

Javascript und php
Client- und serverseitige Anwendungsentwicklung für das
WWW

Bernd Blümel

Version: 26. März 2002

Inhaltsverzeichnis

1	Einleitung	2
2	Erste Beispiele	6
3	Einfügen von JavaScript und php in html-Dateien	24
3.1	Einfügen von JavaScript in html-Dateien	24
3.1.1	<script> und </script>	24
3.1.2	Das src-Attribut des <script>-Tags	24
3.2	Einfügen von php in html-Dateien	26
4	Grundsätzliches	29
4.1	Kommentare	29
4.2	Variablen	30
4.3	Ausdrücke	36
4.4	Operatoren	36
4.4.1	Der Zuweisungsoperator	36
4.4.2	Arithmetische Operatoren	36
4.4.3	Vergleichsoperatoren	40
4.4.4	Logische Operatoren	42
4.4.5	Der ternäre Operator	43
4.5	Konstante	44
5	Beispiele	48
5.1	Euro-Dollar-Umrechnung	48
5.2	Volatilität	50
5.3	Das Raketenbeispiel	54
6	Algorithmen und Kontrollstrukturen (Steuerungsstrukturen)	63
6.1	Algorithmen	63
6.2	Die if- Anweisung (Ein- und Zweiseitige Auswahl)	64
6.2.1	Motivation	64
6.2.2	Syntax	67
6.2.3	php und Get und Post	72
6.2.4	Beispiele	75
6.3	Der Switch (Mehrfachauswahl)	88
6.3.1	Motivation	88
6.3.2	Syntax	90
6.3.3	Beispiele	94

7 Funktionen	105
7.1 Motivation	105
7.2 Syntax	111
7.3 Beispiele	115
7.3.1 Euro-Dollar-Umrechnung (fortgesetzt)	115
7.3.2 Volatilitäten (fortgesetzt)	124
7.3.3 Raketenbeispiel (fortgesetzt)	124
7.4 Referenz- und Wertparameter	132
8 Klassen und Objekte	141
8.1 Motivation	141
8.2 Beispiele in php	144
8.2.1 Euro-Dollar-Umrechnung objektorientiert	144
8.2.2 Volatilitäten (objektorientiert)	148
8.2.3 Raketenbeispiel (objektorientiert) und die Nutzung von Konstruktoren	148
8.3 Klassenmethoden	157
8.4 Klassen und Objekte in JavaScript	162
9 JavaScript-Objekte und das Event-Modell	167
9.1 Das Window-Objekt und die Objekthierarchie	167
9.2 Das document-Objekt	172
9.3 Event-Handler	174
9.4 Das link-Objekt	176
9.5 Das Form-Objekt	181
9.5.1 Eingabekontrollen	185
10 Das große Ganze: Teil 1	202
10.1 Aufgabenstellung und Analyse	202
10.2 Einige Bemerkungen zur Nutzung von Datenbanksystemen in php	204
10.3 Erste Lösung, Stringbehandlung	207
10.4 Verbesserte Lösung: Sessions und ein wenig Anwendungsarchitektur	223
11 Lösungen	247

Kapitel 10

Das große Ganze: Teil 1

In diesem Kapitel wollen wir das bisher Gelernte am Euro-Dollar-Umrechnungsbeispiel vertiefen. Darüber hinaus werden Sie einige neue Sachen, wie die Behandlung von Strings oder Datenbankzugriff aus php, lernen. Als Datenbank setzen wir MySQL ein. MySQL ist ein unter der GPL stehendes Datenbanksystem und gerade bei Internet-Anwendungen in Verbindung mit php weit verbreitet. Als Werkzeug zur Administration der Datenbank benutzen wir das ebenfalls unter der GPL stehende phpmyadmin.

10.1 Aufgabenstellung und Analyse

Wir setzen das Euro-Dollar-Umrechnungsbeispiel fort: Es soll ernst werden. Die Euro-Dollar-Umrechnung soll in den Internet-Auftritt der Schwanen Gesellschaft GmbH¹ eingebunden werden. Auf der Produktseite soll, wie in Kapitel 7 erstmals behandelt, der Preis unserer Produkte in Euro und Dollar ausgewiesen werden. Um die Anziehungskraft der Produktseite zu erhöhen und damit für Internet-Benutzer einen zusätzlichen Anreiz zu schaffen, diese Seite zu besuchen, soll im “oberen Bereich”² der Seite ein Euro-Dollar-Umrechner angeboten werden. Die Anwendung soll leicht erweiterbar sein. Das bedeutet, wenn unser Marketing auf die Idee kommt, die Preise auch in Schweizer Franken oder in japanischen Yen auszuweisen, soll auch das möglich sein. Der Euro-Dollar-Umrechner muss dann selbstverständlich zu einen Euro-Dollar-Franken- bzw. Euro-Dollar-Yen-Umrechner erweitert werden. Der gegenwärtige Dollarkurs soll von Sachbearbeitern der Produktpflege in das System eingegeben werden. Dies soll jederzeit möglich sein. Der eingegebene Kurs liegt dann allen Berechnungen zu Grunde. Die Sachbearbeiter der Produktpflege sind allerdings alle Computer-Laien. Von ihnen kann nicht erwartet werden, Quellcode zu editieren. Für sie muss daher eine einfache Möglichkeit zur Kurspflege vorgesehen werden. Da wir zur Zeit nur drei Produkte über das Internet anbieten wollen, ist nicht daran gedacht, die Produktpreise aus der vorhandenen Produktdatenbank einzulesen³. Für sämtliche Eingaben sollen sowohl Komma als auch Punkt als Dezimaltrenner vorgesehen werden. Die Ausgabe erfolgt immer mit zwei Nachkommastellen und dem Komma als Dezimaltrenner.

Zunächst schreiben wir die Anforderungen (Anwendungsfälle) etwas strukturierter auf:

1. Der Dollarkurs muss über eine einfach zu bedienende Oberfläche eingegeben werden. Als Dezimaltrenner sind Komma und Punkt erlaubt. Die Eingaben werden von Sachbearbeitern der Produktpflege vorgenommen.

¹Wegen des Logos, werden Sie später sehen!

²Was auch immer damit gemeint sein soll!

³Das machen wir in Kapitel ??.

2. Dieser Dollarkurs liegt allen Ausgaben zu Grunde.
3. Die Euro-Preise unserer Produkte werden automatisch in Dollar umgerechnet. Diese Preise werden zusätzlich auf der Produktseite ausgegeben.
4. In die Produktseite wird ein Euro-Dollar-Umrechner integriert. Bei den Eingaben sind Punkt oder Komma als Dezimaltrenner erlaubt, alle Ausgaben erfolgen mit zwei Nachkommastellen und dem Komma als Dezimaltrenner.
5. Die Anwendung muss auf weitere Währungen erweiterbar sein.

Als Nächstes erstellen wir ein Use-Case-Diagramm. Use-Case-Diagramme zeigen die Anwendungsfälle (engl. Use Case) und die Anwender (Akteure) die mit den Anwendungsfällen interagieren⁴:

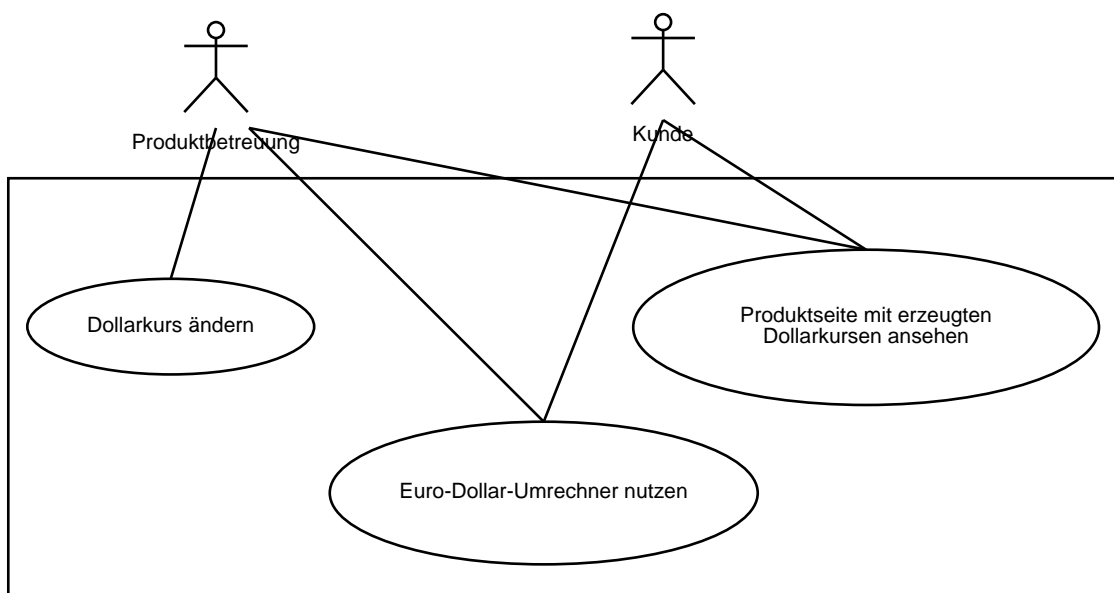


Abbildung 10.1: Das Use-Case-Diagramm zur Euro-Dollar-Problemstellung

Abb. 10.19 sollte selbsterklärend sein.

