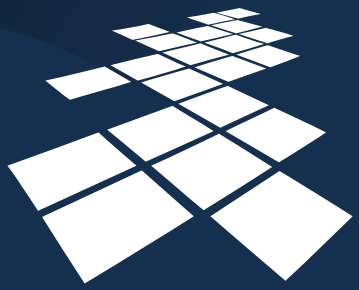
AJAX

Wykład prowadzi:
Marek Wojciechowski



UCZELNIA
ONLINE

Plan wykładu

- Czym jest AJAX?
- Technologie składowe AJAX
- XMLHttpRequest: Asynchroniczna komunikacja z serwerem
- Przykład prostej aplikacji AJAX
- Przykładowe ogólnodostępne aplikacje AJAX
- Zalety i wady AJAX
- Podsumowanie

Celem wykładu jest przedstawienie techniki tworzenia aplikacji internetowych AJAX. Wykład obejmuje: omówienie założeń AJAX i motywacji dla powstania tej techniki, przedstawienie technologii składowych AJAX i ich ról oraz ilustrację sposobu tworzenia aplikacji AJAX na prostym przykładzie. Przedstawione i krótko omówione będą też dwa popularne rzeczywiste serwisy internetowe oparte o AJAX, które miały duże znaczenie dla rozpropagowania tej techniki. Wykład zakończy wskazanie zalet i wad techniki AJAX oraz krótkie podsumowanie.

Czym jest AJAX?

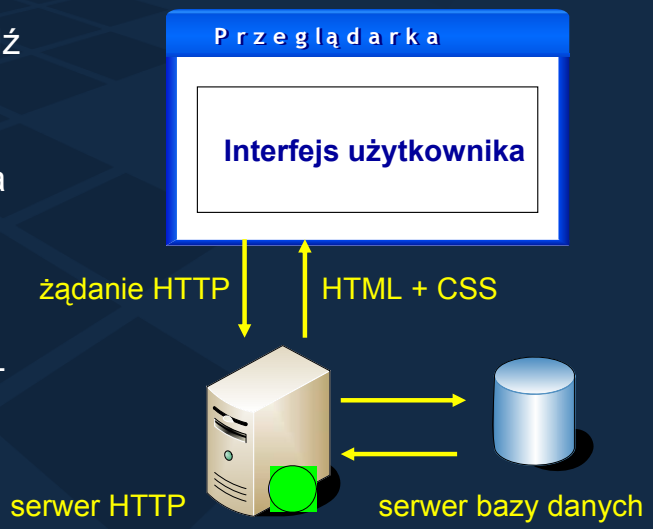
- AJAX = **A**synchronous **J**avaScript **A**nd **X**ML
- AJAX nie jest nową samodzielną technologią
- AJAX jest nową techniką tworzenia aplikacji internetowych w oparciu o (X)HTML, CSS, JavaScript, DOM, XML
- Nowe wcielenie techniki Remote Scripting
- Nowy sposób myślenia o tworzeniu aplikacji internetowych
- Oferuje poziom interaktywności aplikacji zbliżony do aplikacji desktopowych

AJAX (Asynchronous JavaScript And XML) nie jest nową technologią, ale nowym sposobem wykorzystania kombinacji istniejących technologii takich jak m.in. JavaScript, DOM i CSS, które w ostatnich latach osiągnęły dojrzałość i uzyskały powszechne wsparcie w przeglądarkach. Nazwę AJAX zaproponował Jesse James Garrett, jako niewątpliwie chwytliwy termin opisujący technikę tworzenia interaktywnych aplikacji, w dużym stopniu polegających na kodzie JavaScript po stronie klienta, realizującym asynchroniczne odwołania do serwera. AJAX bazuje na koncepcji Remote Scripting, zaproponowanej wcześniej przez firmę Microsoft, polegającej na pobieraniu danych z serwera z poziomu skryptu uruchamianego w przeglądarce. Pobieranie danych odbywa się w tym wypadku bez konieczności odświeżenia całej strony w przeglądarce, a pobierane dane są najczęściej generowane przez skrypt lub aplikację innego typu po stronie serwera.

Bez wątpienia AJAX stanowi rewolucję w sposobie myślenia o tworzeniu aplikacji internetowych, pozwalając na uzyskanie stopnia interaktywności zarezerwowanego dotychczas dla aplikacji desktopowych. Przykładowo, internetowy interfejs serwisu pocztowego GMail udostępnianego przez Google (http://gmail.google.com) pozwala na przeglądanie wielu wiadomości jednocześnie, odświeża prezentowaną zawartość skrzynki w trakcie gdy użytkownik edytuje wiadomość, itp.

Klasyczny model aplikacji internetowej

- Praca w trybie żądanie-odpowiedź
- Logika biznesowa w całości po stronie serwera
- Interakcja z serwerem poprzez linki i formularze HTML



W klasycznym modelu aplikacji internetowych interakcja użytkownika z aplikacją pracującą na serwerze aplikacji odbywa się w trybie żądanie-odpowiedź. Interfejs użytkownika ma postać powiązanych ze sobą regułami nawigacji stron WWW, w całości lub częściowo generowanych programowo przez aplikację po stronie serwera. Praca użytkownika z daną stroną interfejsu aplikacji kończy się wybraniem łącza prowadzącego do kolejnej strony aplikacji lub zatwierdzeniem formularza do wprowadzania danych za pomocą przycisku na stronie. W obu przypadkach interakcja użytkownika ze stroną kończy się wysłaniem żądania HTTP do serwera, a serwer w odpowiedzi wysyła do przeglądarki kolejną stronę aplikacji.

Wady modelu klasycznego

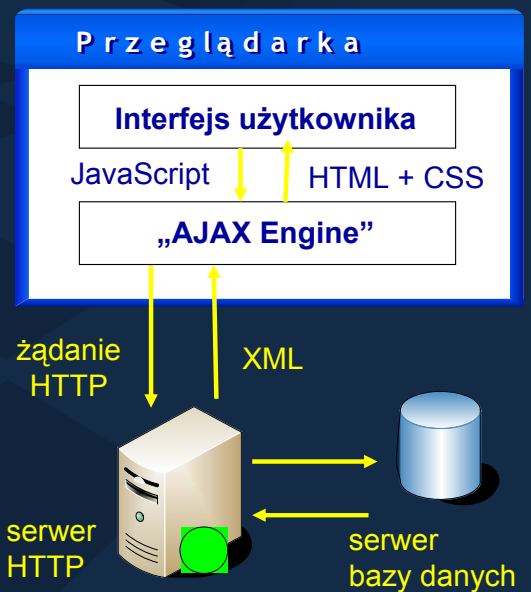
- Model będący konsekwencją pierwotnego zastosowania WWW, jakim było udostępnianie dokumentów
- Model niezorientowany na udostępnianie aplikacji
  - niski poziom interaktywności aplikacji
  - ponowne ładowanie strony po akcji użytkownika
  - marnowanie czasu serwera i przepustowości sieci
  - konieczność oczekiwania na całkowite załadowanie strony
  - długi czas ładowania złożonych stron

Klasyczny model aplikacji internetowych jest konsekwencją pierwotnego zastosowania usługi WWW, jakim było publikowanie w sieci Internet dokumentów hipertekstowych, powiązanych ze sobą łączami umożliwiającymi nawigację z jednego dokumentu do drugiego. Architektura WWW odpowiednia dla udostępniania treści nie jest jednak optymalna dla udostępniania aplikacji.

Podstawowym problemem klasycznych aplikacji internetowych jest oczywiście ich niski stopień interaktywności w porównaniu z aplikacjami desktopowymi, uruchamianymi na komputerze użytkownika. W trakcie gdy użytkownik np. wprowadza dane do formularza, serwer nie zauważy żadnej aktywności użytkownika aż do momentu zatwierdzenia formularza. Z kolei gdy serwer generuje i przesyła kolejną stronę do przeglądarki, użytkownik musi poczekać aż nowa strona zostanie w całości odebrana przez przeglądarkę. W przypadku skomplikowanych aplikacji wymagających np. dostępu do bazy danych i obciążonego serwera aplikacji, czasy przestoju mogą być znaczące. Należy też zauważyć, że często kolejna strona wysłana przez serwer do przeglądarki niewiele się różni od poprzedniej, a mimo to generowana jest od początku i przesyłana w całości. Mamy więc w tym wypadku do czynienia z marnowaniem czasu serwera i przepustowości sieci.

