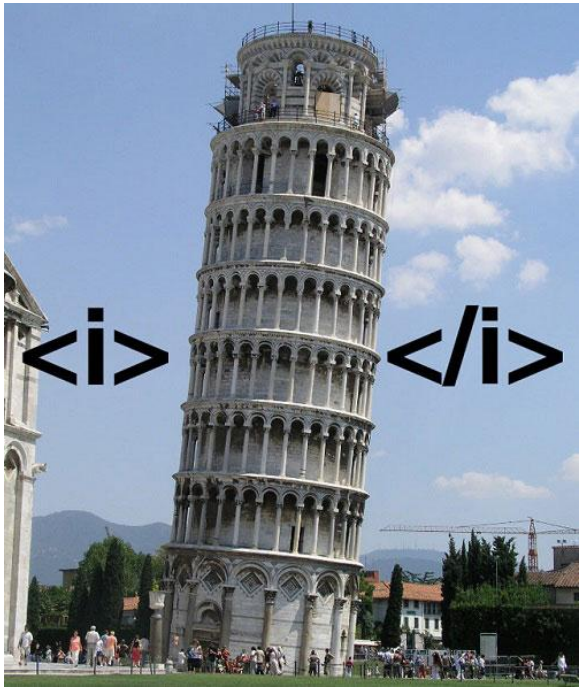
The Web as a Frontend to Database Services

www.w3schools.com
www.webdesign.com
…



History: The Internet and the Web

- 1945 linking microfiches , by Vannevar Bush
- 1960s Internet as (D)ARPA project:
  fault-tolerant, heterogeneous WAN (cold war!)
  term "Hypertext" coined by Ted Nelson at ACM 20th National Conference
- 1976 Queen Elizabeth sends her first email. She's the first state leader to do so.
- 1980 Berners-Lee at CERN writes notebook program to link arbitrary nodes
- 1989 Berners-Lee makes a proposal on information management at CERN
- 1990 Berners-Lee's boss approves purchase of a NeXT cube
  Berners-Lee begins hypertext GUI browser+editor and dubs it "WorldWideWeb"
  First web server developed
- 1991 May 17 – general release of WWW on central CERN machines
- 1992 more browsers: Viola & Erwise released
- 1994 > 200 web servers by start of year
  Mosaic: easy to install, great support, first inline images ("much sexier")
  Andreessen & colleagues form "Mosaic Comm. Corp"; later "Netscape"

Internet & Web: Basic Concepts

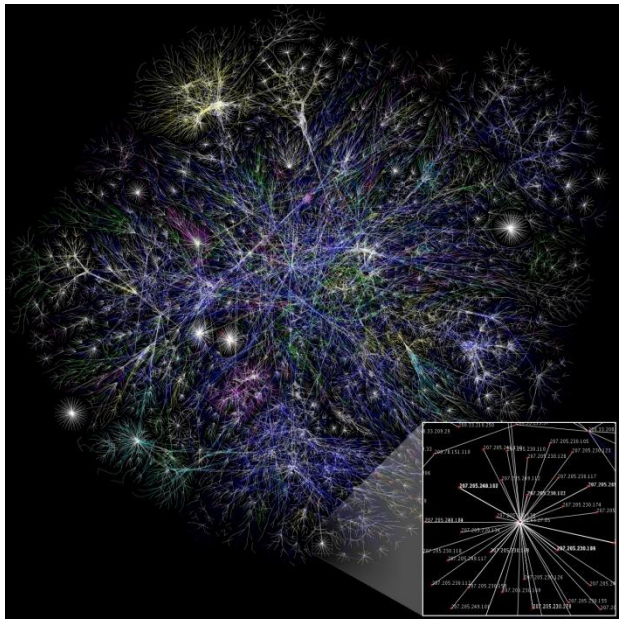
Internet & WWW

- Internet originally 4 basic services, based on TCP & IP:
  - telnet, ftp, mail, news
  - Later many more: IRC, SSL, NTP, ...
- Each computer has worldwide unique id
  - IP address: n.n.n.n (32 bit IPv4, 128 bit IPv6)
  - Domain name: subdomain.host.top-level-domain
  - DNS to resolve
- World-Wide Web just another Internet service
  - HTTP: Hypertext Transfer Protocol
  - HTML: Hypertext Markup Language
  - URIs (Uniform Resource Identifiers)

telnet, ftp, ..., http
(application layer)

TCP
(transport layer)

IP
(network layer)



[wikipedia]

