Vorlesungsreihe
Entwicklung webbasierter Anwendungen

PHP
- Teil 1 - Grundlagen -

Prof. Dr.-Ing. Thomas Wiedemann /

email: wiedem@informatik.htw-dresden.de

HOCHSCHULE FÜR TECHNIK UND WIRTSCHAFT DRESDEN (FH)
Fachbereich Informatik/Mathematik

Gliederung zu PHP

- Historie , PHP-Versionen im Überblick
- Grundlagen der PHP-Programmierung
  - Einbindung in HTML
  - Syntax, Datentypen und Variablen
  - Operatoren , Anweisungen , Funktionen
  - Web-Formularanbindung
  - Datenbankanbindung
  - Funktionsbibliotheken im Überblick

Quelle(n) :

- **PHP-Homepage : www.php.net www.php.de**
- **Online-Kurse: www.selfphp.de www.w3schools.com/php**

Historie von PHP

- Mitte der 90er Jahre entstand der Vorläufer PHP/FI (=Personal Home Page / Forms Interpreter) durch Rasmus Lerdorf (Kanada) als ein Set von C-Binaries und Perl Skripten zur Erfassung der Zugriffe auf einen Webauftritt
- später mit **C-Sourcecode** als allgemeine Toolsammlung neu implementiert und als Open-Source frei gegeben (vgl. https://en.wikipedia.org/wiki/PHP)

PHP3 – Akronym für „PHP: Hypertext Preprocessor“ - ab Mitte 1998

- Komplette Überarbeitung von PHP/FI durch eine Kooperation von Andi Gutmans, Zeev Suraski und Rasmus Lerdorf mit den Schwerpunkten : Erweiterbarkeit (-> andere Entwickler -> PHP-Boom), DB-Schnittstellen, Netz-Protokollzugriff etc.)

PHP4 – Freigabe ab Mitte 1998

- viele neue Leistungsmerkmale in einer neuen Kernel-Engine – als 'Zend Engine bezeichnet (aus Zeev / Andi), stark verbesserte Performance und Unterstützung für viele weitere Webserver, HTTP-Sessions, viele neue Sprachkonstrukte.

PHP5 – ab 2004 nochmalige Leistungsverbesserungen in einer neuen ZEND-Engine 2.0, Detailänderungen bei der Einbindung verschiedener Bibliotheken

Aktuell: PHP 7.x -> PHP 6 mit Unicode-Support wurde 2010 gestoppt

PHP7 – Hauptziel: nochmals verbesserte Performance (ca. 30% schneller), teilweise aber nicht abwärtskompatibel

PHP im Überblick - grundlegende Sprachmerkmale

Einordnung :

- Hauptanwendung als Server-seitige Skriptsprache, eingebettet in HTML oder als eigenständiges Programm mit HTML-Ausgabe
- relativ einfach zu erlernen (schnelle Umsetzung typischer Webaufgaben)
- **schnelle Turn-around-Zyklen** (Code-Änderung im Sourcetext -> Serverupload)
- optional auch als Standalone-Komandozeilensprache im Rahmen von Batches oder als Clientseitige GUI Applikationen (selten) einsetzbar

Sprachmerkmale (im Vergleich zu C) :

- relativ große Ähnlichkeit zu C, aber nicht voll kompatibel
- portabel auf verschiedene Rechnerplattformen
- integrierte Speicherverwaltung bei Strings und Objekten (Garbage Collector)
- keine expliziten Pointer, keine separaten Header-Dateien
- bessere Laufzeitüberwachung bei Matrizen und anderen dynamischen Objekten
- gutes Fehlermanagement (stark beeinflussbar -> abschaltbare Meldungen)
- relativ sicher bei korrekter Programmierung (trotz pot. Vollzugriff auf Server)

Installation

- unterstützt werden (fast) alle Unix-Versionen und Windows mit Binaries oder generell mit Sourcecode (-> Neukompilieren auf System unter Beachtung der zusätzlich einzubindenden Bibliotheken – insbes. MySQL-Support !!)
- Die Anbindung kann über CGI – dann erfolgt bei jedem Aufruf ein Neustart des PHP-Parsers - oder über die SAPI-Schnittstelle von PHP erfolgen.
- Installation als CGI-Applikation (langsam, teilweise unsicher, -> Tests):

```
ScriptAlias /php/ "c:/php/"
AddType application/x-httpd-php .php
Action application/x-httpd-php "/php/php.exe"
```

=> httpd.conf

- Installation als Modul unter Apache 2.0 (schneller, effizienter)

```
LoadModule php5_module „c:/php/sapi/php5apache2.dll"
AddType application/x-httpd-php .php
```

