Chapter 2: Interactive Web Applications

2.1 Interactivity and Multimedia in the WWW architecture
2.2 Server-Side Scripting (Example PHP, Part I)
2.3 Interactivity and Multimedia for Web Browsers
2.4 Interactive Server-Side Scripting (Example PHP, Part II)
2.5 Interactive Client-Side Scripting (Example HTML5/JavaScript)
2.6 Data Storage in Web Applications (Example Database Access in PHP)
2.7 Asynchronous Interactivity in the Web (Example AJAX)
Chapter 2: Interactive Web Applications

2.5 Interactive Client-Side Scripting (Example JavaScript)
2.6 Data Storage in Web Applications
   (Example Database Access in PHP)
2.7 Asynchronous Interactivity in the Web (Example AJAX)

Literature:
S. Fulton, J. Fulton: HTML5 Canvas, O’Reilly 2011
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 "onsubmit" 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

```html
<script type="text/javascript">
  function validateForm() {
    if (document.blogentry.name.value =="") {
      alert("Name is required");
      return false;
    }
    var emailinput=document.blogentry.email.value;
    var atpos=emailinput.indexOf("@" Malone
    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>
<form name="blog-entry" onsubmit="return validateForm();">
  <label for="name">Name: </label>
  <input name="name" id="name" type="text"> </br>
  <label for="email">Email: </label>
  <input name="email" id="email" type="text">
  <input type="submit" value="Submit">
</form>
```

formvalidate.html
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

```html
<!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>
```
HTML5 Interactive Controls

• Standard controls for interactive applications have been integrated into HTML5
  – “range” element (slider control)
  – “color” element (interactive color picker)

• Potential:
  – Higher client-side (stand-alone) interactivity
  – Typical applications: Drawing, image editing
  – See discussion of “canvas” element below
Example: Slider in HTML5

```html
<!DOCTYPE html>

<html>
<head>
  <title>Slider in HTML5</title>
  <style type="text/css">
    input[type=range]::before {content: attr(min)}
    input[type=range]::after {content: attr(max)}
    input[type=range]
      {width:500px; color:red; font-size:1.5em;}
  </style>
</head>

<body oninput="current.value=slider.value">
  <input name="slider" type="range" min="1" max="11"/>
  <output name="current">5</output>
</body>
</html>
```
HTML5 Canvas

• “HTML5 Canvas is an immediate mode bitmapped area of the screen that can be manipulated with JavaScript.” (Fulton/Fulton)

• 2D Drawing Context:
  – Object associated with a Canvas object
  – Used as handler in JavaScript to address the canvas (drawing API)

• Typical drawing primitives:
  – Draw shapes
  – Render text
  – Display images
  – Apply colors, rotations, transparency, pixel manipulations, fills, strokes

• Canvas works on (low) pixel level
  – Browser redraws whole canvas each time the Canvas is modified using JavaScript
Demo: Harmony

Tested with Opera
Example: Drawing on the Canvas

<!doctype html>
<html>
<head>
  <title>Canvas Hello World</title>
  <script type="text/javascript">
    window.addEventListener("load", drawScreen, false);
    function drawScreen() {
      var c = document.getElementById("theCanvas");
      var ctx = c.getContext("2d");
      ctx.fillStyle = "lightgrey";
      ctx.fillRect(0, 0, c.width, c.height);
      ctx.font = "italic bold 32px sans-serif";
      ctx.fillStyle = "red";
      ctx.fillText("Hello World!", 50, 50);
    }
  </script>
</head>
<body>
  <canvas id="theCanvas" width=300 height=80>
    Your browser does not support Canvas!
  </canvas>
</body>
</html>

 canvashello.html
Example: Interactive Gradient (1)

<!doctype html>
<html>
<head>
    <title>Canvas Gradient Fill</title>
    <meta charset="UTF-8">
    <script type="text/javascript">
        window.addEventListener("mousemove", drawScreen, false);
        function drawScreen(event) {
            var c = document.getElementById("theCanvas");
            var ctx = c.getContext("2d");
            var mx = Math.min(event.clientX, c.width);
            var my = Math.min(event.clientY, c.height);
            var grad =
                ctx.createRadialGradient(mx, my, 0, mx, my, c.width*1.5);
            grad.addColorStop(0,"#f00");
            grad.addColorStop(1,"#00f");
            ctx.fillStyle = grad;
            ctx.fillRect(0, 0, c.width, c.height);
        }
    </script>
</head>
<body>
</body>
Example: Interactive Gradient (2)

... 