Aus der Aufgabenstellung leiten wir zunächst folgende Entscheidungen ab:

- Wir werden zwei html-Seiten benötigen:
 - Eine für die Produktbetreuung zur Pflege des Dollarkurses.
 - Eine für die Kunden, mit Produktpreisen und dem Rechner.
- Wir werden den Dollarkurs in einer Datenbank vorhalten. Dazu werden wir eine Tabelle Währung anlegen und als zunächst einzigen Eintrag den Dollarkurs vornehmen. Dies ist flexibel für mögliche spätere Erweiterungen (weitere Währungen). Außerdem werden wir irgendwann auch die Produktpreise aus der Datenbank einlesen, und müssen dann sowieso mit der Datenbank arbeiten. Alternative wäre, den Kurs in einer Datei im Dateisystem des www-Servers abzuspeichern und das ist genauso aufwändig, wie die Nutzung einer Datenbank.

⁴Das Use-Case-Diagramm ist ebenfalls Bestandteil der bereits in Kapitel 8 erwähnten UML.

10.2 Einige Bemerkungen zur Nutzung von Datenbanksystemen in php

Datenbanksysteme sind Serversysteme, die ähnlich wie www-Server auf clientseitige Anforderungen warten. Genau wie die Browser von www-Servern html-Seiten anfordern, können unsere php-Programme von Datenbanksystemen Daten anfordern. Genau wie bei www-Servern muss man dazu den Namen des Rechners, auf dem die Datenbanksoftware läuft (das kann natürlich auch der www-Server selber sein) wissen. Anders als bei www-Server-Software muss man sich bei Datenbanken mit Benutzernamen und Passwort legitimieren.

Wenn man das tut, erhält man eine bidirektionale Verbindung zur Datenbank. Über diese Verbindung kann man Abfragen und Kommandos an die Datenbank schicken. Die Datenbank antwortet über diese Verbindung. Dies ist dann entweder das Ergebnis der Abfrage oder eine Bestätigung, das Kommando durchgeführt zu haben⁵.

Die Abfragen oder Kommandos müssen in SQL (Structured Query Language)⁶ formuliert sein.

Ich zeige die grundsätzliche Vorgehensweise an einem Zugriff von php auf ein MySQL-Datenbanksystem. Der Verbindungsaufbau zu einer Datenbank erfolgt mit der php-Funktion `mysql_pconnect`:

```
$link=mysql_pconnect("localhost", "bb", "meinPW");
```

Wie Sie sehen, erwartet `mysql_pconnect` drei Übergabeparameter:

- Den Namen des Rechners: Dies ist hier `localhost`. Hiermit wird der Rechner, auf dem das php-Programm selber läuft, angesprochen. Alternativ kann hier der Name eines anderen Rechners stehen. Der Verbindungsaufbau über das Netz erfolgt automatisch.
- Den Namen des Benutzers: Dies ist hier `bb`. Hier kann der Name eines jeden beim Datenbanksystem registrierten Nutzers stehen.
- Das Passwort des Benutzers: Hier `meinPW`.

Im Erfolgsfall enthält die Variable `$link` nun die bidirektionale Verbindung zur Datenbank. `$link` wird von nun an als Übergabeparameter aller php-Funktionen, die mit der Datenbank arbeiten, benötigt. Abb. 10.2 gibt übrigens einen Überblick über die MySQL-Funktionen von php.

Als nächstes wird die Datenbank ausgewählt. Ein Datenbanksystem kann beliebig viele⁷ Datenbanken enthalten. Abb. 10.3 zeigt die Datenbanken eines auf einem Server des eBusiness-Labors installierten Datenbank-Systems.

Die Auswahl erfolgt mittels des SQL-Kommandos `use`.

```
$query="use intranet";  
mysql_query($query, $link);
```

In der ersten Zeile obigen php-Codes wird das SQL-Kommando auf der Variablen `$query` abgespeichert. Hier wird die Datenbank `intranet` ausgewählt. Dies ist übrigens die unserem Informationssystem zu Grundliegende Datenbank. Mit dem Kommando `mysql_query` wird das SQL-Kommando dann an die Datenbank geschickt und dort ausgeführt. Wie wir sehen, erwartet `mysql_query` zwei Übergabeparameter:

- `$query`: Das SQL-Kommando, das vom Datenbank-System ausgeführt werden soll.

⁵Oder eben auch nicht, falls etwas schiefgegangen ist.

⁶SQL ist nicht Thema dieser Ausarbeitung. Unterlagen zu SQL finden Sie in den Datenbank-Unterlagen meines Kollegen Johannes, die über meine Internet-Seite verfügbar sind.

⁷Wird natürlich durch Plattenspeicher und Leistungsfähigkeit der Server-Hardware begrenzt.

XXXIII. MySQL functions

These functions allow you to access MySQL database servers.

More information about MySQL can be found at <http://www.mysql.com/>.

Table of Contents

[mysql_affected_rows](#) ? Get number of affected rows in previous MySQL operation
[mysql_change_user](#) ? Change logged in user on active connection
[mysql_close](#) ? close MySQL connection
[mysql_connect](#) ? Open a connection to a MySQL Server
[mysql_create_db](#) ? Create a MySQL database
[mysql_data_seek](#) ? Move internal result pointer
[mysql_db_query](#) ? Send an MySQL query to MySQL
[mysql_drop_db](#) ? Drop (delete) a MySQL database
[mysql_errno](#) ? Returns the number of the error message from previous MySQL operation
[mysql_error](#) ? Returns the text of the error message from previous MySQL operation
[mysql_fetch_array](#) ? Fetch a result row as an associative array
[mysql_fetch_field](#) ? Get column information from a result and return as an object
[mysql_fetch_lengths](#) ? Get the length of each output in a result
[mysql_fetch_object](#) ? Fetch a result row as an object
[mysql_fetch_row](#) ? Get a result row as an enumerated array
[mysql_field_name](#) ? Get the name of the specified field in a result
[mysql_field_seek](#) ? Set result pointer to a specified field offset
[mysql_field_table](#) ? Get name of the table the specified field is in
[mysql_field_type](#) ? Get the type of the specified field in a result
[mysql_field_flags](#) ? Get the flags associated with the specified field in a result
[mysql_field_len](#) ? Returns the length of the specified field
[mysql_free_result](#) ? Free result memory
[mysql_insert_id](#) ? Get the id generated from the previous INSERT operation
[mysql_list_fields](#) ? List MySQL result fields
[mysql_list_dbs](#) ? List databases available on on MySQL server
[mysql_list_tables](#) ? List tables in a MySQL database
[mysql_num_fields](#) ? Get number of fields in result
[mysql_num_rows](#) ? Get number of rows in result
[mysql_pconnect](#) ? Open a persistent connection to a MySQL Server
[mysql_query](#) ? Send an SQL query to MySQL
[mysql_result](#) ? Get result data
[mysql_select_db](#) ? Select a MySQL database
[mysql_tablename](#) ? Get table name of field

Abbildung 10.2: Die mysql-Funktionen von php

- \$link: Die bidirektionale Verbindung zur Datenbank.

Nun ist die Datenbank ausgewählt. Wir können weitere, beliebige SQL-Befehle an die Datenbank schicken:

```
$query="select * from room";
$result=mysql_query($query, $link);
```

Obiges SQL-Kommando selektiert alle Datensätze der Tabelle room aus der Datenbank intranet. mysql_query schickt die Abfrage an die Datenbank. Das Ergebnis steht auf der Variablen \$result. Beispiel 10.1 zeigt eine Ausgabe obigen SQL-Kommandos in einer direkten Schnittstelle des Datenbanksystems.

Beispiel 10.1 Ausgabe eines SQL-Kommandos

```
mysql> select * from room;
+-----+-----+-----+
| room_nr | name      | group_of_seats |
+-----+-----+-----+
|      1 | 01-36    |                2 |
|      2 | 01-37    |                2 |
|      3 | 0-38     |                5 |
|      4 | 0-39     |                5 |
|      5 | 1-39     |                3 |
```


erzeugte Ergebnismenge. Die nächsten beiden Übergabeparameter sind Zeile und Spalte des gewünschten Wertes der Ergebnismenge, wobei, wie häufig in der Informatik, von Null hochgezählt wird. \$ersterDatensatzErstesFeld hat in unserem Beispiel den Wert 1 (vgl. Beispiel 10.1). Um den Wert "1-39" zu erhalten, müsste das Kommando

```
$ersterDatensatzErstesFeld=mysql_result($result,4,1);
```

abgesetzt werden (vgl. Beispiel 10.1).

10.3 Erste Lösung, Stringbehandlung

Die Datenbank

Zunächst müssen wir das Datenbankmodell entwickeln. Das machen wir mit Entity Relationship-Modellierung (ERM)⁸. Hier ist dies ganz einfach: Es gibt nur eine Tabelle, die nennen wir "waehrung". Die Attribute sind: "waehrung_nr", "name", "kurs". Abb. 10.4 zeigt das ERM.

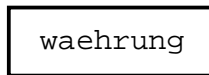


Abbildung 10.4: Das ERM zur Euro-Dollar-Problemstellung

Zunächst müssen wir eine Datenbank und dann die Tabelle im Datenbanksystem anlegen. Dies können wir entweder direkt oder mit phpmyadmin tun.

Das Ergebnis ist in Abb. 10.5 zu sehen.