Model aplikacji internetowej AJAX

- Asynchroniczna komunikacja z serwerem z poziomu kodu JavaScript w przeglądarce
- Serwer wysyła dane, fragmenty kodu, a nie całe strony
- Dokument w przeglądarce modyfikowany programowo poprzez interfejs DOM



W modelu AJAX przeglądarka uruchamia aplikację, a nie tylko prezentuje strony stanowiące interfejs użytkownika. Logika aplikacji pracująca po stronie klienta (przeglądarki) jest zaimplementowana w języku JavaScript stanowiąc tzw. „AJAX engine”, a odwołania do serwera w celu pobrania potrzebnych danych są realizowane asynchronicznie bez przerywania pracy użytkownika i bez konieczności przeładowania strony w przeglądarce. Po pobraniu przez aplikację z serwera danych lub fragmentów kodu (XML, HTML, CSS, JavaScript), dokument wyświetlany w przeglądarce (stanowiący interfejs użytkownika) jest modyfikowany programowo poprzez interfejs DOM (Document Object Model) z poziomu kodu JavaScript. W ten sposób „AJAX engine” pośredniczy miedzy użytkownikiem i serwerem, odpowiadając za prezentację interfejsu użytkownika i komunikując się z serwerem w jego imieniu.

Technologie składowe AJAX

- HTML (lub XHTML) + CSS – do prezentacji danych
- JavaScript – do implementacji części logiki aplikacji po stronie klienta
- Document Object Model (DOM) – do programowej modyfikacji dokumentu w przeglądarce
- XML – jako podstawowy format przesyłanych danych
  - alternatywą jest czysty tekst lub format JSON
- Obiekt XMLHttpRequest – do asynchronicznej komunikacji z serwerem
  - alternatywą są zagnieżdżone ramki IFRAME

Podstawowy zestaw technologii wykorzystywany w aplikacjach AJAX to HTML (lub XHTML) oraz CSS, JavaScript (lub JScript), interfejs DOM, XML i obiekt XMLHttpRequest.

Język HTML (a najlepiej nowsza, zgodna z regułami XML jego wersja - XHTML) oraz arkusze stylów CSS (Cascading Style Sheets) służą do prezentacji danych.

Język JavaScript jest wykorzystywany do implementacji części logiki aplikacji po stronie klienta.

Interfejs DOM (Document Object Model) jest wykorzystywany z poziomu języka JavaScript do programowej modyfikacji dokumentu w przeglądarce np. dodawania nowych elementów, zmiany zawartości elementów, modyfikacji stylu elementów.

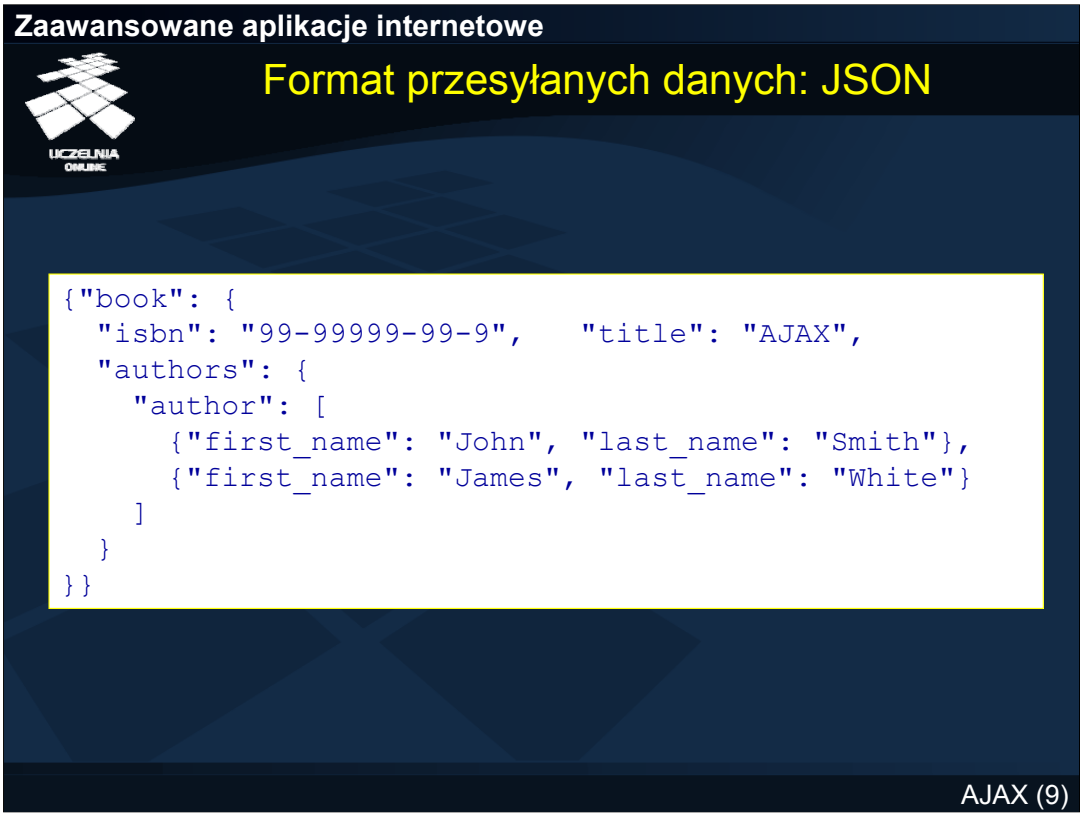
XML jest podstawowym formatem danych pobieranych przez aplikację JavaScript w przeglądarce asynchronicznie z serwera. Rozwinięcie skrótu AJAX jest w tym zakresie nieco mylące, gdyż może sugerować iż AJAX polega na XML jako jedynym obsługiwanym formacie przesyłania danych. W rzeczywistości dane mogą być przesyłane nawet czystym tekstem, a XML jest po prostu naturalnym wyborem ze względu na łatwość parsowania. Ważną alternatywę dla XML w aplikacjach AJAX stanowi opracowany specjalnie dla języka JavaScript język JSON (JavaScript Object Notation).

Ostatnim, ale kluczowym elementem AJAX jest obiekt XMLHttpRequest. Służy on do obsługi żądań HTTP wysyłanych przez kod JavaScript asynchronicznie do serwera. Wbrew nazwie, obiekt XMLHttpRequest umożliwia pobieranie z serwera danych w innych formatach niż XML (np. JSON lub czysty tekst). Obiekt XMLHttpRequest jest obecnie zalecanym sposobem asynchronicznej komunikacji z serwerem, ale należy zwrócić uwagę, że podobną funkcjonalność można osiągnąć w oparciu o zagnieżdżone ramki <iframe>.

Format przesyłanych danych: XML

```
<book>
  <isbn>99-99999-99-9</isbn>
  <title>AJAX</title>
  <authors>
    <author>
      <first_name>John</first_name>
      <last_name>Smith</last_name>
    </author>
    <author>
      <first_name>James</first_name>
      <last_name>White</last_name>
    </author>
  </authors>
</book>
```


Podstawowym formatem danych pobieranych z serwera za pomocą asynchronicznych żądań HTTP w aplikacjach AJAX jest język XML. Dla przypomnienia struktury dokumentów XML, na slajdzie przedstawiono przykładowy zapis w XML informacji o książce. Taką postać mogłyby mieć np. dane pobierane z serwera przez aplikację AJAX do obsługi internetowej biblioteki.



Ważną alternatywę dla XML jako formatu przesyłania danych stanowi JSON (JavaScript Object Notation). Jest to format umożliwiający zapisanie w postaci tekstowej obiektu JavaScript.

