Aplikacje webowe

dr inż. Aleksander Smywiński-Pohl

Elektroniczne Przetwarzanie Informacji
Konsultacje: czw. 14.00-15.30, pokój 3.211

Plan prezentacji

URL

Komunikacja HTTP

Formularze

CGI

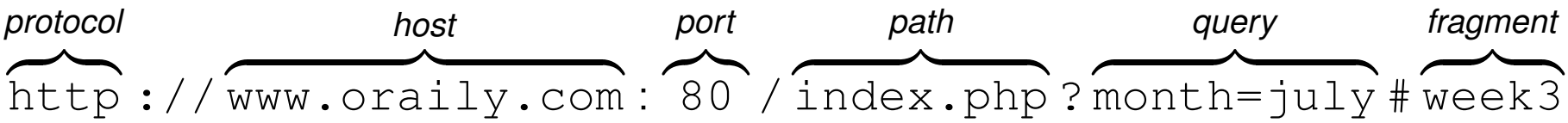
JavaScript

Frameworki webowe

REST

Plan prezentacji

URL – Unified Resource Locator



- *protocol* – protokół komunikacyjny HTTP, HTTPS
- *host* – nazwa domenowa hosta
- *port* – port na którym działa usługa HTTP
- *path* – ścieżka do zasobu (może być obsługiwana dynamicznie)
- *query* – zapytanie, zawiera parametry żądania GET
- *fragment* – adres fragmentu dokumentu (identyfikowany przez atrybut `name` znacznika `a`)

URL względny i bezwzględny

- URL *bezwzględny* – zawiera protokół, nazwę hosta oraz port (opcjonalnie)
- URL *względny* – nie zawiera powyższych, dzieli się na:
  - ścieżkę *bezwzględną* – rozpoczyna się od znaku `/`, np. `/index.html`
  - ścieżkę *względną* – nie rozpoczyna się od znaku `/`, np. `index.html`, oznacza zasób relatywny względem aktualnej ścieżki

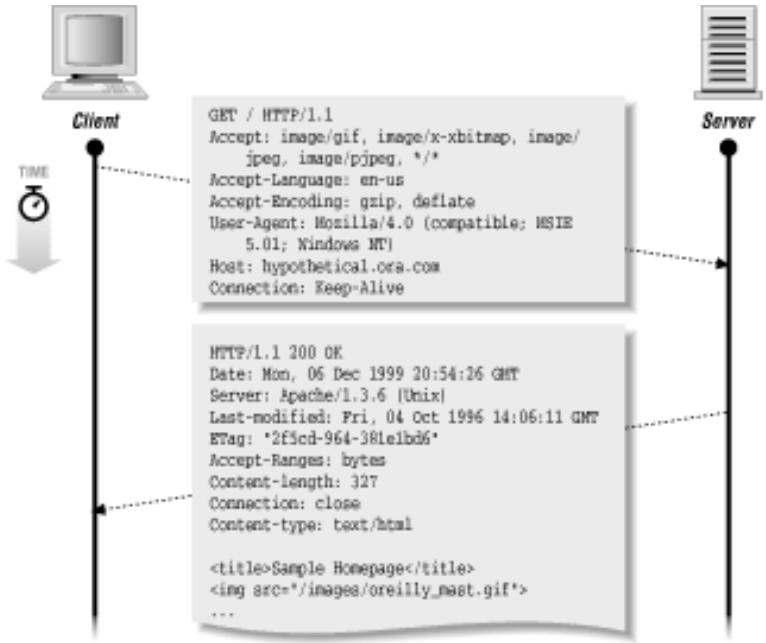
Kodowanie w URLach

- Niektóre znaki mają specjalne znaczenie w adresie URL, np.: `?`, `#`, `/`
- Treść, która ma zostać wysłana na serwer niejednokrotnie musi zawierać te znaki (np. treść formularza)
- Konieczne jest zakodowanie znaków specjalnych:
  - znaki specjalne zapisywane są w kodzie szesnastkowym
  - rozpoczynają się od znaku `%`, po którym następuje szesnastkowy kod znaku, np. `#` $\rightarrow$ `%23`.

Plan prezentacji

Cykl żądanie-odpowiedź (request-response)

- Klient (przeglądarka) wysyła żądanie
- Serwer wysyła odpowiedź



Rysunek :
http://oreilly.com/catalog/httppr/chapter/http_pkt.html

Żądanie przeglądarki

```
GET / HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/
    jpeg, image/pjpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE
    5.01; Windows NT)
Host: some.host.com
Connection: Keep-Alive
```

Wiersz żądania

- Metoda żądania: GET, POST, HEAD, PUT, DELTE, CONNECT, OPTIONS, TRACE
- URL zasobu: bezwzględny ścieżka do zasobu
- Protokół: HTTP, HTTPS + wersja

