

Unit 2: Web Technologies

Core Technologies. Server-Side Technologies. Restful Web Services

Unit 2: Web Technologies

- ▶ Core techs
 - ◆ URIs
 - ◆ HTTP
- ▶ Server-Side techs
 - ◆ Java Servlets
- ▶ Restful Web Services

World Wide Web

*“The **World Wide Web** (known as "WWW", "Web" or "W3") is the universe of network-accessible information, the embodiment of human knowledge”*

W3C <http://www.w3.org/WWW/>

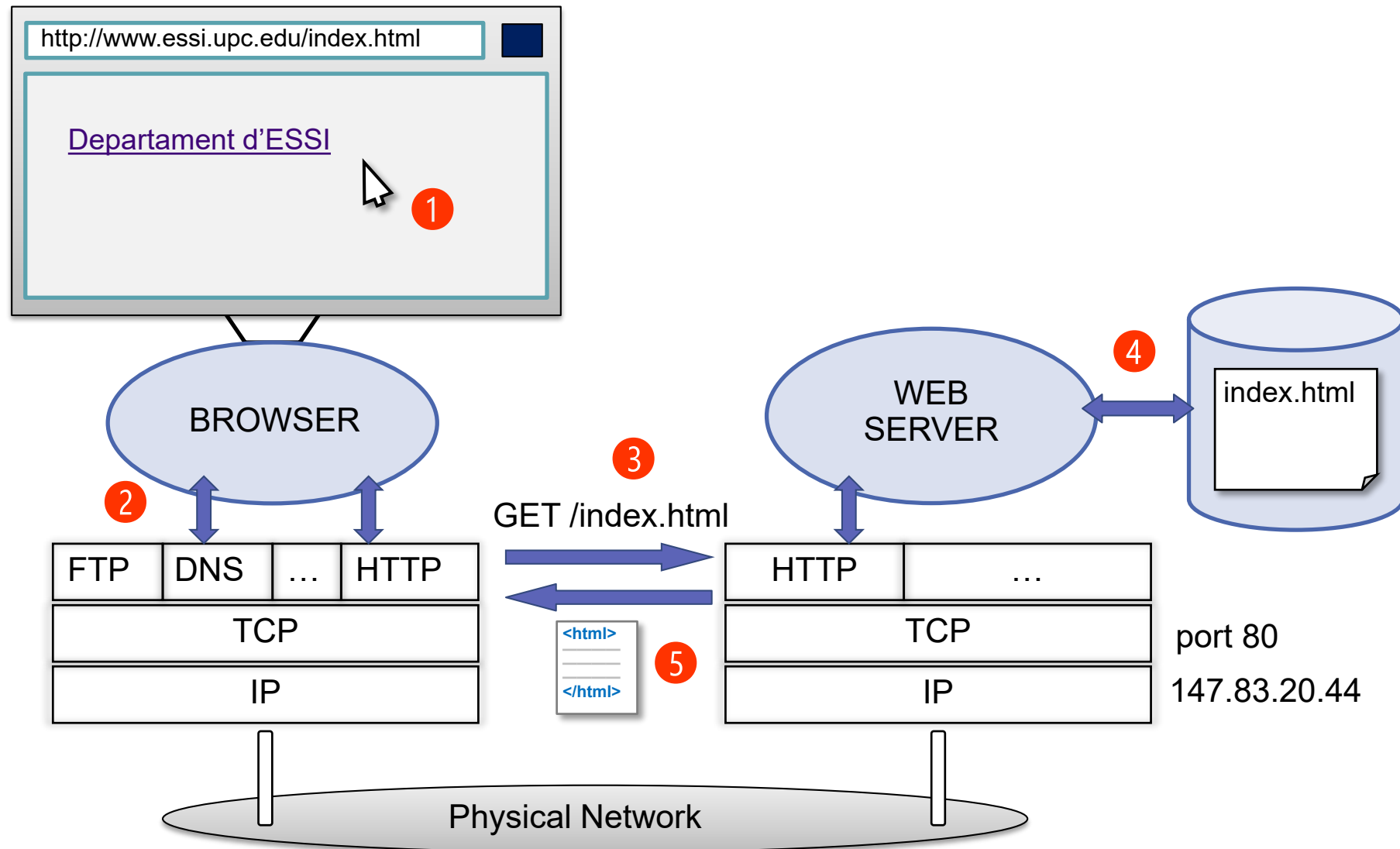
“Legend has it that every new technology is first used for something related to sex or pornography. That seems to be the way of humankind”