PHP- Konfiguration

- Im Systemverzeichnis von PHP sind grundlegende Einstellungen in der php.ini abgelegt (je nach Server werden auch parallel mehrere .ini-Dateien für Debugg- oder Testzwecke unterstützt)
- Der generelle Aufbau entspricht der http.conf-Datei von Apache.
- Die wichtigsten Einstellungen (komplette Liste siehe PHP-Handbuch) sind:
  - Tag-Optionen bzgl. HTML-Einbindung
  - Sicherheitseinstellungen
  - Resourcen- und Fehlerhandling
  - Logging and Debugging
  - Übergabe von Daten aus Formularen an PHP – legt ggf. automatisch Variablen zu GET / POST / Cookies und System an
  - Zusatzfunktionen für Upload, Email-Versand und Datenbanken

```
short_open_tag = On
; Allow ASP-style <% %> tags.
asp_tags = On
max_execution_time = 1200     ;
html_errors = On
error_log = c:\www\error.log
register_globals = OFF
variables_order = "GPCS“
mysql.default_user =““
....
```

PHP- Sicherheit der Installation

- je nach Anwendungszweck und Konfigurationsausrichtung (Entwicklung / Produktivserver) sind entsprechende Sicherheitseinstellungen notwendig
- zuerst nur Disk. der Sicherheit der Grundinstallation (Prog. später):

Option 1 (veraltet) : **safe_mode** *boolean*

- *"This feature has been DEPRECATED as of PHP 5.3.0 and REMOVED as of PHP 5.4.0."*
- Hintergrund: safe_mode ist eine Blackbox und ggf. trotz Aktivierung unsicher !!

Option2 : statt dessen oder zusätzlich **disable_functions** *list_der_ funktionen*

"chdir,apache_child_terminate, apache_get_modules, apache_get_version, apache_getenv, apache_note, apache_setenv, curl_exec, curl_multi_exec, define_syslog_variables, disk_free_space, diskfreespace, dl, error_log, escapeshellarg, escapeshellcmd, exec, ftp_connect, ftp_exec, ftp_get, ftp_login, ftp_nb_fput, ftp_put, ftp_raw, ftp_rawlist, ini_alter, ini_get_all, ini_restore, link, mysql_pconnect, openlog, passthru, pfsockopen, php_uname, popen, posix_getpwuid, posix_kill, posix_mkfifo, posix_setpgid, posix_setsid, posix_setuid, posix_uname, proc_close, proc_get_status, proc_nice, proc_open, proc_terminate, set_time_limit, shell_exec, symlink, syslog, system, tmpfile, virtual

- Selbst diese Liste ist NICHT als definitiv sicher zu sehen (neue PHP-Codefehler können einzelne Befehle unsicher machen) -> Google: php disable_functions list

➔ bitte immer Check in aktueller Doku : **http://php.net/manual/en/ini.core.php**

PHP- Einbindung

- Option 1 – beliebige PHP-Abschnitte innerhalb von HTML

```
<html><body>
 <?php  echo "Hallo Welt !";     ?>
</body> </html>
```

echo – gibt Argument an Output-stream

- Option 1b – PHP kann selbst temporär wieder zu HTML wechseln, ohne daß dies Bedingungen oder Ausdrücke stört

```
<?php  if ($expression) {?>
   <strong>Das ist richtig.</strong>
   <?php   } else {   ?>
   <strong>Das ist falsch.</strong>
   <?php }  ?>
```

- Option 2 – komplette HTML-Ausgabe mit PHP

```
<?php  echo "<strong>Das ist HTML</strong>“ ?>
```

PHP- Grundlegende Sprachreferenz

- Abschluss aller Anweisungen mit Semikolon
- Kommentare wie in C /* beliebig langer Kommentar */ // nur bis Zeilenende
- Variablen werden durch $ am Anfang des Bezeichners markiert

```
$wert1 = 1;
```

- Es wird zwischen Groß- und Kleinschreibung unterschieden

```
$zaehler != $Zaehler
```

- PHP unterstützt keine expliziten Typ-Definitionen bei der Deklaration von Variablen; der Typ einer Variablen wird durch den Zusammenhang bestimmt in dem die Variable benutzt wird.

```
$wert1 = 1;    // Wert1 ist eine Integer-Variable
$wert2 = “Hallo“;    // Wert2 ist eine String-Variable
```

- Unterdrückung potentieller Fehlermeldungen durch vorangestelltes @

Verfügbare primitive Datentypen

- Boolean : $flag = TRUE ; $Flag2 = FALSE;
- Integer : $i = -123; $z_oktal = 0123; $z_hex = 0x1A;

  (Integer-Überlauf führt automatisch zu float -> keine Gefahr !)
- Fließkomma-Zahl (float) : $d1 = 1.234; $d2 = 1.2e-3;
- String / Zeichenkette : können beliebig lang sein, 3 Arten der Def.:

  $t1 = ‘single quoted text‘ // Keine Auswertung durch Parser!!!

  $t2 = “Das waren $anzahl werte !“; // double quoted !!!