Wiersze pól nagłówkowych żądania

Field name `Content-Type` : *Field value* `text/html`

- `Host` – nazwa docelowego hosta
- `Content-Length` – długość żądania (w bajtach)
- `Content-Type` – typ nośnika żądania
- `Authorization` – określa nazwę i hasło użytkownika
- `User-Agent` – określa nazwę, wersję i platformę klienta
- `Referer` – określa adres, spod którego użytkownik trafił na ten adres
- `Cookie` – zawiera wartość ciasteczek ustawionych wcześniej przez serwer

Odpowiedź serwera

```
HTTP/1.1 200 OK
Date: Mon, 06 Dec 1999 20:54:26 GMT
Server: Apache/1.3.6 (Unix)
Last-Modified: Fri, 04 Oct 1996 14:06:11 GMT
ETag: "2f5cd-964-381e1bd6"
Accept-Ranges: bytes
Content-length: 327
Connection: close
Content-type: text/html

<title>Sample Homepage</title>
...
```

Wiersz stanu

$$\overbrace{\texttt{HTTP/1.1}}^{\textit{protocol}}\ \overbrace{\texttt{200 OK}}^{\textit{status}}$$

- `2xx` – żądanie jest poprawne
- `3xx` – przekierowanie
- `4xx` – wystąpił błąd, którego źródłem jest klient
- `5xx` – wystąpił błąd, którego źródłem jest serwer

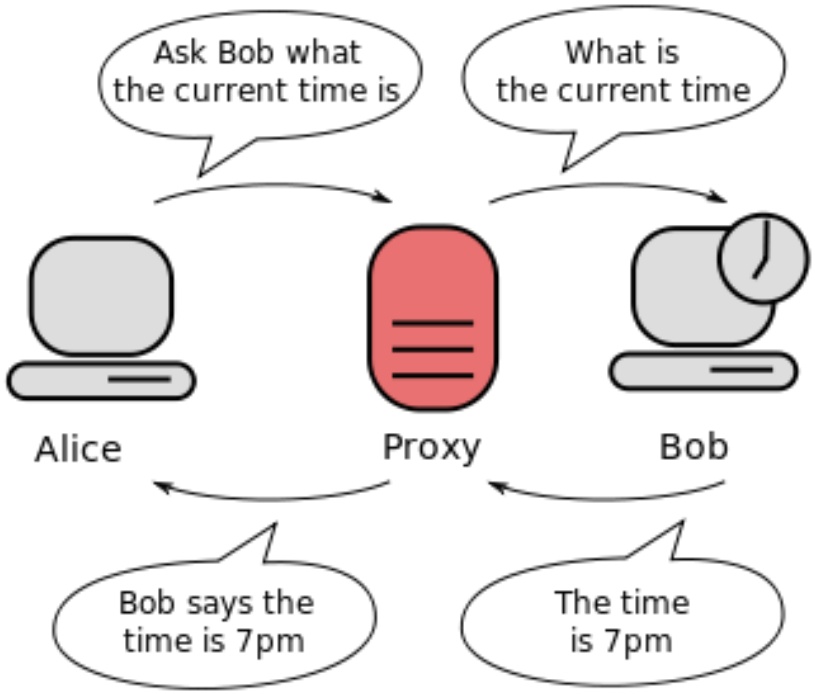
Wiersze pól nagłówkowych odpowiedzi

- `Content-Base` – określa bazy adres URL dla wszystkich względnych adresów URL
- `Content-Length` – określa długość treści zasadniczej odpowiedzi
- `Content-Type` – określa typ odpowiedzi (stosując kody MIME)
- `Date` – określa datę wysłania odpowiedzi
- `ETag` – określa unikalny znacznik treści (ulega zmianie przy zmianie treści)

Wiersze pól nagłówkowych odpowiedzi – cd.

- `Last-Modified` – określa datę ostatniej modyfikacji zasobu
- `Location` – w przypadku przekierowania określa docelowy adres URL
- `Server` – określa nazwę i wersję serwera
- `Set-Cookie` – wysyła żądanie ustawienia Cookie
- `WWW-Authenticate` – określa sposób uwierzytelniania