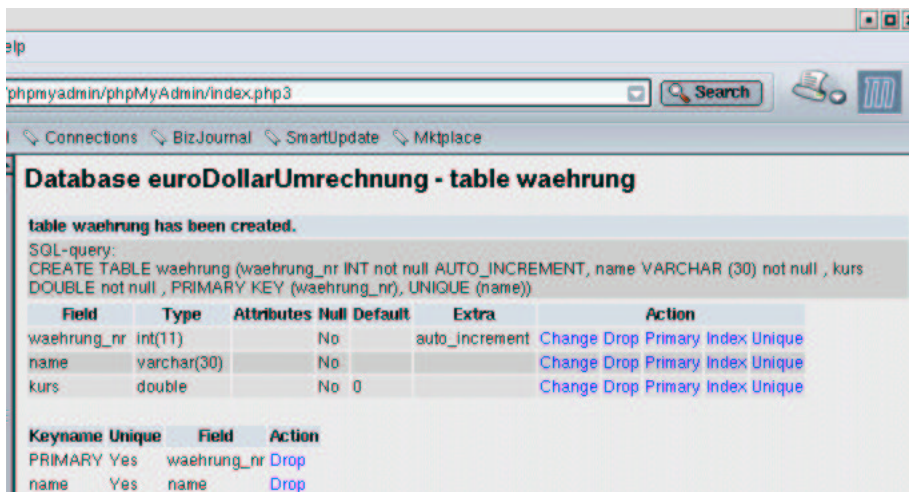


Abbildung 10.5: Die Tabelle Währung

"waehrung_nr" ist der selbst hoch zählende Primärschlüssel. "name" ist auf unique gesetzt, damit ist garantiert, dass der Name einer Währung innerhalb der Tabelle eindeutig ist. Zum Abschluss unserer Datenbankaktivitäten tragen wir einen Datensatz in die Tabelle ein. Dies zeigt Abb. 10.6.

Beachten Sie, dass Sie für "waehrung_nr" keinen Wert angeben müssen. Weil "waehrung_nr" auf "autoincrement" steht, wird der Wert von "waehrung_nr" von der Datenbank vergeben.

⁸Ebenfalls in den Unterlagen von Herrn Johannes zu finden.



Abbildung 10.6: Der Eintrag eines Datensatzes

Die Seite zur Kurspflege

Nun folgt die Entwicklung der Seite zur Pflege des Dollarkurses. Funktion und Aussehen dieser Seite könnten wir, wie in Abb. 10.7 dargestellt⁹, programmieren.



Abbildung 10.7: Die Seite zur Kurspflege

Das Programm dazu sieht im ersten Ansatz so aus:

Beispiel 10.2 Die Kurspflege Anwendung

```
<!-- Programm zur Euro-Dollar Umrechnung
      Teil Das grosse Ganze
      Kurspflege
      Dateiname: grosseGanzel/kurspflege.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title> Euro-Dollar Umrechnung Kurspflege</title>
</head>
<body>
```

⁹Ich bin halt kein Designer!

```

<table border=2 align="center">
  <tr>
    <td>
      
    </td>
    <td>
      Die Schwanen Gesellschaft
    </td>
  </tr>
</table>
<h2 align="center">
  Kurspflege
</h2>
<hr>
<?php
// Wir pruefen ob die Anfrage ueber get oder post erfolgte
if($REQUEST_METHOD!="POST")
{
// erster Aufruf, das Formular muss praesentiert werden
  echo "<form name='kurspflege' action='$PHP_SELF' method='post'>";
?>
  <table border align="center">
    <tr>
      <td>
        Neuer Kurs
      </td>
      <td>
        <input type="text" name="kurs" size=12>
      </td>
    </tr>
    <tr>
      <td colspan="2" align="center">
        <input type="submit" name="Button1" value="Abschicken">
      </td>
    </tr>
  </table>
</form>
<?php
}
else
{
  // Verbindung zur Datenbank herstellen
  $link=mysql_pconnect("localhost", "root", "");
  // Datenbank euroDollarUmstellung einstellen
  $query="use euroDollarUmrechnung";
  mysql_query($query, $link);
  // update sql-Kommando
  $updateQuery ="update waehrung set kurs='$kurs' ";
  $updateQuery.="where name='Dollar'";
  mysql_query($updateQuery, $link);
  echo "Kurs ge&auml;ndert auf $kurs.";
}
?>

```

```
</body>
</html>
```

Neues beginnt hier im else-Teil. In der Zeile

```
$link=mysql_pconnect("localhost", "root", "");
```

wird eine Verbindung zum Datenbank-System aufgebaut. Der Server, auf dem das Datenbank-System implementiert ist, ist derjenige, auf dem auch das php-Script läuft (localhost). Der Benutzer, der angemeldet wird, heißt root. Er hat kein Passwort (""). Die Kennung der Verbindung ist auf der Variablen \$link abgespeichert.

```
$query="use euroDollarUmrechnung";
mysql_query($query, $link);
```

In der ersten obigen Zeile wird das SQL-Kommando zum Anmelden der Datenbank euroDollarUmrechnung der Variablen \$query zugewiesen. In der zweiten Zeile wird das Kommando an die Datenbank geschickt und dort ausgeführt.

```
$updateQuery = "update waehrung set kurs='$kurs' ";
$updateQuery.="where name='Dollar' ";
mysql_query($updateQuery, $link);
```

In den ersten beiden oben dargestellten Zeilen wird das SQL-Kommando zum Update der Datenbank (Änderung des Dollar-Kurses) zusammengestellt. Beachten Sie die Konstruktion:

```
$updateQuery.="where name='Dollar' ";
```

Dies ist einfach eine abkürzende Schreibweise für die Stringverkettung (.=). Dies hatten wir in Kapitel 4 besprochen. Variablen oder Zeichenketten, die einem Feld der Datenbank zugewiesen werden sollen, werden in Apostrophe (") eingeschlossen.

Zwei Schönheitsfehler hat die Anwendung zur Kurspflege noch:

- Die Benutzer können keine Kommas als Dezimaltrenner eingeben.
- Es wird nicht überprüft, ob der eingegebene Wert eine Zahl ist.

Eingabe-Überprüfungen realisieren wir, wie in Kapitel 9 dargestellt, in JavaScript. Dadurch ändert sich auch die Definition des Formulars in der php-Datei (vgl. Beispiel 9.16), da die Übertragung des Formulars ja jetzt durch JavaScript geschehen muss. Die Funktion zur Überprüfung, ob eine Eingabe eine Zahl ist, hatten wir bereits implementiert. Sie ist in Beispiel 9.14 dargestellt.

Nun müssen wir noch mit der Eingabe eines Kommas als Dezimaltrenner klarkommen. Aber auch das ist gar nicht so schwer. Wir untersuchen die Eingabe des Benutzers auf die Eingabe eines Kommas. Finden wir ein Komma, ersetzen wir es durch einen Punkt. Die Realisierung findet sich in Beispiel 10.3.

Beispiel 10.3 *Kommas in Punkte umwandeln*

```
// Funktionen zur Stringbearbeitung
//
// Dateiname: stringbearbeitung.js
function komma2punkt(str)
```

```
{
    str=str.replace(/,/g, ".");
    return str;
}
```

Strings in JavaScript sind hybride Variablen. Das bedeutet:

- Benutze ich einen String ganz normal als Variable, wird er von JavaScript auch so behandelt. Ich kann ihn mit anderen Strings verketteten, einem anderen String zuweisen, vergleichen und was der Dinge mehr sind, die man mit Variablen machen kann und die Sie bereits alle gelernt haben.
- Schließe ich aber an eine String-Variable einen Punkt an, denkt JavaScript, ich möchte den String gerne als Objekt behandeln¹⁰. JavaScript konvertiert den String dann in ein String-Objekt und ich kann alle Methoden der String-Klasse auf meine String-Variable anwenden¹¹. Benutze ich das String-Objekt, zu dem meine Variable ja geworden ist, wieder als normale Variable, macht JavaScript alles rückgängig und ich habe wieder meine normale Variable.

So erklärt sich also, dass ich in der Zeile

```
str=str.replace(/,/g, ".");
```

einen Methodennamen über einen Punkt mit einer String-Variablen verbinden kann. `replace` ist nun eine Methode der String-Klasse und ersetzt in dem String-Objekt, das diese Methode aufruft, den ersten Übergabeparameter durch den zweiten. Ein wenig gewöhnungsbedürftig ist die Syntax, wie die Übergabeparameter angegeben werden müssen. Der erste Übergabeparameter (die Zeichenkette, die ersetzt werden soll) wird in Schrägstriche (Slash, /) eingeschlossen. Steht hinter dem schließenden Schrägstrich ein `g`, so bedeutet dies, dass die zu ersetzende Zeichenkette jedesmal, wenn sie auftaucht, ersetzt wird. Fehlt das `g` hinter dem schließenden Schrägstrich, wird die Zeichenkette nur einmal ersetzt. Die Zeichenkette, die die erste ersetzen soll, wird, wie wir das gewohnt sind, in Anführungszeichen ("") eingeschlossen¹². Was die in Beispiel 10.3 dargestellte Funktion `komma2punkt()` also tut, ist, jedes Komma im ihr übergebenen String in einen Punkt umzuwandeln. Die Funktion `komma2punkt()` speichern wir im Verzeichnis `javascript` unterhalb des Verzeichnisses, in dem unsere `kurspflege.php` liegt.

Jetzt müssen wir also nur noch das JavaScript-Script schreiben, das unsere Funktionen `komma2punkt()` und `istKeineZahl()` aufruft.

Beispiel 10.4 Die Kurspflege Anwendung verbessert

```
<!-- Programm zur Euro-Dollar Umrechnung
    Teil Das grosse Ganze
```

¹⁰Was ich dann auch besser möchte!

¹¹Die man wiederum in der JavaScript-Referenz nachschlagen kann, die String-Methoden!