Uniform Resource Identifiers

- Uniform naming schema to identify resources on the Internet
  - resource can be anything: index.html, mysong.mp3, picture.jpg
  - Syntax: scheme ":" [authority] [path] ["?" query]
  - Ex: http://www.cs.wisc.edu/index.html, mailto:webmaster@bookstore.com, telnet:127.0.0.1
- Structure of an http URI: http://www.cs.wisc.edu/~dbbook/index.html
  - Naming scheme (http)
  - Name of host computer + optionally port# (//www.cs.wisc.edu:80) – 80 is default
  - Name of resource (~dbbook/index.html)
- URL = Uniform Resource Locator (subset of URIs; old term)
  - Identification via network "location"

HTTP

Hypertext Transfer Protocol

- What is a communication protocol?
  - Set of rules that defines the structure of messages & communication process
  - Examples: TCP, IP, HTTP
- What happens if you click on www.cs.wisc.edu/~dbbook/index.html?
  - Client connects to server, transmits HTTP request to server
  - Server generates response, transmits to client
  - Both disconnect
- HTTP header describes content/action (text = ISO-8859-1), content for data
  - RFC 2616

HTTP Sample Request/Response

- Client sends:

```
GET ~dbbook/index.html HTTP/1.1
User-agent: Mozilla/4.0
Accept: text/*, image/gif, image/jpeg
```

- Server responds:

```
HTTP/1.1 200 OK
Date: Mon, 04 Mar 2002 12:00:00 GMT
Server: Apache/1.3.0 (Linux)
Last-Modified: Mon, 01 Mar 2002 09:23:24 GMT
Content-Length: 1024
Content-Type: text/html

<html> <head></head>
<body>
<h1>Burns and Nobble Internet Bookstore</h1>
Our inventory:
<h3>Science</h3>
<b>The Character of Physical Law</b>
...
</body></html>
```

Try this:
$ telnet google.com 80
GET / HTTP/1.1
<3x newline>

HTTP Request Structure

- Request line

  GET ~/index.html HTTP/1.1

  - Http method field (GET and POST, more later)
  - local resource field
  - HTTP version field
- Type of client

  User-agent: Mozilla/4.0
- What types of files (MIME types) the client will accept

  Accept: text/*, image/gif, image/jpeg

  - MIME = Multipurpose Internet Mail (!) Extensions = file type naming system
  - MIME types other than text/*, image/jpeg, image/gif, image/png need browser plug-in or helper application

HTTP Response Structure

- Status line — HTTP/1.1 200 OK
  - HTTP version: HTTP/1.1
  - Status code
  - Server message, textual

  •200 OK: Request succeeded
  •400 Bad Request: Request could not be fulfilled by the server
  •404 Not Found: Requested object does not exist on the server
  •505 HTTP Version not supported
- Date when the object was created — Last-Modified: Mon, 01 Mar 2002 09:23:24 GMT
- Number of bytes being sent — Content-Length: 1024
- What type is the object being sent — Content-Type: text/html
- *…plus potentially many more items, such as server type, server time, etc.*
- The payload! — <html>…</html>

HTTP Doesn't Remember!

- HTTP stateless on the granularity of requests
  - No “sessions”
  - Every message completely self-contained
  - No previous interaction “remembered” by protocol
- Implication for applications:
  Any state information (shopping carts, user login information, …) need to be encoded in every HTTP request *and* response!
  - More later!

Conventions

- index.html (Windows: index.htm), .php, ...
  - If local path ends with directory, this file is assumed
    - *Ex: http://www.myserver.foo/Downloads*
  - If not found: directory listing is displayed
    - *Put dummy index.html if you don't want this, or disable default in server*
- Local path *~name/path*
  - leads to ~*name*/public_html/*path* where *name* is local user name

HTML & Friends

HTML Primer

- HTML is a data exchange format
  - Unformatted ASCII
    - *Proper indentation increases readability*
  - Text interspersed with tags, some with attributes; usually start and end tag:

```
<h1 align="center">headline</h1>
```

  - Opening tags: “<” element name “>”
  - Closing tags: “</” element name “>”
  - Tags can be nested:

```
<h1><em>my</em> text</h1>
```

- Many editors automatically generate HTML directly from your document
  - But you need to know HTML too, want to generate it lateron!
  - And tool's code sometimes has bad quality, cf. Microsoft Word “Save as html”

HTML Primer (contd.)

```
<a name="top">

<h1>An important heading</h1>

<h2>A slightly less important heading</h2>

<p>This is the <em>first</em> paragraph.</p>

<img src="peter.jpg" width="150" height="200" alt="me">

My link list:
<ul>
  <li>This is a link to <a href="http://www.w3.org/">W3C</a>
  <li>This a link to <a href="peter.html">Peter's page</a>
  <li>Go to <a href="#top">top</a>
  <li><a href="/"><img src="logo.gif" alt="home page"></a>
</ul>
```

HTML Primer (contd.)