Tim Berners-Lee <http://www.theguardian.com/technology/2005/aug/12/uknews.onlinesupplement>

World Wide Web

- ▶ The Web was created in 1989 by **Tim Berners-Lee** while working at the European Organization for Nuclear Research (CERN) in Geneva.
- ▶ The three initial basic components:
 - ◆ **HTML**: Markup language for formatting hypertext documents
 - ◆ **URI**: Uniform notation scheme for addressing accessible resources over the network
 - ◆ **HTTP**: Protocol for transporting messages over the network

World Wide Web: Typical Interaction



Uniform Resource Identifier (URI)

- ▶ “The Web is an information space. Human beings have a lot of mental machinery for manipulating, imagining, and finding their way in spaces. URIs are the points in that space” (W3C)
- ▶ A URI is a compact sequence of characters that identifies an abstract or physical resource. Its generic syntax defines four parts:

<scheme name> : <hierarchical part> [? <query>] [# <fragment>]

- ▶ Example:

`http://user:pass@example.com:992/animal/bird?species=seagull#wings`

The diagram illustrates the components of the URI `http://user:pass@example.com:992/animal/bird?species=seagull#wings` using curly braces and labels below the string:

- scheme**: `http` (blue text)
- authority**: `//user:pass@example.com:992` (red text)
 - login**: `user:pass` (red text)
 - host**: `example.com` (red text)
 - port**: `:992` (red text)
- hierarchical part**: `/animal/bird` (red text)
- path**: `/animal/bird` (red text)
- query**: `?species=seagull` (green text)
- fragment**: `#wings` (red text)

URLs and URNs

- ▶ A "Uniform Resource Locator" (URL) is a URI that provides also a mean for locating the resource by describing its primary access mechanism. Examples:

<ftp://ftp.is.co.za/rfc/rfc1808.txt>

<mailto:John.Doe@example.com>

- ▶ A "Uniform Resource Name" (URN) refers to a URI under the "urn" scheme (e.g [urn:isbn:1892295490](#)) or to any other URI with the properties of a name

HyperText Transfer Protocol (HTTP)

- ▶ Request/response communication model
 - ◆ HTTP Request
 - ◆ HTTP Response
- ▶ Typical interaction (HTTP/1.0 or lower):
 - ◆ Web browser establishes a TCP/IP connection with the target Web server
 - ◆ Web browser sends a HTTP ASCII request over the TCP/IP connection.
 - ◆ Web server returns a HTTP MIME-like response and releases the TCP/IP connection
- ▶ HTTP/1.1 improvements:
 - ◆ Persistent connections: several request/response interactions in a single connection.
 - ◆ Request pipelining: multiple HTTP requests are sent without waiting for the corresponding responses.

HTTP Message Format

REQUEST_start_line | RESPONSE_start_line

← start line

<Field-name-1>: <Value>
<Field-name-2>: <Value>
...

← header fields

← blank line (CRLF)

<lorem> ipsum dolor sit amet, consectetur adipiscing
elit. Duis et ante nulla, ac vulputate metus. Nunc
fringilla dignissim mauris quis placerat</lorem>

← body (optional)

Request and response messages have the same generic format

HTTP Request Message

REQUEST_start_line := METHOD <Resource_path> HTTP_Version

METHOD := GET | POST | HEAD | ...

HTTP_Version := HTTP/1.0 | HTTP/1.1 | ...

Example:

```
GET /index.html HTTP/1.1
```

```
Host: www.essi.upc.edu
```

```
Accept: text/html
```

HTTP Request Methods

GET	get a resource from the server
HEAD	get the header only (no response body)
POST	send data (usually in the request body) to the server
PUT	store request body on server
PATCH	apply partial modifications to a resource
TRACE	get the "final" request as it is received by the server (after it has potentially been modified by proxies)
OPTIONS	get a list of HTTP methods supported by the server
DELETE	delete a resource on the server

HTTP Header Fields in Req & Res

Field name	Description	Req/Res	Example
Accept	Content-Types that are acceptable	Req	Accept: text/plain
Authorization	Authentication credentials for HTTP authentication	Req	Authorization: Basic QWxhZGRpbjpvGVuIHNIc2FtZQ==
Cache-Control	Tells all caching mechanisms from server to client whether they may cache this object. Measured in seconds	Res	Cache-Control: max-age=3600
Cookie	an HTTP cookie previously sent by the server with Set-Cookie	Req	Cookie: UserID=JohnDoe;...
Content-Length	The length of the request body in octets (8-bit bytes)	Req/Res	Content-Length: 348
Content-Type	The mime type of this content	Req/Res	Content-Type: text/html; charset=utf-8
ETag	An identifier for a specific version of a resource, often a message digest	Res	ETag: "737060cd8c284af7ad3082f209582d"
Set-Cookie	an HTTP cookie	Res	Set-Cookie: UserID=JohnDoe;...