Proxy



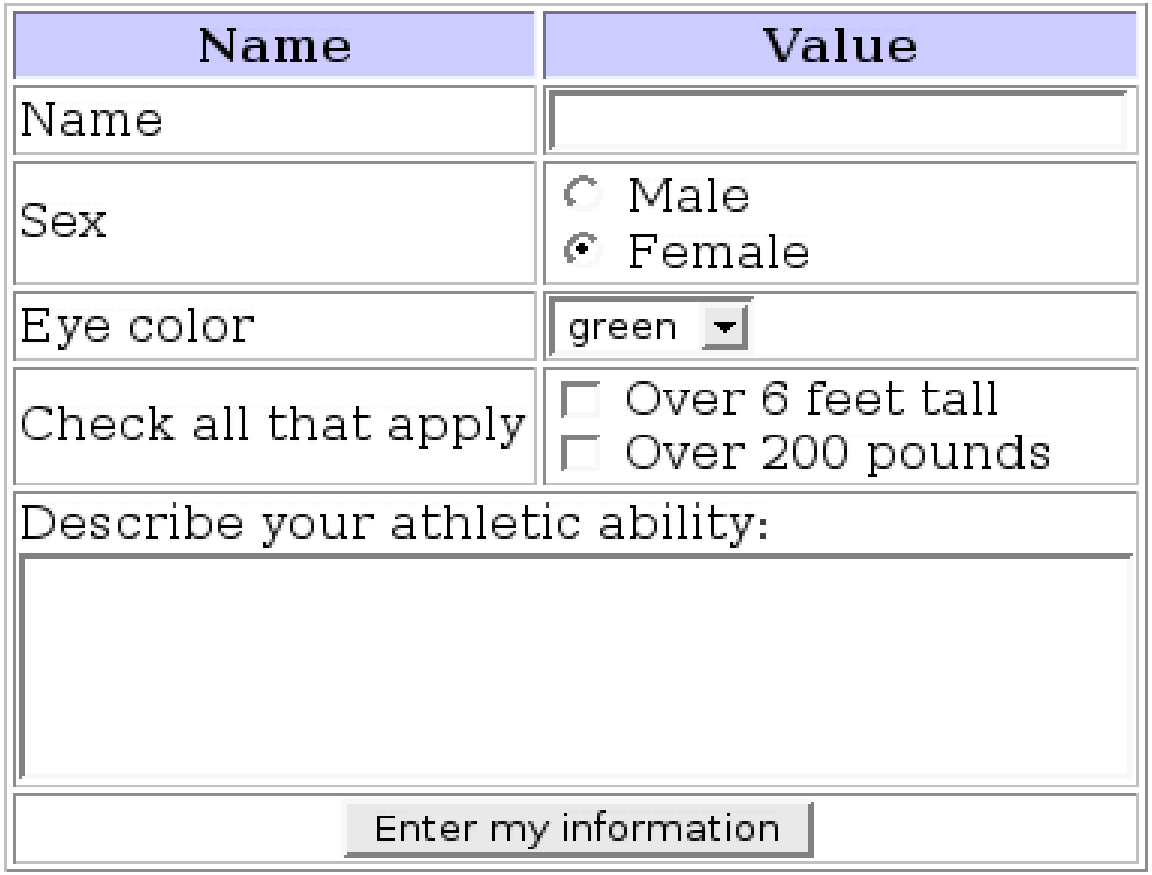
Rysunek : http://en.wikipedia.org/wiki/File:Proxy_concept_en.svg

Działanie proxy

- Przy pierwszym żądaniu zasobu serwer proxy kontaktuje się z oryginalnym serwerem
- Odpowiedź serwera jest zapisywana w pamięci podręcznej proxy
- Przy kolejnym żądaniu do tego samego zasobu proxy wysyła do serwera żądanie HEAD
- Jeśli znacznik `ETag` jest identyczny jak w przypadku pierwszego żądania serwer proxy nie pobiera ponownie tego zasobu
- W przeciwnym razie serwer proxy ponownie pobiera zasób

Plan prezentacji

Przykładowy formularz



Rysunek : http://en.wikipedia.org/wiki/File:Sample_web_form.png

Formularz GET

- Formularz

```
<form action="/app.cgi" method="GET">
 User Name: <input name="user" type="text" />
 <input type="submit" />
</form>
```

- Zawartość komórki `user`

  `Smywiński-Pohl`
- Nagłówek

```
GET /app.cgi?user=Smywi%C5%84ski-Pohl HTTP 1.1
Host: some.host.com
...
```

- Treść żądania

  brak treści

Formularz POST

- Formularz

```
<form action="/app.cgi" method="POST">
 User Name: <input name="user" type="text" />
 <input type="submit" />
</form>
```

- Zawartość komórki `user`

  `Smywiński-Pohl`

- Nagłówek

```
POST /app.cgi HTTP 1.1
Host: some.host.com
...
```

- Treść żądania

  `user=Smywi%C5%84ski-Pohl`

Znacznik `form`

- `method` – określa rodzaj żądania do serwera: GET lub POST
- `action` – określa adres, pod który zostanie wysłane żądanie
- `enctype` – dopuszczalne wartości (dla żądania POST):
  - `application/x-www-form-urlencoded` (wartość domyślna)
  - `multipart/form-data` – dla formularzy do przesyłania plików
- `autocomplete` – pozwala wyłączyć automatyczne uzupełnianie formularza
- `novalidate` – pozwala wyłączyć mechanizm walidacji po stronie przeglądarki

Znacznika `input`

- `name` – nazwa pola formularza (użyta jest do zakodowania treści formularza)
- `value` – wartość pola formularza (domyślna wartość ustawiana przez serwer)
- `placeholder` – treść zachęcająca do wypełnienia formularza
- `required` – wskazuje, że pole jest wymagane
- `pattern` – wyrażenie regularne służące do walidacji pola
- `min`, `max` – wartość minimalna i maksymalna
- `type` – typ pola formularza