Na slajdzie przedstawiono przykład zapisania struktury obiektu reprezentującego książkę w notacji JSON. Obiekt „book” posiada atrybuty tekstowe „isbn” i „title” oraz atrybut „authors” zawierający tablicę obiektów reprezentujących autorów. Każdy z autorów jest opisany atrybutami tekstowymi: „first_name” i „last_name”.

Wybór formatu przesyłanych danych

- Zalety XML:
  - powszechnie znany format ogólnego przeznaczenia
  - bogactwo narzędzi po stronie serwera (np. parsery)
  - możliwość przetwarzania odebranych z serwera danych za pomocą transformacji XSLT
  - silne wsparcie ze strony biznesu
- Zalety JSON:
  - łatwość i szybkość parsowania w języku JavaScript (funkcja eval())

Jako format danych przesyłanych przez serwer do przeglądarki w odpowiedzi na asynchroniczne żądanie HTTP, aplikacje AJAX najczęściej wykorzystują XML ze względu na bogactwo wspierających go bibliotek, parserów i innych narzędzi. Przykładowo, istnieje możliwość wykorzystania transformacji XSLT w celu przetworzenia danych XML pobranych z serwera do postaci fragmentu HTML gotowego do umieszczenia w dokumencie głównym. (Transformacje XSLT są realizowane po stronie klienta w języku JavaScript poprzez obiekt XSLTProcessor, stanowiący część DOM API.) Argumentem nie bez znaczenia jest też w wypadku XML silne wsparcie ze strony dużych korporacji, mających duży wpływ na rozwój technologii informatycznych.

Zaletą JSON jest łatwość i szybkość jego parsowania po stronie klienta z poziomu kodu JavaScript. Parsowanie sprowadza się do wywołania funkcji eval(), przekazując jej jako parametr treść wysłanego przez serwer dokumentu JSON. Przykładowo, wynikiem wartościowania funkcją eval() łańcucha znaków przedstawionego na poprzednim slajdzie będzie obiekt zawierający jedną właściwość o nazwie „book”, będącą obiektem reprezentującym książkę.

Wadą JSON jest mniejsze niż w przypadku XML wsparcie przez języki i narzędzia pracujące po stronie serwera oraz fakt, że JSON jest notacją specjalizowaną dla języka JavaScript, a XML jest powszechnie znanym formatem ogólnego przeznaczenia.

Obiekt XMLHttpRequest

- JavaScript, CSS i DOM były już wcześniej wykorzystywane łącznie do zwiększenia atrakcyjności stron WWW (jako DHTML)
- Nową jakość w AJAX stanowią żądania HTTP asynchronicznie wysyłane do serwera przez kod JavaScript w przeglądarce
- Podstawowym sposobem realizacji asynchronicznych żądań HTTP z przeglądarki jest obiekt XMLHttpRequest

JavaScript, CSS i DOM wykorzystywane były od lat w połączeniu ze sobą stanowiąc tzw. dynamiczny HTML (DHTML) w celu „ożywienia” statycznych stron WWW poprzez animacje i często ciekawe efekty wizualne. Nową jakością cechującą AJAX są tak naprawdę żądania HTTP wysyłane przez kod JavaScript asynchronicznie do serwera. Po odebraniu odpowiedzi od serwera, aplikacja AJAX modyfikuje wymagające zmiany fragmenty dokumentu za pomocą interfejsu DOM. Najczęściej aplikacje AJAX realizują asynchroniczne odwołania do serwera poprzez obiekt XMLHttpRequest. (Alternatywnym rozwiązaniem są zagnieżdżone ramki <iframe>.)

Obiekt XMLHttpRequest nie jest elementem żadnego oficjalnego standardu, ale ze względu na powszechne wsparcie w przeglądarkach stał się standardem de facto. Jako pierwsza obiekt XMLHttpRequest zaoferowała programistom firma Microsoft w swojej przeglądarce Internet Explorer 5.0 w postaci obiektu ActiveX, dostępnego z poziomu języków skryptowych wspieranych przez przeglądarkę (JScript, VBScript). Wkrótce Mozilla, Apple i Opera Software opracowały swoje implementacje XMLHttpRequest dla swoich przeglądarek. Obecnie można przyjąć, że wszystkie w miarę aktualne wersje przeglądarek udostępniają obiekt XMLHttpRequest, choć sposób jego tworzenia jest odmienny w Internet Explorer niż w pozostałych przeglądarkach. Na szczęście różnice w implementacji XMLHttpRequest kończą się na sposobie tworzenia obiektu i dalsze odwołania do niego mają charakter w pełni przenaszalny.

Korzystanie z XMLHttpRequest (1/3)

- Utworzenie obiektu

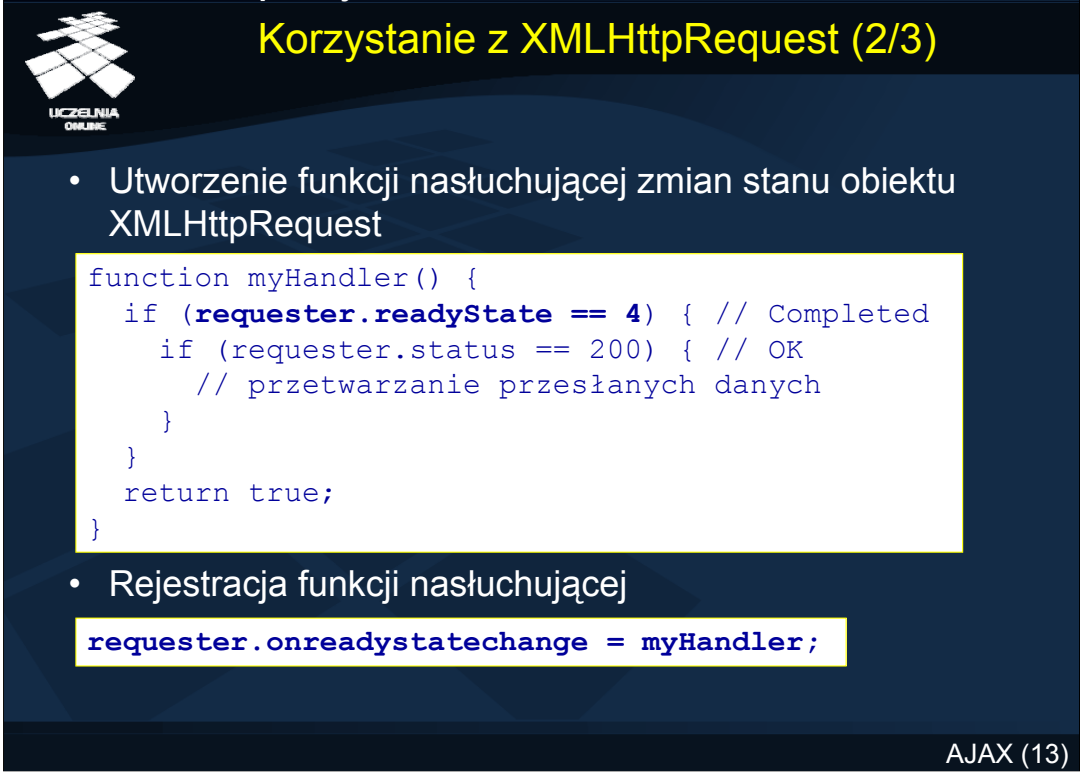
```
if (window.XMLHttpRequest) {
   // Mozilla, Safari, Opera ...
   requester = new XMLHttpRequest();
} else if (window.ActiveXObject) {
   // Internet Explorer
   requester = new ActiveXObject("Microsoft.XMLHTTP");
}
```

- Asynchroniczne wysłanie żądania

```
requester.open("GET", "/skrypt.php");
requester.send(null);
```


Na slajdzie przedstawiono dwa fragmenty kodu. Pierwszy z nich pokazuje sposób tworzenia obiektu XMLHttpRequest metodą odpowiednią dla wykorzystywanej przeglądarki. Przeglądarką, która różni się w tym zakresie od pozostałych jest Microsoft Internet Explorer. W przeglądarce firmy Microsoft obiekt XMLHttpRequest jest dostępny w postaci obiektu ActiveX, przez co dla jego działania konieczne jest oprócz włączenia przez użytkownika obsługi skryptów również możliwości uruchamiania komponentów ActiveX.