  -> PARSER, d.h. PHP-Variablen IM TEXT werden ausgegeben !
- Alternative Heredoc-Syntax :

  - frei gewählter Bezeichner –

    hier EOFT1 begrenzt den Text

    - Bez. muß in 1. Spalte stehen !

```
$t3  = <<<EOFT1
Beispiel eines Strings über mehrere
Script-Zeilen durch Gebrauch
der heredoc-Syntax.
EOFT1;
```

PHP- Sprachreferenz - Kontrollstrukturen

- Bedingungen if () und switch (wie in C)

```
if ($a > $b) {  print "a ist > b"; } else
{ print "a NICHT > b !";  }
```

```
switch ($i) { case 0: echo "i ist 0";
break;
case 1:  echo "i ist gleich 1";  break;
// ...
```

- While (Bedingung) Anw.

```
$i = 1; while ($i <= 10) { echo $i++;  }
```

- Do ... While (Bedingung)

```
$i = 1; do { echo $i++;  }
while ($i <= 10)
```

- Zählschleife (analog C)

```
for ($i = 1; $i <= 10; $i++)
 { echo $i;  }
```

- For each – Schleife zum Durchlaufen von Arrays

```
$arr = array("eins", "zwei", "drei");
foreach ($arr as $value) {
    echo "Wert:  $value<br />\n";
```

- PHP erlaubt eine alternative Syntax mit : if (...) : endif;

Weitere Datentypen

- Arrays : - sind eigentlich Listen mit Zugriff über beliebige ID´s
- Indexed Arrays (analog wie C-Array)

```
$a1 = array( 1, 1, 1, 1, 1, 8=>1, 4=>1, 19, 3=>13);
$months  = array(1 => 'Januar', 'Februar', 'März');
$a1[2] = 25;  //  Zugriff auf einzelne Elemente
print_r($ months);// gibt alle Elemente aus
```

Iteration über Indexed Arrays

-
```
$arrlength = count($ months);
for($x = 0; $x < $arrlength; $x++) {
        echo $ months[$x];   echo "<br>";    }
```

PHP- Sprachreferenz - Datentypen III

- Arrays : - sind eigentlich Listen mit Zugriff über beliebige ID´s
- Assoziative Arrays : definieren einen Key (kann auch Text sein und zugeh. Inhalt)

```
$persons = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
$persons["Ben"] = 25;     //  Zugriff auf einzelne Elemente
print_r($erstesquartal);  // gibt alle Elemente aus
```

Iteration über Assoziative Arrays

- **foreach**($persons as **$x_key => $x_value**) {
  echo "Key=" . $x_key . ", Value=" . $x_value; echo "
"; }

- Sortieren von Arrays mit **sort**($a1) oder **ksort**($persons)
  (und weiteren 10 Sortieroptionen (vorwärts/ rückw. / nach Key)

PHP- Sprachreferenz - Superglobale Arrays

- Viele Daten werden in PHP auf superglobalen Arrays abgelegt (d.h. der Zugriff ist IMMER und VOB ÜBERALL aus möglich):

$GLOBALS - universelle Ablage (auch eigener) Var.

$_SERVER - Serverinfo (Path / IP –Adr. usw.)

$_REQUEST - Daten zum aktuellen Request

$_POST - Daten einer Postanfrage

$_GET - Daten einer Get-Anfrage

$_FILES - Liste der geuploadeten Files

$_ENV - übergebene Befehlsargumente an Skript

$_COOKIE - Cookieverwaltung (siehe PHP VL2)

$_SESSION - Sessionverwaltung (siehe PHP VL2)

PHP- Grundlegende Sprachreferenz – Variablen Sichtbarkeit

- In PHP wird unterschieden zwischen
  - Globalen Variablen, die im gesamten Skript bekannt sind und
  - Lokalen Variablen, die nur in der aktuellen Funktion bekannt sind.
- Beispiel: Lokale Variable

```
function test_lokal($zahl) {
    $zweitezahl = 10;  // diese Varialble ist nur in Fkt. bekannt
    return $zahl * $zweitezahl;
}
```

- Beispiel: Globale Variable

```
function test_global($zahl) {
    global $zweitezahl;   // Zugriff auf globale Variable
    return $zahl * $zweitezahl;
}

$zweitezahl = 10; // globale Variable definieren
echo test_global(10);
```

- Objekte (entsprechen bekannter OOP mit Daten + Methoden)
  - PHP_Klassendefintion mit entsprechenden Sichtbarkeitsregeln (public / private / protected)

```
class Person
{  private var $name;  // Deklaration der Daten
    public function Person()     // Klassenmethode
    { $this->name = $GLOBALS['firstname']; } }
$p1  = new Person();  // Instanziierung
 ?>
```

  - spezielle Methoden (in PHP als Magic functions bezeichnet)

```
__CLASS__ - liefert Namen der Klasse
public function __construct($initvalue) //  Constructor
    { echo 'Class '", __CLASS__, '" initiated! />';  }
public function __destruct    () { } //  Constructor
public function __toString()  // zur Stringumwandlung
    { return $this->$name;   }
  echo   $p1;  // mit  definierter ___toString-Temode geht dies
```

PHP- Sprachreferenz - OO – Vererbung

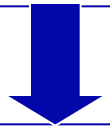
- Vererbung erlaubt Aufbau von Klassenhierarchien
  - mit Erweiterung um neue Daten und Methoden
  - mit Überschreiben bestehender Methoden

```
 //  Vererbung
class Student extends Person
{ private  var  StudentID;
   public   getStudentID() { return this->StudentID; }

   // Überschreiben der bestehenden Methode unter
 //  Einbeziehung der „alten“ Methodenm mit  parent::
 public function Person()     // Klassenmethode
    {   parent::__Person();
        $this->StudentID = $GLOBALS['StudentID']; } }
}
```