<body>
  <canvas id="theCanvas" width=500 height=500>
    Your browser does not support Canvas!
  </canvas>
</body>
</html>
Interactive Animations with HTML5 / JavaScript

- Example “GeoBlaster” ("Asteroid"-like game by Fulton/Fulton)
Example Excerpt: Animation Principles (1)

• Timer-based loop:

```javascript
const FRAME_RATE=40;
frameRateCounter=new FrameRateCounter(FRAME_RATE);
var intervalTime=1000/FRAME_RATE;
setInterval(runGame, intervalTime);
```

• `runGame` function is called again and again in fixed frequency

• For each frame (varying dependent on game state):

```javascript
function gameStatePlayLevel()
{
  checkKeys();
  update();
  render();
  checkCollisions();
  checkForExtraShip();
  checkForEndOfLevel();
  frameRateCounter.countFrames();
}
```
Example Excerpt: Animation Principles (2)

- For each frame, all objects are updated:
  
  ```javascript
  function update() {
    updatePlayer();
    updatePlayerMissiles();
    updateRocks();
    updateSaucers();
    updateSaucerMissiles();
    updateParticles();
  }
  ```

- Example: Update of player’s ship:
  
  ```javascript
  function updatePlayer() {
    player.missileFrameCount++;
    player.x+=player.movingX*frameRateCounter.step;
    player.y+=player.movingY*frameRateCounter.step;
    if (player.x > xMax) {
      player.x=-player.width;
    }else if (player.x<-player.width){
      player.x=xMax;
    }...
  }
  ```
Example Excerpt: Animation Principles (3)

- Rendering the player's ship:

```javascript
function renderPlayerShip(x, y, rotation, scale) {
    // transformation
    context.save(); // save current state in stack
    context.globalAlpha = parseFloat(player.alpha);
    var angleInRadians = rotation * Math.PI / 180;
    var sourceX = Math.floor((player.rotation / 10) % 10) * 32;
    var sourceY = Math.floor((player.rotation / 10) / 10) * 32;
    if (player.thrust) {
        context.drawImage(shipTiles2, sourceX, sourceY, 32, 32, player.x, player.y, 32, 32);
    } else {
        context.drawImage(shipTiles, sourceX, sourceY, 32, 32, player.x, player.y, 32, 32);
    }
    ...
}
```
HTML5 Canvas vs. SVG vs. Flash/Silverlight

• HTML5 Canvas:
  – *Immediate* mode (direct drawing on canvas, no structuring of image)
  – Bitmap level

• SVG, Flash, Silverlight:
  – *Retained* mode (internal “scene graph” model rendered to canvas, incremental updates possible)
  – Bitmap or vector level

• HTML5 Canvas & SVG:
  – Open standards, dependent on acceptance by browser manufacturers

• Flash, Silverlight etc.:
  – Closed, proprietary standards (browser plugins needed)
  – Popularity determines “penetration” of plugins to browser population

• In any case, “political” decisions are important:
  – Example: Open standard support in MS Internet Explorer
  – Example: Lack of Flash support on Apple iOS devices (iPhone, iPod, iPad)
Chapter 2: Interactive Web Applications

2.1 Interactivity and Multimedia in the WWW architecture
2.2 Server-Side Scripting (Example PHP, Part I)
2.3 Interactivity and Multimedia for Web Browsers
2.4 Interactive Server-Side Scripting (Example PHP, Part II)
2.5 Interactive Client-Side Scripting (Example HTML5/JavaScript)
2.6 Data Storage in Web Applications (Example Database Access in PHP)
2.7 Asynchronous Interactivity in the Web (Example AJAX)
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)
  – Powerful solution: SQL database access from server scripts

• Note: Discussion is focused on Relational Databases and SQL due to their overwhelming popularity
  – Object-oriented databases?
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
  - Server 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

Web Storage Example: Code Snippets

- Storing data:
  ```javascript
  function storeFirstName() {
    var firstName =
        document.getElementById("firstName").value;
    localStorage.firstName=firstName;
    showStorage();
  }
  ```
  - Formally, there is an interface `Storage` defining the methods `getItem`, `setItem`, `removeItem`, which can be used in this shortcut style.