https://en.wikipedia.org/wiki/List_of_HTTP_header_fields

HTTP Response Message

Response_start_line := HTTP_Version Status_Code [explanation]

Status_Code := 100 ... 599

Example:

```
HTTP/1.1 200 OK
```

```
Date: Fri, 19 Feb 2010 16:48:36 GMT
```

```
Server: Apache/2.2.14 (Unix)
```

```
Last-Modified: Tue, 02 Feb 2010 14:36:59 GMT
```

```
Content-Length: 17008
```

```
Content-Type: text/html
```

```
<html>
```

```
...
```

```
</html>
```

HTTP Standard Status codes

100 Continue
200 OK
201 Created
202 Accepted
203 Non-Authoritative
204 No Content
205 Reset Content
206 Partial Content
300 Multiple Choices
301 Moved Permanently
302 Found
303 See other
304 Not Modified
305 Use Proxy
307 Temporary Redirect

4xx Client's fault

400 Bad Request
401 Unauthorized
402 Payment Required
403 Forbidden
404 Not Found
405 Method Not Allowed
406 Not Acceptable
407 Proxy Authentication Required
408 Request Timeout
409 Conflict
410 Gone
411 Length Required
412 Precondition Failed
413 Request Entity Too Large
414 Request-URI Too Long
415 Unsupported Media Type
416 Requested Range Not Satisfiable
417 Expectation Failed

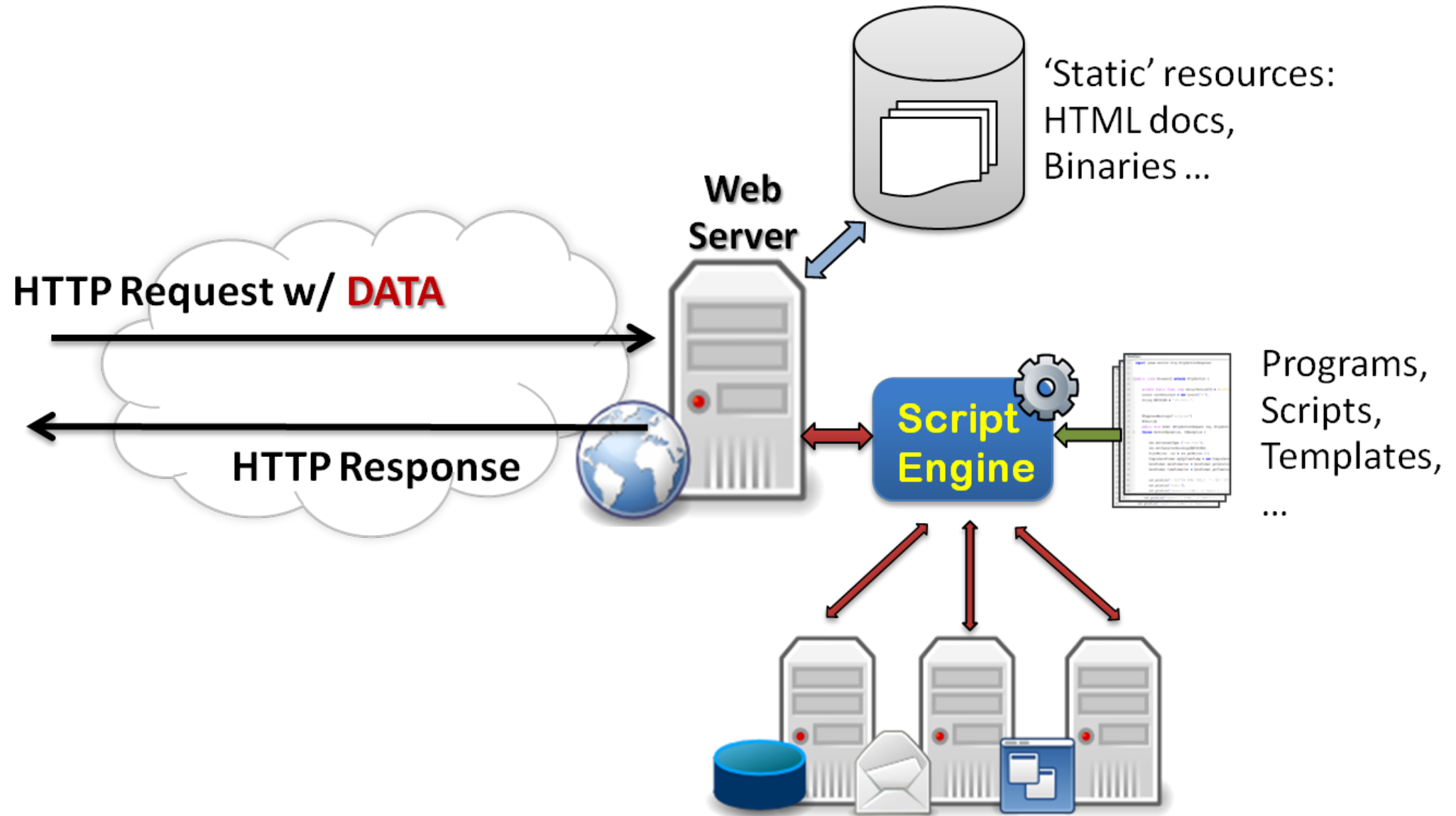
500 Internal Server Error
501 Not Implemented
502 Bad Gateway
503 Service Unavailable
504 Gateway Timeout
505 HTTP Version Not Supported

