Chapter 2: Interactive Web Applications

Example: Fibonacci Function in PHP (Version 1)

```
<body> ...
  <h2>
     <?php
          function fib($n){
               if ($n==0)
                    return 0;
               else
                    if ($n==1)
                         return 1;
                    else
                         return fib($n-1)+fib($n-2);
          };
          echo "fib(3) = ", fib(3), "<br>";
          echo "fib(8) = ", fib(8), "<br>";
     ?>
  </h2>
</body>
</html>
```

fibonacci1.php

HTTP Basics

- HTTP = HyperText Transfer Protocol, see http://www.w3.org/Protocols/
- Client-Server communication:
  - Client opens (TCP) connection to server (usually on port 80)
  - Client sends request (as text lines)
  - Server sends response (as text lines)
  - Client closes connection (HTTP is *stateless*)
- Format of all HTTP messages (requests and responses):

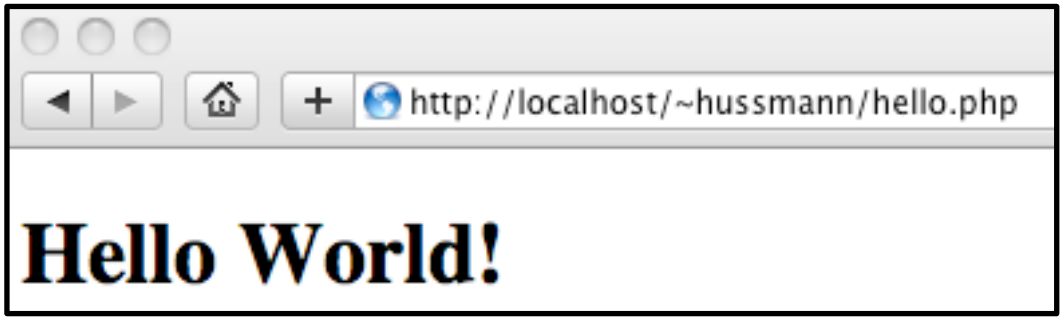
```
Initial line
Header lines (zero or more)
Blank line
Message body (optional)
```

- Example HTTP request:

```
GET /lehre/ws1314/mmn/index.html HTTP/1.1
Host: www.medien.ifi.lmu.de:80
<blank line!>
```

Sample HTTP Request (GET)

```
GET /~hussmann/hello.php HTTP/1.1
ACCEPT: text/html,application/xhtml+xml,application/
xml;q=0.9,*/*;q=0.8
ACCEPT_ENCODING: gzip, deflate
ACCEPT_LANGUAGE: en-us
CONNECTION: keep-alive
HOST: localhost
USER_AGENT: Opera/9.80 (Macintosh; Intel Mac OS X 10.8.5;
U; en) Presto/2.9.168 Version/11.52
CONTENT_TYPE:
```



HTTP Server Responses

- Message sent back from HTTP server always contains an initial response line which gives the *status* of the request processing.
- Example (success):

```
HTTP/1.1 200 OK
```

- Example (error):

```
HTTP/1.1 404 Not found
```

- Status codes:
  - 1xx: Informational message
  - 2xx: Success of some kind
  - 3xx: Redirection to other URL
    - e.g. 303: See other URL (given in Location: header)
  - 4xx: Client side error
  - 5xx: Server side error
    - e.g. 500: Server error

Example HTTP Response

- Experimenting manually with HTTP client/server dialogues:
  - "**`telnet <host> 80`**" in UNIX shell
- Retrieving a HTML page:

```
GET /~hussmann/hello.php HTTP/1.1
Host: localhost:80
```

- Response:

```
HTTP/1.1 200 OK
Date: Mon, 21 Oct 2013 13:07:14 GMT
Server: Apache/2.2.24 (Unix) DAV/2 PHP/5.3.26 mod_ssl/
2.2.24 OpenSSL/0.9.8y
X-Powered-By: PHP/5.3.26
Content-Length: 214
Content-Type: text/html

<!DOCTYPE html> … <html> … </html>
```

Passing CGI-Style Parameters in GET Request

- Convention for passing parameter values to server-side programs
  - Introduced by the *Common Gateway Interface (CGI)*
  - Not part of the HTML protocol!
  - Interpreted by server programs, e.g. PHP module
- Syntax:
  - Parameter data stream is appended to URL after a "?"
  - Keyword/value pairs, separated by "=", e.g. "**`fibinput=12`**"
  - Multiple parameter groups are separated by "&"
  - Spaces in strings are replaced by "+"
  - Non-ASCII characters (and special characters "&", "+", "=", "%") are replaced by "%*xx*" (hexadecimal code of character in used character set)