PHP- Verarbeitung von GET/Post-Daten

- Standardformular sendet Request an PHP-Skript

```
<form action="action.php"
method="POST"> Ihr Name: <input
type="text" name="name" /> Ihr Alter: <input
type="text" name="alter" /> <input
type="submit"> </form>
```



- PHP-Parser ersetzt in Skript die $_POST[“...“] – Tags durch die Inhalte aus dem Request

```
Hallo <?php echo $_POST["name"]; ?>.
Sie sind <?php echo $_POST["alter"];
?> Jahre alt.
```

- HTML-Ergebnis

```
Hallo Tom. Sie sind 25 Jahre alt.
```

Anforderungen an die Formularauswertung

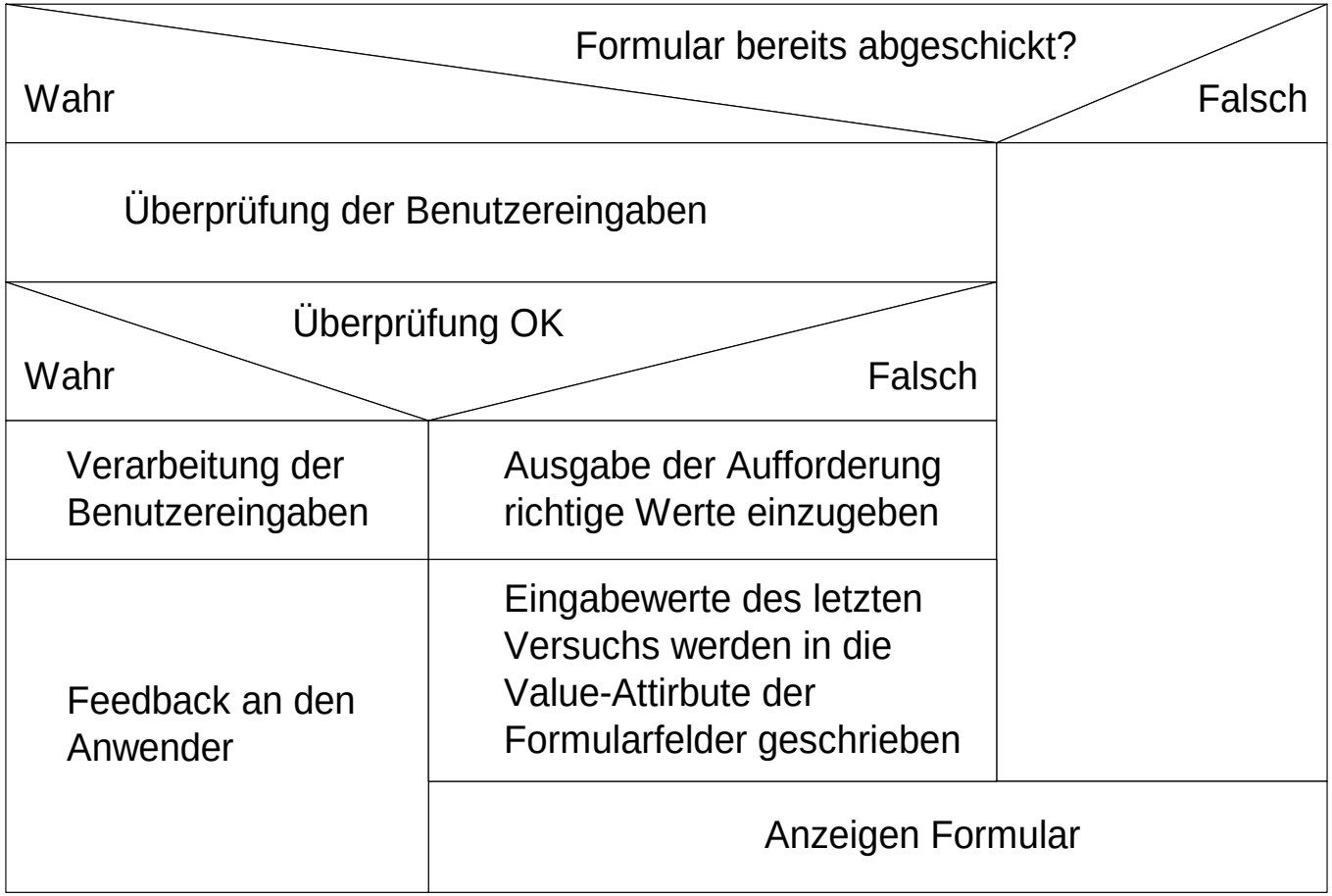
1. Vollständigkeit
   - Alle notwendigen Formularfelder sollten ausgefüllt werden.
   - Falls nicht
     - sollte der Benutzer einen Hinweis auf fehlende Felder erhalten und
     - es sollte keine Neueingabe bereits eingegebener Daten erforderlich sein.