- Text structuring
  - Headlines
  - Paragraphs, text emphasis
- Links
  - External
  - Relative
  - Internal
- Images
- Text structuring (contd.)
  - Lists

```
<a name="top">

<h1>An important heading</h1>

<h2>A slightly less important heading</h2>

<p>This is the <em>first</em> paragraph.</p>

<img src="peter.jpg" width="150" height="200" alt="me">

My link list:
<ul>
 <li>This is a link to <a href="http://www.w3.org/">W3C</a>
 <li>This a link to <a href="peter.html">Peter's page</a>
 <li>Go to <a href="#top">top</a>
 <li><a href="/"><img src="logo.gif" alt="home page"></a>
</ul>
```

HTML Primer (contd.)

- Text structuring (contd.)
  - tables
  - row
  - column heading
  - regular column

Year	Sales
2000	$18M
2001	$25M
2002	$36M

```
<table>
  <tr>
    <th>Year</th>
    <th>Sales</th>
  </tr>
  <tr>
    <td>2000</td>
    <td>$18M</td>
  </tr>
  <tr>
    <td>2001</td>
    <td>$25M</td>
  </tr>
  <tr>
    <td>2002</td>
    <td>$36M</td>
  </tr>
</table>
```

HTML Forms

- Common way to communicate data from client to server
- General format of a form:

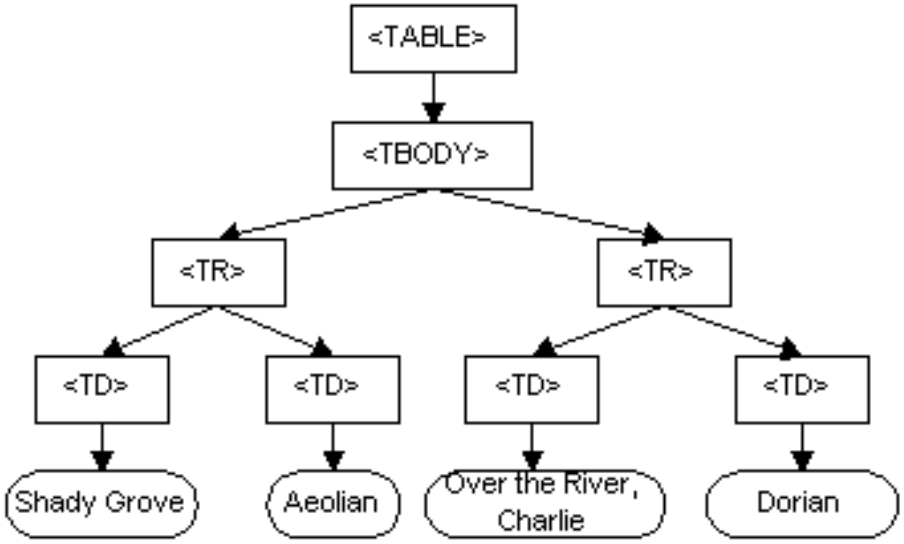
```
<form action="page.jsp" method="GET" name="loginForm">
  <input type=… value=… name=…>
</form>
```

- Components of an HTML form tag:
  - action: URI that handles the content
  - method: HTTP GET or POST
  - name: Name of the form; can be used in client-side scripts to refer to the form

HTML and DOM

```
<TABLE>
  <TBODY>
    <TR>
      <TD>Shady Grove</TD>
      <TD>Aeolian</TD>
    </TR>
    <TR>
      <TD>Over the River, Charlie</TD>
      <TD>Dorian</TD>
    </TR>
  </TBODY>
</TABLE>
```

Exercise:
draw DOM tree
for some HTML snippet



Document Object Model

- HTML document actually describes a tree structure
  - ...that becomes manifest as "real" tree only within browser
- So far: how can I describe such a tree for input into rendering engine?
- Dynamic HTML: *manipulate* tree representation while being displayed
- Document Object Model (DOM) =
  platform and language neutral interface that allows programs and scripts to dynamically access and update content & structure of HTML documents
  - Intro: http://www.w3schools.com/htmldom/default.asp
  - Definition: http://www.w3.org/TR/DOM-Level-2-HTML

CSS: Cascading Style Sheets

- Idea: Separate display style from structure & contents
  - W3C recommendation = standard
- File reference to CSS, placed in HTML <head> section
  - <link rel=“style sheet” type=“text/css” href=“books.css”>
- Media specific style sheets
  - <link rel="stylesheet" type=“text/css” media="screen" href="website.css">
    <link rel="stylesheet" type=“text/css” media="print, embossed" href="print.css">
    <link rel="stylesheet" type=“text/css” media="aural" href="speaker.css">