Fibonacci Function in PHP: Using Request Data

```
<body>
   <h1>
            Fibonacci Function (Result)
   </h1>
   <h2>
            <?php
                  $fibinput = $_REQUEST['fibinput'];
                  function fib($n){ as in version 1 };
                  echo "fib($fibinput) = ";
                  echo fib($fibinput);
                  echo "<br>";
            ?>
            <br>
            <a href="fibonacci2a.html">New Computation</
   a>
   </h2>
</body>
```

fibonacci2b.php

Example GET Request with Parameter

- Request:

```
GET /~hussmann/fibonacci2b.php?fibinput=10 HTTP/1.1
Host: localhost
```

- Response:

```
HTTP/1.1 200 OK
Date: Mon, 21 Oct 2013 13:18:38 GMT
Server: Apache/2.2.24 (Unix) DAV/2 PHP/5.3.26 mod_ssl/
2.2.24 OpenSSL/0.9.8y
X-Powered-By: PHP/5.3.26
Content-Length: 337
Content-Type: text/html

<!DOCTYPE html>

<html>
<head> … fib(10) = 55 … </html>
```

GET and POST Methods in HTTP

Hypertext Transfer Protocol (HTTP) supports two methods for passing parameter values to called documents/scripts:

- GET Method:
  - Values of variables are coded and transmitted within URL:
    `http://host.dom/pfad/fibonacci2.php?fibinput=10`
  - Parameters can be passed just by creating a certain URL (without forms)
  - Suitable for simple requests
- POST Method:
  - Values of variables coded and transmitted in the HTTP message body data
  - Values of variables not visible in URL
  - Web server reads parameter values from message (like browser reads HTML text)
- Variable encoding is not part of HTTP (but specified for HTML forms)
  - For POST requests, the coding method is given in the Content-Type header
    - » application/x-www-form-urlencoded (CGI conventions)
    - » multipart/form-data (segmented data, better for large data blocks)

Example POST Request with Parameter

- Request:

```
POST /~hussmann/fibonacci2b.php HTTP/1.1
Host: localhost
Content-Type: application/x-www-form-urlencoded
Content-Length: 11

fibinput=12
```

- Response:

```
HTTP/1.1 200 OK
Date: Mon, 21 Oct 2013 13:24:10 GMT
...
Content-Type: text/html

<!DOCTYPE html>

<html>
<head> … fib(12) = 144 … </html>
```

Variables, Parameter Passing and Security

- Global arrays **`$_REQUEST`**, **`$_GET`**, **`$_POST`**
  - for accessing external values determined at call time (like form input)
  - **`$_REQUEST`** contains all parameters given in request,
    `$_GET` and **`$_POST`** contains all parameters passed by the resp. method
  - Obtaining individual variable values by array lookup:
    `$_REQUEST['var'];`

- Older PHP versions (up to 4.2.0):
  - Huge security hole by not distinguishing between external parameters (e.g. input from HTML forms) and local variables
    - » External values were directly accessible through variables (like "**`$fibinput`**")
  - Manipulations of URL (GET parameter values) may enable setting of internal variables (e.g. “$authorization_successful”...!)
  - Old behavior can still be enabled by PHP server configuration

HTML Reminder: Forms

- User input in HTML:
  `<form>` Element
- Sub-element:
  - **`<input type=`*`ty`* **`name=`*`name`*`>`**
    Selected classic (HTML 4) types (*ty)*:

`checkbox`	Check box (Attribute **`checked`**)
`radio`	Radio button (Attribute **`checked`**)
`text`	Text input line
`textarea`	Multi-line text input area
`password`	Text input area not displaying the input
`file`	File selection
`button`	General button
`submit`	Button to send form contents
`reset`	Button to reset form contents

  - **`<select name=`*`name`*`>`** Pop-up menu for selection from options
    List of options: Sub-elements **`<option>`**
    `<option selected>` defines "pre-selected" values

HTML Form Example

```
<body>
    <form action="test.php"
          method="GET"
          enctype="application/x-www-form-urlencoded">
          <label>Name <input type="text" name="name"
                 maxlength="10"/></label><br>
          Sex:
          <input type="radio" name="sex"
                 value="male"></input> male
          <input type="radio" name="sex"
                 value="female"></input> female <br>
          <input type="checkbox" name="married"
                 value="yes">Married</input><br>
          <input type="submit" value="Submit" />
    </form>
</body>
```