Typy pól formularza – HTML 4

- `button` – przycisk generujący akcję
- `checkbox` – pole opcji tak/nie
- `file` – plik
- `hidden` – pole ukryte
- `password` – pole hasła
- `radio` – pole opcji typu „radio”
- `reset` – przycisk resetowania formularza
- `submit` – przycisk wysyłki formularza
- `text` – pole tekstowe

Typy pól formularza – HTML 5

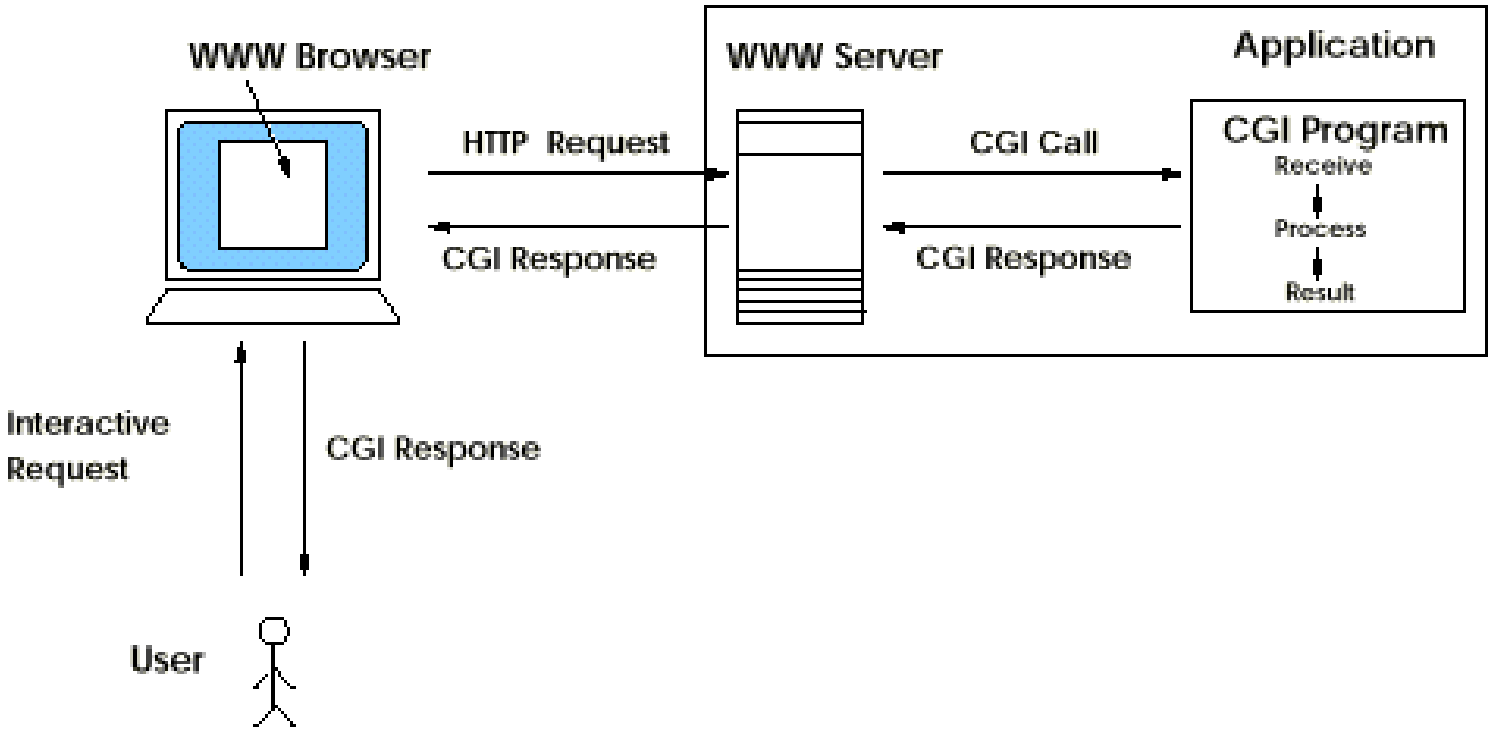
- `color` – wybór koloru
- `date` – wybór daty
- `datetime` – wybór daty i czasu
- `email` – adres e-mail
- `image` – obrazek
- `month` – miesiąc
- `number` – liczba
- `range` – zakres
- `search` – pole wyszukiwania
- `...`

Pozostałe elementy formularza

- `select` – wybór jednej z wielu opcji
- `option` – opcja do wyboru
- `textarea` – duże pole tekstowe

Plan prezentacji

Common Gateway Interface



Rysunek : http://www.irt.org/articles/js184/

CGI

- pierwszy jednolity mechanizm komunikacji serwera z aplikacjami webowymi
- definiuje interfejs pomiędzy serwerem a aplikacją
  - standardowe wejście (`STDIN`) – treść żądania
  - standardowe wyjście (`STDOUT`) – nagłówek i treść odpowiedzi
  - zmienne środowiskowe – parametry żądania
- CGI jest niezależne względem języka programowania
- skrypty są wykonywane w momencie, w którym żądanie dociera do serwera