5xx Server's fault

Cookies

- ▶ A cookie is a piece of data that a Web server ask a Web browser to hold on to, and to return when the browser makes a subsequent HTTP request to that server.
- ▶ Initially, a cookie is sent from a server to a browser by adding a line to the HTTP headers:
 - ◆ Set-Cookie: WoTCookie32=2a2879cb3837f4d767615612a4f2355ede564793cf2042f561f33b5d227e2436; Expires=Thu, 23-Sep-2021 09:52:30 GMT
- ▶ A cookie can have up to six parameters:
 - ◆ Name (required)
 - ◆ Value (required)
 - ◆ Expiration date
 - ◆ Path
 - ◆ Domain
 - ◆ Requires a secure connection

Server-side computing



Passing data through HTTP requests

- ▶ Query string pairs key=value

- ❖ GET /directory.php?userid=3439

- ▶ Path attributes:

- ❖ DELETE /users/3439

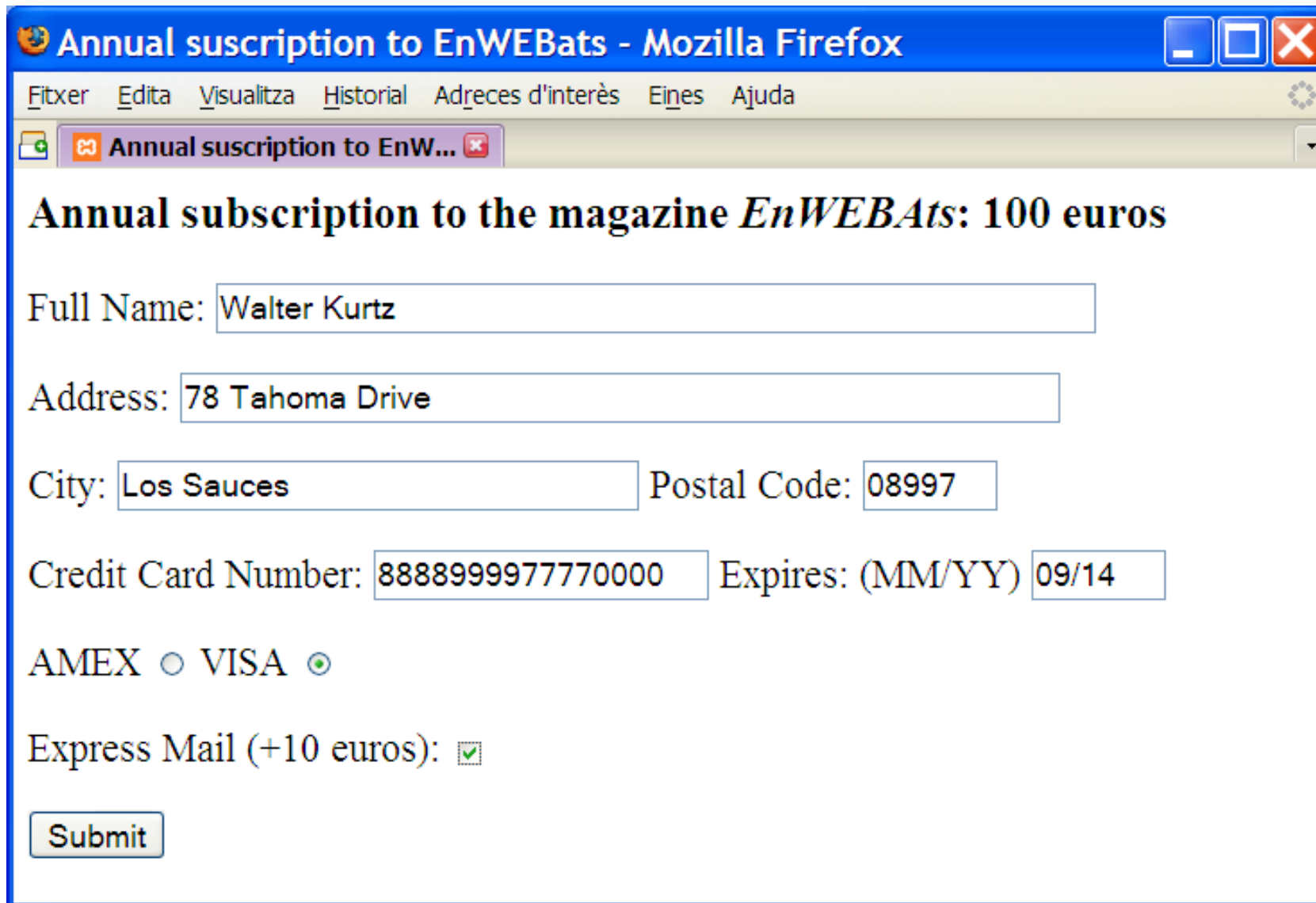
- ▶ HTTP Request Body (POST & PUT)

```
POST /asw01ss/wot HTTP/1.1
Content-type:application/x-www-form-urlencoded
author=Menganito&tweet_text=Text+de+prova
```