2. Plausibilität
   - Alle Daten sind auf Plausibilität zu prüfen (eMail-Adresse? Kreditkartennr.?).

3. Exkurs: Sicherheit
   - Nochmals: Eine client-seitige Prüfung reicht nicht aus, da JavaScript deaktiviert werden kann.
   - Es sind Angriffe auf den Server mittels Formularinhalten möglich:
     - Über Zeichen wie ‘‘ oder ; kann versucht werden Befehle beispielsweise in der Datenbank auszuführen.
     - Der Server bzw. die Datenbank kann mit Megabyte großen Formularinhalten überschwemmt werden. (Denial-of-Service-Angriff)
   - Fazit: Bei professionellen Anwendungen sind Eingaben zusätzlich auf Größe und sicherheitsrelevante Zeichen zu prüfen.

Professionelle Auswertung von Formularen

- Das Formular-Skript ruft sich selber auf (Affenformular-Technik).
- Darstellung der Anwendungslogik als Struktorgramm:



Code: Affenformular (Anfang)

```
<html><head><title>Formulardemo</title></head><body>
<?  // Pruefen ob Formular abgeschickt wurde oder nicht
 if (isset($abgeschickt)) {
     // Validierung: Pruefen ob das Feld ausgefuellt wurde
     $fehlermeldung = "";
     if ($name == "") { $fehlermeldung = $fehlermeldung . "Name"; }
     // Pruefen ob ein Fehler gefunden wurde
     if ($fehlermeldung == "") {
          // Verarbeiten der Formulareingaben z.B. in Datenbank schreiben
          // Feedback an den Benutzer
          ?>
          <p>Vielen Dank!</p>
          <?
     } else {
          // Ausgabe der Fehlermeldung
          ?>
          <p>Bitte f&uuml;llen Sie folgende Felder aus: <? echo $fehlermeldung ?></p>
          <?
     } // Ende: Fehler oder nicht
} // Ende: Abgeschickt oder nicht
```

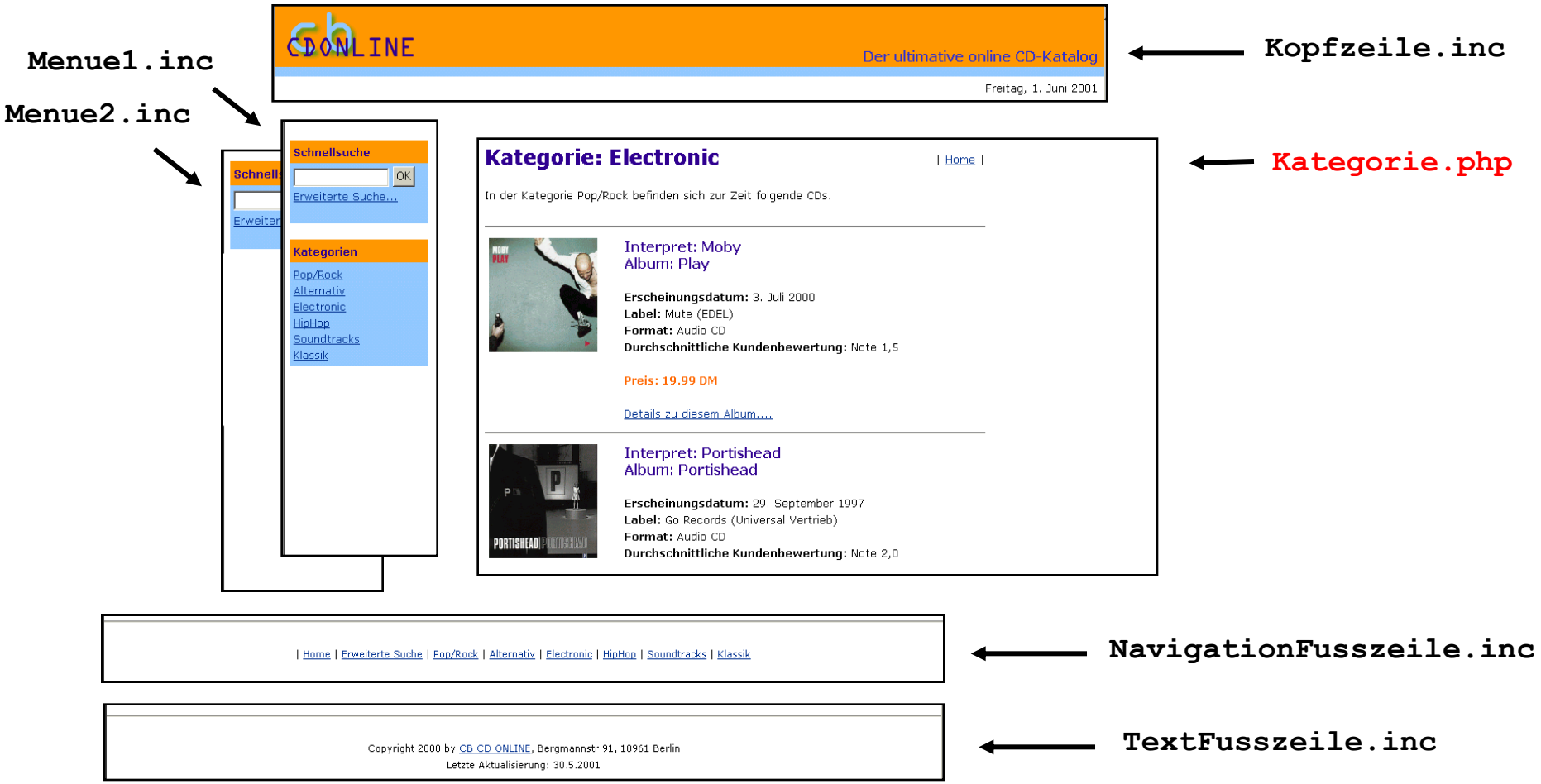
Code: Affenformular (Fortsetzung)

```
// Formular anzeigen, falls :
// 1. Der Benutzer zum ersten mal auf die Seite kommt oder
// 2. das Formular schon abgeschickt wurde und die Prüfung einen
Fehler ergab
if (!isset($abgeschickt) or ($fehlermeldung !="")) {
?>
  <h1>Bitte geben Sie ihren Namen ein:</h1>
  <form name="suche" method="post"
action="rezensionschreiben_kurz.php">
  <input type="text" name="name" value='<? echo $name ?>'>
  <input type="submit" name="abgeschickt" value="Submit">
  </form>
<? } ?>
</body>
</html>
```