¹²Für diese merkwürdig erscheinende Syntax gibt es tatsächlich einen Grund. Ich habe hier nämlich nicht die ganze Wahrheit verraten. Tatsächlich erzeugen die Schrägstriche einen regulären Ausdruck (regular Expression, Reg Exp). Reguläre Ausdrücke kommen aus dem Unix und sind eine mächtige Waffe bei der Stringbearbeitung. Mit ihnen kann man nicht nur einfache Ersetzungen durchführen, sondern auch auf "recht einfache Art" (wenn man es denn kann), Bedingungen formulieren wie: Ersetze abc durch xyz, und zwar nur dann, wenn in dem Wort, was abc enthält, nur die Buchstaben abcde vorkommen und wenn das Wort das dritte im Absatz ist und nicht mehr als 8 Buchstaben enthält und jedes dritte Mal lass aus, selbst wenn alle anderen Bedingungen zutreffen ... Reguläre Ausdrücke werden wir aber nicht weiter behandeln.

```

    Kurspflege
    Dateiname: grosseGanze1/kurspflege2.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title> Euro-Dollar Umrechnung Kurspflege</title>
  <script language = "JavaScript"
    src="./javascript/feldkontrolle.js">
  </script>
  <script language = "JavaScript"
    src="./javascript/stringbearbeitung.js">
  </script>

  <script language = "JavaScript">
    function teste()
    {
      document.kurspflege.kurs.value=komma2punkt(
        document.kurspflege.kurs.value);

      if(istKeineZahl(document.kurspflege.kurs,
        "Kurs ist keine Zahl"))
      {
        return false;
      }
      document.kurspflege.method="post";
      document.kurspflege.action=document.kurspflege.action.value;
      document.kurspflege.submit();
    }
  </script>
</head>
<body>
  <table border=2 align="center">
    <tr>
      <td>
        
      </td>
      <td>
        Die Schwanen Gesellschaft
      </td>
    </tr>
  </table>
  <h2 align="center">
    Kurspflege
  </h2>
  <hr>
  <?php
    // Wir pruefen ob die Anfrage ueber get oder post erfolgte
    if($REQUEST_METHOD!="POST")
    {
      // erster Aufruf, das Formular muss praesentiert werden
    ?>
      <form name='kurspflege'>
        <input type="hidden" name="action"

```

```

        value="<?php echo $PHP_SELF ?>" >
<table border align="center">
  <tr>
    <td>
      Neuer Kurs
    </td>
    <td>
      <input type="text" name="kurs" size=12>
    </td>
  </tr>
  <tr>
    <td colspan="2" align="center">
      <input type="button" name="Button1" onClick="teste()"
        value="Abschicken">
    </td>
  </tr>
</table>
</form>
<?php
}
else
{
  // Verbindung zur Datenbank herstellen
  $link=mysql_pconnect("localhost", "root", "");
  // Datenbank euroDollarUmstellung einstellen
  $query="use euroDollarUmrechnung";
  mysql_query($query, $link);
  // update sql-Kommando
  $updateQuery ="update waehrung set kurs='$kurs' ";
  $updateQuery.="where name='Dollar' ";
  mysql_query($updateQuery, $link);
  echo "Kurs ge&auml;ndert auf $kurs.";
}
?>
</body>
</html>

```

Die Zeilen

```

<form name='kurspflege'>
  <input type="hidden" name="action"
    value="<?php echo $PHP_SELF ?>" >

```

zusammen mit der Umimplementierung des Button

```

<input type="button" name="Button1" onClick="teste()"
  value="Abschicken">

```

sorgen dafür, dass unser Formular nur über JavaScript abgeschickt werden kann.

```

onClick="teste()"

```

sorgt dafür, dass die JavaScript-Funktion `teste()` aufgerufen wird, wenn der Benutzer auf den Button klickt. Die Funktion `teste()` ist im `<head>`-Teil der php-Datei deklariert. In `teste()` wird zunächst die Funktion `komma2punkt` auf den Wert des Input-Feldes `kurs` angewendet. Das bedeutet, etwaige Kommas, die unsere Benutzer eingegeben haben werden durch Punkte ersetzt. Das Ergebnis von `komma2punkt` wird in das Input-Feld zurückgeschrieben:

```
document.kurspflege.kurs.value=komma2punkt(
    document.kurspflege.kurs.value);
```

Danach checkt unsere bereits in Kapitel 9 entwickelte Funktion `istKeineZahl()`, ob die Eingabe des Benutzers eine Zahl war. Beachten Sie die Reihenfolge. Wir müssen zuerst die Kommas durch Punkte ersetzen, dann erst können wir überprüfen, ob die Eingabe eine Zahl war. Der Grund ist einfach: `istKeineZahl()` benutzt die JavaScript interne Methode `isNaN()`. Und `isNaN` würde bei Kommas in Zahlen nie Zahlen erkennen. War alles okay, setzt `teste()` die Anfragemethode (`post`), setzt die "action" auf die php-Datei selbst und schickt das Formular ab. Dies entspricht aber Beispiel 9.16 aus Kapitel 9.

Die Produktseite

Wir starten mit einem grafischen Prototyp der Seite. Unsere Benutzer können dann beurteilen, ob das Design und die Funktionalität der Seite das ist, was sie erwarten. Die erste Näherung der Produktseite zeigt Abb. 10.8.

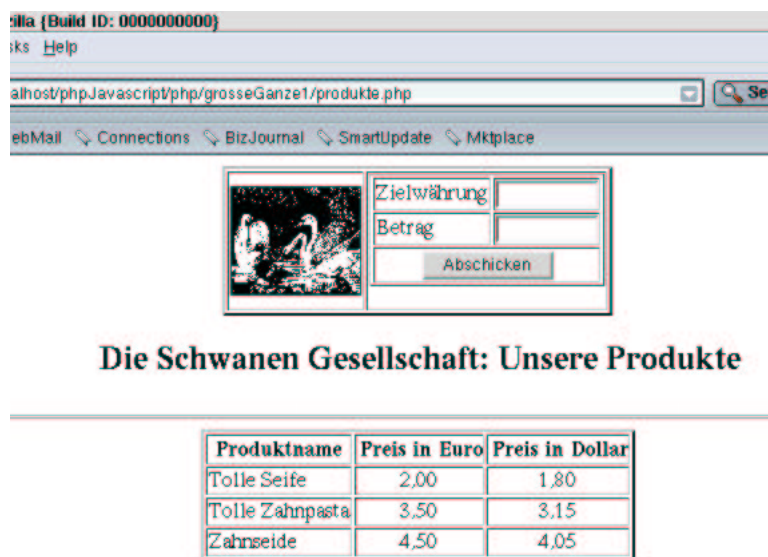


Abbildung 10.8: Produktseite, die erste Näherung

Allein, der Prototyp findet keine Gnade vor den Augen unserer Benutzer (der Marketingabteilung). Die meinen nämlich (nicht ganz zu unrecht), dass der Euro-Dollar-Umrechner dazu dienen soll, Surfer auf unsere Produktseite zu ziehen, damit die, während sie den Rechner benutzen, unsere Produkte im Auge haben und dann vielleicht die "tolle Seife" bestellen. Und bei unserem Prototyp sieht es ja so aus, dass sich für die Nutzer, wenn sie auf "Abschicken" klicken, eine neue Seite öffnet. Außerdem findet Die Abteilung Marketing die Bedienung zu kompliziert. Besser ist, wenn das eine Input-Feld mit Euro beschriftet ist und das andere mit Dollar. Nimmt der Benutzer einen Eintrag oder

eine Änderung vor, wird gerechnet und das Ergebnis im jeweils anderen Feld dargestellt. Es ergibt sich das Design aus Abb. 10.9.



Abbildung 10.9: Produktseite, so soll es aussehen

Uns stellt das natürlich vor diverse Probleme. Fangen wir mit den einfachen an. Zunächst sollen die aus den Europreisen berechneten Dollarpreise auf der Produktseite unserem Anforderungskatalog entsprechend mit zwei Nachkommastellen und dem Komma als Dezimaltrenner ausgegeben werden. Das ist in php nicht wirklich schwierig. In php gibt es diverse Funktionen für formatierte Ausgaben. Für unsere Zwecke geeignet ist `number_format()`. `number_format()` erwartet vier Übergabeparameter:

- Der erste Übergabeparameter ist die Zahl, die formatiert ausgegeben werden soll.
- Der zweite Übergabeparameter gibt die Anzahl Nachkommastellen an.
- Der dritte Übergabeparameter repräsentiert den Dezimaltrenner.
- Der vierte Übergabeparameter ist der Tausendertrenner.

Als nächstes müssen wir den Dollarkurs aus der Datenbank lesen. Das können wir aber auch. Wir sprechen mit `mysql_pconnect` das Datenbank-System an, wählen mit “use euroDollarUmrechnung” eben jene Datenbank aus, schicken einen `select`-Befehl an die Datenbank, und holen aus der Ergebnismenge das gewünschte Feld.

Ansonsten können wir für den Produktbereich der gewünschten php-Seite die Lösung aus Beispiel 7.10 fast unverändert übernehmen.

Etwas anders ist die Situation beim Euro-Dollar-Umrechner. Die Logik können wir mit JavaScript eigentlich leicht lösen. Das Formular entspricht nämlich Beispiel 9.13. Es gibt nur ein Problem: In Beispiel 9.13 hatten wir den Dollarkurs fest in das JavaScript-Programm aufgenommen. Jetzt steht er in einem Feld einer Tabelle einer Datenbank auf einem unserer Server. Und JavaScript läuft im Browser des Clients. Es gibt keinerlei Möglichkeit, von JavaScript aus auf Datenbanken des Servers zuzugreifen. Und hier machen wir den ganz großen Trick: Wir lesen beim Aufbau der Produktseite den Dollarkurs mit php aus der Datenbank. Dann schreiben wir mit dem `php-echo`-Befehl das ganze

JavaScript-Script in die Datei. Das hört sich jetzt ungeheuer kompliziert an, ist es aber gar nicht so wirklich. Wenn gleich der Code im Einzelnen besprochen wird, werden Sie das (wie ich hoffe) leicht verstehen. Und zum Schluss müssen die Ausgaben noch mit zwei Dezimalstellen und dem Komma als Dezimaltrenner in die Textfelder von Abb. 10.9 geschrieben werden. In JavaScript gibt es keine Funktionen zur formatierten Ausgabe von Zahlen. Also müssen wir selber eine schreiben, die das macht.