- ▶ HTTP Headers: Standard and Custom

- ▶ Cookies

“EnWEBats” Example: The Web Form



The screenshot shows a Mozilla Firefox browser window titled "Annual suscription to EnWEBats - Mozilla Firefox". The address bar shows a single tab titled "Annual suscription to EnW...". The main content area displays the following form:

Annual subscription to the magazine *EnWEBats*: 100 euros

Full Name:

Address:

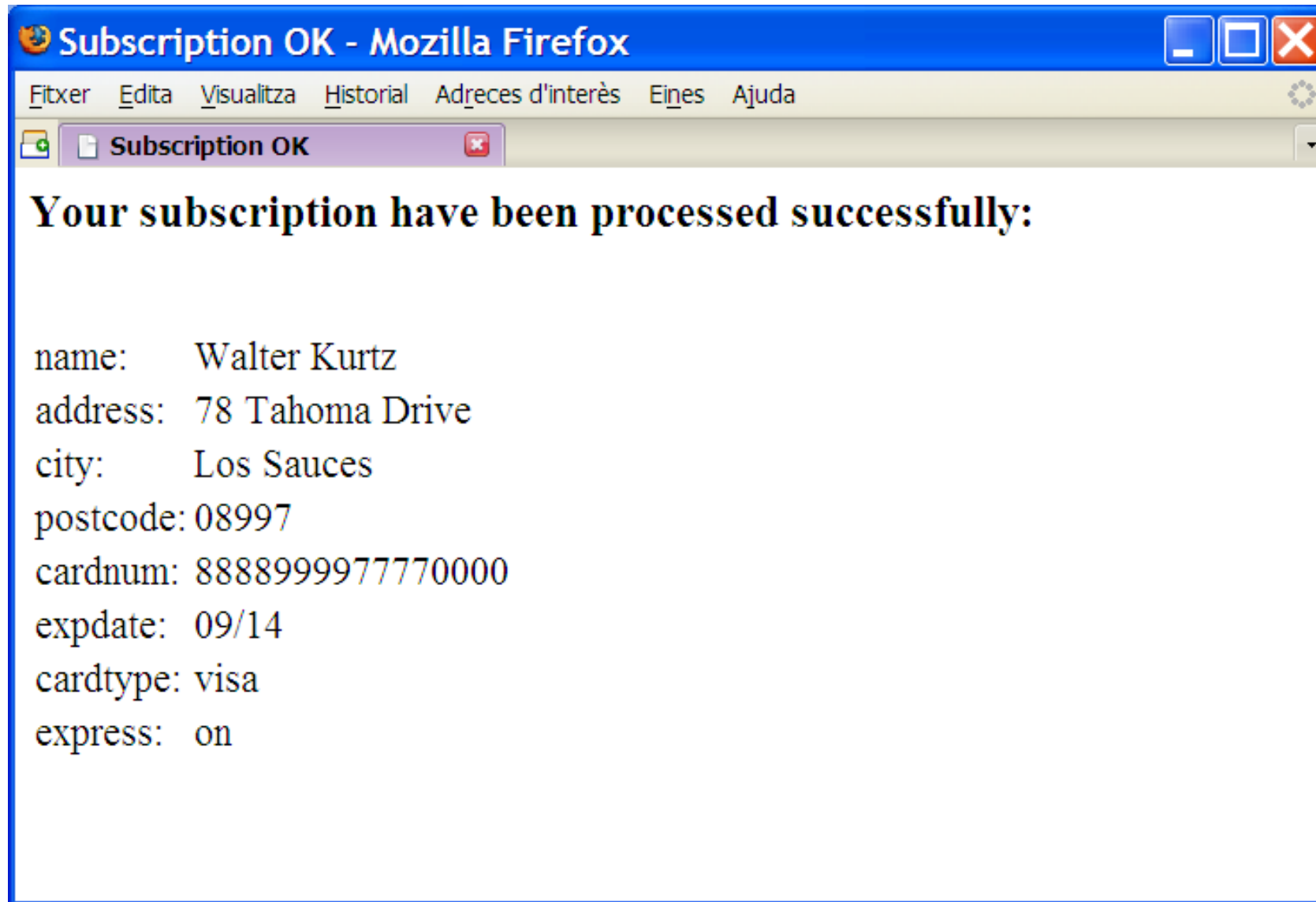
City: Postal Code:

Credit Card Number: Expires: (MM/YY)

AMEX ☐ VISA ☒

Express Mail (+10 euros): ☒

“EnWEBats” Example: The Output



“EnWEBats” Example: enwebats.html

```
<html>
....
<form action="subscribe" method="post">
Full Name: <input name="name" size="57">
Address: <input name="address" size="57">
City: <input name="city" size="32">
Postal Code: <input name="postcode" size="5">
Credit Card Number: <input name="cardnum" size="19">
Expires: (MM/YY) <input name="expdate" size="5">
AMEX <input name="cardtype" value="amex" type="radio"> VISA
    <input name="cardtype" value="visa" type="radio">
Express Mail (+10 euros): <input name="express" type="checkbox">
<input value="Submit" type="submit"> </p> </form>
....
</html>
```

Exemple “EnWEBats”: HTTP Request

```
POST /subscribe HTTP/1.1
```

```
Host: www.enwebats.com
```

```
Content-type: application/x-www-form-urlencoded
```

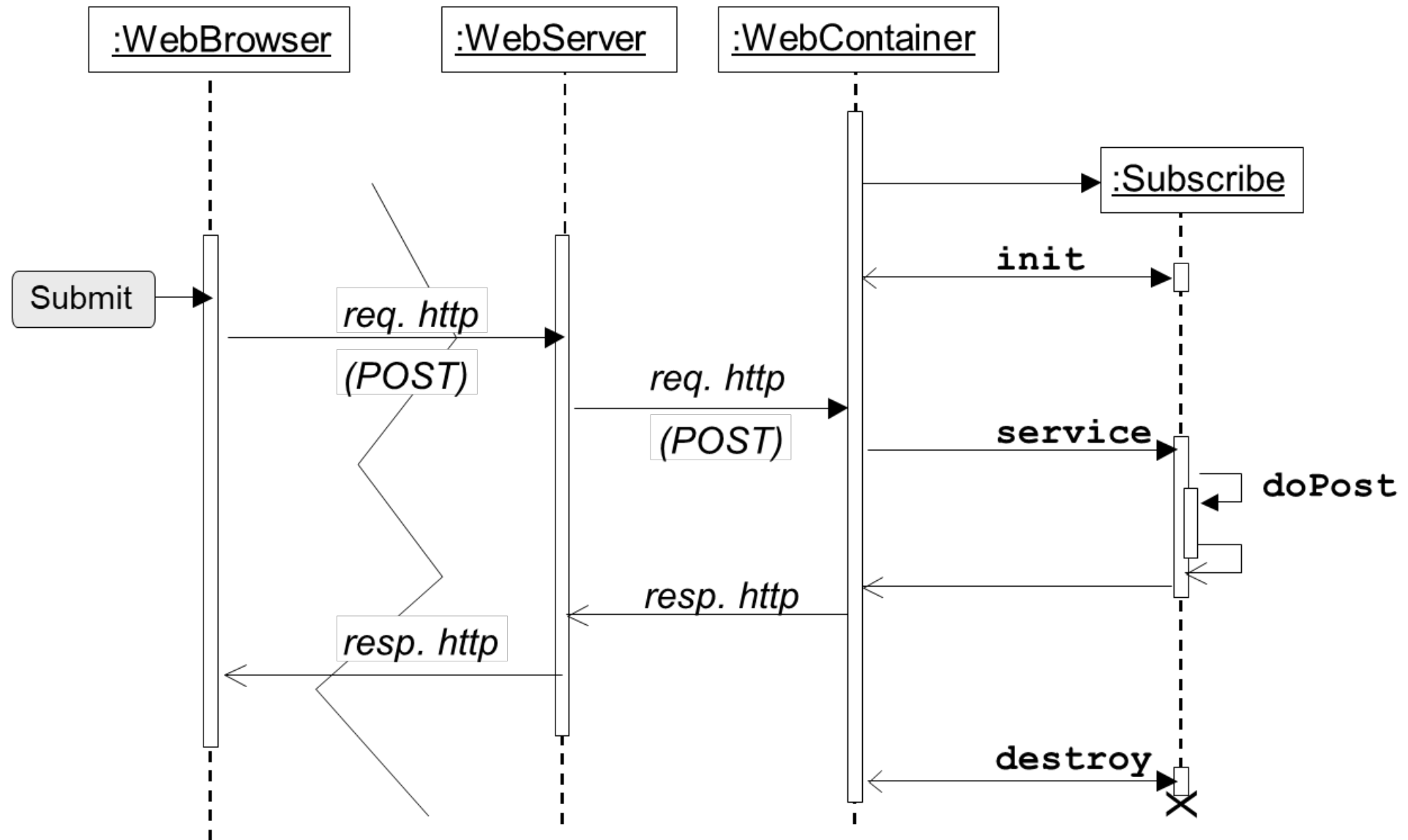
```
name=Walter+Kurtz&address=78+Tahoma+Drive&  
city=Los+Sauces&postcode=08997&cardnum=88889  
99977770000&expdate=09%2F14&cardtype=visa&express=on
```

Exemple “EnWEBats”: Subscribe.java

@WebServlet("/subscribe")

```
public class Subscribe extends HttpServlet {  
    public void doPost(javax.servlet.http.HttpServletRequest req,  
        javax.servlet.http.HttpServletResponse res) throws ServletException, IOException  
{ res.setContentType("text/html");  
    PrintWriter output = res.getWriter();  
    output.println("<html>");  
    output.println("<title>Subscription OK</title>");  
    output.println("<body bgcolor=white>");  
    output.println("<h3>Your subscription have been processed successfully:</h3>");  
    output.println("<table border=0>");  