Einbau eventuell schon vorhandener Daten

Seitengenerierung mittels Require-Befehlen

Die Wartung von Webapplikationen lässt sich erleichtern, indem wiederkehrende Layout-Elemente zentral in eigenen Dateien gespeichert werden, die anschließend in eine Vielzahl von Seiten integriert werden.



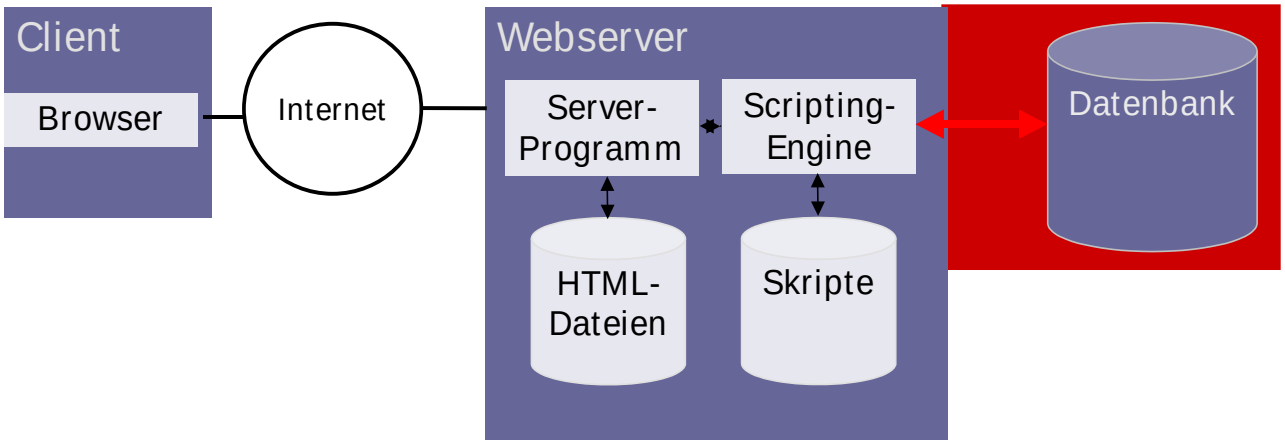
Zusammenführung der Layoutelemente

Die Layout-Elemente werden anschließend mittels Require-Befehlen wieder zusammengesetzt.

index.php

```
<?
// Variablen die in den Includes verwendet werden festlegen
$Seitentitel = "CB CD-Shop – Kategorie: Electronic";

// Einbinden zentraler Funktionsbibliotheken (z.B. Formularprüffunktionen)
require(".\_includes\PhpScript\SeitenFunktionen.inc");

// Einbinden der Layoutbausteine Kopfzeile und Menue links
require(".\_includes\PageElements\Header.inc");
require(".\_includes\PageElements\KopfZeile.inc");
require(".\_includes\PageElements\MenuSucheKategorien.inc");

?>

       <h1>Kategorie: Electronic</h1>
        <p>Hier steht der eigentliche Inhalt der Seite.</p>

<?
// Einbinden der Fusszeilen
require(".\_includes\PageElements\NavigationFusszeile.inc");
require(".\_includes\PageElements\TextFusszeile.inc");

?>
```