CSS Syntax

- CSS syntax (simplified)
  - css-file ::= css-def*
  - css-def ::= selector "{" (prop ":" val)* "}"
  - selector ::= tag
    | [tag] "." class
    | [tag] ":" pseudo
  - elem ::= STRING
  - class ::= STRING
  - pseudo ::= “link" | "visited" | …
  - prop ::= <predefined prop names>
  - val ::= STRING
    | NUMBER ["px" | "cm" | …]

```
body      { font-family:Arial,sans-serif; }
a:link    { color:red }
.special  { color:green; font-size:large; }
```

- Effect on HTML page display:
  - same effect as:
    <h1 style=”font-family:Arial,sans-serif”>
    but applies to all <h1>
  - Style used in a tag:
    <a href="…"> is red
    (overriding a default & a definition in CSS)
  - Style can be used with any tag:
    <p class=”special”>

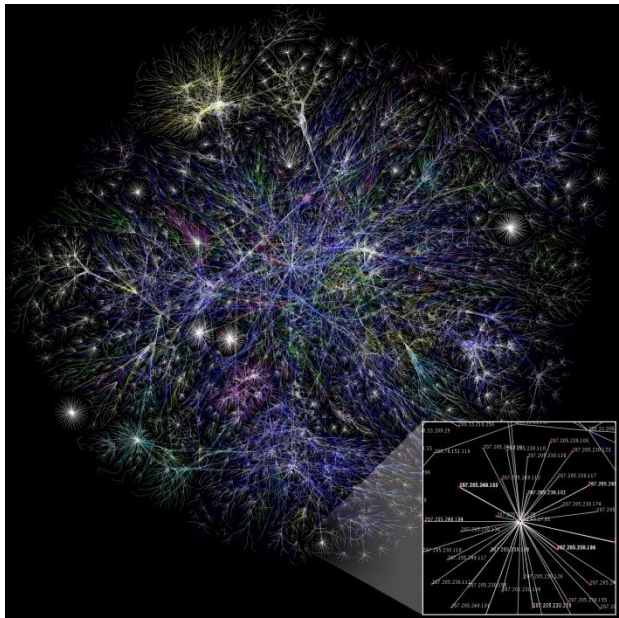
Internet & WWW

- Internet originally 4 basic services, based on TCP & IP:
  - telnet, ftp, mail, news
  - Later many more: IRC, SSL, NTP, ...
- Each computer has worldwide unique id
  - IP address: n.n.n.n (32 bit IPv4, 128 bit IPv6)
  - Domain name: subdomain.host.top-level-domain
  - DNS to resolve
- World-Wide Web just another Internet service
  - HTTP: Hypertext Transfer Protocol
  - HTML: Hypertext Markup Language
  - URIs (Uniform Resource Identifiers)

telnet, ftp, ..., http
(application layer)

TCP
(transport layer)

IP
(network layer)



[wikipedia]

Hypertext Transfer Protocol

- What is a communication protocol?
  - Set of rules that defines the structure of messages & communication process
  - Examples: TCP, IP, HTTP
- What happens if you click on www.cs.wisc.edu/~dbbook/index.html?
  - Client connects to server, transmits HTTP request to server
  - Server generates response, transmits to client
  - Both disconnect
- HTTP header describes content/action (text = ISO-8859-1), content for data
  - RFC 2616

HTTP Request Structure

- Request line

  GET ~/index.html HTTP/1.1

  - Http method field (GET and POST, more later)
  - local resource field
  - HTTP version field
- Type of client

  User-agent: Mozilla/4.0
- What types of files (MIME types) the client will accept

  Accept: text/*, image/gif, image/jpeg

  - MIME = Multipurpose Internet Mail (!) Extensions = file type naming system
  - MIME types other than text/*, image/jpeg, image/gif, image/png need browser plug-in or helper application

HTTP Response Structure

- Status line — HTTP/1.1 200 OK
  - HTTP version: HTTP/1.1
  - Status code
  - Server message, textual

  •200 OK: Request succeeded
  •400 Bad Request: Request could not be fulfilled by the server
  •404 Not Found: Requested object does not exist on the server
  •505 HTTP Version not supported
- Date when the object was created — Last-Modified: Mon, 01 Mar 2002 09:23:24 GMT
- Number of bytes being sent — Content-Length: 1024
- What type is the object being sent — Content-Type: text/html
- *…plus potentially many more items, such as server type, server time, etc.*
- The payload! — <html>…</html>