CGI – zmienne środowiskowe

- `CONTENT_LENGTH` – długość treści żądania
- `CONTENT_TYPE` – typ treści
- `QUERY_STRING` – treść zapytania dla żądania GET
- `REMOTE_ADDR` – adres IP klienta
- `REMOTE_PORT` – port klienta
- `SERVER_ADDR` – adres IP serwera
- `SERVER_PORT` – port serwera
- `HTTP_ACCEPT` – treści akceptowane przez klienta
- `HTTP_COOKIE` – treść Cookie
- `HTTP_REFERER` – URL dokumentu, który skierował użytkownika do tego zasobu

Program w języku C jako skrypt CGI

```c
#include <stdio.h>
#include <stdlib.h>

int main (){
  printf("Content-Type: text/html\n\n");
  printf("<html>\n");
  printf("<body>\n");
  printf("User IP: %s<br>\n",getenv("REMOTE_ADDR"));
  printf("User Port: %s<br>\n",getenv("REMOTE_PORT"));
  printf("Server IP: %s<br>\n",getenv("SERVER_ADDR"));
  printf("Server Port: %s<br>\n",getenv("SERVER_PORT"));
  printf("Query string: %s<br>\n",getenv("QUERY_STRING"));
  printf("</body>\n");
  printf("</html>\n");
}
```

Wady CGI

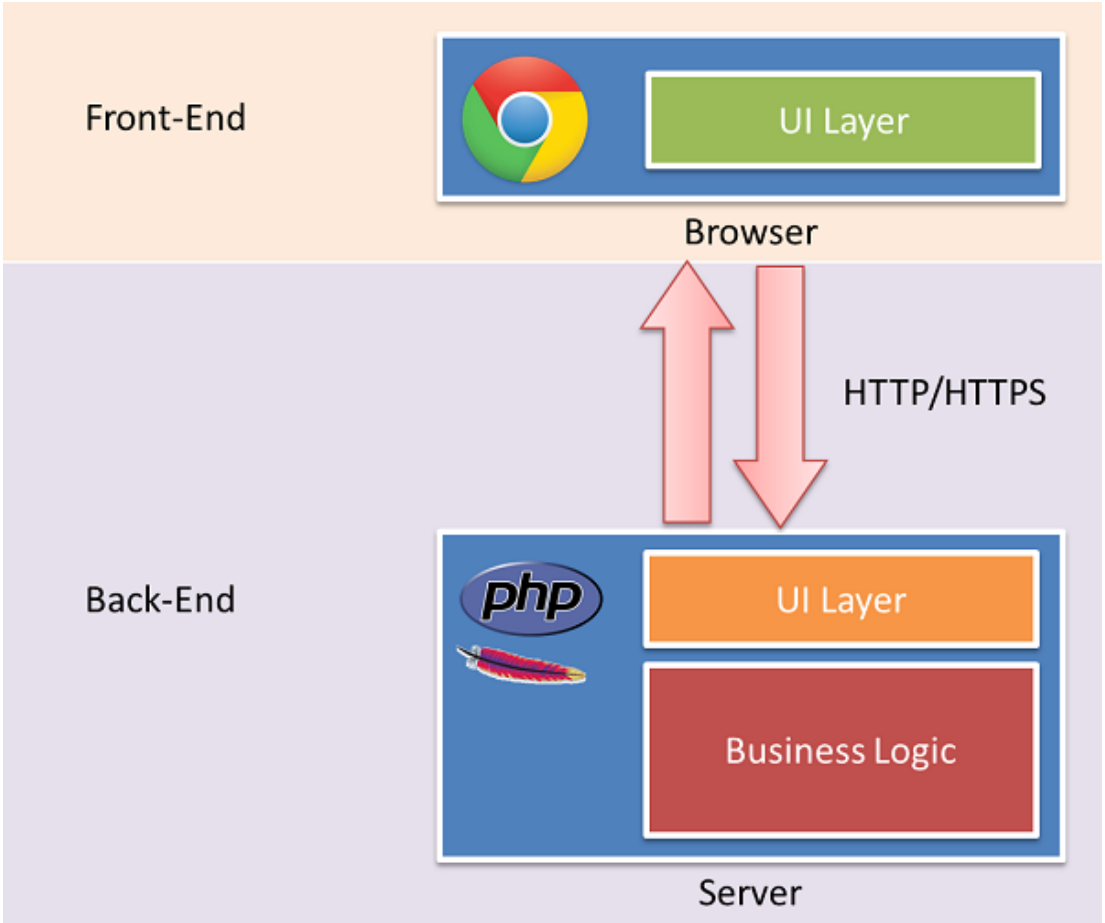
- niskopoziomowy dostęp do danych żądania
- konieczność dekodowania danych pochodzących z przeglądarki
- konieczność kodowania danych wysyłanych do przeglądarki (np. adresy URL)
- programy CGI wykonują się jako osobne procesy
- implementacja interfejsu CGI zależna jest od systemu operacyjnego
- brak oddzielenia treści od algorytmów przetwarzających

Plan prezentacji

JavaScript

- obiektowy język skryptowy
- działa w większości przeglądarek
- pozwala na programowanie po stronie klienta
- podstawowy problem – kompatybilność implementacji
- nie mylić z Javą!

