** i **wyświetlają**. HTML jest językiem, który określa formatowanie i zawartość dokumentu. Taki dokument składa się głównie ze statycznego tekstu i obrazów. Zwróć uwagę, że HTML nie był przeznaczony do tworzenia złożonych aplikacji internetowych z interaktywną zawartością czy przyjaznym dla użytkownika interfejsem. Kiedy chcesz udać się na kolejną stronę HTML, korzystając z protokołu HTTP, musisz zainicjować przeładowanie pełnej strony. I oczywiście strona HTML musi istnieć we wskazanej lokacji, zanim wyślesz żądanie jej wyświetlenia. Wyraźnie widać, że te ograniczenia nie zachęcają do tworzenia niczego interesującego.

Pomimo to HTTP i HTML są parą cieszącą się niesłabnącym powodzeniem i rozumianą zarówno przez serwery, jak i klientów (przeglądarki). Stanowią one fundament internetu, który znamy. Rysunek 1.1 przedstawia prostą transakcję, podczas której użytkownik wysyła żądanie wyświetlenia strony z internetu, używając protokołu HTTP.



Rysunek 1.1. Proste żądanie HTTP

Trzy rzeczy, o których musisz pamiętać.

1. Transakcje HTTP zawsze mają miejsce między ***klientem sieciowym*** (oprogramowaniem wysyłającym żądanie, na przykład przeglądarką internetową) a ***serwerem sieciowym*** (oprogramowaniem odpowiadającym na żądanie, na przykład **Apache** lub **IIS**). **Od tej pory, pisząc o „kliencie", będziemy mieli na myśli klienta sieciowego, a pisząc „serwer", będziemy odnosić się do serwera sieciowego.**
2. Użytkownik to osoba korzystająca z klienta.
3. Nawet jeśli HTTP (czy jego bezpieczna wersja — ***HTTPS***) jest najważniejszym protokołem wykorzystywanym w internecie, to nie jest jedynym. Różne typy serwerów wykorzystują różne protokoły to wykonywania poszczególnych zadań, które z reguły nie mają nic wspólnego z surfowaniem po sieci. Najczęściej będziemy tu mówić o protokole HTTP i przez „żądanie w sieci" będziemy rozumieli żądanie wysłane za pośrednictwem protokołu HTTP. W innym razie dokładnie określimy, jaki protokół mamy na myśli.

Z pewnością kombinacja HTTP-HTML ma spore ograniczenia. Pozwala jedynie na uzyskiwanie statycznej zawartości (strony HTML) z internetu. Aby nadrobić te braki, rozwinięto kilka technologii.

Chociaż wszystkie żądania, o których powiemy, będą używać do przesyłania danych protokołu HTTP, to same dane mogą być tworzone dynamicznie na serwerze (na przykład przy użyciu w tym celu informacji z bazy danych). Co więcej, wszystkie one mogą zawierać nie tylko zwykły HTML, ale mogą pozwolić klientowi na zastosowanie pewnych funkcji, nie ograniczając go do zwykłego wyświetlenia statycznej strony.

Technologie, które pozwalają inteligentniej wykorzystać sieć, zgrupowaliśmy w dwóch kategoriach:

- ***Technologie po stronie klienta*** pozwalają klientowi wykonywać bardziej interesujące zadania niż tylko wyświetlanie statycznych dokumentów. Zwykle są to rozszerzenia języka HTML, które nie mogą go całkowicie zastąpić.
- ***Technologie po stronie serwera*** to te, które pozwalają przechowywać na serwerze logikę służącą do tworzenia strony „w locie".

PHP i inne technologie strony serwera

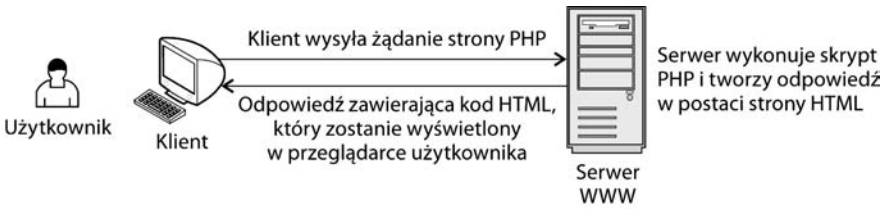
Technologie występujące po stronie serwera sprawiają, że serwer może wykonywać znacznie więcej zadań niż tylko zwracać żądane pliki HTML. Z ich pomocą może wykonywać złożone obliczenia, wykonywać aplikacje zorientowane obiektowo, pracować z bazami danych, a to nadal nie wyczerpuje jego możliwości.

Wyobraź sobie tylko, ile danych musi przetworzyć Amazon, by wykonać obliczenia zgodnie z indywidualnymi ustaleniami dla każdego klienta. Pomyśl, jakie działania wykonuje Google podczas wyszukiwania w swej ogromnej bazie, aby tylko odpowiedzieć na Twoje żądanie. Tak, to właśnie przetwarzanie danych po stronie serwera jest motorem rewolucji internetowej. Dzięki niemu internet jest dla nas tak użyteczny.

Ważne jest, abyś pamiętał, że niezależnie od rodzaju działań wykonywanych po stronie serwera, odpowiedź, którą otrzymuje klient, musi być podana w sposób dla niego zrozumiały. Na przykład, możesz wykorzystać kod HTML, pamiętając o jego ograniczeniach, które podaliśmy wcześniej.

PHP jest jedną z technologii używanych do implementacji specyficznej logiki strony serwera. W rozdziale 3. przedstawimy krótki wstęp do tej technologii. Będziemy też wykorzystywać PHP podczas budowy kodu ***AJAX***. Ważne, byś zapamiętał, że PHP ma wielu różnych konkurentów, np. ***ASP. NET*** (Active Server Pages, technologię projektowania aplikacji wprowadzoną przez Microsoft), ***Java Server Pages (JSP)***, ***Perl***, ***Coldfusion***, ***Ruby on Rails*** i wiele innych. Każdy z nich wymusza na programiście inny sposób tworzenia funkcjonalności po stronie serwera.

PHP nie jest jedynie technologią strony serwera. Jest to również język skryptowy, z którego możesz korzystać przy pracy nad skryptem PHP. Na rysunku 1.2 pokazaliśmy, jak realizowane jest żądanie przekazania strony PHP o nazwie *index.php*. Tym razem serwer nie odsyła jej zawartości, tylko wykonuje skrypt i zwraca wynik swojego działania. Wynik musi być w postaci HTML bądź innej zrozumiałej dla klienta.



Rysunek 1.2. Żądanie strony PHP wysyłane przez klienta

Po stronie serwera będziesz z reguły korzystał z usług ***serwera bazy danych***, by sprawnie zarządzać swoimi danymi. W przykładach, które zamieściliśmy w tej książce, używaliśmy bazy ***MySQL***, ale założenia są takie same dla każdej innej bazy. Podstawy pracy z bazami danych i PHP poznasz w rozdziale 3.

Jednak nawet korzystając z PHP, który tworzy odpowiedzi oparte na zawartości bazy danych, przeglądarka nadal wyświetla nudny, statyczny i niezbyt inteligentny dokument HTML.

To właśnie potrzeba istnienia lepszych, inteligentniejszych narzędzi po stronie klienta spowodowała powstanie oddzielnej gałęzi technologii, które nazywamy technologiami strony klienta. Dzisiejsze przeglądarki parsują z powodzeniem nie tylko prosty kod HTML. Zobaczmy, jak to działa.