- Retrieving data:
  ```javascript
  function showStorage() {
    var firstName = localStorage.firstName;
    var lastName = localStorage.lastName;
    if (firstName!=null) {
      document.getElementById("firstNameDisplay").innerHTML = firstName;
    }
    ...
  }
  ```
**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:
  ```json
  { "student": {
    "identification": [
      { "name": "firstname", "value": "Max" },
      { "name": "lastname", "value": "Muster" }
    ],
    "grades": [...] } 
  }
  ```
Advanced Client Side Storage:
Web SQL Databases

• Idea: Browser-local storage of data which can be queried in SQL-style like from a Relational Database
• API standardized by W3C and supported by some browsers (Chrome, Opera, Safari), not by Firefox
• Development stalled in November 2010
Database Management Systems: A Quick Reminder

• Database:
  – Structured collection of data items
  – Stored persistently
  – Provides access to a common data pool for multiple users

• Database Management System (DBMS):
  – Collection of programs for administration and usage of a database
  – Various base models for DBMS:
    » Old: network model, hierarchical model
    » Dominant: relational model
    » Alternative: object-oriented model

• Relational databases:
  – Good methodological support for design of data schema
  – Standardized language interface SQL (Structured Query Language)
Prerequisites and Basic Architecture

- Database server
  - Database 1
    - Table 1.1
    - Table 1.2
  - Database 2
    - Table 2.1
    - Table 2.2
- Administration software
- User programs
- Other Servers (e.g. Web Server)
MySQL

- Open source software system
  - Frequently used also in commercial context
  - [www.mysql.com](http://www.mysql.com)

- Software package providing:
  - Database server (mysqld)
  - Administration program (mysqladmin)
  - Command line interface (mysql)
  - Various utility programs

- Communication between programs on local host: *socket* interface
  - Bidirectional data stream exchange between programs
  - Similar to files
Before Creating Anything in the Database...

- Using a database requires careful *information design*.
- Which are the data to be stored?
- Are there existing data to connect to?
- What is the **schema** of the data to be stored?
  - Eg. Entity-Relationship diagrams as a tool
  - Transformation into relational database schema (table design)
- Once a database if filled with data and in use, it is rather difficult to modify!
  - Database schema design has to be carried out with great care!
- Most important rule: Avoid redundant storage of information
Creating Database Tables (1)

• Prerequisites:
  – Database server running
  – Socket connection between programs intact
  – User accounts with adequate privileges known

• First step: Create **database**
  – Container for many tables
  – Requires special privileges
  – Example SQL:
    
    ```
    create database music;
    ```

• Second step: **Choose used** database
  – Sets the context for further interactions
  – Example SQL:
    
    ```
    use music
    ```
Creating Database Tables (2)

- Third step: Create **tables**
  - According to earlier design
  - Each table should provide a unique identifier (**primary key**)
  - SQL Example:
    
    ```sql
    create table song (code VARCHAR(5), title VARCHAR(20), artist VARCHAR(20), composer VARCHAR(20), runtime INT);
    ```
  - Further steps: Defining keys, indices etc.

- Fourth step: Fill tables with **data**
  - Simplest case: Individual SQL commands
  - Better: Import from structured data file
  - Frequent: Special programs for importing and creating data
  - SQL Example:
    
    ```sql
    insert into song
    values ('1','One','U2','Adam Clayton, Bono, Larry Mullen & The Edge',272);
    ```
SQL Monitor Output

```
mysql> describe song;
+------------+------------+------+-----+-------------+----------+
| Field      | Type       | Null | Key | Default     | Extra    |
+------------+------------+------+-----+-------------+----------+
| code       | varchar(5) | YES  |     | NULL        |          |
| title      | varchar(20)| YES  |     | NULL        |          |
| artist     | varchar(20)| YES  |     | NULL        |          |
| composer   | varchar(20)| YES  |     | NULL        |          |
| runtime    | int(11)    | YES  |     | NULL        |          |
+------------+------------+------+-----+-------------+----------+
5 rows in set (0.01 sec)
mysql>  
```
## Queries with SQL

```sql
mysql> select * from song;
+-----------+---------+-----------------+-------------------+-------+
<table>
<thead>
<tr>
<th>code</th>
<th>title</th>
<th>artist</th>
<th>composer</th>
<th>runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>One</td>
<td>U2</td>
<td>Adam Clayton, Bono</td>
<td>272</td>
</tr>
<tr>
<td>2</td>
<td>In the End</td>
<td>Linkin Park</td>
<td></td>
<td>219</td>
</tr>
<tr>
<td>3</td>
<td>Wheel in the Sky</td>
<td>Journey</td>
<td></td>
<td>252</td>
</tr>
<tr>
<td>4</td>
<td>Lady in Black</td>
<td>Uriah Heep</td>
<td></td>
<td>281</td>
</tr>
<tr>
<td>5</td>
<td>Smoke on the Water</td>
<td>Deep Purple</td>
<td></td>
<td>481</td>
</tr>
</tbody>
</table>
+-----------+---------+-----------------+-------------------+--------+
5 rows in set (0.00 sec)
```

```sql
mysql> select title from song where runtime>250;
+---------+
<table>
<thead>
<tr>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>One</td>
</tr>
<tr>
<td>Wheel in the Sky</td>
</tr>
<tr>
<td>Lady in Black</td>
</tr>
<tr>
<td>Smoke on the Water</td>
</tr>
</tbody>
</table>
+---------+
4 rows in set (0.00 sec)
```
Server-Side Databases, PHP and MySQL