Conventions

- index.html (Windows: index.htm), .php, ...
  - If local path ends with directory, this file is assumed
    - *Ex: http://www.myserver.foo/Downloads*
  - If not found: directory listing is displayed
    - *Put dummy index.html if you don't want this, or disable default in server*
- Local path *~name/path*
  - leads to ~*name*/public_html/*path* where *name* is local user name

HTTP Sample Request/Response

- Client sends:

```
GET ~dbbook/index.html HTTP/1.1
User-agent: Mozilla/4.0
Accept: text/*, image/gif, image/jpeg
```

- Server responds:

```
HTTP/1.1 200 OK
Date: Mon, 04 Mar 2002 12:00:00 GMT
Server: Apache/1.3.0 (Linux)
Last-Modified: Mon, 01 Mar 2002 09:23:24 GMT
Content-Length: 1024
Content-Type: text/html

<html> <head></head>
<body>
<h1>Burns and Nobble Internet Bookstore</h1>
Our inventory:
<h3>Science</h3>
<b>The Character of Physical Law</b>
...
</body></html>
```

Try this:
$ telnet google.com 80
GET / HTTP/1.1
<3x newline>

HTML Primer

- HTML is a data exchange format
  - Unformatted ASCII
    - *Proper indentation increases readability*
  - Text interspersed with tags, some with attributes; usually start and end tag:

```
<h1 align="center">headline</h1>
```

  - Opening tags: “<” element name “>”
  - Closing tags: “</” element name “>”
  - Tags can be nested:

```
<h1><em>my</em> text</h1>
```

- Many editors automatically generate HTML directly from your document
  - But you need to know HTML too, want to generate it lateron!
  - And tool's code sometimes has bad quality, cf. Microsoft Word “Save as html”

HTML Primer (contd.)

- Text structuring
  - Title (for browser title bar)
  - Headlines
  - Paragraphs, text emphasis
- Links
  - External
  - Relative
  - Internal
- Images
  - use alt, width, height attributes!
- Text structuring (contd.)
  - Lists

```
<a name="top">
<title>My first HTML document</title>
<h1>An important heading</h1>
<h2>A slightly less important heading</h2>
<p>This is the <em>first</em> paragraph.</p>
<img src="peter.jpg" width="150" height="200" alt="me">
My link list:
<ul>
  <li>This is a link to <a href="http://www.w3.org/">W3C</a>
  <li>This a link to <a href="peter.html">Peter's page</a>
  <li>Go to <a href="#top">top</a>
  <li><a href="/"><img src="logo.gif" alt="home page"></a>
</ul>
```

HTML Primer (contd.)

- Text structuring (contd.)
  - tables
  - row
  - column heading
  - regular column

Year	Sales
2000	$18M
2001	$25M
2002	$36M

```
<table>
  <tr>
    <th>Year</th>
    <th>Sales</th>
  </tr>
  <tr>
    <td>2000</td>
    <td>$18M</td>
  </tr>
  <tr>
    <td>2001</td>
    <td>$25M</td>
  </tr>
  <tr>
    <td>2002</td>
    <td>$36M</td>
  </tr>
</table>
```

CSS: Cascading Style Sheets

- Idea: Separate display style from structure & contents
  - W3C recommendation = standard
- Define appearance of particular items
  - HTML element:
  - Self-defined:
  - Special:

```
body      { font-family: Arial,sans-serif; }
a:link    { color: red; }
.special  { color: green; font-size: large; }
```

```
<html>
<body>
    <h1>Title in Arial, but bold</h1>
    <div id="special">I am different</div>
    <a href="#somewhere">link in red</a>
</body>
</html>
```

- All HTML code of site references common CSS file → Corporate Design

```
<link rel="style sheet" type="text/css" href="books.css">
```

Summary: WWW and HTML

- WWW: another Internet service,
  aimed at easily traversing interconnected documents
- Protocol: HTTP, data exchange format: HTML
  - captures document structure according to fixed schema
- Browser = program that
  - gets page address; fetches HTML (+ likely additional files); renders page for display
- Separation of concerns:
  - HTML for structure and contents
  - CSS for layout
  - JavaScript for Dynamic HTML (see next: AJAX)

HTTP: GET, POST ...and the REST

GET Requests

- HTTP defines request types: GET, POST, PUT, DELETE, …
- Request modification through key/value pairs
  - ?
  - &
- Client sends:

http://acme.com/srv ? mybasket=6570616275 & article=656e44204456

Request Parameters: How Passed?