    Enumeration paramNames = req.getParameterNames();  
    while (paramNames.hasMoreElements()) {  
        String name = (String) paramNames.nextElement();  
        String value = req.getParameter(name);  
        output.println("<tr><td>"+name+" : </td><td>"+value+"</td></tr>"); }  
    output.println("</table></body></html>");  
    output.close(); } }
```

Java Servlets: Request Scenario



Java Servlets

- ▶ A **Servlet** is an object that receives a request and generates a response based on that request.
- ▶ A **Web container** is essentially the component of a Web server that interacts with the servlets. The Web container is responsible for
 - ◆ Managing the lifecycle of servlets.
 - ◆ Mapping a URI to a particular servlet.
 - ◆ Ensuring that the URI requester has the correct access rights.

Exemple “EnWEBats”: subscribe.php

```
<?php function print_row($item, $key) {  
    echo "<tr><td>$key: </td><td>$item</td></tr>\n"; }?>  
<html>  
<head>  
<title>Subscription OK</title>  
</head>  
<body bgcolor=white>  
<h3>Your subscription have been processed successfully:</h3>  
<BR>  
<BR>  
<table border=0 width=50%>  
<?php array_walk($_POST, 'print_row'); ?>  
</table>  
</body>  
</html>
```

HTTP: Two Important Remarks

▶ HTTP is **stateless**:

- ◆ The Web Server handles each HTTP request independently and there is no easy way to keep track of each client request and to associate it with a previous one.
- ◆ However, managing state is important for many applications: a single use case scenario often involves navigating through a number of Web pages. Without a state management mechanism, you would have to continually supply all previous information entered for each new Web page.