Drugi fragment kodu pokazuje sposób asynchronicznego wysłania żądania HTTP (taki sam dla wszystkich popularnych przeglądarek). W celu wysłania żądania HTTP do serwera za pomocą utworzonego obiektu XMLHttpRequest należy kolejno wywołać na jego rzecz dwie metody: open() i send(). Metoda open() inicjalizuje żądanie i wymaga podania jako argumentów metody żądania HTTP (GET, POST, itd.) oraz adresu URL. Metoda send() aktywuje połączenie i realizuje żądanie. Argumentem send() w przypadku żądania POST jest łańcuch znaków z zakodowanymi danymi do przesłania, a w przypadku żądania GET - null.



Po wywołaniu metody send(), obiekt XMLHttpRequest połączy się z serwerem i pobierze z niego dane. Żądanie to będzie zrealizowane w sposób asynchroniczny i czas, po którym proces pobierania danych zostanie zakończony nie jest znany. Przypomnijmy, że jest to kluczowa cecha modelu AJAX, dzięki której użytkownik aplikacji po wykonaniu akcji skutkującej odwołaniem do serwera może kontynuować interakcję z aplikacją, nie czekając na zakończenie przesyłania danych przez serwer. Aby aplikacja mogła zareagować na zakończenie pobierania danych z serwera, konieczne jest (przed wywołaniem żądania!) zarejestrowanie dla obiektu XMLHttpRequest funkcji nasłuchującej zmian stanu jego właściwości o nazwie readyState. Zmienna ta informuje o stanie połączenia obiektu z serwerem i może przyjmować następujące wartości liczbowe: 0 - uninitialized, 1 - loading, 2 - loaded, 3 - interactive, 4 - completed.

Pierwszy fragment kodu przedstawiony na slajdzie zawiera szkielet typowej funkcji nasłuchującej. Przetwarzanie przesłanych danych (zależne od aplikacji, na slajdzie mające postać komentarza) nastąpi po zakończeniu połączenia z serwerem i tylko wtedy gdy dla zrealizowanego w ramach połączenia żądania HTTP serwer zwróci kod statusu 200 (OK).

Drugi przedstawiony na slajdzie fragment kodu pokazuje sposób rejestracji funkcji nasłuchującej zmian stanu połączenia realizowanego poprzez obiekt XMLHttpRequest.

Korzystanie z XMLHttpRequest (3/3)

- Dostęp do danych pobranych z serwera:
  - poprzez właściwości obiektu XMLHttpRequest
  - po zakończeniu pobierania danych
  - typowo z poziomu funkcji nasłuchującej
- Właściwość responseXML:
  - zawiera drzewo DOM zawartości XML przesłanej przez serwer
- Właściwość responseText:
  - zawiera dane w postaci jednego łańcucha znaków

Po zakończeniu pobierania danych z serwera, pobrane dane można odczytać poprzez właściwości obiektu XMLHttpRequest (typowo z poziomu funkcji nasłuchującej). Właściwości obiektu XMLHttpRequest udostępniające pobrane z serwera dane to responseXML i responseText. Pierwsza z nich zawiera drzewo DOM zawartości XML przesłanej przez serwer, a druga – dane w postaci jednego łańcucha znaków. W przypadku danych przesłanych jako text/plain lub text/html jedynie właściwość responseText zawiera dane. Dla danych text/xml responseText zawiera alternatywny tekstowy zapis zawartości XML dostępnej w postaci DOM poprzez właściwość responseXML.

IFRAME jako alternatywa dla XMLHttpRequest

- Pobieranie danych z serwera bez konieczności przeładowywania całej strony można zrealizować również w oparciu o zagnieżdżone ramki IFRAME

```
<iframe id="data"
  name="data"
  style="width:0px; height:0px; border: 0px"
  src="blank.html"></iframe>
```

```
<a href="URL_on_server" target="data">
Call the server!</a>
```


Obiekt XMLHttpRequest nie jest jedynym sposobem pobierania danych z serwera bez konieczności przeładowywania całej strony. Starszą, ale ciągle niekiedy stosowaną techniką jest wykorzystanie do tego celu niewidocznych zagnieżdżonych ramek <iframe>.

Na slajdzie przedstawiono kod HTML definiujący niewidoczną zagnieżdżoną ramkę <iframe> oraz kod definiujący łącze, którego wybranie spowoduje załadowanie wskazanego dokumentu do zagnieżdżonej w dokumencie głównym ramki <iframe>.

Dokument załadowany do zagnieżdżonej ramki w odpowiedzi na kliknięcie łącza może wywołać funkcję JavaScript z dokumentu głównego i w ten sposób zainicjować przetwarzanie danych załadowanych z serwera po zakończeniu pobierania dokumentu. Ramka <iframe> ma zerowe wymiary, co powoduje że nie jest widoczna (jawne ustawienie ramki jako niewidocznej za pomocą style="display: none;" w niektórych przeglądarkach powoduje, że zawartość ramki w ogóle nie jest pobierana z serwera!).

XMLHttpRequest czy IFRAME?

- XMLHttpRequest jest rozwiązaniem nowszym i opracowanym specjalnie do realizowania asynchronicznych odwołań do serwera
  - szybszy niż IFRAME
  - wbudowana obsługa XML
- Wykorzystanie do tego celu IFRAME ma charakter „sztuczki”
  - wspierane również przez starsze przeglądarki
  - nie wymaga włączenia obsługi ActiveX w Internet Explorer

Obiekt XMLHttpRequest jest rozwiązaniem nowszym niż znacznik <iframe>, a co ważniejsze opracowanym specjalnie do realizacji asynchronicznych żądań HTTP. Wykorzystanie ramek <iframe> do tego celu ma raczej charakter „sztuczki” i nie było ich planowanym zastosowaniem. Zalety XMLHttpRequest w porównaniu z ramkami <iframe>:

(a) XMLHttpRequest najczęściej jest szybszy,

(b) XMLHttpRequest ma „wbudowaną” obsługę XML,

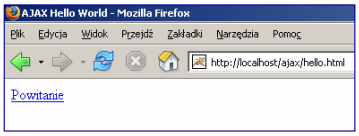
(c) załadowanie dokumentu do ramki <iframe> jest uwzględniane w historii przeglądarki, co w aplikacji AJAX powoduje nienaturalne z punktu widzenia użytkownika efekty operacji Back i Refresh.

Domyślnym sposobem realizacji asynchronicznego pobierania danych z serwera bez konieczności przeładowania całej strony jest dziś z pewnością obiekt XMLHttpRequest. Czynniki, które mogą skłonić twórcę aplikacji do wykorzystania niewidocznej ramki <iframe> to m.in.: wsparcie dla <iframe> w starszych przeglądarkach i konieczność umożliwienia uruchamiania obiektów ActiveX w przeglądarce Internet Explorer w celu skorzystania z XMLHttpRequest.

Przykład prostej aplikacji (1/10)

- Aplikacja „Hello World”

```
<HTML>
   <HEAD>
      <TITLE>AJAX Hello World</TITLE>
      <META HTTP-EQUIV="Content-Type"
            CONTENT="text/html; charset=windows-1250">
      <SCRIPT type='text/javascript'>
         function getData() {...}
         function myHandler() {...}
      </SCRIPT>
   </HEAD>
   <BODY>
      <DIV><A href="javascript:getData()">Powitanie</A></DIV>
      <DIV id="hello"></DIV>
   </BODY>
</HTML>
```



Pierwszym etapem praktycznego poznawania nowych technologii jest typowo implementacja aplikacji „Hello World”, wyświetlającej tekst powitalny. W przypadku aplikacji AJAX tekst do wyświetlenia w przeglądarce będzie pobrany asynchronicznie z serwera w odpowiedzi na żądanie użytkownika, a następnie „wbudowany” w dokument prezentowany w przeglądarce.