/test.php?name=Max+Muster&sex=male&married=yes

HTML Forms and Server-Side Scripts

- HTML page containing forms usually calls separate script page and transfers form data as variable values
- **`action`** attribute for HTML tag **`<form>`**
  - Specifies the server page to process the input
  - Can contain embedded script
- **`method`** attribute for HTML tag **`<form>`**
  - Specifies the HTTP method to be used to transfer form data to the server
  - Possible values: GET (default), POST
- **`enctype`** attribute for HTML tag **`<form>`**
  - Specifies the encoding method to be used for form data
  - Possible values:
    - » application/x-www-form-urlencoded (CGI conventions) (default)
    - » multipart/form-data (segmented data)

Example: POST Request with Multipart Encoding

- HTML:

```
<form action="test.php"
    method="POST" enctype="multipart/form-data">
```

- Generated HTTP request:

```
POST /test.php HTTP/1.1
Host: localhost …
Content-Type: multipart/form-data;
boundary=---------------------------103832778631715
Content-Length: 355

-----------------------------103832778631715
Content-Disposition: form-data; name="name"

Max Muster
-----------------------------103832778631715
Content-Disposition: form-data; name="sex"

male
-----------------------------103832778631715
Content-Disposition: form-data; name="married"

yes
-----------------------------103832778631715--
```

Fibonacci Function in PHP (Version 2): Input Form Calling PHP Script

```
<body>
  <h1>
     Fibonacci Function (Input)
  </h1>
  <h2>
     Please enter number:
     <form name="fibform" action="fibonacci2b.php">
          <input type="text" name="fibinput"
            value="0"><br>
          <input type="submit" value="Compute">
     </form>
  </h2>
</body>
</html>
```

fibonacci2a.html

Combination of Input and Result Pages

```
<body>
   <h1>
       Fibonacci Function
   </h1>
   <h2>
       <?php
              function fib($n){  as above  };
              $eingabe = $_REQUEST['fibinput'];
              echo "fib($eingabe) = ";
              echo fib($eingabe);
              echo "<br>";
       ?>
       <br>
       Please enter number:
       <form name="fibform" action="fibonacci2.php">
              <input type="text" name="fibinput" value="0"><br>
              <input type="submit" value="Compute">
       </form>
   </h2>
</body>
```

action="fibonacci2.php" can be omitted

fibonacci2.php

Form Validation, Traditional Style

- Data entered into input forms needs to adhere to specific constraints:
  - Some fields required, some optional
  - Special formats like date, URL, email address
- Checking the constraints (“validating” the input)
  - Performed by client-side script code (JavaScript)
  - Typically an event handler for the “submit” event
  - Only if validation returns true, data is submitted
- Client-side validation saves server time and network traffic
  - Nevertheless, server usually validates received data again!

Example: Traditional Form Validation

```
<form id="blogentry">
  <label for="name">Name: </label>
  <input name="name" type="text"></br>
  <label for="email">Email: </label>
  <input name="email" type="text">
  <input type="submit" value="Submit">
</form>
<script type="text/javascript">
  blogentry = document.getElementById("blogentry");
  blogentry.addEventListener("submit", validateForm, false);
  function validateForm() {
    if (blogentry.name.value =="") {
      alert("Name is required");
      return false;
    };
    var emailinput=blogentry.email.value;
    var atpos=emailinput.indexOf("@");
    var dotpos=emailinput.lastIndexOf(".");
    if (atpos<1 || dotpos<atpos+2 || dotpos+2>=emailinput.length) {
      alert("Not a valid e-mail address");
      return false;
    };
    return true;
  }
</script>
```

formvalidate.html

Email validation code taken from w3schools.org

Detour: Accessing HTML Elements in JavaScript

- Old-fashioned JavaScript document tree:
  - Array access: **`document.forms[`*`f`*`].elements[`*`e`*`]`**
  - Shorthand: **`document.forms.`*`f`*`.elements.`*`e`** (associative array)
  - Even shorter: **`document.`*`f`*`.`*`e`**
- Strict DOM style:
  - **`document.getElementById("`*`f`*`")`**
- HTML5 Candidate Recommendation (Aug 6, 2013), Sect. 5.2.4:
  - The Window interface supports named properties. The supported property names at any moment consist of the following, in tree order, ignoring later duplicates:
  - the browsing context name of any child browsing context of the active document whose name is not the empty string,
  - the value of the name content attribute for all a, applet, area, embed, form, frameset, img, and object elements in the active document that have a non-empty name content attribute, and
  - the value of the id content attribute of any HTML element in the active document with a non-empty id content attribute.
- Note that **`window`** is equivalent to **`self`** in JavaScript and can be omitted!