JavaScript i inne technologie po stronie klienta

Technologie występujące po stronie klienta różnią się od tych po stronie serwera pod wieloma względami, poczynając od sposobu, w jaki są ładowane i wykonywane. ***JavaScript*** jest językiem skryptowym. Jego kod możesz tworzyć w zwykłym notatniku i włączać w kod strony HTML w celu zwiększenia możliwości Twojej strony. Klient wysyła żądanie dotyczące przesłania strony HTML, która może zawierać kod JavaScript. JavaScript jest obecnie wspierany przez wszystkie przeglądarki i nie wymaga instalowania dodatkowego oprogramowania.

JavaScript jest językiem, który rządzi się własnymi prawami (teoretycznie nie jest związany z rozwojem aplikacji internetowych). Wspiera go większość klientów pod prawie wszystkimi platformami systemowymi. Do tego ma pewne cechy języka zorientowanego obiektowo. JavaScript nie jest językiem kompilowanym, więc nie nadaje się do wykonywania zaawansowanych obliczeń czy pisania sterowników. Ponadto, aby przeglądarka interpretowała go poprawnie, musi być do niej dostarczony w całości, a to z kolei nie sprzyja podnoszeniu bezpieczeństwa. Za to doskonale sprawdza się w aplikacjach internetowych.

To właśnie JavaScript dał projektantom stron internetowych możliwość tworzenia stron, na których pada śnieg, czy takich, na których użytkownik nie spowoduje przeładowania całej strony (i utracenia wpisanych danych), gdy nie wypełni poprawnie wszystkich pól stworzonego formularza. Jednak mimo tych możliwości JavaScript nigdy nie był konsekwentnie wykorzystywany przy poprawianiu interfejsu aplikacji internetowych.

Innymi popularnymi technologiami, jakie rozwinęły się po stronie klienta, są aplety Java i pliki Macromedia Flash. Aplety pisane są w popularnym i potężnym języku Java, ale ich wykonanie wymaga instalacji ***Java Virtual Machine***. Aplety są niewątpliwie środkiem, który pozwala bardziej rozwinąć projekt, ale ostatnio straciły na popularności ze względu na zajmowanie zbyt wielu zasobów komputera. Zdarza się, że ładują się bardzo długo. Zasadniczo są zbyt potężnym narzędziem do realizowania niewielkich potrzeb aplikacji internetowych.

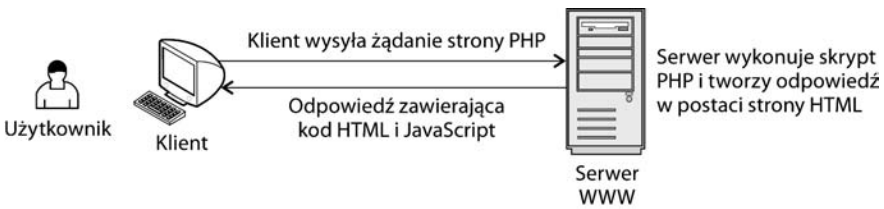
Oprogramowanie Macromedia Flash dostarcza równie potężnych narzędzi do tworzenia animacji i wszelkich efektów graficznych. De facto stało się standardem, w którym tego typu programy są rozprzestrzeniane w internecie. Flash również wymaga instalowania specjalnej ***wtyczki*** w przeglądarce. Aplikacje oparte o Flasha stały się bardzo popularne i ich ilość ciągle rośnie.

Nasuwa się więc następujący wniosek — łączenie HTML z technologiami po stronie serwera i klienta daje Ci duże możliwości budowania aplikacji internetowych.

Czego brakuje?

Widać, że mamy pewne możliwości. Po co więc ludzie poszukują nowych rozwiązań? Czego nam brakuje?

Jak zauważyliśmy na początku tego rozdziału, wszelkie technologie istnieją po to, by służyć potrzebom rynku. A część rynku potrzebuje większych możliwości po stronie klienta bez użycia apletów Java, Flasha czy innych tego typu technologii. Są zbyt krzykliwe lub z jakichś powodów za duże dla użytkowników. W takich przypadkach projektanci zadowalali się stronami WWW tworzonymi w HTML, JavaScript i PHP (czy dowolnej innej technologii strony serwera). Przykład realizacji żądania w opisany sposób staraliśmy się zilustrować na rysunku 1.3.

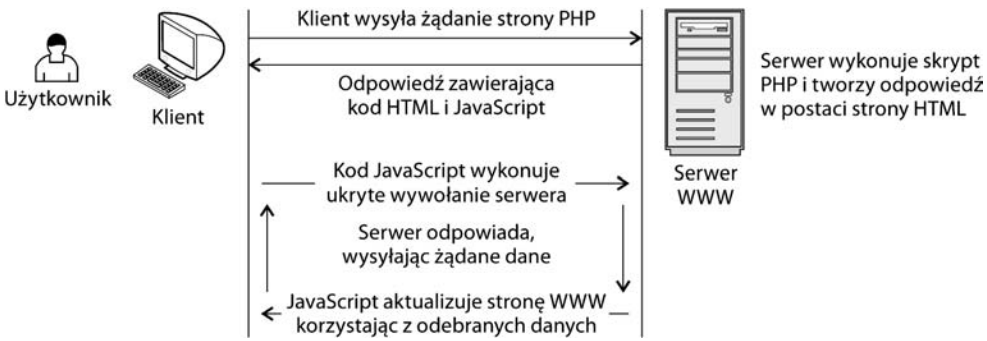


Rysunek 1.3. HTTP, HTML i JavaScript w akcji

Widzisz na nim żądanie przesłane protokołem HTTP i odpowiedź będącą kombinacją HTML i JavaScript zbudowaną za pomocą skryptu PHP. Ukrytym problemem tego rozwiązania jest fakt, że za każdym razem kiedy klient potrzebuje nowych danych z serwera, tworzy się nowe żądanie HTTP. To oznacza przeładowanie strony, czyli zamrożenie działań użytkownika. To właśnie przeładowanie strony jest „złem" dzisiejszego świata aplikacji internetowych, a AJAX zdąża nam na ratunek.

Rozumienie technologii AJAX

Akronim AJAX rozwija się w nazwę ***Asynchroniczny JavaScript i XML (ang. Asynchronous JavaScript and XML)***. Jeśli jesteś zdania, że to niewiele mówiące wyjaśnienie, to jesteśmy skłonni przyznać Ci rację. Mówiąc wprost, AJAX można traktować jako „wzbogacony JavaScript". W zasadzie oferuje tę samą technikę po stronie klienta co JavaScript, ale wykorzystuje ją do wysyłania żądań w tle w celu dostarczenia dodatkowych danych. Takie rozwiązanie zabezpiecza nas przed wielokrotnym przeładowaniem strony i zapewnia aktualizację danych. Rysunek 1.4 jest wizualizacją typowej transakcji, w której użytkownik żąda strony z włączonym kodem AJAX.



Rysunek 1.4. Typowe wywołanie AJAX

Mówiąc ogólnie, AJAX ma nam zapewnić równowagę między funkcjonalnością po stronie klienta i funkcjonalnością po stronie serwera podczas wykonywania żądań użytkownika. Aż do chwili obecnej obie te funkcjonalności były rozpatrywane oddzielnie, zakładaliśmy ich działanie po kolei. AJAX daje nam możliwość równomiernego rozłożenia obciążenia na obie strony — serwera i klienta, pozwalając im na komunikację w tle **w czasie, gdy użytkownik pracuje na stronie WWW**.