Datenbankanbindung mit PHP

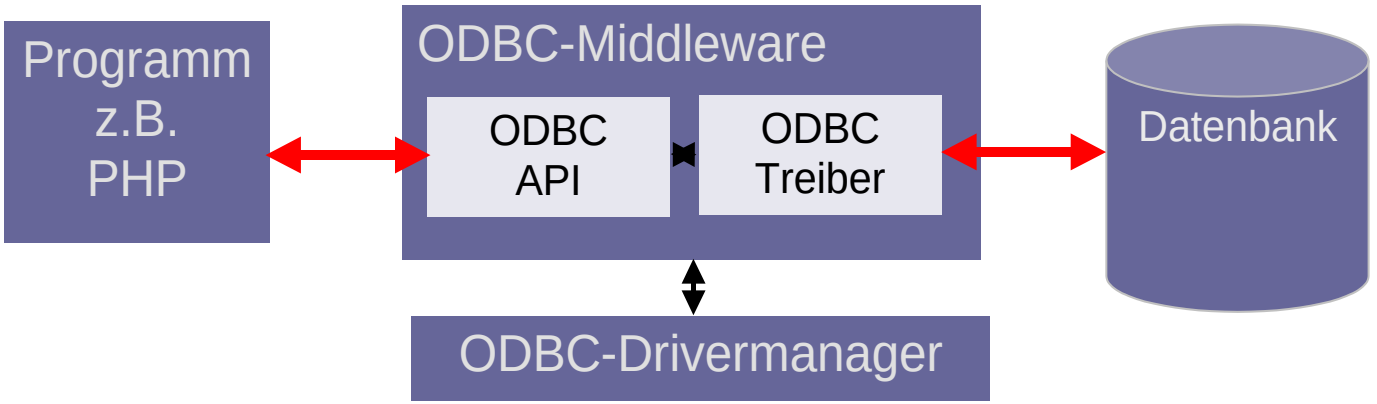


- PHP verfügt über datenbankspezifische Schnittstellen zu:
  - MySQL, MS SQL-Server, Oracle, Sybase, dBase, Informix
  - InterBase, mSQL, PostgreSQL, DBM
- Datenbankzugriffe in PHP erfolgen über:
  - datenbankspezifische Schnittstellen (hohe Performance)
  - Connectivity-Middleware wie ODBC (geringere Performance)

- **Zwei Optionen für native DB-operationen in PHP**
  - Alte (deprecated) MySql-operation (bis 2012)
  - Neue improved MySQL**i** – syntax mit object oriented syntax

Open Database Connectivity (ODBC)

- ODBC ist eine Middleware-Komponente, die den Zugriff auf unterschiedliche Datenbanken über eine einheitliche Schnittstelle (API) erlaubt.



- Der ODBC-Standard wurde von Microsoft eingeführt.
- Es gibt ODBC-Treiber für nahezu jede Datenbank.
- Vorteil: Datenbanken sind austauschbar, ohne die Applikation umschreiben zu müssen (falls Standard-SQL verwendet, ANSI 89).
- Nachteil: Performance geringer als bei datenbank-spezifischen APIs.

Beispiel 1: Staus aus der Datenbank holen

- Relationstyp stau_meldungen in der Datenbank staudb.sql:

  stau_meldungen (id, datum, aktiv, autobahn, info)

```
// MySQL-Verbindung zur Datenbank aufbauen
$conn = new mysqli("server","user","pwd", „staudb") or
                die(mysql_error());


// SQL Query-String definieren
$query_String = "SELECT * FROM stau_meldungen
                WHERE aktiv = '1'
                 ORDER BY autobahn, datum desc;";


// Abfrage ausfuehren
$result = $conn->query ($query_String);
```

Beispiel 1: Ausgabe der Informationen

- Über eine While-Schleife werden alle Tupel des Abfrageergebnisses durchlaufen und
- die Inhalte in den HTML-Code der Seite geschrieben.

```
<?
if ($result->num_rows > 0) {
 // Schleife ueber alle Tupel bei Ergebn.
while($Row = $result->fetch_assoc ())
{ ?> <p>
   Autobahn: <? echo $Row[autobahn] ?><br>
   Meldung: <? echo $Row[info] ?><br>
   Datum: <? echo $Row[datum] ?>
   </p>
   <hr size="2">
<?}
} ?>
```

Beispiel 2: Autobahn-Auswahl

```
// Informationen zu den Autobahnen aus DB holen
$Query_String = "SELECT DISTINCT autobahn
                 FROM stau_meldungen
                 WHERE aktiv = '1'
                 ORDER BY autobahn;";
$Result_Menue = $conn->query ($Query_String);


// Ausgabe der Autobahninformationen
while(($Row = $result->fetch_assoc($Result_Menue )) {
     echo "<p><a href='index.php?autobahn="
          . $Row[3]
          . "'>"
          . $Row["autobahn"]
          . "</a></p>";
};
```

- Ergebnis:

```
<p><a href='index.php?autobahn=A01'>A01</a></p>
<p><a href='index.php?autobahn=A07'>A07</a></p>
....
```

Situationsspezifische SQL-Befehle

- Die meisten Datenbankabfragen in Webapplikationen sind situationsspezifisch.
- Beispiel: Nur die Staus einer vom Nutzer gewählten Autobahn aus der Datenbank holen
- Vorgehen:
  - Dem Skript wird über einen Parameter das Auswahlkriterium übergeben
  - Ein passendes SQL-Kommando wird zusammengebaut und ausgeführt

```
// SQL-Query-String situationsspezifisch zusammenbauen

$autobahn = $_GET["autobahn"];
$Query_String = "SELECT * FROM stau_meldungen
                 WHERE aktiv = '1'
                 AND autobahn ='" . $autobahn . "'
                 ORDER BY datum;";
// SQL-Kommando ausfuehren
$Result_ID = $conn->query ($Query_String);
```