Form Validation with HTML5

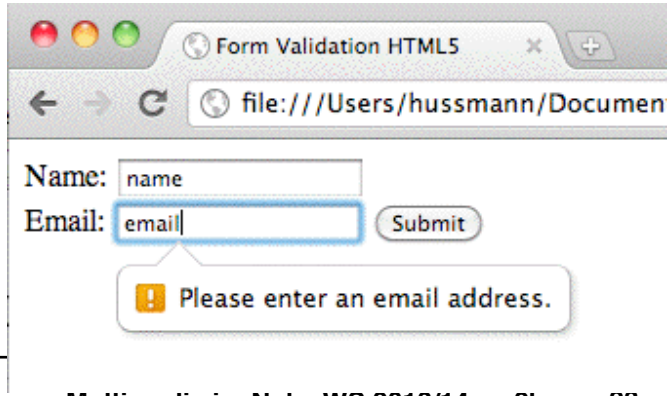
- Standard scenarios of form validation are integrated into HTML5 standard
  - Input types: email, URL, date, time, number, range, search, phone number, color
  - Attributes: Required, min, max, step, pattern
- Frequent phenomenon:
  - ***Procedural*** features are transformed to ***declarative*** features
- Using HTML5, JavaScript code can be removed
  - Just using declarative HTML
  - New code is less error-prone
  - New code is more precise (regarding definition of input syntax)
  - New code automatically benefits from upgrades
  - Special devices (e.g. smartphones) can choose best representation
- Transition problem:
  - For “legacy browsers”, traditional code has to remain for some time

Example: Form Validation with HTML5

```
<!DOCTYPE html>

<html>
  <head>
    <title>Form Validation HTML5</title>
  </head>

  <body>
    <form name="blogentry">
      <label for="name">Name: </label>
      <input id="name" type="text" required></br>
      <label for="email">Email: </label>
      <input id="email" type="email" required>
      <input type="submit" value="Submit">
    </form>
  </body>
</html>
```



formvalidate5.html Google Chrome

Chapter 2: Interactive Web Applications

Literature:

B. Lawson, R. Sharp: Introducing HTML5, New Riders 2011

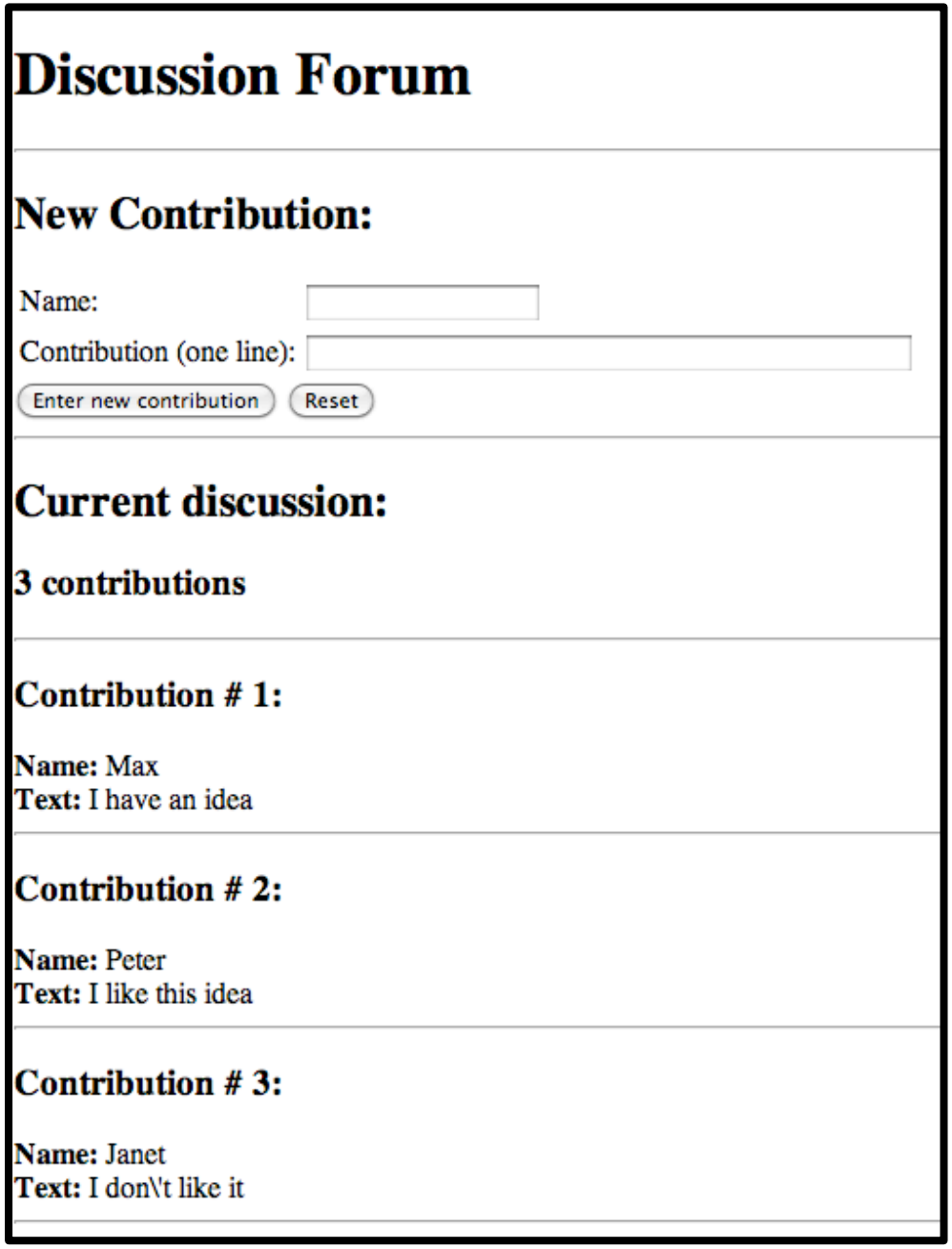
S. Fulton, J. Fulton: HTML5 Canvas, O'Reilly 2011