Spróbujemy przybliżyć Ci to przy użyciu prostego przykładu. Wyobraź sobie formularz, w którym użytkownik musi podać pewne dane (na przykład imię i nazwisko, adres email, hasło, numer karty kredytowej itd.). Poprawność tych danych powinna być sprawdzona przed dotarciem do transakcyjnej części Twojej aplikacji. Można to osiągnąć na dwa sposoby bez użycia AJAX. Przede wszystkim, można pozwolić użytkownikowi ***wysłać*** formularz i dokonać weryfikacji

danych na serwerze. W tym wariancie użytkownik musi odczekać tzw. **martwy czas**, w którym strona ponownie się ładuje. Alternatywą jest wykonanie sprawdzenia po stronie klienta. Pamiętaj, że nie zawsze jest to możliwe (lub wykonalne), ponieważ wiąże się z dostarczeniem zbyt wielu danych do klienta (wyobraź sobie sytuację, w której musisz sprawdzić, czy podane przez użytkownika miasto znajduje się we wpisanym przez niego państwie).

W wariancie obsługiwanym przez technologię AJAX, który chcemy Ci przedstawić, aplikacja zweryfikuje poprawność wprowadzonych danych, wykonując wywołania serwera w tle. Na przykład, kiedy użytkownik wybierze państwo, przeglądarka „w locie" wyśle do serwera żądanie pobrania listy miast w danym kraju. Zwróć uwagę, że użytkownik nieprzerwanie wypełnia formularz podczas tej aktywności przeglądarki i serwera. Przykład formularza weryfikującego podane dane opisaliśmy w rozdziale 4.

Na co dzień możesz się zetknąć z niezliczonymi przykładami zastosowania technologii AJAX. Udaj się pod podane niżej adresy i sprawdź na własnej skórze tę różnicę:

- **Google Suggest** pomaga Ci w wyszukiwaniach **Google**. Jej możliwości są prawdziwie spektakularne. Wypróbuj *http://www.google.com/webhp?complete=1*. **Yahoo! Instant Search** oferuje podobne właściwości — *http://instant.search.yahoo.com/*. (W rozdziale 6. nauczymy Cię tworzyć podobne funkcjonalności).
- **Gmail** (*http://www.gmail.com*) jest już tak popularny, że nie trzeba go przedstawiać. Inni usługodawcy kont emailowych opartych o aplikację internetową, jak **Yahoo! Mail** i **Hotmail**, poszły za tym przykładem i oferują już funkcjonalności oparte o AJAX.
- **Google Maps** (*http://maps.google.com*), **Yahoo Maps** (*http://maps.yahoo.com*) i **Windows Live Local** (*http://local.live.com*).
- Inne usługi, jak *http://www.writely.com* i *http://www.basecamphq.com*.

W dalszej części książki pokażemy Ci jeszcze inne przykłady.

> Uważaj, bo możesz nadużyć lub użyć źle technologii AJAX. Samo umieszczenie kodu AJAX na Twojej stronie nie zagwarantuje Ci, że stanie się ona lepsza. Poprawne wykorzystanie tej technologii jest w Twoich rękach.

Wiesz już zatem, że AJAX służy do tworzenia bardziej wszechstronnych i interaktywnych stron WWW. Udaje mu się to dzięki asynchronicznym wywołaniom serwera podczas pracy użytkownika. AJAX pozwala projektantom tworzyć bardziej inteligentne aplikacje internetowe, które będą znacznie lepiej współpracować z człowiekiem niż ich poprzednicy.

Wszystkie technologie, które składają się na AJAX, są już zaimplementowane w przeglądarkach internetowych takich jak **Mozilla Firefox**, **Internet Explorer** czy **Opera**. Oznacza to, że klient nie musi instalować dodatkowych modułów, aby uruchomić stronę korzystającą z AJAX. A oto, co składa się na AJAX:

- JavaScript jest zasadniczym składnikiem AJAX. To on pozwala Ci tworzyć funkcjonalności po stronie klienta. Aby manipulować częściami strony HTML, w funkcjach JavaScript będziesz intensywnie wykorzystywać ***obiektowy model dokumentu — DOM (ang. Document Object Model)***.
- Obiekt ***XMLHttpRequest*** pozwala skryptowi JavaScript asynchronicznie dostać się do serwera. Użytkownik nie musi przerywać pracy na czas trwania wykonywanych w tle działań. Przez dostęp do serwera rozumiemy tu proste żądanie pliku lub skryptu na serwerze wysłane przez protokół HTTP. Takie żądania są proste do zrealizowania i nie napotykają przeszkód ze strony zapory ogniowej.
- Technologia po stronie serwera obsługuje żądanie wysłane przez klienta. W tej książce wykorzystamy do tego skrypty PHP.

Aby komunikacja klient-serwer przebiegała sprawnie, musimy mieć sposób **przekazywania danych** i ich **interpretowania**. Przekazywanie danych jest tą prostszą częścią. Skrypt klienta dostaje się na serwer, używając obiektu `XMLHttpRequest`, i może wysłać pary nazwa-wartość, korzystając z poleceń ***`GET`*** lub ***`POST`***. Odczytanie tych wartości nie jest trudne. Wystarczy jakikolwiek skrypt po stronie serwera.

Skrypt na serwerze po prostu wysyła odpowiedź przez protokół HTTP. Jednak nie jest to zwykły kod HTML. Serwer użyje formatu, który może być parsowany przez kod JavaScript po stronie klienta. Sugerujemy Ci format XML, którego zaletą jest dobre wsparcie i wiele bibliotek pozwalających manipulować dokumentem XML. Ale oczywiście możesz wybrać inny format, jeśli masz na to ochotę, choćby zwykły dokument tekstowy. Popularną alternatywą dla XML jest ***Obiektowa Notacja JavaScript (JSON — ang. JavaScript Object Notation)***.

Pisząc tę książkę, założyliśmy, że znasz już składniki technologii AJAX, może z wyjątkiem obiektu `XMLHttpRequest`, który jest mniej popularny. Jednak chcemy mieć pewność, że za nami nadążasz, więc przedstawimy Ci, jak te elementy działają — razem i osobno. O tym będą traktować rozdziały 2. i 3. W pozostałej części rozdziału 1. zajmiemy się ogólnym stanem rzeczy. Napiszemy też program w technologii AJAX, żeby uspokoić bardziej niecierpliwych czytelników.

Żaden ze składników technologii AJAX nie jest nowy ani rewolucyjny (żaden nawet nie ewoluuje), choć coś przeciwnego może sugerować nagły zgiełk powstały wokół niej. Wszystkie składniki istnieją od mniej więcej 1998 roku. Sam AJAX narodził się w 2005 roku w artykule Jesse Jamesa Garreta. Z jego treścią możesz zapoznać się pod adresem *http://www.adaptivepath.com/publications/essays/archives/000385.php*. Popularność przyniosły mu zastosowania w aplikacjach Google.

Nowością w AJAX jest to, że po raz pierwszy na rynku pojawiła się potrzeba standaryzacji i skierowania rozwoju na konkretne tory. Wynikiem tych działań jest mnogość pojawiających się bibliotek AJAX i tworzenie coraz to nowych stron w tej technologii. Wkład Microsoft w rozwój AJAX to projekt Atlas.