Damit beginnen wir. Wir erweitern die JavaScript-Datei "stringbearbeitung.js" um eine Funktion zur schönen Ausgabe von Zahlen. Es handelt sich dabei um die in Beispiel 10.5 dargestellte Funktion "schoeneAusgabe()".

Beispiel 10.5 Komma als Dezimaltrenner in Javascript

```
// Funktionen zur Stringbearbeitung
//
// Dateiname: stringbearbeitung2.js
function komma2punkt(str)
{
    str=str.replace(/,/g,".");
    return str;
}
function schoeneAusgabe(str)
{
    b=str.indexOf (".");
    if (b== -1)
    {
        str=str+ ",00"; // keine Nachkommastellen
    }
    else
    {
        if((str.substring(b+1,str.length)).length == 1)
        {
            str=str.substring(0,b)+","+str.substring(b+1,b+2)+"0";
        }
        else
        {
            str=str.substring(0,b)+","+str.substring(b+1,b+3);
        }
    }
    return str;
}
}
```

"schoeneAusgabe()" erwartet einen Übergabeparameter: Den String, der einen Punkt enthalten kann und der demzufolge umgewandelt werden muss.

```
function schoeneAusgabe(str)
```

Bei der Umwandlung wenden wir weitere Methoden der Stringklasse an:

```
b=str.indexOf (".");
```

gibt als Ergebnis die Stelle zurück, an der der erste Punkt im String str steht. Enthält der String keinen Punkt, wird -1 zurückgegeben. In diesem Fall gibt es keine Nachkommastellen und ,00 wird angehängt.

Gibt es hingegen einen Punkt, teilen wir den String mit der String-Methode substring auf. substring extrahiert einen Teilstring aus dem String-Objekt. Der Teilstring beinhaltet das Zeichen am ersten substring übergebenen Parameter, sowie alle Zeichen zwischen dem ersten und dem zweiten Parameter, das Zeichen am zweiten übergebenen Parameter allerdings nicht.

```
str.substring(0,b)
```

enthält also alle Zeichen bis zum Punkt, den Punkt selber aber nicht mehr. Hierbei müssen wir noch unterscheiden, ob es eine oder mehrere Nachkommastellen gibt. Bei einer Nachkommastelle müssen wir nämlich eine Null anfügen. Sind es zwei oder mehr, werden die über zwei hinausgehenden Stellen abgeschnitten. Um dies zu ermitteln, benutzen wir die Eigenschaft length des String-Objekts. length enthält die Anzahl Zeichen eines Strings.

```
str.substring(b+1,str.length)).length
```

ist also folgendermaßen zu lesen: Bilde einen Substring des übergebenen Strings, der direkt nach dem Punkt beginnt (Stelle b+1) und alle Zeichen bis zum letzten Zeichen enthält (str.length) und berechne von dem so erzeugten String die Länge. Ist diese Eins, so setzen wir den String wieder zusammen, mit der Ausnahme, dass wir an die Stelle wo der Punkt stand, ein Komma einsetzen. Zusätzlich fügen wir eine Null an:

```
str=str.substring(0,b)+", "+str.substring(b+1,b+2)+"0";
```

Gibt es zwei oder mehr Stellen nach dem Komma, fügen wir das Komma ein und schneiden nach der zweiten Stelle ab:

```
str=str.substring(0,b)+", "+str.substring(b+1,b+3);
```

Beachten Sie, dass diese Funktion nicht ganz korrekt ist, da sie nach der zweiten Stelle abschneidet. Eigentlich müsste dort gerundet werden. Sie können sich überlegen, wie man vorgehen müsste, um die Funktion vollständig richtig zu implementieren!

Als nächstes betrachten wir die Funktionen zur Umrechnung von Euro in Dollar bzw. umgekehrt. Dabei können wir auf die bereits in Beispiel 7.7 erstellte Umrechnungsfunktion zurückgreifen. Wir müssen sie nur an die neuen Gegebenheiten anpassen:

Beispiel 10.6 Euro-Dollar-Umrechnung mit Datenbank

```
<?php
// Funktion zur Dollar-Euro oder Euro-Dollar Umrechnung
// Datei:euroDollarUmrechnung.inc.php
// Verzeichnis: includes
function holeDollarKurs()
{
    $link=mysql_pconnect("localhost", "root", "");
    $query="use euroDollarUmrechnung";
    mysql_query($query, $link);
    $query="Select kurs from waehrung where name='Dollar'";
    $result=mysql_query($query, $link);
    $kurs=mysql_result($result,0,0);
    return $kurs;
}
```

```

}
function euroDollarUmrechnung($zielwaehrung, $betrag)
{
    $kurs=holeDollarKurs();
    if(($zielwaehrung=="Dollar")||($zielwaehrung=="dollar"))
    {
        $dollarbetrag=$kurs*$betrag;
        return $dollarbetrag;
    }
    if(($zielwaehrung=="Euro")||($zielwaehrung=="euro"))
    {
        $eurobetrag=(1/$kurs)*$betrag;
        return $eurobetrag;
    }
}
function erzeugeEuroDollarUmrechnungsScript()
{
    $kurs=holeDollarKurs();
    echo "<script language=\"JavaScript\">\n";
    echo "function euroDollarUmrechnung(zielwaehrung, betrag)\n";
    echo "{\n";
    echo "    var kurs=$kurs;\n";
    echo "    if((zielwaehrung==\"Dollar\")||(zielwaehrung==\"dollar\"))\n";
    echo "{\n";
    echo "        dollarbetrag=kurs*betrag;\n";
    echo "        return(dollarbetrag);\n";
    echo "    }\n";
    echo "    if((zielwaehrung==\"Euro\")||(zielwaehrung==\"euro\"))\n";
    echo "{\n";
    echo "        eurobetrag=(1/kurs)*betrag;\n";
    echo "        return(eurobetrag);\n";
    echo "    }\n";
    echo "}\n";
    echo "</script>\n";
}
?>

```

```
$kurs=holeDollarKurs();
```

ist die einzige Zeile, in der sich Beispiel 10.6 von Beispiel 7.7 unterscheidet. Das ist eigentlich auch einsichtig, denn, wenn wir den Dollarkurs aus der Datenbank gelesen haben, ist der Algorithmus ja auch identisch. Betrachten wir nun holeDollarKurs(): Die ersten Zeilen kennen wir schon. Sie stellen die Verbindung zum Datenbank-System her und stellen die Datenbank euroDollarUmrechnung ein.

```
$query="Select kurs from waehrung where name='Dollar'";
```

ist das SQL-Kommando, um den Dollarkurs aus der Datenbank zu lesen. Wie bereits weiter oben besprochen, erhalten wir von der Datenbank eine Ergebnismenge zurück, die wir mit mysql_result bearbeiten können. Da die Ergebnismenge nur ein Element enthalten kann, gibt es in der Ergebnismenge nur eine Zeile und eine Spalte. Daher erhalten wir mit

```
$kurs=mysql_result($result,0,0);
```

den Dollarkurs.

Ganz neu ist `erzeugeEuroDollarUmrechnungsScript()`. Denken wir uns bei dieser Funktion einmal die `echo`-Befehle am Anfang jeder Zeile weg, so sehen wir, dass es zu Beispiel 7.12 eigentlich nur einen Unterschied gibt: Die JavaScript-Variable `kurs` wird in Beispiel 10.6 mit dem Inhalt der php-Variablen `$kurs` besetzt, auf der vorher der aus der Datenbank gelesene Dollarkurs gespeichert wurde. Die anderen Änderungen sind marginal: Der `\` vor den Anführungszeichen sorgt dafür, dass die Anführungszeichen nicht als Ende des jeweiligen Strings interpretiert werden¹³. Das am Ende einer Zeile eingefügte `\n` ist das Zeichen für den Zeilenvorschub. Dies sorgt nur dafür, dass die `“view source”`-Funktion des Browsers den JavaScript-Source-Code untereinander und nicht in einer Zeile¹⁴ anzeigt. Wenn also `erzeugeEuroDollarUmrechnungsScript()` in einer html-Datei aufgerufen wird, wird, wie Sie ja wissen, der php-Code durch seinen Output ersetzt und der Output ist in diesem Fall eine JavaScript-Funktion, die Euros in Dollar und umgekehrt umrechnen kann.

Jetzt sind wir soweit, uns die Implementierung der Produktseite ansehen zu können. Beispiel 10.7 zeigt sie.

Beispiel 10.7 Die Produktseite mit Euro-Dollar-Umrechnung

```
<!-- Programm zur Euro-Dollar Umrechnung
    Teil Das grosse Ganze
    Produktseite mit Euro-Rechner
    Dateiname: grosseGanze/produkte.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
    <title> Euro-Dollar Umrechnung Kurspflege</title>
<?php
    require_once("../includes/euroDollarUmrechnung.inc.php");
    erzeugeEuroDollarUmrechnungsScript();
?>
<script language = "JavaScript"
    src = "../javascript/feldkontrolle.js">
</script>
<script language = "JavaScript"
    src = "../javascript/stringbearbeitung2.js">
</script>
<script language = "JavaScript">
    function dollar2euro()
    {
        document.euro6.dollar.value=komma2punkt
            (document.euro6.dollar.value);
        if(istKeineZahl(document.euro6.dollar,
            "Keine Zahl eingegeben!"))
        {
            return;
        }
        else
        {
            document.euro6.euro.value=
                schoeneAusgabe(euroDollarUmrechnung(
```

¹³Sie werden durch den `\` escaped, wir hatten das ja viel früher schon einmal besprochen.