Na slajdzie przedstawiono kod dokumentu HTML zawierającego taką właśnie prostą aplikację AJAX oraz jej wygląd w przeglądarce (przed wybraniem łącza inicjującego pobieranie tekstu powitalnego z serwera). Ciało dokumentu zawiera element , w którym zostanie umieszczony tekst powitalny, oraz łącze, którego wybranie spowoduje uruchomienie funkcji JavaScript o nazwie getData(). Ciała funkcji getData() i pomocniczej funkcji myHandler() realizujących pobranie danych z serwera zostaną przedstawione na kolejnych slajdach.

Przykład prostej aplikacji (2/10)

hello.txt

```
Witaj świecie!
```

- Aplikacja „Hello World”: funkcja getData()

```
function getData()
{
  if (window.XMLHttpRequest) {
    requester = new XMLHttpRequest();
  }
  else
    if (window.ActiveXObject) {
      requester = new ActiveXObject("Microsoft.XMLHTTP");
    }

  requester.onreadystatechange = myHandler;

  requester.open("GET", "hello.txt");
  requester.send(null);
}
```

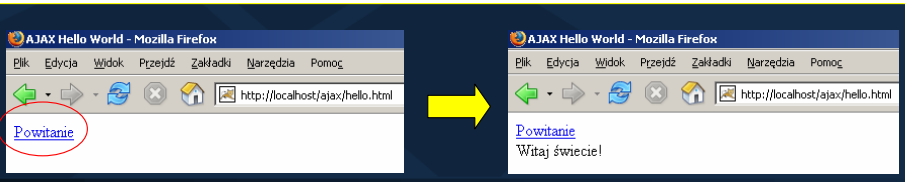

Funkcja getData() wysyła żądanie HTTP do serwera w celu pobrania zawartości pliku tekstowego hello.txt. Działanie funkcji rozpoczyna się od utworzenia obiektu XMLHttpRequest w sposób odpowiedni dla wykorzystywanej przez użytkownika przeglądarki. Następnie rejestrowana jest funkcja myHandler() jako funkcja nasłuchująca zmian stanu wykorzystywanego obiektu XMLHttpRequest (kod funkcji myHandler() na następnym slajdzie). Na końcu następuje samo wysłanie żądania HTTP do serwera.

W celu poprawnej obsługi polskich znaków, plik tekstowy zawierający tekst „Witaj świecie!” musi zostać zakodowany w systemie UTF-8, gdyż takie kodowanie zakłada obiekt XMLHttpRequest dla danych, które nie mają postaci XML.

Przykład prostej aplikacji (3/10)

- Aplikacja „Hello World”: funkcja myHandler()

```
function myHandler()
{
  if (requester.readyState == 4) {
    if (requester.status == 200) {
      el = document.getElementById("hello");
      txt = document.createTextNode(requester.responseText);
      el.appendChild(txt) }
  }
  return true;
}
```



Funkcja myHandler(), zarejestrowana jako funkcja nasłuchująca, będzie uruchamiana po każdej zmianie stanu połączenia z serwerem, w ramach którego realizowane jest asynchroniczne żądanie HTTP. Jeśli realizacja żądania została zakończona (readyState = 4) i serwer pomyślnie odpowiedział na żądanie (status HTTP = 200), funkcja myHandler() zmodyfikuje dokument HTML w przeglądarce, umieszczając w jego strukturze węzeł tekstowy z tekstem pochodzącym z pliku pobranego z serwera. Nowy węzeł tekstowy jest dodany jako węzeł potomny elementu o identyfikatorze „hello”.

Efekt działania aplikacji przedstawiono u dołu slajdu: z lewej strony wygląd ekranu przed, a z prawej strony po kliknięciu na łącze „Powitanie”.

Przykład prostej aplikacji (4/10)

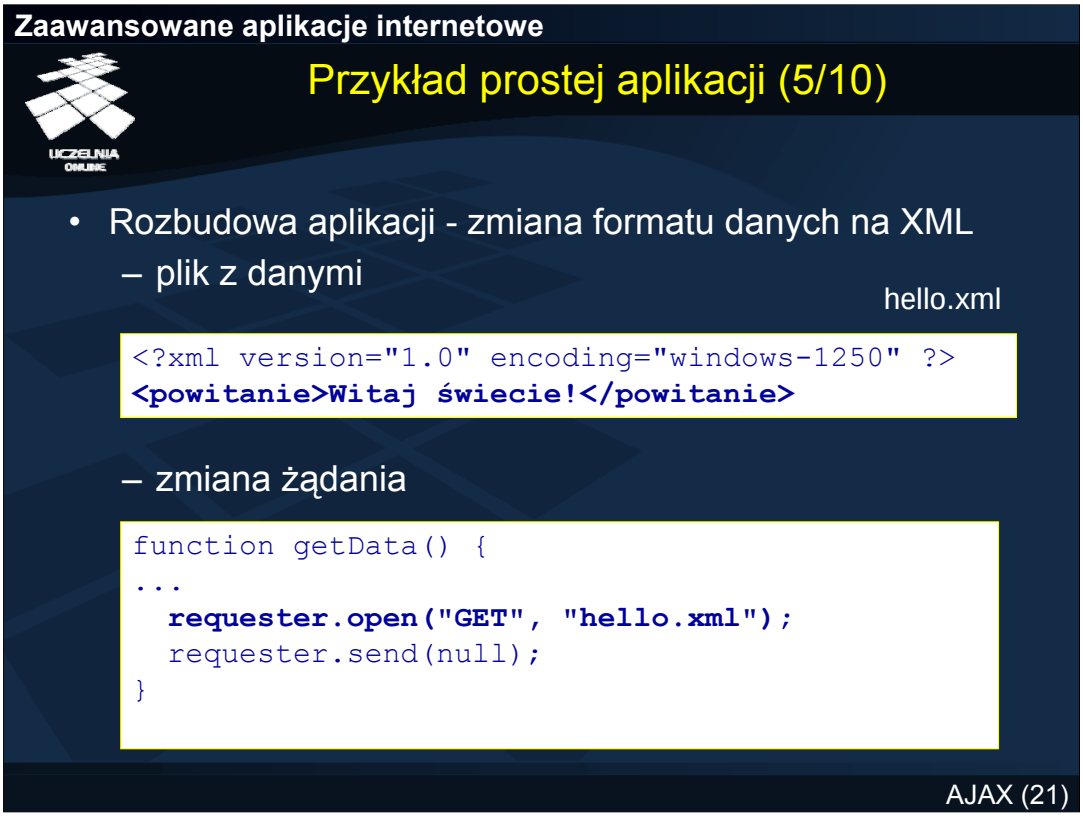
- Rozwiązanie alternatywne (właściwość innerHTML)
  - umieszczenie dowolnie złożonej zawartości HTML w elemencie

```
function myHandler()
{
  if (requester.readyState == 4) {
    if (requester.status == 200) {
      el = document.getElementById("hello");

      el.innerHTML += requester.responseText;
    }
  }
  return true;
}
```


Kod przedstawionej na poprzednich slajdach przykładowej prostej aplikacji AJAX można nieznacznie uprościć wykorzystując do modyfikacji struktury dokumentu zamiast metod interfejsu DOM createTextNode() i appendChild() właściwość innerHTML elementów drzewa dokumentu, obsługiwaną przez wszystkie popularne przeglądarki.

Zaletą właściwości innerHTML jest to, że umożliwia ona umieszczenie za pomocą jednej instrukcji dowolnie złożonej zawartości HTML we wskazanym elemencie. Zawartość ta może zawierać zagnieżdżone znaczniki HTML. Korzystanie z właściwości innerHTML nie jest jednak zalecane w dokumentach XHTML, ze względu na niespójny sposób obsługi tego rozwiązania przez różne przeglądarki.