Back-end vs. front-end



Rysunek : http://www.nczonline.net/blog/2013/10/07/node-js-and-the-new-web-front-end/

HTML + CSS + JS + back-end

HTML	struktura, semantyka, treść
CSS	wygląd, elementy interakcji
JavaScript	złożona interakcja, żądania asynchroniczne
back-end	generowanie treści, przechowywanie danych

Zadania JS

- kontrola nad przeglądarką
- asynchroniczna komunikacja
- modyfikowanie dokumentu HTML
- ograniczenie komunikacji z serwerem
- szybszy interfejs użytkownika
- Single Page Applications (SPA)

Kompatybilność

Najczęściej osiągana dzięki frameworkom, np. jQuery

- najbardziej popularna biblioteka JS
- ułatwia operacja takie jak:
  - wybieranie elementów HTML
  - tworzenie animacji
  - obsługę zdarzeń
  - tworzenie żądań AJAX
  - parsowanie JSONa

Inne frameworki

- AngularJS – dzieło Googla
- Ember.js – rozwijany w współpracy z Ruby on Rails
- Ext JS – zawiera wiele gotowych komponentów UI
- Google Web Toolkit – pozwala pisać kod w Javie
- SproutCore – protoplasa Ember.js

Plan prezentacji

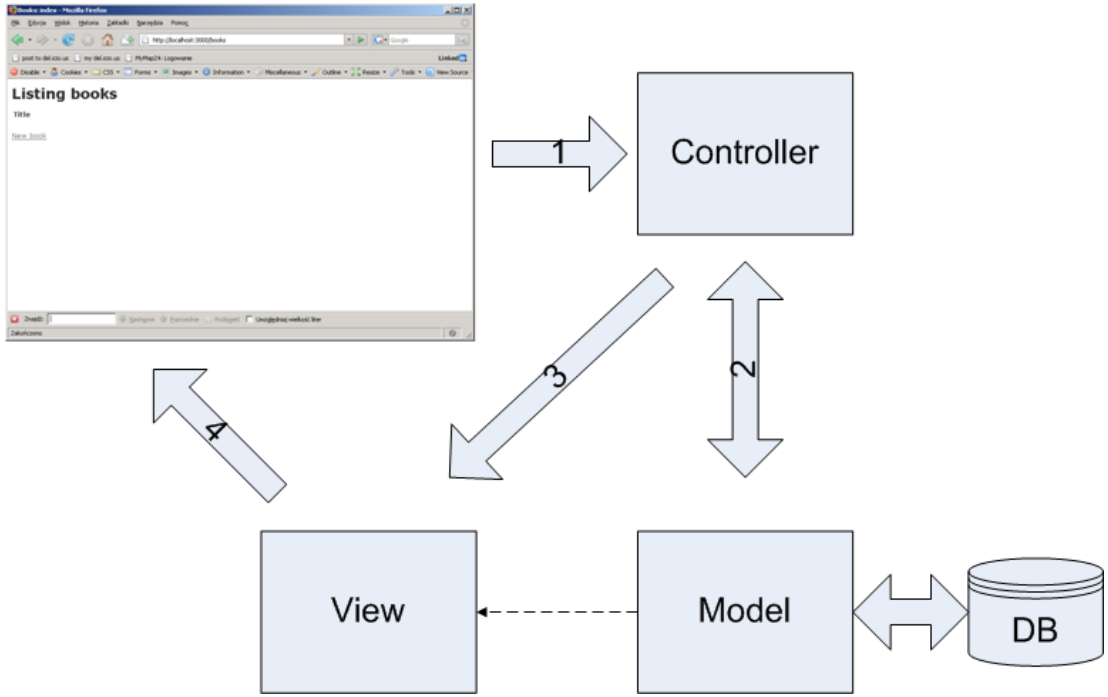
Przegląd języków programowania

- C/C++ – systemy operacyjne, systemy wbudowane, narzędzia systemowe
- Java – aplikacje korporacyjne i duża plikacje serwerowe
- C# – programy dla systemu Windows, aplikacje webowe
- Objective-C – programy dla systemu MacOS i iOS
- PHP – aplikacje webowe
- Ruby, Python, Perl – programowanie skryptowe, programowanie systemowe, aplikacje webowe
- JavaScript – programowanie po stronie klienta (w przeglądarce), ostatnio również po stronie serwera (node.js)
- ActionScript – programowanie po stronie klienta (Flash)

Popularne frameworki webowe

- PHP
  - CakePHP
  - CodeIgiter
  - Symfony
- Python
  - Django
  - Pylons
- Ruby
  - Ruby on Rails
  - Sinatra
- Java
  - Spring MVC
  - JSF
  - GWT

MVC – Model Widok Kontroler



Sinatra – formularz (1)