MySQL - Injection

- Eine SQL Injection nennt man das Einfügen von SQL Kommandos die an dieser Stelle nichts zu suchen haben:

```
// SQL-Query-String situationsspezifisch zusammenbauen
$Query_String = "SELECT * FROM stau_meldungen
                 WHERE aktiv = '1'
                 AND autobahn ='".$_GET[$autobahn]."'
                 ORDER BY datum;";


// was passiert wenn man z.B im Eingabefeld nun:
 A13'; DROP staudb;-- angibt.


Es wird nun diese Query erzeugt:

SELECT * FROM stau_meldungen WHERE aktiv = '1'
                 AND autobahn ='A13'; DROP DATABASE staudb;--';
```

- Injection vermeiden:
  - if (!get_magic_quotes_gpc()) {
    $autobahn = addslashes($autobahn); }

Formulareingaben in die Datenbank schreiben

- Nach ihrer Validierung werden die Formulareingaben über ein „INSERT INTO“Kommando in die Datenbank geschrieben:

```
// SQL-Kommando zur Schreiben der Daten zusammenbauen
$Query_String = "INSERT INTO stau_meldungen
                        (autobahn, info, aktiv, datum)
                 VALUES
                        ('" . $autobahn . "','" . $meldung .
                         "','1','" . $datum . "');";

// SQL-Kommando ausfuehren
$Result_ID = $conn->query ($Query_String);
```

- Hinweis:
  - Beachten: Anführungszeichen im SQL-Kommando sind durch Hochkommata ‘ zu ersetzen.

PHP- Datenbankanbindung -Komplettbeispiel

```
<?php /* Verbindung aufbauen, auswählen einer Datenbank */
$link = new mysqli("localhost", "root", "##pw##", "test")
    or die("Keine Verbindung möglich: " . mysql_error());
echo "<P>Verbindung zum Datenbankserver erfolgreich";
$query = "SELECT * FROM test";
$result =  $link-> query  ($query) or die("Anfrage fehlgeschlagen: " . mysql_error());
/* Ausgabe der Ergebnisse in HTML */
if ($result->num_rows > 0) {
echo "<table>\n";
while ($line = $result->fetch_assoc())
  {  echo "\t<tr>\n";
    foreach ($line as $col_value) {    // Universal-Ausgabe für beliebige Tabelle !
        echo "\t\t<td>$col_value</td>\n";
    }     echo "\t</tr>\n";
  } echo "</table>\n";
}
/* Freigeben des Resultsets */         $link-close();
```

PHP- Funktionsbibliotheken

- Neben der Open-Source – Verfügbarkeit von PHP sind die historisch gewachsenen Funktionsbibliotheken einer der Hauptgründe für den Erfolg von PHP

Einige der interessantesten Bibliotheken sind :

- Bzip2 - direkter Zugriff auf ZIP-Dateien
- ClibPDF - Erzeugung und Manipulation von PDF-Dateien
- File-Zugriffsfunktionen in großer Analogie zu C
- FTP-Funktionen zum Remote-Abruf von Dateien
- Netzwerk und Socket-Funktionen für Zugriff auf Remote-Server oder eigene Protokolle
- Grafikfunktionen zur Erzeugung beliebiger Grafiken im GIF / JPEG und anderen Formaten durch PHP-Zeichenfunktionen
- IMAP, POP3 und NNTP – Emailfunktionen
- XML –Funktionen (DOM / SimpleXML)
- Verschiedenste Datenbankzugänge : MYSQL / MS-SQL / Oracle / Informix

Die große Anzahl von Bibliotheken wird durch spezielle Projekte verwaltet:

- PEAR - "PHP Extension and Application Repository" mit code distribution mech. / style guides für Bibl. / PHP Foundation Classes (PFC)
- PECL - PHP Extension Community Library (PECL) für C-Erweiterungen

PHP- im Vergleich

PHP im Vergleich zu anderen Skriptsprachen und Optionen :

- **ASP .NET :** sind an MS-Technologien (IIS, Windows) gebunden, bei Vorhandensein von VB-Code eventuell noch sinnvoll, PHP stabiler und flexibler auf anderen Plattformen
- **Cold FUSION** : langsamer und weniger flexibel als PHP, wird nicht auf allen Plattformen unterstützt
- **PERL :** relativ komplex, da für allgemeine Programmieraufgaben entwickelt, PHP entstand ursprünglich als eine Sammlung von PERL-Skripten, Schwerpunkt auf größere Textmanipulationen und Batch-Verarbeitung in UNIX-Environments
- **JAVA :** genauso portabel, Java benötigt eine längere Turn-around-Zeit bei der Entwicklung von Applikationen, kann infolge der Vorab-Compilierung und Java Bean jedoch komplexere und besser strukturierte Programme erzeugen
- **JavaScript** könnte in Kombination mit serverseitigen JavaScript-Umgebungen wie NodeJS zu einer Alternative zu PHP werden …