Data Storage Options in the Web: Overview

- Client-side storage:
  - Necessary to maintain continuity of client interaction
  - Session level: Linking consecutive request/response pairs
  - Long-term level: Personalization, preferences
  - Implemented in browser
  - Traditional solution: Cookies
  - Modern solutions (HTML5): Web Storage, Web SQL Databases
- Server-side storage:
  - Necessary to get access to and modify global information
  - Implemented on server
  - Simple solution: Server files (see PHP discussion forum example below)
  - Powerful solution: SQL database access from server scripts
- Note: Discussion is focused on Relational Databases and SQL due to their overwhelming popularity
  - Object-oriented and other database concepts beyond SQL (“NoSQL”)?

A Simple Discussion Forum (1)

- Interactive submission of text contributions
- Display of all submissions available on server
- Server uses simple text file for storage
- Altogether approx. 50 lines of HTML+PHP !



A Simple Discussion Forum (2)

Contents of file "forum.txt":

- Each two consecutive lines represent one contribution.
- First line: Name
- Second line: Text

```
Max
I have an idea
Peter
I like this idea
```

A Simple Discussion Forum (3)

Display of the full content of the file 'forum.txt'

- Used file function:
  - `file()`: Converts file content to string array
- Used array function:
  - `count()`: Length of array

```
<h2>Current discussion:</h2>
<?php
   $content = file("forum.txt");
   echo "<h3>", count($content)/2, " contributions</h3>";
   echo "<hr>";
   $i = 0;
   while ($i < count($content)) {
       echo "<h3>Contribution # ", ($i+2)/2, ":</h3>";
       echo "<b>Name: </b>", $content[$i++], "<br>";
       echo "<b>Text: </b>", $content[$i++], "<br>";
       echo "<hr>";
   }
?>
```

forum.php

A Simple Discussion Forum (4)

Extending the file 'forum.txt' with a new contribution

- Parameter **`$newcontrib`** indicates whether the "enter contribution" button was pressed

- Used file functions:
  - **`fopen(), fclose():`** Open file ("a"=append), close file
  - **`fputs():`** Write string to file

```
<?php
  $newcontrib = $_REQUEST['newcontrib'];
  $name = $_REQUEST['name'];
  $contrib = $_REQUEST['contrib'];
  if ($newcontrib != "" && $name != "" && $contrib != "") {
     $file = fopen("forum.txt", "a");
     if ($file) {
           fputs($file,$name . "\n");
           fputs($file,$contrib . "\n");
           fclose($file);
     }
  }
?>
```

Sessions and States

- HTTP is stateless
  - Server does not “remember” any data from previous transactions
- Linking several transactions to a “session” with common data storage
  - Client-side: Storing all data on client and re-transmit for every transaction
  - Server-side: Storing all data on server, client has to identify the session
- Common solution:
  - Server-side software offers session support
    - » E.g. session support in PHP
  - Client stores “session id”
  - Methods for linking request to session id:
    - » Variable/value pair in GET or POST request
    - » HTTP “Cookie”