▶ HTTP is one-way:

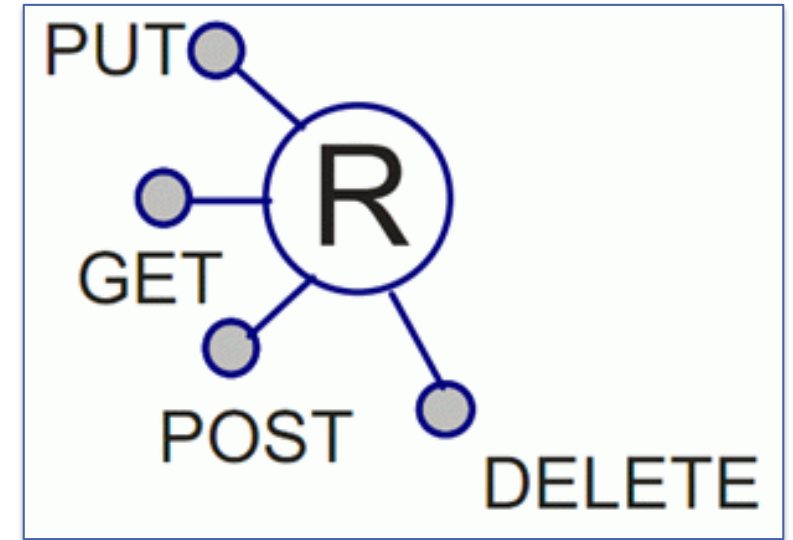
- ◆ Clearly separated roles: Clients -Web browsers- make requests to -Web-Servers, no vice versa.
- ◆ HTTP 2.0 now supports push notifications.

Session State Management in Web Apps

Solution	Implementation	Benefits	Drawbacks
On the Client	<ul style="list-style-type: none">• Hidden Form Fields• HTTP Cookies• URI Rewriting• Browser's localStorage & sessionStorage	No problems with load-balanced server clusters	Security concerns if data not encrypted
On the Web container	HttpSession (Java), \$_SESSION (PHP), and the like	Easy-to-use APIs	Load-balanced server clusters require special treatments
On a DB	Stored in a DB table	<ul style="list-style-type: none">• Sharable• Recoverable	May penalize DB performance

RESTful Web Services


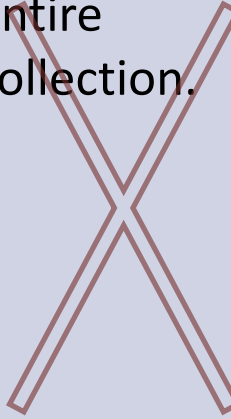
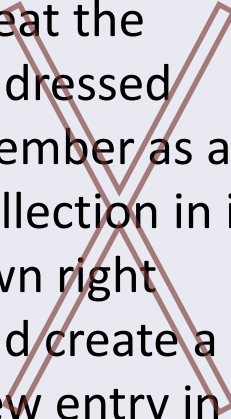
- ▶ RESTful Web Services expose their data and functionality through **resources** identified by **URI**
- ▶ **Uniform Interface Principle**: Clients interact with resources through a fixed set of verbs. Example HTTP:



POST (create) | **GET** (read) | **PUT** (update) | **DELETE**

- ▶ **Multiple representations** (MIME types) for the same resource: XML, JSON, ...
- ▶ **Hyperlinks** model resource relationships and valid state transitions for dynamic protocol description and discovery

RESTful WS: Example (adapted from *Wikipedia*)

Resource	GET	PUT	POST	DELETE
http://www.x.com/api/products	List the members (URIs and perhaps other details) of the collection. For example list all the products.	Replace the entire collection with another collection. 	Create a new entry in the collection. The new entry's ID is assigned automatically and is usually returned by the operation.	Delete the entire collection. 
http://www.x.com/api/products/41714	Retrieve a representation of the addressed member of the collection, expressed in an appropriate Internet media type.	Update the addressed member of the collection, or if it doesn't exist, create it.	Treat the addressed member as a collection in its own right and create a new entry in it. 	Delete the addressed member of the collection.

References

- ▶ SHKLAR, Leon et al. **Web Application Architecture: Principles, Protocols and Practices, Second Edition**. John Wiley & Sons, 2009.
- ▶ www.w3c.org
- ▶ www.w3schools.com