Aplikacje AJAX najczęściej pobierają dane z serwera w formacie XML. Przewagą XML nad czystym tekstem jest przede wszystkim możliwość reprezentacji struktury i znaczenia danych za pomocą znaczników. W przypadku aplikacji AJAX, XML ma jeszcze jedną zaletę, która ma znaczenie nawet w przypadku tak prostych danych jak te przesyłane w naszej przykładowej aplikacji. Dla danych XML obiekt XMLHttpRequest akceptuje różne kodowania znaków, biorąc pod uwagę informację o kodowaniu dokumentu zawartą w prologu dokumentu XML. Dzięki temu wybierając XML jako format wymiany danych w aplikacji AJAX twórcy aplikacji mają swobodę wyboru kodowania znaków dla przesyłania danych. Przypomnijmy, że dla czystego tekstu musiał to być UTF-8.

Na slajdzie przedstawiono dokument XML zwierający tekst powitalny oraz modyfikację żądania HTTP w funkcji getData() (zmiana adresu URL pobieranego dokumentu). Oczywiście zmiana formatu przesyłanych danych w naszej przykładowej aplikacji pociąga za sobą konieczność wydobycia tekstu powitania z odebranego dokumentu XML, co zostanie przedstawione na kolejnym slajdzie.

Przykład prostej aplikacji (6/10)

- Rozbudowa aplikacji - zmiana formatu danych na XML
  - parsowanie odebranych danych XML

```
function myHandler() {
...
  greeting = requester.responseXML
             .getElementsByTagName("powitanie")[0]
             .firstChild.nodeValue;
  el = document.getElementById("hello");
  txt = document.createTextNode(greeting);
  el.appendChild(txt);
...
}
```


Dostosowanie przykładowej aplikacji do nowego formatu danych przesyłanych przez serwer obejmuje wykorzystanie właściwości responseXML obiektu XMLHttpRequest zamiast responseText oraz wydobycie z przesłanego dokumentu XML tekstu powitania za pomocą interfejsu DOM.

Właściwość responseXML udostępnia drzewo DOM pobranego z serwera dokumentu XML. Aby wyłuskać z niego treść powitania, w pierwszym kroku wyszukiwany jest węzeł o nazwie „powitanie”. Ponieważ metoda getElementsByTagName() wyszukująca węzły o podanej nazwie zwraca tablicę węzłów, konieczne jest pobranie z tablicy pierwszego elementu (znamy strukturę dokumentu i wiemy, że zawiera on tylko jeden taki węzeł). Tekst zawarty w danym węźle jest w drzewie DOM reprezentowany jako zagnieżdżony węzeł typu tekstowego. Dlatego też w drugim kroku, po znalezieniu węzła „powitanie”, pobierany jest z niego węzeł tekstowy (pierwszy i jedyny węzeł potomny), a następnie odczytana wartość tego węzła tekstowego.

Przykład prostej aplikacji (7/10)

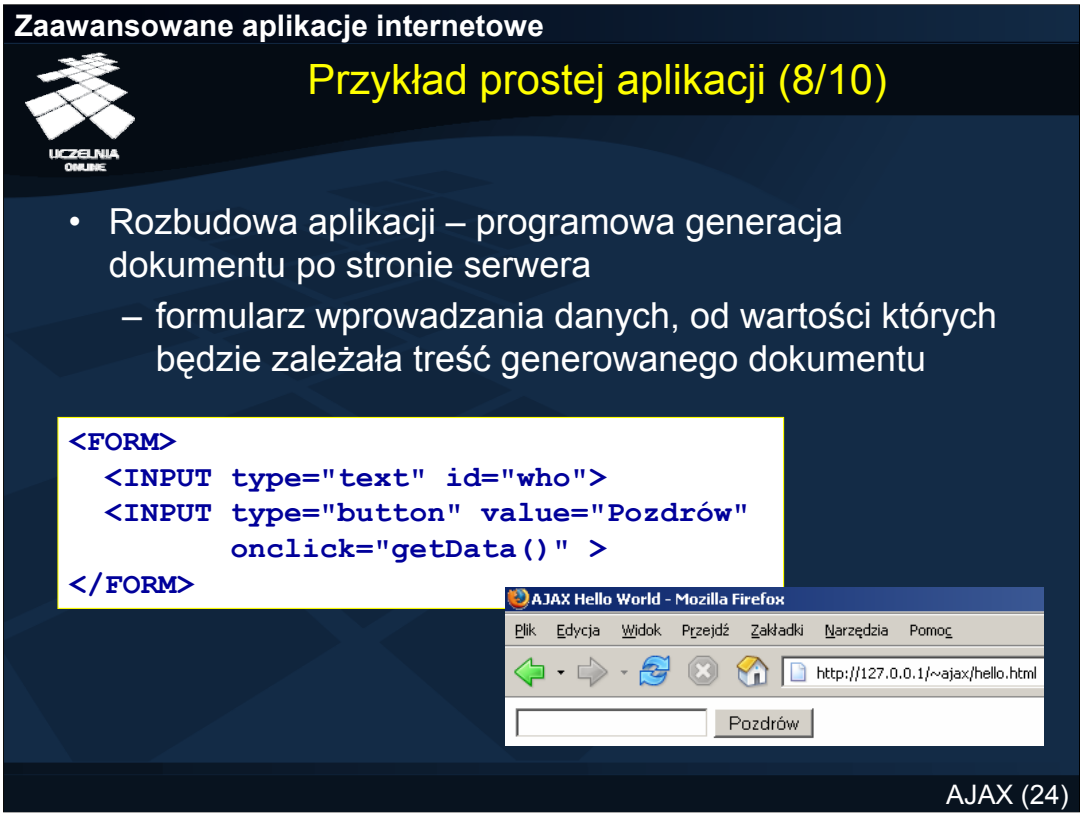
- Rozbudowa aplikacji – programowa zmiana stylu elementu
- Sposoby zmiany stylu elementu poprzez interfejs DOM:
  - ustawienie wartości właściwości className węzła
  - bezpośrednie modyfikowanie właściwości style, mającej postać tablicy właściwości CSS

```
el.style.color = "red";
el.style.fontWeight = "bold";
```

Kolejna modyfikacja przykładowej aplikacji to zmiana stylu elementu prezentującego dane pobrane z serwera.

Istnieją dwa sposoby zmiany stylu elementu poprzez interfejs DOM. Pierwszy polega na ustawieniu wartości właściwości className węzła, a drugi na bezpośrednim modyfikowaniu właściwości style, mającej postać tablicy właściwości CSS. Pierwszy ze sposobów opiera się o arkusze stylów i pozwala na jednoczesne zaaplikowanie do elementu wielu reguł CSS. Drugi sposób jest najczęściej wykorzystywany w połączeniu z pierwszym w celu „dostrojenia” wyglądu elementu poprzez uzupełnienie czy przesłonięcie reguł z arkusza stylów.

W naszym prostym przykładzie wykorzystany został drugi sposób do zmiany koloru i grubości czcionki elementu.



W dotychczasowej formie, przykładowa aplikacja pobiera asynchronicznym żądaniem HTTP z serwera zawartość statyczną tj. serwer przesyła plik tekstowy lub plik XML odczytany z systemu plików serwera. W rzeczywistych aplikacjach AJAX żądania wysyłane do serwera są sparametryzowane, a zawartość wysyłana w odpowiedzi przez serwer jest generowana przez skrypty, serwlety, itp. Aby zilustrować sposób przesyłania parametrów w asynchronicznych żądaniach HTTP, zmodyfikujemy aplikację przykładową tak, aby tekst pozdrowienia zawierał imię wprowadzone przez użytkownika do formularza. Na slajdzie przedstawiono kod formularza HTML, którym należy zastąpić łącze dotychczas inicjujące pobieranie danych. Kliknięcie przycisku w formularzu spowoduje wywołanie funkcji getData(). Funkcja ta będzie teraz musiała w wysyłanym żądaniu do serwera zawrzeć tekst wprowadzony do pola tekstowego formularza.