- Special libraries for database access:
  - "Database extensions"
  - Generic for all database systems
- For specific database systems:
  - "Vendor specific database extensions"
- For MySQL:
  - MySQL-specific database extensions to PHP
Connecting to a Database from PHP

• First step: **Connect** to server
  – Establish a connection for data exchange between Web Server/PHP plugin and database server
  – Often local (sockets), if both programs on same machine
  – Requires hostname, (database) username, password
  – PHP function: `mysqli_connect()`
    » Returns a link (resource) which can be used for `mysqli_close()`

• Second step: **Select** a database
  – Corresponds to the SQL command **use**
  – Requires database name (and possibly link to server)
  – PHP function: `mysqli_select_db()`
    » Returns Boolean result (success)
Example: Connecting to Database

```php
$link = mysql_connect('localhost','root','demopw')
    or die ('Could not connect: '.mysql_error());
echo 'Connected.<br/>';

mysql_select_db('music')
    or die ('Could not select db.');
echo 'DB selected.<br/>';

...

?>
Sending Database Queries from PHP

• Basic idea (as in all programming language/database integrations):
  – SQL queries are given as strings to library functions

• Most important function in MySQL extensions to PHP:
  `mysql_query()`
  – Requires SQL query as parameter (optionally link to server as 2nd param.)
  – "Query" includes also **INSERT, UPDATE, DELETE, DROP** (SQL)!

• Return value in case of **SELECT, SHOW, DESCRIBE** and similar:
  – Result set represented by resource value
  – Special functions to retrieve result data as PHP data structures
    – `mysql_num_rows()`
      » Number of rows returned
    – `mysql_fetch_array()`
      » Reads one row of data and transforms it into an array
      » Makes the next row available
Example: Reading Data From a Query in PHP

```php
<?php
...
$query = 'SELECT * FROM song';
$result = mysql_query($query);

while ($row = mysql_fetch_array($result, MYSQL_ASSOC)) {
    foreach ($row as $element) {
        echo $element;
        echo ', ';
    }
    echo ('"<br/>"');
}
...
?>
```
Creating HTML Output From SQL Query (1)

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>
<head>
  <title>Database table in HTML</title>
</head>

<?php
$link = mysql_connect('localhost','root','demopw')
or die ('Could not connect: '.mysql_error());
mysql_select_db('music') or die ('Could not select db.');
?>
Creating HTML Output From SQL Query (2)

...  

```php
<body>
  <h1>The following table is retrieved from MySQL:</h1>
  <table>
    <?php
    $query = 'SELECT * FROM song';
    $result = mysql_query($query)
      or die ('Query failed'.mysql_error());
    while ($row = mysql_fetch_array($result, MYSQL_ASSOC)) {
      echo "\t<tr>\n";
      foreach ($row as $element) {
        echo "\t\t<td>";
        echo $element;
        echo "<\td>\n";
      }
      echo "\t</tr>\n";
    }
  echo "\t</table>\n";
  ?>
</table>
```
Creating HTML Output From SQL Query (3)

...  

```php
<?php
    mysql_free_result($result);
    mysql_close($link);
?>

</body>
</html>
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:

  \[
  \text{The time on your computer is 0:25:38 and the time on the server is 10:38:33}
  \]

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

  \[
  \text{The time on your computer is 0:28:30 and the time on the server is unavailable, you need to be connected to get the server time}
  \]

• 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.