Oto lista korzyści płynących z tworzenia nowej aplikacji z użyciem AJAX:

- możesz tworzyć lepsze i bardziej komunikatywne strony i aplikacje,
- swoją popularnością sprzyja rozwojowi szablonów, które pozwalają Ci uniknąć wyważania otwartych drzwi podczas rozwiązywania podstawowych problemów,
- wykorzystuje dostępne technologie,
- wykorzystuje umiejętności, które już posiadasz,
- cechy AJAX idealnie współpracują z istniejącymi możliwościami przeglądarek (np. zmiana rozmiaru strony, nawigacja na stronie).

Najczęstsze przykłady stosowania AJAX:

- Natychmiastowa weryfikacja danych na serwerze, niezmiernie przydatna w wypadkach, gdy nie możemy wysłać do klienta danych, które są niezbędne do weryfikacji strony. Rozdział 4. zawiera opis działania takiego formularza;
- Tworzenie prostej aplikacji czat online, która nie wymaga zewnętrznych bibliotek typu Java Runtime Machine czy Flash. Taki program stworzysz w rozdziale 5.;
- Pisanie funkcjonalności typu Google Suggest pokażemy Ci w rozdziale 6.,
- Bardziej efektywne wykorzystanie istniejących technologii. W rozdziale 7. napiszesz kod wyświetlający wykres w czasie rzeczywistym. Wykorzystasz do tego ***skalowaną grafikę wektorową***. A w rozdziale 10. użyjesz zewnętrznej biblioteki AJAX do napisania prostej listy *przeciągnij i upuść*;
- Tworzenie interaktywnych **tabel danych**, które aktualizują „w locie" bazę danych na serwerze. Taką aplikację pokażemy Ci w rozdziale 8.;
- Tworzenie aplikacji, które wymagają bieżących aktualizacji z zewnętrznych źródeł. W rozdziale 9. stworzysz prosty agregator RSS.

Pamiętaj o potencjalnych problemach:

- Stały adres strony pisanej z użyciem AJAX powoduje problemy z dodawaniem jej do zakładek. W przypadku aplikacji AJAX dodawanie do zakładek ma inne znaczenie, zależne od konkretnej aplikacji. Oznacza to, że musisz w jakiś sposób zachować stan aplikacji (pomyśl tylko o aplikacjach biurowych — tam w ogóle nie ma zakładek);
- Silniki wyszukiwarek mogą mieć problemy z poprawnym indeksowaniem Twojej strony;
- Naciśnięcie guzika *Wstecz* powoduje inne rezultaty niż w przypadku klasycznych aplikacji internetowych, ponieważ wszystkie działania zachodzą w obrębie jednej strony;
- Można zablokować wykonywanie skryptów JavaScript po stronie klienta, co sprawi, że Twoja aplikacja przestanie działać. W razie takich wypadków warto mieć plan awaryjny, żeby nie tracić gości.

W końcu, zanim zajmiemy się pisaniem pierwszego programu w AJAX, chcemy podać Ci kilka adresów, które mogą być przydatne w poznawaniu świata AJAX:

- *http://ajaxblog.com* — blog poświęcony technologii AJAX,
- *http://www.fiftyfoureleven.com/resources/programming/xmlhttprequest* — pełna kolekcja artykułów dotyczących AJAX,
- *http://www.ajaxian.com* — strona Bena Galbraitha i Dion Almaer, autorów *Pragmatic AJAX*,
- *http://www.ajaxmatters.com* — informacyjna strona na temat AJAX, która zawiera mnóstwo użytecznych odnośników,
- *http://ajaxpatterns.org* — dotyczy wzorców projektowych AJAX,
- *http://www.ajaxinfo.com* — źródło artykułów o AJAX,
- *http://ajaxguru.blogspot.com* — popularny blog,
- *http://www.sitepoint.com/article/remote-scripting-ajax* — doskonały artykuł Camerona Adamsa, „Usable Interactivity with Remote Scripting",
- *http://developer.mozilla.org/en/docs/AJAX* — strona Mozilli dotycząca AJAX,
- *http://en.wikipedia.org/wiki/AJAX* — strona AJAX w Wikipedii.

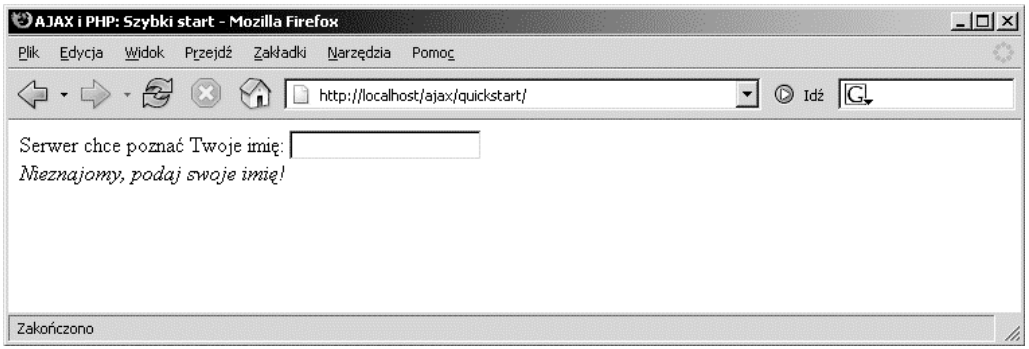
Ta lista w żadnym razie nie jest zamknięta. Jeśli interesują Cię inne źródła w sieci, Google z pewnością posłuży Ci pomocą. W kolejnych rozdziałach podamy Ci więcej odnośników, ale będą one dotyczyć bardziej szczegółowych zagadnień.

Tworzenie prostej aplikacji w AJAX i PHP

W takim razie bierzmy się za pisanie kodu. Na kolejnych kartach pokażemy Ci, jak stworzyć prostą aplikację AJAX.

> To ćwiczenie przeznaczyliśmy dla naszych najbardziej niecierpliwych czytelników, którzy chcą zacząć programować jak najszybciej. Założyliśmy tu, że znasz podstawy JavaScript, PHP i XML. Jeśli jest inaczej lub jeśli pomimo Twojej wiedzy zagadnienie będzie dla Ciebie zbyt trudne, bez wahania udaj się wprost do rozdziału 2. W nim i w rozdziale 3. przyjrzymy się bliżej technologiom i technikom tworzącym AJAX, wtedy wszystko stanie się jasne.

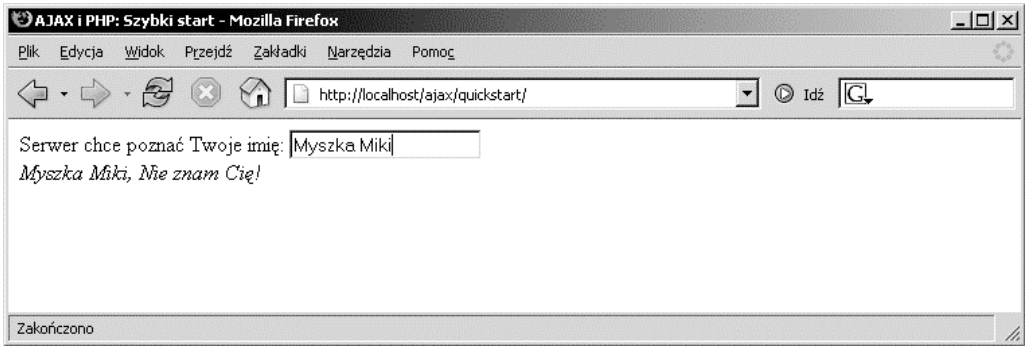
Zaraz stworzysz prostą aplikację AJAX, którą nazwiemy **quickstart**. Użytkownik będzie w niej poproszony o podanie imienia, a serwer będzie przesyłał odpowiedzi w trakcie pisania. Rysunek 1.5 pokazuje stronę startową aplikacji index.html (zwróć uwagę, że plik index.html ładuje się automatycznie podczas otwierania folderu quickstart bez podania jawnie jego nazwy).