- app1.rb

```
require 'rubygems'
require 'sinatra'
require 'sinatra/reloader' if development?

get '/' do
  erb :index
end
```

- views/index.erb

```
Wprowadź swoje imię:
<form method="post">
  <input type="text" name="name"/>
</form>
```

Sinatra – formularz (1)

- app1.rb

```
require 'rubygems'
require 'sinatra'
require 'sinatra/reloader' if development?

get '/' do
  erb :index
end
```

- views/index.erb

```
Wprowadź swoje imię:
<form method="post">
  <input type="text" name="name"/>
</form>
```

Sinatra – formularz (2)

- app1.rb – cd.

```
post '/' do
  @message = "Witaj " + params[:name]
  erb :result
end
```

- views/result.erb

```
<%= @message %>
```

- views/layout.erb

```
<html>
  <body style="width: 900px;margin: auto">
    <h2>Aplikacja formularz</h2>
    <div>
      <%= yield %>
    </div>
  </body>
</html>
```

Sinatra – formularz (2)

- app1.rb – cd.

```
post '/' do
  @message = "Witaj " + params[:name]
  erb :result
end
```

- views/result.erb

```
<%= @message %>
```

- views/layout.erb

```
<html>
  <body style="width: 900px;margin: auto">
    <h2>Aplikacja formularz</h2>
    <div>
      <%= yield %>
    </div>
  </body>
</html>
```

Sinatra – formularz (2)

- app1.rb – cd.

```
post '/' do
  @message = "Witaj " + params[:name]
  erb :result
end
```

- views/result.erb

```
<%= @message %>
```

- views/layout.erb

```
<html>
  <body style="width: 900px;margin: auto">
    <h2>Aplikacja formularz</h2>
    <div>
      <%= yield %>
    </div>
  </body>
</html>
```

Plan prezentacji

URL

Komunikacja HTTP

Formularze

CGI

JavaScript

Frameworki webowe

REST

Ograniczenia języka HTML

- efekt końcowy zrozumiały dla ludzi, ale nie dla maszyn
- formularze są niepotrzebne, jeśli znamy format danych
- aplikacje na urządzenia mobilne nie integrują się z natywnym interfejsem
- nadmiarowość informacji, np. wielokrotne przesyłanie szablonu strony

Podział aplikacji na backend i frontend

- jeden backend wspólny dla wielu frontendów/klientów
  - aplikacja HTML-owa
  - natywna aplikacja na urządzeniu mobilnym
  - inna aplikacja korzystająca z API
- backend nie generuje żadnego HTML-a
- odpowiada jedynie na żądania HTTP
- na wyjściu generuje XML lub JSON (JavaScript Object Notation)
- frontend HTML-owy napisany w JavaScripcie
- komunikacja z backendem z wykorzystaniem AJAXa (Asynchronous JavaScript and XML)

REST – Representational State Transfer

- architektura zaprojektowana dla rozproszonych serwisów webowych, np.:
  - Amazon S3
  - CKAN
  - CouchDB
- wykorzystuje dobrze znane cechy protokołu HTTP
  - adresy URL
  - czasowniki: `GET`, `POST`, `PUT`, `DELETE`
  - formaty: `HTML`, `JSON`, `XML`
  - pozostałe: proxy, firewall, caching, mime, etc.
- centralne pojęcie: **zasób**

REST – Representational State Transfer

- architektura zaprojektowana dla rozproszonych serwisów webowych, np.:
  - Amazon S3
  - CKAN
  - CouchDB
- wykorzystuje dobrze znane cechy protokołu HTTP
  - adresy URL
  - czasowniki: `GET`, `POST`, `PUT`, `DELETE`
  - formaty: `HTML`, `JSON`, `XML`
  - pozostałe: proxy, firewall, caching, mime, etc.
- centralne pojęcie: **zasób**

REST – Representational State Transfer

- architektura zaprojektowana dla rozproszonych serwisów webowych, np.:
  - Amazon S3
  - CKAN
  - CouchDB
- wykorzystuje dobrze znane cechy protokołu HTTP
  - adresy URL
  - czasowniki: `GET`, `POST`, `PUT`, `DELETE`
  - formaty: `HTML`, `JSON`, `XML`
  - pozostałe: proxy, firewall, caching, mime, etc.
- centralne pojęcie: **zasób**

REST – Representational State Transfer

- architektura zaprojektowana dla rozproszonych serwisów webowych, np.:
  - Amazon S3
  - CKAN
  - CouchDB
- wykorzystuje dobrze znane cechy protokołu HTTP
  - adresy URL
  - czasowniki: `GET`, `POST`, `PUT`, `DELETE`
  - formaty: `HTML`, `JSON`, `XML`
  - pozostałe: proxy, firewall, caching, mime, etc.
- centralne pojęcie: **zasób**