¹⁴Ich werde Ihnen nämlich noch einen Screenshot zeigen, in dem man sehen kann, dass tatsächlich eine JavaScript-Funktion an den Browser übertragen wird.

```

        "euro", document.euro6.dollar.value).toString());
    }
}
function euro2dollar()
{
    document.euro6.euro.value=komma2punkt
        (document.euro6.euro.value);
    if(istKeineZahl(document.euro6.euro,
        "Keine Zahl eingegeben!"))
    {
        return;
    }
    else
    {
        document.euro6.dollar.value=
            schoeneAusgabe(euroDollarUmrechnung(
                "dollar", document.euro6.euro.value).toString());
    }
}
</script>
</head>
<body>
    <table border=2 align="center">
        <tr>
            <td>
                
            </td>
            <td valign="bottom">
                <form name='euro6'>
                    <table border>
                        <tr>
                            <td>
                                Dollar
                            </td>
                            <td>
                                <input type="text" name="dollar"
                                    onChange="dollar2euro()" size=12>
                            </td>
                        </tr>
                        <tr>
                            <td>
                                Euro
                            </td>
                            <td>
                                <input type="text" name="euro"
                                    onChange="euro2dollar()" size=12>
                            </td>
                        </tr>
                    </table>
                </form>
            </td>
        </tr>
    </table>

```

```

<h2 align="center">
    Die Schwanen Gesellschaft: Unsere Produkte
</h2>
<hr>
<table border="2" align="center">
<tr>
    <th> Produktname </th>
    <th> Preis in Euro </th>
    <th> Preis in Dollar </th>
</tr>
<tr>
    <td> Tolle Seife </td>
    <td align="center"> 2,00 </td>
    <td align="center">
<?php
        echo(number_format(euroDollarUmrechnung("Dollar", 2),2,""," "))
?>
    </td>
</tr>
<tr>
    <td> Tolle Zahnpasta </td>
    <td align="center"> 3,50 </td>
    <td align="center">
<?php
        echo(number_format(euroDollarUmrechnung("Dollar", 3.5),2,""," "))
?>
    </td>
</tr>
<tr>
    <td> Zahnseide </td>
    <td align="center"> 4,50</td>
    <td align="center">
<?php
        echo(number_format(euroDollarUmrechnung("Dollar", 4.5),2,""," "));
?>
    </td>
</tr>
</table>
</body>
</html>

```

Direkt im <head>-Teil wird `erzeugeEuroDollarUmrechnungsScript()` aufgerufen. Dies bedeutet, `erzeugeEuroDollarUmrechnungsScript()` wird durch den erzeugten JavaScript-Code ersetzt. Der Browser erhält die in Abb. 10.10 dargestellten Daten.

Der Rest der Produktseite besteht aus der Zusammensetzung der Beispiele 7.10 und 9.11. Der Unterschied zu Beispiel 9.11 besteht darin, dass von den mit den `onChange`-Event-Handle­rn der Textfelder verbundenen Funktionen zunächst `komma2punkt()` aufgerufen wird, da laut Anforderungskatalog Kommas als Dezimaltrenner zugelassen sind. Vor dem Zurückschreiben in das Input-Text-Feld wird ebenfalls im Unterschied zu Beispiel 9.11 `schoeneAusgabe` aufgerufen, damit die Ausgabe des errechneten Wertes mit zwei Dezimalstellen und dem Komma als Dezimaltrenner erfolgt. Betrachten wir diese Zeile genauer:

```

<!-- Programm zur Euro-Dollar Umrechnung
Teil Das grosse Ganze
Produktseite mit Euro-Rechner
Dateiname: grosseGanze/produkte.php /-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title> Euro-Dollar Umrechnung Kurspflege</title>
  <script language="JavaScript">
function euroDollarUmrechnung(zielwaehrung, betrag)
{
var kurs=0.97;
if((zielwaehrung=="Dollar")||(zielwaehrung=="dollar"))
{
dollarbetrag=kurs*betrag;
return(dollarbetrag);
}
if((zielwaehrung=="Euro")||(zielwaehrung=="euro"))
{
eurobetrag=(1/kurs)*betrag;
return(eurobetrag);
}
}
</script>
<script language = " JavaScript"
src="./javascript/feldkontrolle.js">
</script>
<script language = " JavaScript"
src="./javascript/stringbearbeitung2.js">
</script>
<script language = " JavaScript">
function dollar2euro()
{

```

Abbildung 10.10: Die erzeugte JavaScript-Funktion

```

document.euro6.dollar.value=
  schoeneAusgabe(euroDollarUmrechnung(
    "dollar", document.euro6.euro.value).toString);

```

Zunächst erkennen wir wieder die Schachtelbarkeit von Funktionsaufrufen. Zunächst wird `euroDollarUmrechnung()` aufgerufen. `schoeneAusgabe()` verarbeitet das Ergebnis von `euroDollarUmrechnung()`. Ein bisschen stört noch `“.toString“`. `euroDollarUmrechnung()` gibt aber eine Zahl zurück, `schoeneAusgabe()` hingegen erwartet einen String. `“.toString“` an eine eine Zahl enthaltende Variable sorgt dafür, dass der Inhalt der Variablen in einen String umformatiert wird. Warum JavaScript das in diesem Fall allerdings nicht automatisch macht, wo es doch sonst so viel automatisch macht, ist mir allerdings auch unklar.

Damit müsste der obere Teil, das Euro-Dollar-Umrechnungsformular, jetzt verständlich sein. Den zweiten Teil, die Umrechnung der Produktpreise, müssten Sie eigentlich direkt verstehen. Der Unterschied zu Beispiel 7.10 liegt nur darin, dass zwischen das `echo`-Kommando und den Aufruf von `euroDollarUmrechnung()` (diesmal die `php`-Funktion wohlgermerkt) ein Aufruf der Funktion `number_format()` geschaltet wurde. Der erste Parameter dieser Funktion ist das Ergebnis von `euroDollarUmrechnung()`, also die Zahl, die ausgegeben werden soll, der zweite die Anzahl Dezimalstellen (2), der dritte der Dezimaltrenner (,) und der vierte der Tausendertrenner (“ ”).

Zusammenfassung

Eine solche vergleichsweise doch überschaubare Aufgabenstellung erfordert bereits einen nicht unerheblichen Programmieraufwand. Mit Software wie MS-Frontpage kann man bei diesen Aufgabenstellungen “keinen Blumentopf mehr gewinnen”. Die Aufgabe war eigentlich nur im Zusammenspiel von JavaScript, html, php und Datenbanktechnologie lösbar. Ganz abgesehen von der Anforderungsanalyse, die wir hier nur kurz behandelt haben. Die Gesamtlösung ist auf mehrere Dateien verteilt. Abb. 10.11 zeigt die Dateistruktur der Anwendung.

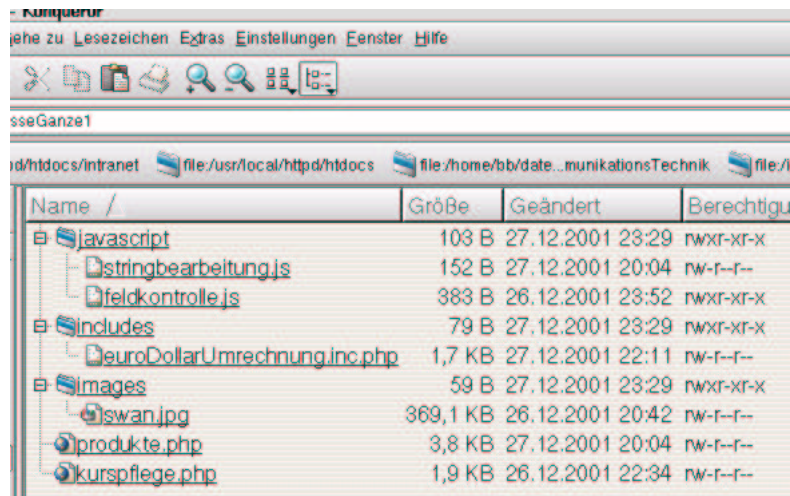


Abbildung 10.11: Die Dateistruktur der Anwendung

10.4 Verbesserte Lösung: Sessions und ein wenig Anwendungsarchitektur

Kaum ist das Produkt fertig und online gegangen, häufen sich die Probleme:

1. Die Marketing-Leute haben mitbekommen, dass es eine Seite gibt, über die man den Euro-Kurs ändern kann. Und machen das manchmal. Was natürlich die Mitarbeiter der Produktpflege nicht glücklich macht.
2. Sogar Kunden ändern manchmal den Euro-Kurs.
3. Das Software-Qualitätsmanagement fragt an, wie wir uns eine etwaige Umstellung auf ein anderes Datenbank-System vorstellen. Schließlich ist nichts für die Ewigkeit (schon gar nicht in deutschen Unternehmen). Bei unseren jetzigen Seiten müssten wir bei einer Änderung des Datenbank-Systems alle Seiten kontrollieren und ggf. anpassen.
4. Überdies findet unsere Anwendungsarchitektur keine Gnade beim Software-Qualitätsmanagement. Wir haben zwar eine Datei, in der wir Währungsfunktionen sammeln, nämlich euroDollarUmrechnung.inc.php, allerdings finden sich weitere Währungsfunktionen auch in kurspflege.php.
5. Auch die Anwender sind nicht ganz zufrieden. Manchmal ist die Datenbank nicht verfügbar. Dieser Fehler wird nicht abgefangen, anstelle dessen werden die Anwender mit unverständlichen Fehlermeldungen gequält.