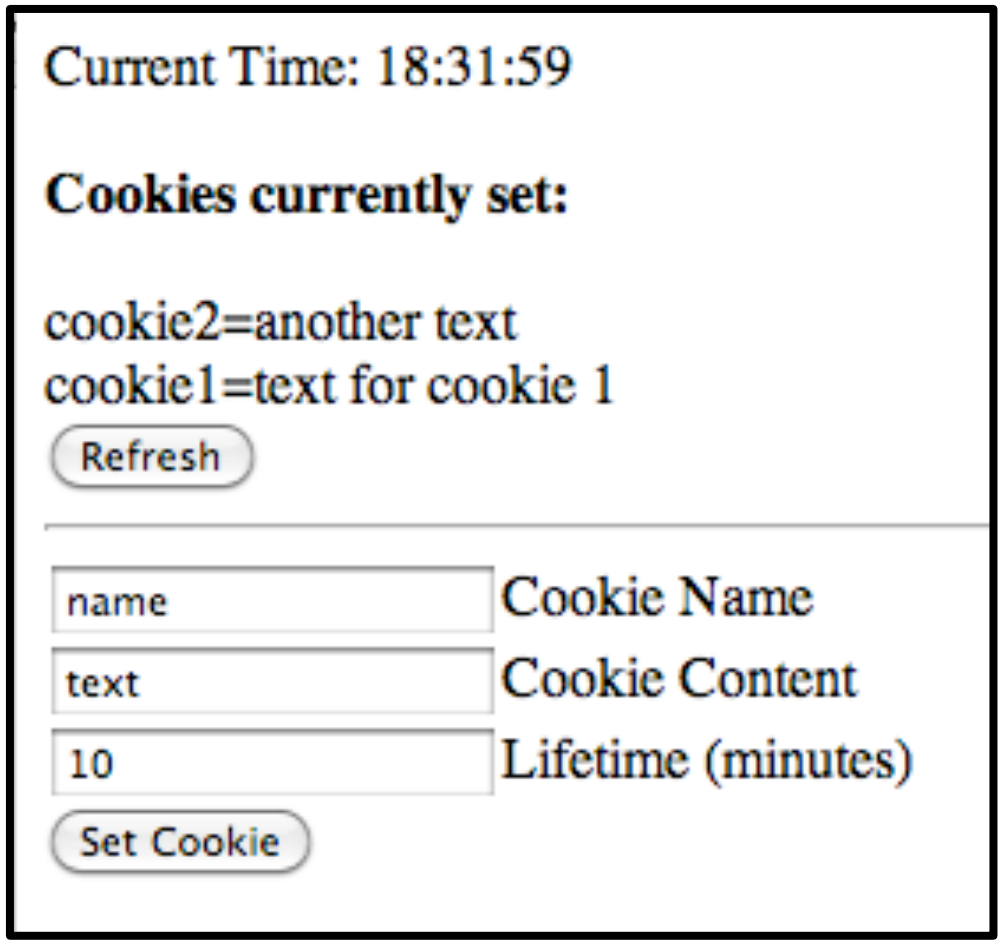
Cookies in HTTP

- Small data units stored in the browser storage area, controlled by browser
- Cookie contains:
  - *Name* (String), also called *key*
  - *Value* (String)
  - *Expiration date*
  - optional: domain, path, security information
- HTTP transfers cookies between client and server
  - In response, server can include header line “Set-Cookie:”
    - » Further information: name + value pair, expiration time
  - Cookie is stored by the browser
  - In further requests to the same server, client includes header line “Cookie:”
    - » Further information: name + value pair
  - Only cookies related to the requested server are transferred

Types of Cookies

- Session cookie
  - Deleted on browser termination
  - No expiration date given = session cookie
- Persistent cookie
  - For tracking, personalization
- Secure cookie
  - Only transmitted when secure connection to server is used
- HttpOnly cookie
  - Access only for HTTP, not for script APIs
- Third party cookie
  - Cookies set for different domain than currently visited server
  - Used for tracking and cross-domain advertising

Cookies in PHP: Screenshot



Accessing Cookies

Displaying a list of all cookies currently set (for this application) by reading from global array $_COOKIE:

```
<html>
  <?php
    date_default_timezone_set('Europe/Berlin');
    echo "Current Time: ", date("G:i:s"), "<br><br>\n";
    echo "<b>Cookies currently set:</b><br><br>\n";
    while (list($k, $v) = each($_COOKIE))
          echo $k, "=", $v, "<br>\n";
  ?>
...
</html>
```

HTML Form for Setting a Cookie

```
<form>
    <input type="text" name="key" value="name">
       Cookie Name<br>
    <input type="text" name="val" value="text">
       Cookie Content<br>
    <input type="text" name="tim" value="10">
       Lifetime (minutes)<br>
    <input type="submit" name="set"
       value="Set Cookie"><br>
</form>
```

- Page loaded via `action` is identical to page containing the form ("cookietest.php") – omitting the `action` attribute is sufficient.
- Due to server-side execution, the actual setting action can only be carried out when the next page is loaded!
- `"name"` attribute of **`submit`** button required for distinction to other buttons ("refresh" in the example).