Rysunek 1.5. Strona startowa Twojej aplikacji quickstart

W trakcie gdy użytkownik pisze, serwer wywoływany jest asynchronicznie w regularnych odstępach czasu. Sprawdzamy w ten sposób, czy serwer rozpoznaje aktualnie podane imię. Żądania wysyłane są automatycznie w odstępach ok. 1 sekundy. To wyjaśnia, dlaczego nie potrzebujemy przycisku np. *Wyślij* do oznaczenia końca wprowadzania danych. (Ta metoda raczej nie jest odpowiednia do rzeczywistych mechanizmów logowania, ale doskonale pokazuje możliwości AJAX).

W zależności od podanego imienia, odpowiedzi będą różne. Spójrz na przykład pokazany na rysunku 1.6.



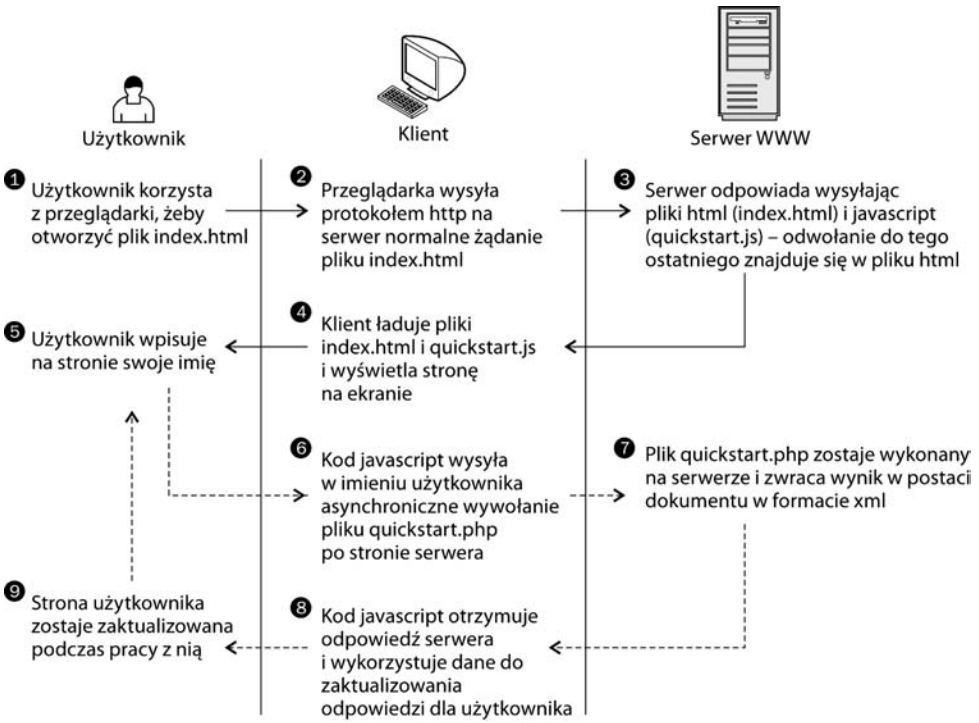
Rysunek 1.6. Użytkownik otrzymuje natychmiastową odpowiedź od aplikacji WWW

Może na pierwszy rzut oka nie widzisz nic niezwykłego. Specjalnie podaliśmy tak prosty pierwszy przykład. Chodzi o to, abyś dokładnie zrozumiał, o co w nim chodzi. W tym przykładzie istotne jest to, że odpowiedzi serwera pojawiają się automatycznie, bez ingerencji w aktualnie wykonywaną przez użytkownika czynność. (Wiadomość pojawia się w trakcie pisania). **Strona nie przeładowuje się, aby wyświetlić nowe dane, pomimo potrzeby łączenia się z serwerem, aby je uzyskać.** Niełatwo byłoby uzyskać taki efekt bez pomocy technologii AJAX.

W skład aplikacji wchodzą:

1. plik *index.html*, który zawiera kod głównej strony aplikacji;
2. plik *quickstart.js*, w którym znajduje się kod JavaScript, ładowany po stronie klienta jednocześnie z plikiem index.html. Ten plik czuwa nad wykonywaniem asynchronicznych żądań, kiedy potrzebna jest działalność serwera;
3. plik *quickstart.php* to skrypt umieszczony na serwerze, który jest wywoływany przez kod JavaScript ze strony klienta.

Rysunek 1.7 pokazuje akcje podejmowane w trakcie działania aplikacji.



Rysunek 1.7. Diagram wyjaśniający działanie wszystkich elementów Twojej aplikacji

Kroki od 1. do 5. są typowym żądaniem HTTP. Po wysłaniu żądania użytkownik musi poczekać na załadowanie strony. W typowym (bez AJAX) rozwiązaniu takie ładowanie ma miejsce przy każdym żądaniu nowych danych z serwera.

Kroki od 5. do 9. demonstrują działanie wywołania typu AJAX. Mówiąc konkretniej, ukazują sekwencję asynchronicznych żądań HTTP. Wykorzystując obiekt `XMLHttpRequest`, łączymy się z serwerem. W tym czasie użytkownik może normalnie korzystać ze strony, tak jak w przypadku aplikacji biurowej. Nie pojawia się żadne odświeżanie ani ładowanie strony, aby otrzymać dane z serwera i zaktualizować jej zawartość.

Nadeszła pora, byś stworzył taki kod u siebie. Zanim pójdziemy dalej, upewnij się, że przygotowałeś sobie środowisko pracy. Dokładny opis tego, co masz zrobić, znajduje się w dodatku A. Przeprowadzimy Cię tam przez instalację i ustawienia PHP i Apache. Pokażemy Ci też, jak postawić bazę, która będzie Ci dalej potrzebna (choć nie w tym przykładzie).

> Wszystkie ćwiczenia w tej książce wymagają instalacji opisanych w środowisku A. Jeśli przeprowadzisz instalację i konfigurację inaczej, możesz potrzebować potem różnych zmian, na przykład używania innych nazw folderów.

Do dzieła! — szybki start z AJAX

1. W Dodatku A pokazaliśmy Ci, jak należy skonfigurować przeglądarkę i założyć folder *ajax*, do którego będziesz miał dostęp przez serwer. W folderze *ajax* załóż podfolder *quickstart*.
2. W folderze quickstart stwórz plik o nazwie *index.html*.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8">
<title>AJAX i PHP: Szybki start</title>
    <script type="text/javascript" src="quickstart.js"></script>
  </head>
  <body onload='process()'>
    Serwer chce poznać Twoje imię:
    <input type="text" id="myName" />
    <div id="divMessage" />
  </body>
</html>
```

3. Utwórz plik o nazwie *quickstart.js* i zawartości:

```
// przechowuje odwołanie do obiektu XMLHttpRequest
var xmlHttp = createXmlHttpRequestObject();

// zwraca obiekt XMLHttpRequest
function createXmlHttpRequestObject()
{
  // przechowa odwołanie do obiektu XMLHttpRequest
  var xmlHttp;
  // jeśli uruchomiony jest Internet Explorer
  if(window.ActiveXObject)
  {
    try
    {
      xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
    }
```