Przykład prostej aplikacji (9/10)

- Rozbudowa aplikacji – programowa generacja dokumentu po stronie serwera
  - przekazanie w żądaniu danych z formularza

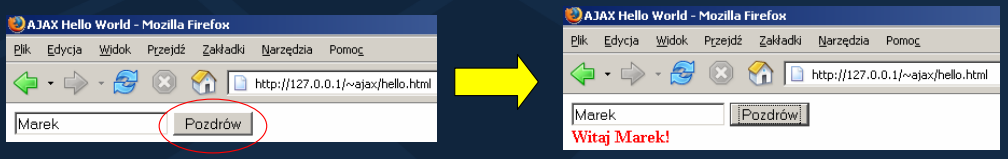
```
function getData()
{ ...
  requester = new XMLHttpRequest();
  ...
  requester.onreadystatechange = myHandler;
  requester.open("GET", "hello.php?who="
            + document.getElementById("who").value);
  requester.send(null);
}
```


Żądanie do serwera jest wysyłane gdy użytkownik naciśnie przycisk w formularzu. URL żądania wskazuje skrypt PHP. Zamiast skryptu PHP można oczywiście wykorzystać inny program generujący XML po stronie serwera HTTP, np. serwlet. Przed wysłaniem żądania, do adresu URL doklejany jest parametr zawierający imię wprowadzone przez użytkownika do pola tekstowego w formularzu.

Przykład prostej aplikacji (10/10)

- Rozbudowa aplikacji – programowa generacja dokumentu po stronie serwera
  - skrypt po stronie serwera hello.php

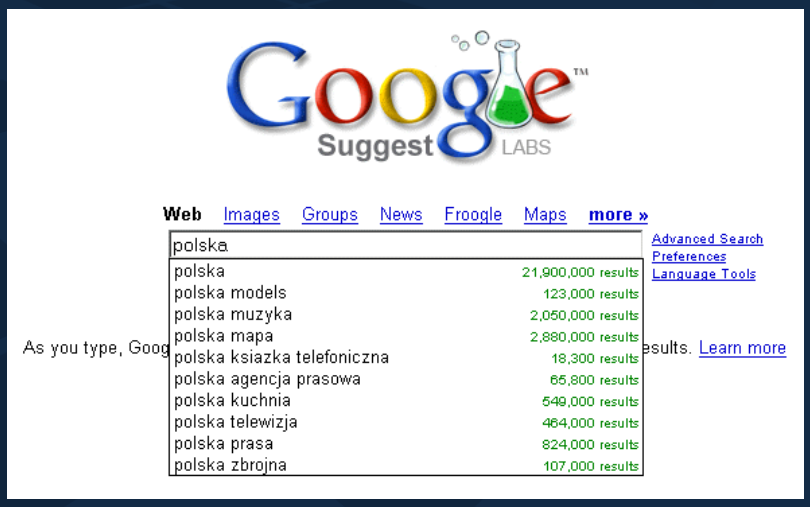
```
<?php header("Content-type: text/xml"); ?>
<?xml version="1.0" encoding="windows-1250" ?>
<powitanie>Witaj <?php echo $_GET['who']; ?>!
</powitanie>
```



Przedstawiony na slajdzie skrypt PHP generuje dokument o takiej samej strukturze jaką miał wcześniejszy statyczny plik XML. Dlatego nie jest konieczne modyfikowanie funkcji myHandler() przetwarzającej dane przesłane przez serwer. Wyróżnione fragmenty skryptu PHP to poinformowanie przeglądarki o typie przesyłanej zawartości (text/xml) oraz odczyt wartości przesłanego parametru żądania (w naszym przypadku imienia).

U dołu slajdu przedstawiono efekt działania zmodyfikowanej aplikacji.

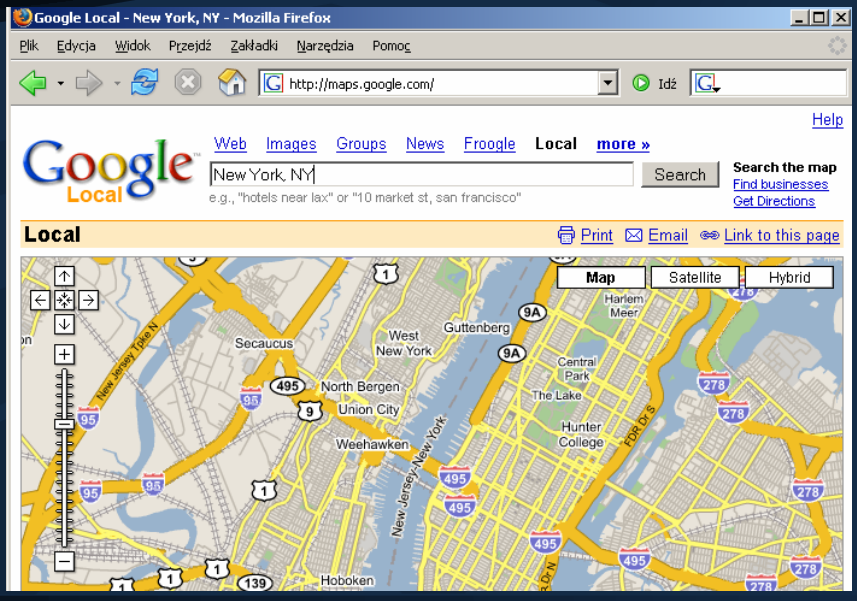
Przykład aplikacji AJAX: Google Suggest



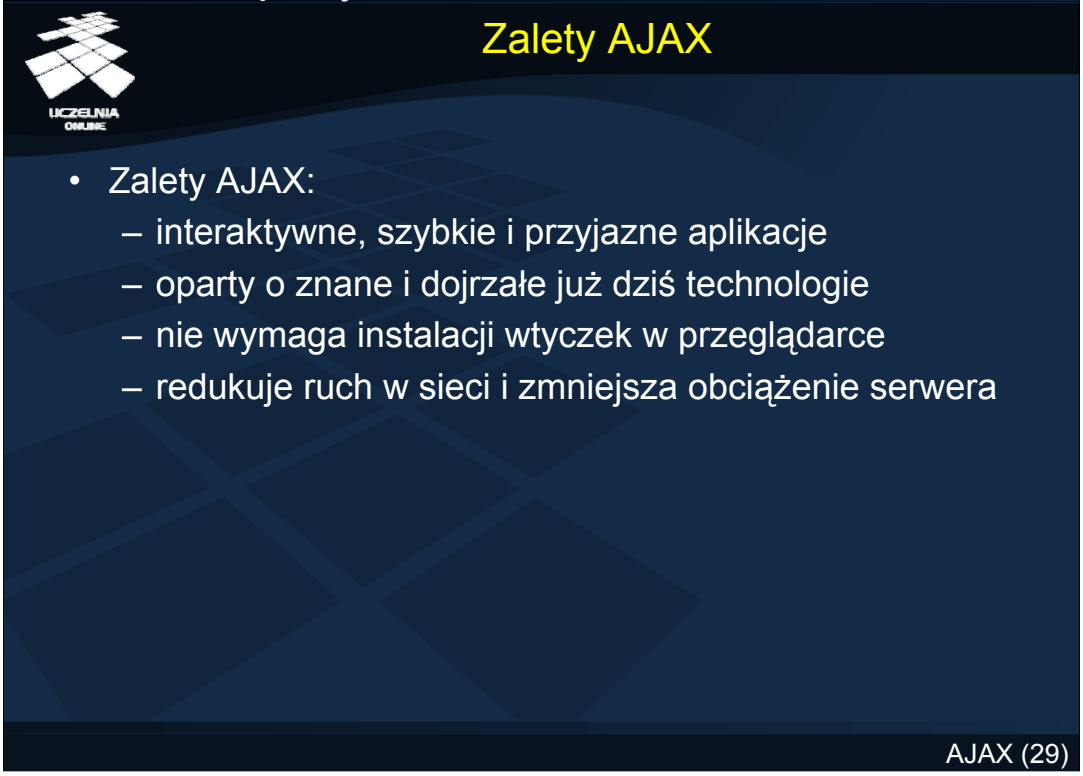
Coraz więcej korporacji m.in. Google, Yahoo i Amazon oferuje serwisy wykorzystujące AJAX. Szczególną rolę w rozwoju i propagowaniu idei AJAX odegrała firma Google, ponieważ to właśnie niespotykana wcześniej interaktywność jej pionierskich serwisów takich jak Google Suggest i Google Maps zwróciła uwagę świata na technikę, dzięki której osiągnięto tak imponujące efekty.