Setting the Cookie

```
<?php
   if ($_GET['set']) {
      $key = $_GET['key'];
      $val = $_GET['val'];
      $tim = $_GET['tim'];
      $exp = time() + $tim * 60;
      setcookie($key, $val, $exp);
   }
?>
<!DOCTYPE html>
<html>
...
```

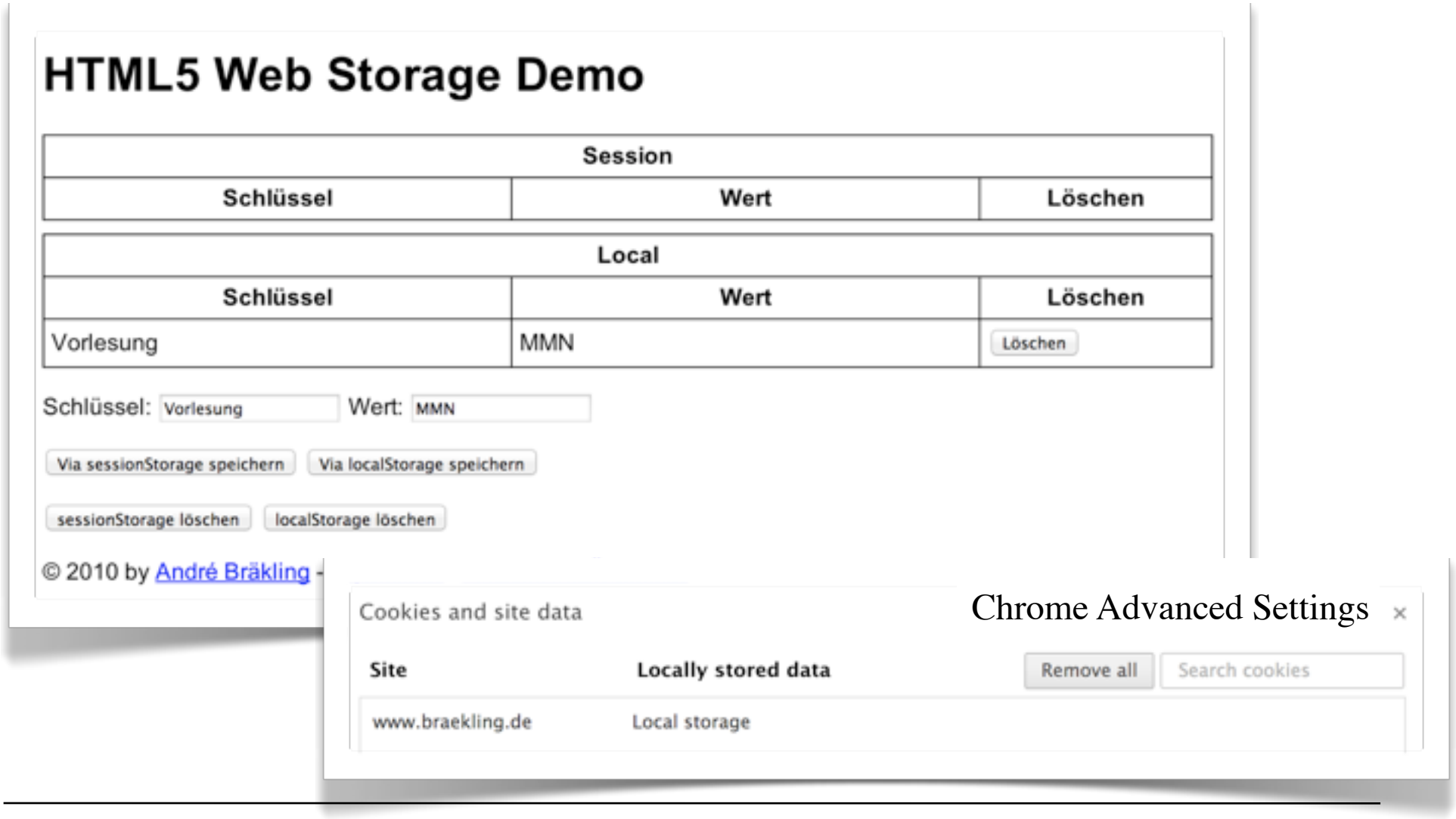
- "name" attribute of **submit** button ('set') is used to decide whether **set** button was pressed
- setcookie() call has to be very first output of page, to be transmitted together with the headers (HTTP requirement).