```
    catch (e)
    {
      xmlHttp = false;
    }
  }
  // jeśli uruchomiona jest Mozilla lub inne przeglądarki
  else
  {
    try
    {
      xmlHttp = new XMLHttpRequest();
    }
    catch (e)
    {
      xmlHttp = false;
    }
  }
  // zwraca utworzony obiekt lub wyświetla komunikat o błędzie
  if (!xmlHttp)
    alert("Błąd podczas tworzenia obiektu XMLHttpRequest.");
  else
    return xmlHttp;
}

// wysyła asynchroniczne żądanie protokołem HTTP, korzystając z obiektu XMLHttpRequest
function process()
{
  // kontynuuje jedynie jeśli obiekt xmlHttp nie jest zajęty
  if (xmlHttp.readyState == 4 || xmlHttp.readyState == 0)
  {
    // pobiera imię wpisane przez użytkownika w formularzu
    name = encodeURIComponent(document.getElementById("myName").value);
    // wykonuje stronę quickstart.php na serwerze
    xmlHttp.open("GET", "quickstart.php?name=" + name, true);
    // definiuje metodę obsługi odpowiedzi serwera
    xmlHttp.onreadystatechange = handleServerResponse;
    // wysyła żądanie do serwera
    xmlHttp.send(null);
  }
  else
    // jeśli połączenie jest zajęte, ponawia próbę po 1 sekundzie
    setTimeout('process()', 1000);
}

// wykonywana automatycznie po otrzymaniu odpowiedzi z serwera
function handleServerResponse()
{
  // kontynuuje jedynie jeśli transakcja została zakończona
  if (xmlHttp.readyState == 4)
```

```
  {
    // status 200 oznacza pomyślne ukończenie transakcji
    if (xmlHttp.status == 200)
    {
      // wyodrębnia wiadomość XML wysłaną z serwera
      xmlResponse = xmlHttp.responseXML;
      // pobiera element główny ze struktury pliku XML
      xmlDocumentElement = xmlResponse.documentElement;
      // pobiera wiadomość tekstową pierwszego potomka elementu document
      helloMessage = xmlDocumentElement.firstChild.data;
      // aktualizuje dane wyświetlane klientowi informacjami otrzymanymi z serwera
      document.getElementById("divMessage").innerHTML = '<i>' + helloMessage
      + '</i>';
      // ponawia sekwencję
      setTimeout('process()', 1000);
    }
    // dla statusu protokołu HTTP innego niż 200 zgłasza błąd
    else
    {
      alert("Wystąpił błąd podczas uzyskiwania dostępu do serwera: "
      + xmlHttp.statusText);
    }
  }
}
```

4. Załóż plik *quickstart.php* o zawartości:

```
<?php
// wygenerujemy dane wyjściowe jako plik XML
header('Content-Type: text/xml');
// generacja nagłówka XML
echo '<?xml version="1.0" encoding="utf-8" standalone="yes"?>';
// utworzenie elementu <response>
echo '<response>';
// pobranie imienia użytkownika
$name = $_GET['name'];
// generacja danych wyjściowych w zależności od podanego imienia
$userNames = array('CRISTIAN', 'BOGDAN', 'FILIP', 'MIHAI', 'YODA');
if (in_array(strtoupper($name), $userNames))
  echo 'Witaj mistrzu ' . htmlentities($name) . '!';
else if (trim($name) == '')
  echo 'Nieznajomy, podaj swoje imię!';
else
  echo htmlentities($name) . ', Nie znam Cię!';
// zamknięcie elementu <response>
echo '</response>';
?>
```

5. Teraz powinieneś móc uruchomić swój nowy program. Wpisz w przeglądarce adres *http://localhost/ajax/quickstart/*. Załadowana strona powinna wyglądać jak na rysunkach 1.5 i 1.6.

> Jeśli pojawią się jakieś problemy, sprawdź, czy przeprowadziłeś poprawnie instalację i konfigurację — zgodnie z wytycznymi z dodatku A. Większość błędów powodują literówki. W rozdziale 2. i 3. pokażemy Ci obsługę błędów w kodzie JavaScript i PHP.

Co właśnie zrobiliśmy?

Teraz zacznie się zabawa — musisz zrozumieć, co dzieje się w kodzie. (Pamiętaj, że szczegóły techniczne dokładniej przedyskutujemy w następnych dwóch rozdziałach).

Zacznijmy od pliku, z którym współpracuje użytkownik — *index.html*. To w nim znajduje się odwołanie do tajemniczego pliku *quickstart.js*. Poza tym ten plik tworzy prościutki interfejs dla przeglądarki. Zwróć uwagę na wyróżnione elementy:

```
<body onload='process()'>
    Serwer chce poznać Twoje imię:
    <input type="text" id="myName" />
    <div id="divMessage" />
  </body>
```

Kiedy strona się ładuje, wykonywana jest funkcja z pliku *quickstart.js* o nazwie `process()`. To sprawia, że element `<div>` wypełniany jest wiadomością z serwera.

Zanim zajmiemy się ciałem funkcji `process()`, przyjrzyjmy się, co dzieje się na serwerze. Tam jest umieszczony skrypt *quickstart.php*. To on będzie tworzył wiadomości XML wysyłane do klienta. Taki plik XML zawiera element `<response>`, który przygotowuje wiadomość wysyłaną następnie do klienta.

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<response>
  ... wiadomość, jaką serwer przesyła do klienta...
</response>
```

Jeśli zawartość pola przechowującego imię wysłana przez klienta jest pusta, serwer wyświetli komunikat `Nieznajomy, podaj swoje imię!`. Jeśli w polu pojawi się imię Cristian, Bogdan, Filip, Mihai lub Yoda, odpowiedzią serwera będzie `Witaj mistrzu <imię użytkownika>!`. Na każde inne imię serwer zareaguje, wysyłając wiadomość `<imię>, Nie znam Cię!`. Więc jeśli Myszka Miki poda swoje dane, to serwer odeśle taką strukturę XML:

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<response>
  Myszka Miki, Nie znam Cię!
</response>
```

Plik *quickstart.php* rozpoczyna swoje działanie wygenerowaniem nagłówka dokumentu XML i otwarciem elementu `<response>`:

```
<?php
// wygenerujemy dane wyjściowe jako plik XML
header('Content-Type: text/xml');
// generacja nagłówka XML
echo '<?xml version="1.0" encoding="utf-8" standalone="yes"?>';
// utworzenie elementu <response>
echo '<response>';
```

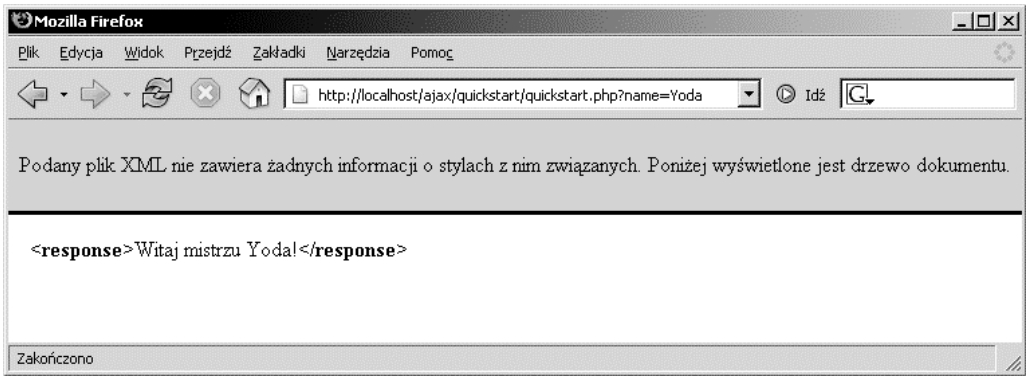
Wyróżniona linia nagłówka określa typ danych wyjściowych — w naszym przypadku jest to dokument XML. To bardzo ważne, ponieważ klient oczekuje właśnie takiego pliku (***API*** użyte do parsowania XML po stronie klienta zgłosi błąd, jeśli w nagłówku nie pojawi się `Content-Type: text/xml`). Po określeniu nagłówka, budujemy odpowiedzi XML metodą sklejania łańcuchów znaków. Właściwy tekst odpowiedzi serwera zawarty jest w elemencie `<response>`, który jest głównym elementem dokumentu XML. Generujemy go, bazując na danych wysłanych przez klienta za pomocą żądania `GET`.