- GET parameters: URL text
  - Can be cached, bookmarked
  - Reload / back in history harmless
  - Data visible in URL

```
GET srv?k1=v1&k2=v2 HTTP/1.1
```

- POST parameters: HTTP message body
  - Not cached, bookmarked
  - Reload / back in history re-submits
  - Data not visible,
    not in history,
    not in server logs

```
POST srv HTTP/1.1
k1=v1&k2=v2
```

http://www.w3schools.com/tags/ref_httpmethods.asp

REST

[Thomas Roy Fielding, 2002]

- REST
  = Representational State Transfer
  - Resource + URI
    - *Web = one address space*
  - representation
  - Client requests follow xlink
    - → *new state*
- Not a standard nor product, but „architectural style“
  - = way to craft Web interface
- URI defines resource being requested
  - Consistent design philosophy
  - easy to follow
- Relies on four basic http operations:
  - GET – *Query*
  - POST – *Update*
  - PUT – *Add*
  - DELETE – *Delete*

Sample RESTful Application

- *Scenario: online shop*
- Fetch information: "shopping basket with id 5873"

GET /shoppingBasket/5873

  - Response:

```
<shoppingBasket xmlns:xlink="http://www.w3.org/1999/xlink">
  <customer xlink:href="http://shop.oio.de/customer/5873">5873</customer>
    <position nr="1" amount="5">
      <article xlink:href="http://shop.oio.de/article/4501" nr="4501">
        <description>lollypop</description>
      </article>
    </position>
  <position nr="2" amount="2">... </position>
</shoppingBasket>
```

  - Client can follow links, that changes its state
  - No side effect (status change) on server side

Sample RESTful Application (contd.)

- Place order:
  "add article #961 to shopping basket #5873"
  - Changes server state

```
POST /shoppingBasket/5873
articleNr=961
```

- Add article
  - Again, changes server state
  - Returns new id

```
PUT /article

<article>
   <description>Rooibush tea</description>
   <price>2.80</price>
   ...
</article>
```

```
HTTP/1.1 201 OK
...

http://shop.oio.de/article/6005
```

- Delete article
  - Server state change

```
DELETE /article/6005
```

Security

- REST: typed requests, firewall can judge → good for security

```
hermes.oio.de - - [26/Nov/2002:12:43:07 +0100] "GET /shoppingBasket/6 HTTP/1.1" 200
hermes.oio.de - - [26/Nov/2002:12:43:08 +0100] "GET /article/12 HTTP/1.1" 200
hermes.oio.de - - [26/Nov/2002:12:43:08 +0100] "GET /article/5 HTTP/1.1" 200
hermes.oio.de - - [26/Nov/2002:12:43:09 +0100] "POST /shoppingBasket/6 HTTP/1.1" 200
hermes.oio.de - - [26/Nov/2002:12:43:13 +0100] "POST /shoppingBasket/6 HTTP/1.1" 200
hermes.oio.de - - [26/Nov/2002:12:43:14 +0100] "GET /Order/3 HTTP/1.1" 200
```

- *→ admins much more inclined to open firewall for REST services than for eg SOAP*

REST: How Powerful?

- Local path uses historical directory syntax → strict hierarchy

  http://.../service-endpoint/MyShop/ShoppingBaskets/14731/Article/67236

  - Standard Web servers, proxies etc can cache
- What breaks hierarchies
  - Multi-dimensional indexing – *Lat/Long/height/time has no particular sequence*
  - SQL: joins – *join tables come in no particular sequence*
  - SQL: complex predicates – *.../filter1/filter2/filter3/... cannot express AND / OR / NOT*
  - SQL: nested queries
- Remedy: old-school KVP http://.../service-endpoint/MyShop?q=*select-from-where*
  - So much more powerful, but no caching etc.

REST: Appraisal

- Strengths
  - Simple paradigm; Web = RESTful resource
  - Caching (except POST)
  - Proven base stds: http, URI, MIME, XML/JSON
    - *Oops: cookies break REST paradigm*
- Weaknesses
  - Assumes addressability by path + identifier (URI!) = single-root hierarchies
    → only fraction of SQL power
  - Schema to represent all URIs is complex
  - response data structure definition outside REST
  - limited support for HTTP PUT & DELETE in popular development platforms
  - Power of http headers not accessible via browser URL

Summary

- Web services: client invokes function on server
  - Remote Procedure Call (RPC)
- Web World is evolving
  - New paradigms emerging (and some disappearing)
  - GET/KVP, POST/XML, SOAP, REST, JSON, OpenAPI, ...
- Service protocol independent from database query languages!
  - GET/KVP: http:/acme.com/access-point?q=select%20*%20from...
  - POST: http:/acme.com/access-point
q=select *from...
  - REST

Interaction:
HTML Forms, AJAX

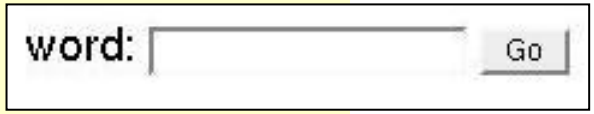
GET Requests

- Request = “command” sent by client to server = text string
  - Ex: http://acme.com/srv/index.html
- HTTP offers “commands” aka “request types”
  - GET obtain information
  - POST upload
  - PUT create new object
  - DELETE well…
  - Etc.