Client-Side Storage in HTML5: Web Storage

- Web Storage/DOM Storage:
  - Standardized by W3C, intended as improvement over Cookies
  - Formerly part of HTML5 specification, now separated
- Purely client-side storage
  - Not transmitted to server with each request
  - Javascript code can issue read and write requests
- Types of storage:
  - Session storage: Related to window/tab (!), deleted on window closing or browser termination
  - Local storage: Related to domain and maintained after browser termination
- Data structure:
  - Simple associative array (key/value pairs, both of string type)
  - Similar to Cookies

Web Storage Example

http://www.braekling.de/testlab/html5-webstorage-demo.html



Web Storage Interface (W3C)

- Interface **`Storage`** (defined independently of implementation language):

```
String getItem(String key);
void setItem(String key, String value);
void removeItem (String key);
void clear();
```

- Top-level browsing context contains two attributes:

```
Storage sessionstorage;
Storage localstorage;
```

- Shorthand notation in JavaScript due to associative array, example:

```
var firstName = localStorage.firstName;
var lastName = localStorage.lastName;
```

- When a storage area changes, an event is fired:

```
StorageEvent storage;
```

JSON Stringification

- What to do if only strings can be stored (somewhere)?
- All data objects (in JavaScript and other languages) can be converted to a String representation
  - XML based
  - Based on JavaScript object constructors: JSON (= JavaScript Object Notation), more space effective
  - **`JSON.stringify():`** Returns string representation
  - **`JSON.parse():`** Converts string representation to JavaScript object
- Example:

```
{"student": {
  "identification": [
        {"name": "firstname",
         "value": "Max"
        },
        {"name": "lastname",
         "value": "Muster"
        }],
  "grades": […]
  }
}
```

Working Offline in Web Applications

- Web applications often rely on connectivity to the server
  - There are still situations/regions without or with restricted/expensive Internet access!
  - Mobile connections are always in danger of temporary failures
- Working offline with server-based applications:
  - Client needs a significant amount of logic to give sense to offline work
  - Application needs to specify which parts of the application data is to be kept locally *(cached)*
    - » Usually a set of files
    - » *Cache manifest* (= list of files)
  - Browser needs to support access to cached data
    - » interpret cache manifest
    - » maintain application cache

HTML5 Cache Manifest

- Cache manifest is a file on the server referenced in the HTML page to be loaded:

```
<!DOCTYPE html>
<html lang="en" manifest="time.manifest">
```

- Cache manifest states the files always to be loaded (even from cache)and the files for which there is an alternative:

```
CACHE MANIFEST
# version 10

CACHE:
index.html
time.js
time.css

FALLBACK:
server-time.js fallback-server-time.js
```

HTML5 Cache Manifest Demo

- If file **`server-time.js`** is available and delivers server time:

> The time on your computer is **0:25:38** and the time on the server is **10:38:33**

- If file **`server-time.js`** is ***not*** available, local **`fallback-servertime.js`** is used:

> The time on your computer is **0:28:30** and the time on the server is **unavailable, you need to be connected to get the server time**

- Distinction between available files and non-available files is done by the application, adequate reaction is carried out.
- Non-realtime data are retrieved from local memory.

Potential Enabled by Server-Side Scripts

- Receive and store user input
  - In various forms of persistent storage
    - » Plain text files, XML files, data base
- Process input and compute results
  - Depending on various information available on server side
- Create output suitable for being displayed in Web browsers
  - HTML, may include JavaScript
- Make use of advanced features offered by Web browsers
  - Examples: Cookies, user agent identification

Applications to Multimedia

- PHP is not directly multimedia-related, but HTML-oriented
- HTML allows media embedding
- The combination of HTML + PHP + media embedding enables the creation of new digital media
- Examples for interactivity added to media playback, realizable by PHP scripts
  - Selection of media, e.g. search functions
    - » Using forms and backend data base
  - User-specific recommendations
    - » Using cookies
  - Aggregating (explicit and implicit) user input
    - » Frequency of use for individual media (charts)
    - » Correlation of use across media (collective recommendation)
    - » Tagging