REST – Representational State Transfer

- architektura zaprojektowana dla rozproszonych serwisów webowych, np.:
  - Amazon S3
  - CKAN
  - CouchDB
- wykorzystuje dobrze znane cechy protokołu HTTP
  - adresy URL
  - czasowniki: `GET`, `POST`, `PUT`, `DELETE`
  - formaty: `HTML`, `JSON`, `XML`
  - pozostałe: proxy, firewall, caching, mime, etc.
- centralne pojęcie: **zasób**

REST – Representational State Transfer

- architektura zaprojektowana dla rozproszonych serwisów webowych, np.:
  - Amazon S3
  - CKAN
  - CouchDB
- wykorzystuje dobrze znane cechy protokołu HTTP
  - adresy URL
  - czasowniki: `GET`, `POST`, `PUT`, `DELETE`
  - formaty: `HTML`, `JSON`, `XML`
  - pozostałe: proxy, firewall, caching, mime, etc.
- centralne pojęcie: **zasób**

REST – zasady działania

- identyfikowanie zasobów poprzez adres URL
  - `http://example.com/cars` – kolekcja zasobów
  - `http://example.com/cars/1` – pojedynczy zasób
- odczytywanie/modyfikowanie zasobu z wykorzystaniem dokumentów opisujących stan

```
<?xml version="1.0" encoding="UTF-8"?>
<book>
  <author-id type="integer">1</author-id>
  <id type="integer">1</id>
  <title>Alef</title>
</book>
```

- CRUD – czasowniki HTTP: `GET`, `POST`, `PUT`, `DELETE`
- linki w dokumencie identyfikują inne możliwe akcje do wykonania

REST – zasady działania

- identyfikowanie zasobów poprzez adres URL
  - `http://example.com/cars` – kolekcja zasobów
  - `http://example.com/cars/1` – pojedynczy zasób
- odczytywanie/modyfikowanie zasobu z wykorzystaniem dokumentów opisujących stan

```
<?xml version="1.0" encoding="UTF-8"?>
<book>
  <author-id type="integer">1</author-id>
  <id type="integer">1</id>
  <title>Alef</title>
</book>
```

- CRUD – czasowniki HTTP: `GET`, `POST`, `PUT`, `DELETE`
- linki w dokumencie identyfikują inne możliwe akcje do wykonania

REST – zasady działania

- identyfikowanie zasobów poprzez adres URL
  - `http://example.com/cars` – kolekcja zasobów
  - `http://example.com/cars/1` – pojedynczy zasób
- odczytywanie/modyfikowanie zasobu z wykorzystaniem dokumentów opisujących stan

```
<?xml version="1.0" encoding="UTF-8"?>
<book>
  <author-id type="integer">1</author-id>
  <id type="integer">1</id>
  <title>Alef</title>
</book>
```

- CRUD – czasowniki HTTP: `GET`, `POST`, `PUT`, `DELETE`
- linki w dokumencie identyfikują inne możliwe akcje do wykonania

REST – zasady działania

- identyfikowanie zasobów poprzez adres URL
  - `http://example.com/cars` – kolekcja zasobów
  - `http://example.com/cars/1` – pojedynczy zasób
- odczytywanie/modyfikowanie zasobu z wykorzystaniem dokumentów opisujących stan

```
<?xml version="1.0" encoding="UTF-8"?>
<book>
  <author-id type="integer">1</author-id>
  <id type="integer">1</id>
  <title>Alef</title>
</book>
```

- CRUD – czasowniki HTTP: `GET`, `POST`, `PUT`, `DELETE`
- linki w dokumencie identyfikują inne możliwe akcje do wykonania

REST – zasady działania

- identyfikowanie zasobów poprzez adres URL
  - `http://example.com/cars` – kolekcja zasobów
  - `http://example.com/cars/1` – pojedynczy zasób
- odczytywanie/modyfikowanie zasobu z wykorzystaniem dokumentów opisujących stan

```xml
<?xml version="1.0" encoding="UTF-8"?>
<book>
  <author-id type="integer">1</author-id>
  <id type="integer">1</id>
  <title>Alef</title>
</book>
```

- CRUD – czasowniki HTTP: `GET`, `POST`, `PUT`, `DELETE`
- linki w dokumencie identyfikują inne możliwe akcje do wykonania

REST – zasady działania

- identyfikowanie zasobów poprzez adres URL
  - `http://example.com/cars` – kolekcja zasobów
  - `http://example.com/cars/1` – pojedynczy zasób
- odczytywanie/modyfikowanie zasobu z wykorzystaniem dokumentów opisujących stan

```
<?xml version="1.0" encoding="UTF-8"?>
<book>
  <author-id type="integer">1</author-id>
  <id type="integer">1</id>
  <title>Alef</title>
</book>
```

- CRUD – czasowniki HTTP: `GET`, `POST`, `PUT`, `DELETE`
- linki w dokumencie identyfikują inne możliwe akcje do wykonania

REST – zalety

- wiele frameworków wspiera REST (np. Ruby on Rails)
- w 2011 roku ok. 75% serwisów udostępniało API w REST
- możliwość wykorzystania istniejącej infrastruktury dla protokołu HTTP (DNS, proxy, uwierzytelnianie, etc.)