```
// pobranie imienia użytkownika
$name = $_GET['name'];
// generacja danych wyjściowych w zależności od podanego imienia
$userNames = array('CRISTIAN', 'BOGDAN', 'FILIP', 'MIHAI', 'YODA');
if (in_array(strtoupper($name), $userNames))
  echo 'Witaj mistrzu ' . htmlentities($name) . '!';
else if (trim($name) == '')
  echo 'Nieznajomy, podaj swoje imię!';
else
  echo htmlentities($name) . ', Nie znam Cię!';
// zamknięcie elementu <response>
echo '</response>';
?>
```

Klient wysyła do serwera tekst wprowadzony przez użytkownika (w założeniu jego imię), używając parametru żądania `GET`. My, zwracając ten tekst do klienta, posłużymy się funkcją PHP `htmlentities`. Pozwoli nam to zastąpić znaki specjalne ich odpowiednikami w HTML. W ten sposób zapewniamy sobie poprawne wyświetlanie wiadomości w przeglądarce i eliminujemy ryzyko obniżenia poziomu bezpieczeństwa.

> Formatowanie tekstu dla klienta po stronie serwera (zamiast bezpośrednio u klienta) jest złym nawykiem. W sytuacji idealnej jedynym obowiązkiem serwera jest wysłanie danych w najprostszej postaci. To po stronie odbiorcy leży obowiązek poradzenia sobie z kwestiami bezpieczeństwa i formatowaniem. Takie założenie jest jeszcze bardziej sensowne, jeśli wyobrazisz sobie, że musisz wstawić taki sam tekst do bazy danych, ale baza będzie żądała innego formatowania. (W takim przypadku to baza, nie serwer odpowiednim skryptem rozwiąże kwestię formatowania). W przykładzie quickstart formatowanie strony HTML skryptem PHP pozwoliło nam skrócić kod.

Jeśli chcesz sprawdzić działanie samego skryptu quickstart.php, wpisz w przeglądarkę *http://localhost/ajax/quickstart/quickstart.php?name=Yoda* i zobacz, co uzyskasz. Wielką zaletą wysyłania parametrów funkcji poleceniem `GET` jest to, że potem możesz łatwo emulować takie żądanie w swojej przeglądarce, bo `GET` oznacza po prostu, że dodałeś takie parametry jak pary nazwa-wartość do łańcucha URL. Powinieneś otrzymać taki efekt:



Rysunek 1.8. Dane XML generowane przez quickstart.php

Po stronie klienta wiadomość XML odczytuje funkcja `handleServerResponse()` umieszczona w pliku *quickstart.js*. Konkretnie odpowiedzialne za to są pierwsze linie kodu.

```
// wyodrębnia wiadomość XML wysłaną z serwera
xmlResponse = xmlHttp.responseXML;
// pobiera element nadrzędny ze struktury pliku XML
xmlDocumentElement = xmlResponse.documentElement;
// pobiera wiadomość tekstową pierwszego potomka elementu document
helloMessage = xmlDocumentElement.firstChild.data;
```

W tym wypadku obiekt `xmlHttp` to obiekt `XMLHttpRequest`, którego używamy do wywołania skryptu *quickstart.php* na serwerze. Jego właściwość `responseXML` ekstrahuje dane z otrzymanego dokumentu XML. Struktura XML jest hierarchiczna. Jej główny element nazywamy elementem dokumentu. W naszym przypadku jest to element `<response>`, który zawiera jednego potomka — interesującą nas wiadomość. Kiedy tylko ją dostaniemy, zostanie wyświetlona na ekranie dzięki technologii DOM, która daje dostęp do elementu `divMessage` w pliku *index.html*.

```
// aktualizuje dane wyświetlane klientowi informacjami otrzymanymi z serwera
document.getElementById("divMessage").innerHTML = '<i>' + helloMessage
+ '</i>';
```

Obiekt `document` jest domyślnym obiektem JavaScript, który pozwala manipulować elementami kodu HTML.

Pozostała część kodu z pliku *quickstart.js* obsługuje żądanie wysłane na serwer, które ma uzyskać wiadomość XML. Funkcja `createXmlHttpRequestObject()` tworzy i zwraca obiekt `XMLHttpRequest`. Jest ona dłuższa, niż być powinna, ponieważ chcieliśmy, by działała w różnych przeglądarkach. Opowiemy o tym szerzej w rozdziale 2. Na razie musisz wiedzieć, co ona robi. Obiekt `XMLHttpRequest`, pojawiający się w funkcji `process()` jako `xmlHttp`, ma powodować asynchroniczne wywołanie serwera. Oto jego postać:

```
// wysyła asynchroniczne żądanie protokołem HTTP, korzystając z obiektu XMLHttpRequest
function process()
{
  // kontynuuje jedynie jeśli obiekt xmlHttp nie jest zajęty
  if (xmlHttp.readyState == 4 || xmlHttp.readyState == 0)
  {
    // pobiera imię wpisane przez użytkownika w formularzu
    name = encodeURIComponent(document.getElementById("myName").value);
    // wykonuje stronę quickstart.php na serwerze
    xmlHttp.open("GET", "quickstart.php?name=" + name, true);
    // definiuje metodę obsługi odpowiedzi serwera
    xmlHttp.onreadystatechange = handleServerResponse;
    // wysyła żądanie do serwera
    xmlHttp.send(null);
  }
  else
    // jeśli połączenie jest zajęte, ponawia próbę po 1 sekundzie
    setTimeout('process()', 1000);
}
```

Kod, który masz przed oczyma, jest sercem AJAX. To on powoduje wysyłanie asynchronicznych żądań do serwera.

Dlaczego wywołania asynchroniczne są tak ważne? Z definicji nie powodują one zamrożenia wykonywanych działań (również przez użytkownika) aż do momentu otrzymania odpowiedzi na wysłane wywołanie. Przetwarzanie asynchroniczne implementuje się na bazie architektury ***sterowanej zdarzeniami***. Przykładem takiej architektury jest graficzny interfejs użytkownika — bez obsługi zdarzeń najprawdopodobniej musiałbyś sprawdzać bez ustanku, czy użytkownik nacisnął przycisk lub przeskalował okno. W przypadku obsługi zdarzeń to przycisk zawiadamia aplikację, że został naciśnięty, więc możesz wtedy podjąć odpowiednie kroki w funkcji obsługującej zdarzenia. W przypadku AJAX ta teoria znajduje zastosowanie przy wysyłaniu żądań do serwera - jesteś automatycznie informowany o nadejściu odpowiedzi.

Jeśli ciekawi Cię, jak działałaby aplikacja oparta o synchroniczne żądania, musisz zmienić trzeci parametr obiektu `xmlHttp.open` na wartość `false` i ręcznie wywołać funkcję `handleServerResponse()`. W czasie takiego uruchomienia okienko, w którym podajesz dane, będzie zamrożone na czas wysyłania żądania do serwera (ponieważ czas zamrożenia jest uzależniony od szybkości połączenia, jest bardzo prawdopodobne, że nie zauważysz zmian, jeśli serwerem jest Twój własny komputer).