How to Pass Back Parameters from Client to Server?

- Client: HTML form

```
<form method='GET' action='http://.../input.php'>
   word:
   <input name='wordKey' type='text'>
   <input type='submit' value='Go'>
</form>
```



- Server: languages typically provides parameters in an array

```
<?
   echo 'You have entered ' . $_GET['wordKey'];
?>
```

Request Parameters: How Passed?

- Key/value pairs (KVPs) appended to service URL
  - URL: http://acme.com/srv ? mybasket=6570616275 & article=656e44204456
  - Server sees: all following "?", separator "&"
- GET: appended to URL

  GET srv?k1=v1&k2=v2 HTTP/1.1
  - Can be cached & bookmarked; reload / back in history ok
  - Data visible in URL
- POST: in HTTP message body

  POST srv HTTP/1.1

  k1=v1&k2=v2
  - Not cached, bookmarked; reload / back in history re-submits
  - Data not visible, not in history, not in server logs

http://www.w3schools.com/tags/ref_httpmethods.asp

We Want More!

- Challenge: want more interactivity than "click link / reload *complete* page“
  - Early attempt: HTML `iframe`
- Microsoft IE5 XMLHttpRequest object part of std DOM
  - Outlook Web Access, supplied with Exchange Server 2000
  - Windows: ActiveX control Msxml2.XMLHTTP (IE5), Microsoft.XMLHTTP (IE6)
- "AJAX" coined by Jesse James Garnett, 2005
  - made popular in 2005 by Google Suggest
  - start typing into Google's search box → list of suggestions

AJAX

- AJAX = Asynchronous Javascript and XML
  - web development technique
- Goal: increase interactivity, speed, functionality, usability
  - Avoid complete page reload → small data loads → more responsive
  - asynchronous: c/s communication independent from normal page loading
- Key idea: Client DOM manipulated to dynamically display & interact
  - Inject response into any place(s) of DOM tree
- standardized components, supported by all major browsers:
  - JavaScript, XML / JSON, HTML, CSS

AJAX by Example

Traditional Style

- Client:

```
<form method='GET' action='http://.../ajax-ex.php'>
   word:
   <input name='wordKey' type='text'>
   <input type='submit' value='Go'>
</form>
```

word: Go

- Server:

```
<?
   echo 'You have entered ' . $_GET['wordKey']
      . ' and your IP is: ' . $_SERVER['REMOTE_ADDR'];
?>
```

- Client, after page reload: You have entered Moribundus, and your IP is: 127.0.0.1

Step 1: Avoid Complete Page Reload

```
<form name='wordForm'>
   word:
   <input name='wordKey' type='text'>
   <input type='button' value='Go' onClick='JavaScript:callBack()'>
   <div id='result'></div>
</form>
```

```
function callBack()
{  var SERVICE = 'http://.../ajax-ex.php';
   var req = new XMLHttpRequest();
   var val = document.forms['wordForm'].wordKey.value;
   req.open( 'GET', SERVICE+'?wordKey='+val, true );
   req.setRequestHeader( 'Content-Type',
                         'application/x-www-form-urlencoded' );
   req.send( null );
   req.onreadystatechange = function()
   {  if (req.readyState == 4)
         document.forms['wordForm'].result.innerHtml =
            req.responseText;
   }
}
```

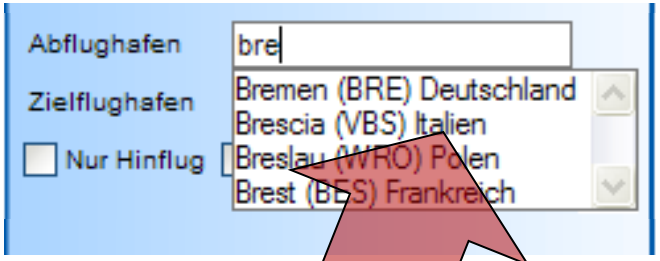
0	request not initialized
1	request set up
2	request sent
3	request in process
4	request complete

word: _____

You have entered Moribundus, and your IP is: 127.0.0.1

Step 2: Avoid SUBMIT Button

- Before: just re-implemented submit; now: allow c/s activity at any time
  - Event handlers
- Ex: suggest keywords with every char typed
  - No submit button!




```
<input name='wordKey' onKeyUp='JavaScript:callBack()'>
```

```
<? ...
   $query = "select entry from Airports
             where entry like '" . $_GET['wordKey'] . "%'";
   $result = mysql_query( $query );
   while ($row = mysql_fetch_array( $result ) )
   {
       print $row[ 'entry' ] . ",";
   }
?>
```

How to ship back & inject data?