Erste Gedanken zur Lösung

Glücklicherweise lassen sich diese Probleme leicht lösen. Probleme (1) und (2) lösen wir mit einem Anmeldeverfahren. Wir werden der Kurspflege-Seite eine Anmeldung vorschalten.

Problem (3) lösen wir, indem wir Funktionen schreiben, die die von uns genutzten MySQL-Funktionen kapseln. Diese sammeln wir in einer Datei. Bei einer Änderung der Datenbank muss dann nur noch

in dieser Datei geändert werden. Mit dieser Vorgehensweise werden wir auch Problem (5) lösen. Es bietet sich nämlich an, hier zu prüfen, ob die Abfragen an die Datenbank auch erfolgreich waren.

Problem (4) lösen wir, indem wir alles, was mit Währung zu tun hat, in der Datei `euroDollarUmrechnung.inc.php` sammeln. Die dort geschriebenen Funktionen werden dann von den Seiten, die der Anwender sieht, aufgerufen (so funktioniert es ja bereits in `produkte.php`).

Anmeldung und Sessions

Doch beginnen wir mit dem Anmeldeverfahren¹⁵. Anmeldungen sind ein grundsätzliches Problem bei Internet-Anwendungen. Der Grund ist, dass das dem WWW zu Grundliegende http-Protokoll “zustandslos” ist. Das bedeutet, dass ein WWW-Server immer genau eine Anfrage bearbeitet und die angeforderte WWW-Seite ausliefert (sofern diese Seite existiert). Danach “vergisst” der WWW-Server den Vorgang. Das würde bedeuten, die Anwender melden sich an, unsere Anwendung akzeptiert die Anmeldung und vergisst den Vorgang gleich wieder. Das wäre gar nicht gut. Es gibt allerdings zwei Möglichkeiten, diesem Zustand abzuhelpfen:

- Man kann die Anmeldeinformation in einen Cookie schreiben. So ein Cookie wird ja, wie Sie wissen, bei jeder erneuten Anfrage an unseren WWW-Server vom Browser an den Server übertragen. Cookies lassen sich von php auswerten und so können wir überprüfen, ob der Benutzer bereits angemeldet ist oder nicht.
- Man kann die in jeder Seite referenzierten URL’s verändern, indem man die Anmeldeinformation mit in die URL aufnimmt (URL-Rewriting). Die URL kann selbstverständlich auch von php aus ausgewertet werden.

Wenn man Anmeldeinformationen über mehrere Seiten einer Internetanwendung vorhält, spricht man von einer “Session”.

In php ist das Erzeugen einer Session relativ einfach. Wir müssen in unseren Programmen Cookies nicht selber auswerten oder schreiben, oder Url’s ändern. php hält für diese Zwecke Funktionen vor. Doch bevor wir anfangen, Sessions zu programmieren, müssen wir andere Dinge bedenken. Denn die Benutzer aus der Produktpflege-Abteilung müssen die Möglichkeit bekommen, sich an unserem System anzumelden. Vorher wollen wir sicherlich keine Session erzeugen.

Um dies abzubilden, müssen wir das Entity-Relationship-Modell ändern. Es bietet sich nämlich an, die Anmeldeinformationen (also die benötigten Informationen über die Mitarbeiter) ebenfalls in der Datenbank vorzuhalten.

Hier wird eine neue Entity und damit die neue Tabelle Mitarbeiter fällig (vgl. Abb. 10.12).

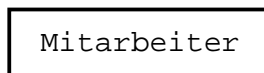


Abbildung 10.12: Das ERM zur Euro-Dollar-Problemstellung, die Entity Mitarbeiter

Wir könnten überlegen, ob die in Abb. 10.4 und Abb. 10.12 dargestellten Entitäten in einer Beziehung stehen (z.B. “darf ändern”). Wir halten das Beispiel jedoch einfach und sagen: Jeder, der sich anmelden kann, darf auch die Währungskurse ändern.

Dann müssen wir uns nur noch die Attribute überlegen. Hier besinnen wir uns auch auf das Notwendigste: `mitarbeiter_nr`, `name`, `vorname`, `benutzerName`, `passwort`. “`mitarbeiter_nr`” ist der selbst

¹⁵Das ist nämlich auch das schwierigste und das machen wir solange wir noch fit sind!

hoch zählende Primärschlüssel. Sie können sich selbst überlegen, aus welchem Grund wir das Feld "benutzerName" benötigen. Abb.10.13 zeigt die mit phpmyadmin neu angelegte Tabelle:

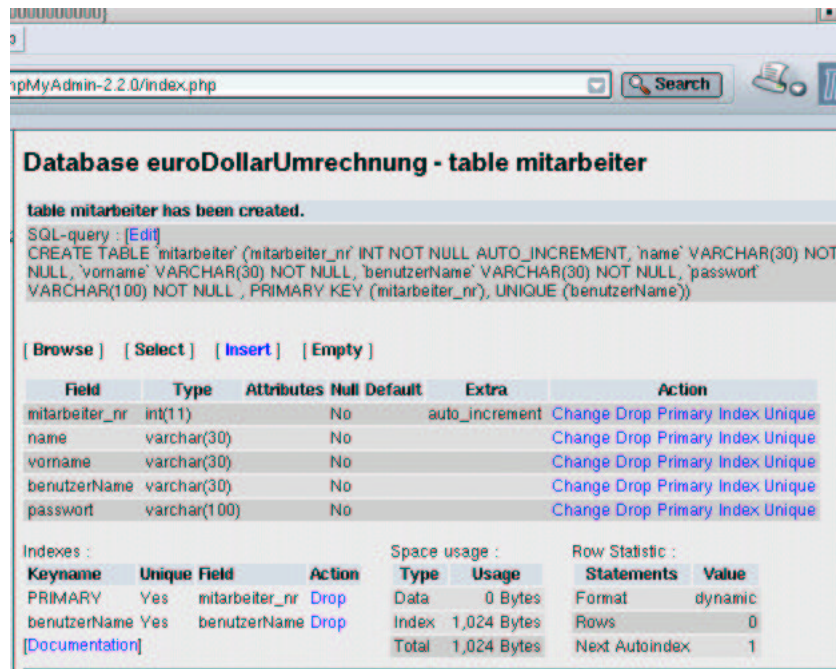


Abbildung 10.13: Die Tabelle Mitarbeiter

Bevor wir mit der Programmierung beginnen, stelle ich kurz die in php integrierte Session-Verwaltung vor. Um unsere Anwendung zu implementieren, benötigen wir folgende php-Funktionen:

- session_start(): Startet eine Session.
- session_register(): Registriert eine Variable in einer Session.
- session_is_registered(): Prüft, ob eine Variable in einer Session registriert wurde.
- session_destroy(): Zerstört eine Session.

session_start() startet, wie der Name schon sagt, eine Session. session_start() muss als erste Anweisung einer html-Seite ausgeführt werden und zwar bevor der Server Output an den Browser schickt, da session_start() header-Informationen verändert. Ein korrekter Aufruf von session_start() in einer html-Datei sieht also folgendermaßen aus:

```
<?php
    session_start();
?>
<html>
```

Beachten Sie, dass nicht einmal ein Leerzeichen vor dem <-Zeichen stehen darf, da der www-Server dann dieses an den Browser sendet. Um das tun zu können, muss der www-Server vorher die vollständigen header-Informationen senden und session_start() kann sie nicht mehr verändern.

session_start() checkt als Erstes, ob es bereits eine Session gibt. Ist dies nicht der Fall (dies betrachten wir zunächst), vergibt session_start() eine Session-Id. Dies ist eine für den Server eindeutige

Zufallszahl, die php dann noch mit dem md5-Algorithmus verschlüsselt. Diese Session-Id wird dann entweder (Cookies sind erlaubt) in einen Cookie geschrieben, oder alle Links werden umgeschrieben (URL-Rewriting). Abb. 10.14 zeigt dies am Beispiel unserer Intranet-Anwendung.

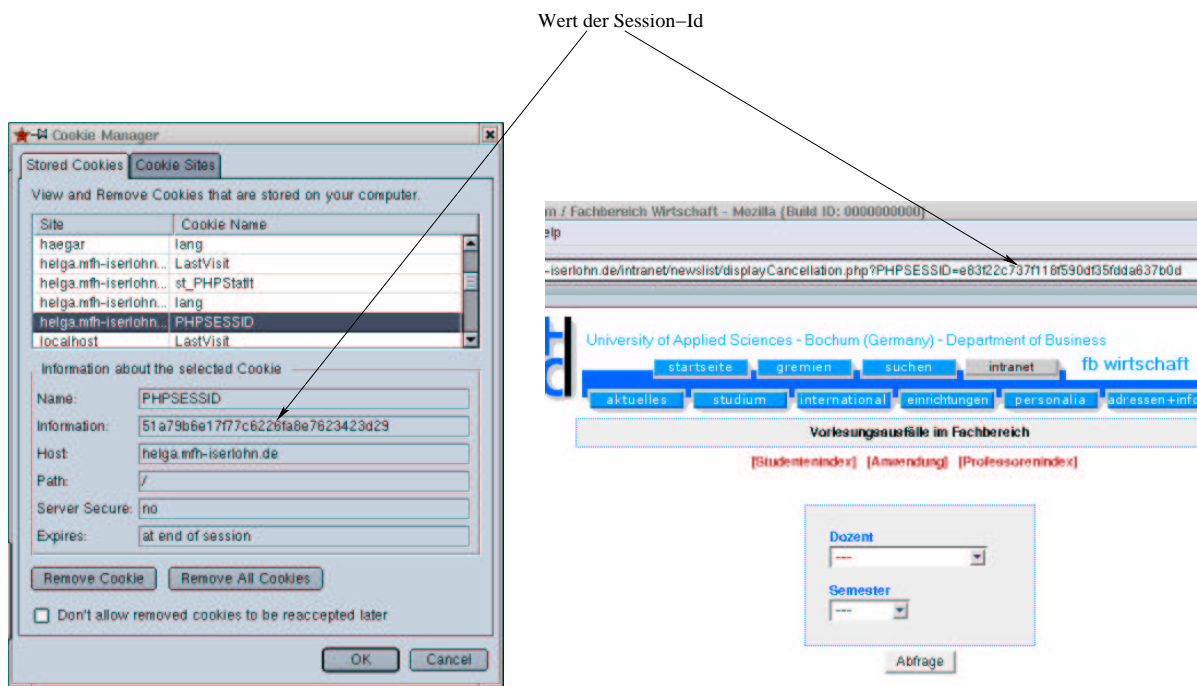


Abbildung 10.14: Session-Variablen als Cookie oder in der URL

Gibt es bereits eine Session, passiert gar nichts. Dies klingt zunächst nicht sonderlich intelligent, denn entweder gibt es keine Session, dann erzeugt `session_start()` eine Session, gibt es hingegen bereits eine, passiert nichts. Wie sollen wir also unterscheiden, ob ein Benutzer angemeldet ist oder nicht?