```
// wysyła żądanie protokołem HTTP, korzystając z obiektu XMLHttpRequest
function process()
{
  // pobiera imię wpisane przez użytkownika w formularzu
  name = encodeURIComponent(document.getElementById("myName").value);
  // wykonuje stronę quickstart.php na serwerze
  xmlHttp.open("GET", "quickstart.php?name=" + name, false);
  // wysyła synchroniczne żądanie do serwera (zamraża proces aż do otrzymania odpowiedzi)
  xmlHttp.send(null);
  //odczytuje odpowiedź
  handleServerResponse();
}
```

Funkcja `process()` inicjalizuje nowe żądanie wysłane do serwera, do czego wykorzystuje obiekt `XMLHttpRequest`. Jest to możliwe tylko wtedy, gdy obiekt nie jest zajęty wykonywaniem innego żądania. W naszym przypadku taka sytuacja pojawi się, gdy serwer będzie odpowiadał dłużej niż 1 sekundę, co zdarza się przy wolnych połączeniach z internetem. Zatem funkcja `process()` rozpoczyna działanie od sprawdzenia, czy może wysłać nowe żądanie:

```
// wysyła asynchroniczne żądanie protokołem HTTP, korzystając z obiektu XMLHttpRequest
function process()
{
  // kontynuuje jedynie jeśli obiekt xmlHttp nie jest zajęty
  if (xmlHttp.readyState == 4 || xmlHttp.readyState == 0)
  {
```

Jeśli połączenie jest zajęte, korzystamy z pomocy metody `setTimeout` i próbujemy po upływie jednej sekundy (drugi argument funkcji określa czas w milisekundach, jaki musi upłynąć przed ponownym wywołaniem funkcji określonej pierwszym argumentem):

```
// jeśli połączenie jest zajęte, ponawia próbę po 1 sekundzie
    setTimeout('process()', 1000);
```

Jeśli linia jest pusta, możesz spokojnie wysłać żądanie. Poniższa linia kodu przygotowuje serwer na nadejście żądania, ale go nie wysyła:

```
// wykonuje stronę quickstart.php na serwerze
    xmlHttp.open("GET", "quickstart.php?name=" + name, true);
```

Pierwszy parametr określa metodę, jaką wysyłane będzie imię użytkownika do serwera. Masz do wyboru metody `GET` i `POST` (więcej o tych metodach dowiesz się w rozdziale 3.). Kolejny parametr to nazwa strony na serwerze, do której chcesz uzyskać dostęp. Jeśli wybrałeś wcześniej parametr `GET`, to drugi z parametrów musisz ustawić jako parę `nazwa=wartość`. Trzeci parametr przyjmuje wartość `true`, jeśli chcesz wywołać serwer asynchronicznie. Przy takim wywołaniu nie musisz czekać na odpowiedź. Wystarczy, że zdefiniujesz kolejną funkcję, która wywoła się **automatycznie** przy zmianie statusu żądania.

```
// definiuje metodę obsługi odpowiedzi serwera
xmlHttp.onreadystatechange = handleServerResponse;
```

Po ustawieniu tej opcji usiądź spokojnie i pozwól systemowi wywołać funkcję `handleServerResponse`, gdy cokolwiek stanie się z Twoim żądaniem. Kiedy wszystko już poustawiasz, inicjalizujesz żądanie wywołaniem metody `send` obiektu `XMLHttpRequest`:

```
// wysyła żądanie do serwera
xmlHttp.send(null);
```

Przyjrzyjmy się teraz funkcji `handleServerResponse()`:

```
// wykonywana automatycznie po otrzymaniu odpowiedzi z serwera
function handleServerResponse()
{
  // kontynuuje jedynie jeśli transakcja została zakończona
  if (xmlHttp.readyState == 4)
  {
    // status 200 oznacza pomyślne ukończenie transakcji
    if (xmlHttp.status == 200)
    {
```

Ta funkcja jest wywoływana za każdym razem gdy zmieni się status żądania. Funkcja zakończy działanie dopiero gdy wartość parametru `xmlHttp.response` będzie równa 4. Możesz też sprawdzać, czy transakcja HTTP zgłasza status 200, który oznacza, że nie wystąpiły problemy podczas żądania HTTP. Dopiero gdy te warunki zostaną spełnione, będziesz mógł odczytać odpowiedź serwera i wyświetlić ją na monitorze użytkownika.

Po otrzymaniu i wykorzystaniu odpowiedzi cały proces powtarza się dzięki funkcji `setTimeout`, która wywołuje funkcję `process()` po upłynięciu jednej sekundy (pamiętaj, że powtarzanie wywołania zadania po stronie klienta nie jest niezbędne czy typowe dla technologii AJAX). Poniżej podajemy wywołanie funkcji `setTimeout`:

```
// ponawia sekwencję
setTimeout('process()', 1000);
```

Powtórzmy na koniec, co dzieje się po załadowaniu strony przez użytkownika (wizualizację tego zagadnienia przedstawia rysunek 1.7):

1. Użytkownik uruchamia stronę *index.html* (odpowiada to krokom 1. – 4. na rysunku 1.7)
2. Użytkownik zaczyna (bądź kontynuuje) wpisywanie swojego imienia (krok 5. na rysunku 1.7)
3. Wykonanie funkcji `process()` z pliku *quickstart.js* powoduje asynchroniczne wywołanie skryptu *quickstart.php* na serwerze. Tekst wpisany przez użytkownika jest przekazywany jako łańcuch zapytania przez funkcję `GET`. Funkcja `handleServerResponse` obsługuje status żądania.
4. Kod z pliku *quickstart.php* wykonuje się na serwerze. Tworzy dokument XML, który zawiera wiadomość z serwera, i próbuje przekazać go do klienta.

5. Metoda `handleServerResponse` jest wielokrotnie (po każdej zmianie statusu) wykonywana po stronie serwera. Ostatnie wywołanie następuje po otrzymaniu pozytywnej odpowiedzi z serwera. Plik XML zostaje odczytany, wiadomość jest wypakowywana i wyświetlana na stronie.
6. Następuje aktualizacja danych wyświetlanych użytkownikowi, co nie przeszkadza mu we wprowadzaniu nowych. Po jednosekundowym opóźnieniu proces powraca do punktu 2.

Podsumowanie

W tym rozdziale zaprezentowaliśmy szybki wstęp do świata technologii AJAX. Jeśli chcesz nauczyć się tworzenia aplikacji w tej technologii, musisz zrozumieć, dlaczego i gdzie się je wykorzystuje. Jak każda inna technologia, AJAX nie rozwiązuje wszystkich problemów, ale pomaga radzić sobie z częścią z nich.

AJAX pozwala Ci łączyć możliwości, jakie daje praca po stronie klienta, z zaletami pracy po stronie serwera. Wszystko to po to, byś mógł poprawić jakość pracy z Twoją aplikacją. Obiekt `XMLHttpRequest` daje Ci możliwość asynchronicznego wywołania strony na serwerze przez kod po stronie klienta. Napisaliśmy tak krótki rozdział celowo i celowo pozostawiliśmy wiele pytań bez odpowiedzi. Cała książka zawiera odpowiedzi na nurtujące pytania i demonstrację wykorzystania funkcjonalności AJAX.