JSON

- JSON = JavaScript Object Notation
  - Lightweight data interchange format
  - MIME type: `application/json` (RFC 4627)
  - text-based, human-readable
- alternative to XML use
  - Subset of JavaScript's object literal notation
  - 10x faster than XML parsing
  - _way_ easier to handle
  - JSON parsing / generating code readily available for many languages

"JSON is XML without garbage"

JSON Example

- Server sends:

```
req.onreadystatechange=function()
{   if(req.readyState==4)
    {   var p = eval( "(" + req.responseText + ")" );
        document.myForm.firstName.value = p.firstName;
    }
}
```

- JSON string sent from server:

```
{    "firstName": "John",
     "lastName": "Smith",
     "address":
     {    "streetAddress": "21 2nd Street",
          "city": "New York",
          "state": "NY",
          "postalCode": 10021
     },
     "phoneNumbers": [ "212 732-1234", "646 123-4567" ]
 }
```

- response parsing code:

```
<?  echo '{' + '"firstName":' + obj.firstName + ','
            + '"lastName":'  + obj.lastName  + ','
     … + '}'
?>
```

JSON Security Concerns

- JavaScript `eval()`
  - most JSON-formatted text is also syntactically legal JavaScript code!
  - built-in JavaScript `eval()` function executes code received
- Invitation to hack:
  embed rogue JavaScript code (server-side attack),
  intercept JSON data evaluation (client-side attack)
  - Safe alternative: `parseJSON()` method,
    see ECMAScript v4 and www.json.org/json.js
- Cross-site request forgery
  - malicious page can request & obtain JSON data belonging to another site

Appraisal: AJAX Advantages

- Reduced bandwidth usage
  - No complete reload/redraw, HTML generated locally, only actual data transferred → payload coming down much smaller in size
  - Can load stubs of event handlers, then functions on the fly
- Separation of data, format, style, and function
  - encourages programmers to clearly separate methods & formats:

Raw data / content	→ normally embedded in XML
webpage	→ HTML / XHTML
web page style elements	→ CSS
Functionality	→ JavaScript + XMLHttp + server code

Appraisal: AJAX Disadvantages

- **Browser integration**
  - dynamically created page not registered in browser history
  - bookmarks
- **Search engine optimization**
  - Indexing of Ajax page contents?
  - (not specific to Ajax, same issue with all dynamic data sites)
- **Web analytics**
  - Tracking of accessing page vs portion of page vs click?
- **Response time concerns from network latency**
  - Web transfer hidden → effects from delays sometimes difficult to understand for users
- **Reliance on JavaScript**
  - JavaScript compatibility issue → blows up code; Remedy: libraries such as `prototype`
  - IDE support used to be poor, changing
  - Can switch off JavaScript in my browser
- **Security**
  - Can fiddle with data getting into browser

Summary

- AJAX allows to add desktop flavour to web apps
  - JSON as lightweight, fast alternative to XML
- Web programming paradigm based on existing, available standards
- Issues: browser compatibility, security, web dynamics
- Many usages:
  - real-time form data validation; autocompletion; bg load on demand; sophisticated user interface controls and effects (trees, menus, data tables, rich text editors, calendars, progress bars, ...); partial submit; mashups (app mixing); desktop-like web app

Resources

- Books:
  - Michael Mahemoff: Ajax Design Patterns. O'Reilly, 2006
  - Mark Pruett: Ajax and Web Services. O'Reilly, 2006
- Web:
  - www.openajaxalliance.org/
  - w3schools.org/ajax
  - Mozilla Developer Center: AJAX:Getting Started
    - *developer.mozilla.org/en/docs/AJAX:Getting_Started*
  - www.json.org

Tool Support: Examples

- jQuery, http://jquery.com/

```
$( "button.continue" ).html( "Next Step..." )
```

- AJAX:

```
$.ajax({
  url: "/api/getWeather",
  data: {
    zipcode: 97201
  },
  success: function( data ) {
    $( "#weather-temp" ).html( "<b>" + data + "</b> degrees" );
  }
});
```

Kore rawa e rawaka te reo kotahi