Google Suggest (http://labs.google.com/suggest/) to aplikacja, która wspiera użytkownika w formułowaniu kryteriów wyszukiwania dla wyszukiwarki Google. W trakcie wprowadzania kryteriów przez użytkownika, kod JavaScript odczytuje dotychczas wprowadzone znaki i wysyła je w tle asynchronicznie do serwera. Serwer w oparciu o zgromadzoną historię kryteriów wyszukiwania, odsyła do przeglądarki listę najbardziej popularnych zapytań rozpoczynających się prefiksem wprowadzonym przez użytkownika. Odebrane z serwera dane są przedstawiane użytkownikowi w formie listy rozwijanej.

Przykład aplikacji AJAX: Google Maps



Google Maps (http://maps.google.com/) to interaktywna mapa oferująca możliwości pomniejszania i powiększania prezentowanego fragmentu i wyszukiwania obiektów na mapie. Najciekawszym i wykorzystującym AJAX elementem serwisu jest sama mapa. Użytkownik obsługuje mapę za pomocą myszy, „przemieszczając się" w czterech kierunkach, zmieniając skalę oraz przełączając się między trybem mapy i zdjęć satelitarnych. Prezentowany obraz składany jest z kawałków, które w miarę potrzeby są asynchronicznie pobierane z serwera bez konieczności przeładowania całej strony. W porównaniu z interaktywnymi mapami opartymi o tradycyjny model aplikacji internetowej, Google Maps posiada dwie istotne dla użytkownika zalety będące konsekwencją zastosowania techniki AJAX: (1) w czasie pobierania danych z serwera możliwa jest interakcja użytkownika z aplikacją, (2) mapa jest szybciej uaktualniana, gdyż pobierane są tylko fragmenty mapy, które nie zostały pobrane wcześniej.



Podstawową zaletą AJAX jest to, że umożliwia on uzyskanie poziomu interaktywności aplikacji dotychczas niedostępnego dla aplikacji internetowych. Aplikacje AJAX są przyjazne użytkownikom gdyż przypominają aplikacje desktopowe. Dodatkowo AJAX pozwala skrócić czas ładowania stron aplikacji dzięki możliwości asynchronicznego pobierania z serwera nie tylko danych, ale również kodu JavaScript i reguł stylistycznych CSS dopiero wtedy gdy będą potrzebne. Uzyskanie tego efektu wymaga oczywiście dokładnego przemyślenia sposobu funkcjonowania aplikacji i podziału kodu na części, które muszą być dostępne od początku i te, które można pobrać później.

Ważną pozytywną cechą AJAX jest też fakt powszechnego wsparcia technologii składowych AJAX przez obecne przeglądarki bez konieczności instalowania wtyczek czy bibliotek runtime, w przeciwieństwie do w pewnym stopniu alternatywnych dla AJAX rozwiązań takich jak aplety Java, aplikacje Java Web Start czy Flash.

Nie bez znaczenia jest też ograniczenie ilości danych przesyłanych z serwera do przeglądarki i przeniesienie części przetwarzania na komputer użytkownika. Dzięki temu redukuje się ruch w sieci i obciążenie serwera aplikacji.

Wady AJAX

- Wady AJAX:
  - język JavaScript nieodpowiedni dla dużych aplikacji
  - konieczność włączenia obsługi JavaScript
  - wrażliwy na opóźnienia w komunikacji sieciowej
  - trudna implementacja, testowanie i debugowanie aplikacji
  - niedojrzałe biblioteki i narzędzia programistyczne

AJAX nie jest techniką pozbawioną wad. Podstawowym problemem jest język JavaScript, w którym implementowana jest logika aplikacji po stronie klienta. JavaScript to skryptowy język, który nie został zaprojektowany jako język do tworzenia dużych aplikacji. Jest on językiem interpretowanym, ze słabą kontrolą typów danych. Mechanizmy obiektowe są oparte o prototypowanie (a nie o klasy).

Z pewnością istotnym mankamentem jest konieczność włączenia obsługi JavaScript w przeglądarce, a w przypadku przeglądarki Internet Explorer również obsługi ActiveX.

Innym problemem są w dalszym ciągu jednak występujące opóźnienia w komunikacji sieciowej. Opóźnienia te nie powodują w przypadku AJAX wstrzymania działań użytkownika, ale sprawiają, że efekty jego interakcji z aplikacją mogą nie być natychmiastowe. Problem ten można rozwiązać poprzez wyświetlanie na stronie informacji o trwającym pobieraniu danych w przypadku czasochłonnych operacji na serwerze.

Na zakończenie należy też przyznać, że implementowanie złożonych aplikacji AJAX jest w chwili obecnej niełatwym zadaniem. Trudne jest również testowanie i debugowanie aplikacji. Z jednej strony wynika to z natury języka JavaScript, a z drugiej z podziału logiki aplikacji między klienta i serwer. Na szczęście pojawia się coraz więcej wzorców projektowych, bibliotek, szkieletów aplikacji i środowisk IDE wspierających AJAX. W chwili obecnej (wiosna 2006) są one jeszcze niedojrzałe.

Podsumowanie

- AJAX = nowa technika tworzenia aplikacji internetowych
- Oparty głównie o:
  - HTML, CSS , JavaScript, DOM, XMLHttpRequest
- Wykorzystuje mechanizm Remote Scripting
- Pozwala na tworzenie interaktywnych aplikacji na wzór aplikacji desktopowych
- Trudna implementacja i testowanie aplikacji

AJAX to nowa technika tworzenia aplikacji internetowych oparta o uznane i dojrzałe już technologie takie jak HTML (lub XHTML), CSS, JavaScript, DOM i XML, wzbogacone o obiekt XMLHttpRequest. AJAX wykorzystuje mechanizm Remote Scripting, polegający na pobieraniu danych z serwera przez skrypt działający w przeglądarce.

Podstawową zaletą AJAX jest to, że umożliwia on uzyskanie poziomu interaktywności aplikacji niedostępnego dla klasycznych aplikacji internetowych. Główną wadą AJAX jest trudność implementacji aplikacji w tej technice ze względu na słabe strony języka JavaScript, słabe wsparcie ze strony narzędzi i podział logiki aplikacji na część wykonującą się w przeglądarce i część pracującą na serwerze. Z tych samych powodów kłopotliwe jest testowanie i debugowanie aplikacji.

Materiały dodatkowe

- J. J. Garrett, Ajax: A New Approach to Web Applications, February 2005
- R. Asleson, N. T. Schutta, Foundations of Ajax, Apress, 2005
- D. Crane, E. Pascarello, D. James, Ajax in Action, Manning Publications, 2005
- Ajax Patterns, http://ajaxpatterns.org/
- The AJAX Revolution. Join in, http://www.telerik.com/AJAX/
- JSON (JavaScript Object Notation), http://www.json.org/

•J. J. Garrett, Ajax: A New Approach to Web Applications, February 2005
•R. Asleson, N. T. Schutta, Foundations of Ajax, Apress, 2005
•D. Crane, E. Pascarello, D. James, Ajax in Action, Manning Publications, 2005
•Ajax Patterns, http://ajaxpatterns.org/
•The AJAX Revolution. Join in, http://www.telerik.com/AJAX/
•JSON (JavaScript Object Notation), http://www.json.org/