Hierzu stellt php die Funktionen `session_register()` und `session_is_registered()` zur Verfügung. `session_register()` registriert Variablen in einer Session, `session_is_registered()` überprüft, ob eine Variable in der aktuellen Session registriert ist.

Eine Variable für eine Session registrieren bedeutet, dass php in einem Verzeichnis für temporäre Dateien des Servers (`/tmp` in Unix/Linux) eine Datei mit dem Namen der vergebenen Session-Id anlegt und in diese Datei den Namen und den Wert der registrierten Variable einfügt. Abb. 10.15 zeigt dies. Unter Nutzung dieser beiden Funktionen erfolgt die Benutzerverwaltung dann in zwei Schritten:

1. Anmeldung:

- Wir starten eine Session.
- Die Benutzer geben Benutzernamen und Passwort an.
- Wir überprüfen dies durch einen Datenbankzugriff.
- Ist die Anmeldung okay, registrieren wir eine Session-Variable z.B. "benutzerName" und weisen dieser den Benutzernamen zu. Ist die Anmeldung nicht okay, zerstören wir die Session wieder (hierfür gibt es die Methode `session_destroy()`).

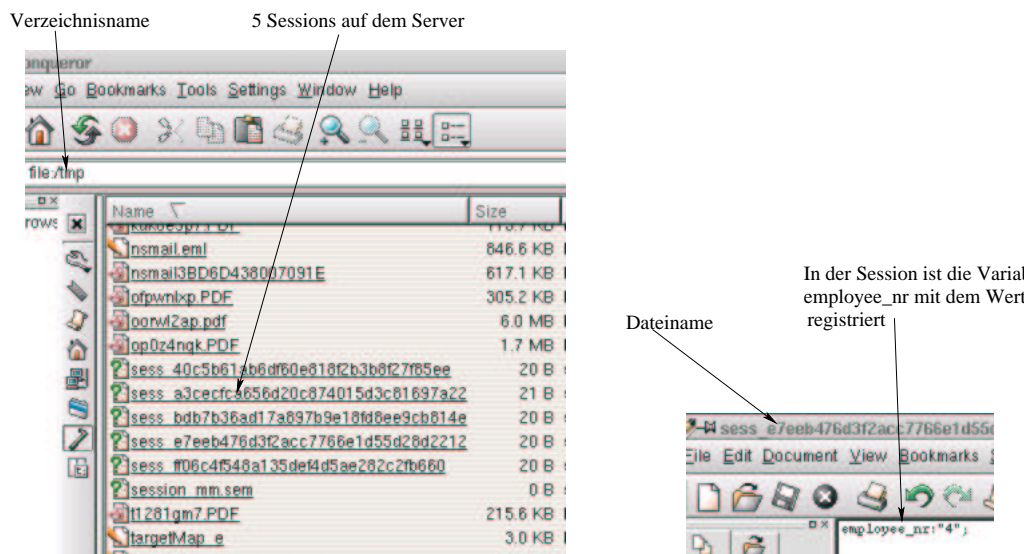


Abbildung 10.15: Variable employee_nr mit Wert 4 in einer Session

2. Überprüfung:

- Wir starten eine Session.
- Wir überprüfen, ob die Variable “benutzerName” registriert ist (mit `session_is_registered()`).
- Ist dies der Fall, so ist der Benutzer angemeldet und darf die durch die Session geschützte Seite benutzen. Ist dies nicht der Fall, zerstören wir die Session wieder (wieder mit der Methode `session_destroy`) und verzweigen den Benutzer auf die Anmeldeseite.

Änderung der Anforderungen

Bevor wir mit der Implementierung beginnen, stimmen wir mit den Anwendern ab, wie die Anmeldung genau ablaufen soll. Hier bieten sich Aktivitätsdiagramme¹⁶ an. Abb. 10.16 zeigt den Ablauf bei der Anmeldung.

Der Ablauf startet mit der Anmeldung des Benutzers. Ist die Anmeldung okay, wird eine Seite aufgerufen, die die Anwendungen referenziert, für die der Benutzer freigeschaltet ist. Ist die Anwendung nicht okay, wird der Benutzer auf die Anmeldungsseite zurückgeführt.

Gedanken zur Implementierung

Implementieren wir dies für unser Beispiel. Zunächst die Anmeldung: Als Erstes müssen wir eine Seite mit einem Formular aufblenden, in das der Benutzer die Anmeldeinformationen eingeben kann. Das ist nicht wirklich schwer, dazu reicht ein Formular mit einer Tabelle für das schöne Aussehen. Das Formular überprüfen wir mit JavaScript auf Eingabefehler (hier können wir nur überprüfen, ob überhaupt Eingaben getätigt wurden) und schicken es, wie gehabt, mit JavaScript ab.

Wenn das Formular abgeschickt wurde, müssen wir schon mehr machen:

1. Eine Verbindung zum Datenbank-System aufbauen und “euroDollarUmrechnung” als Datenbank einstellen.

¹⁶Ebenfalls Teil der UML.

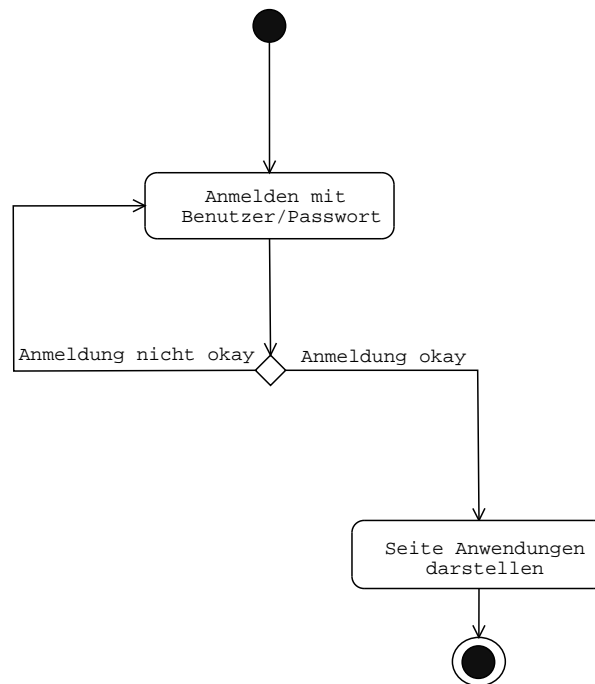


Abbildung 10.16: Ablauf bei der Anmeldung, Darstellung der Seiten anmeldung.php und anwendungen.php

2. Überprüfen, ob es einen Mitarbeiter mit der angegebenen Benutzernamen/Passwort-Kombination gibt.
3. Abhängig vom Erfolg dieses Vergleichs entweder auf die Seite mit den freigeschalteten Anwendungen oder wieder zurück zur Anmeldeseite verzweigen.

Im Licht der Kritik an unserer Anwendung (Punkte (3) und (5)) bietet es sich an, die Verbindung zum Datenbanksystem und die Auswahl der Datenbank “euroDollarUmrechnung” in eine eigene Datei auszulagern. Wir erzeugen hierfür eine Klasse (vgl. 8), die wir DbFunctions nennen und in der wir alle Datenbankanrufe durchführen werden. Sollte sich irgendwann einmal das zu Grundliegende Datenbanksystem ändern, müssen wir nur in dieser Klasse Änderungen durchführen.

Wenn wir Punkt (2) betrachten, fällt uns auf, dass das Feststellen, ob eine Benutzer/Passwort-Kombination übereinstimmt, eigentlich eine Funktionalität der Mitarbeiter ist. Im Kritikpunkt (4) an unserer bisherigen Anwendung, wurde angemerkt, dass wir Funktionalitäten, die eigentlich Währungen zukommen, in anderen Dateien realisiert haben. Um hier vorzubeugen, werden wir eine Klasse Mitarbeiter erzeugen und dort alle Mitarbeiter betreffenden Funktionalitäten sammeln.

Beachten Sie, dass wir die Klassen in diesem Beispiel nicht erzeugen, um wirklich objektorientiert zu entwickeln. Wir werden keine Objekte dieser Klassen erzeugen. Bezugnehmend auf meine Argumentation in Kapitel 8 erstelle ich hier Klassen, um getrennte Namensräume zu erhalten und damit der Leser der Programme sofort sieht, in welcher Datei welche Funktion abgespeichert wird¹⁷.

Bei Punkt (3) müssen wir eigentlich nur abhängig vom Ausgange der Überprüfung der Benutzernamen/Passwort-Kombination auf andere (oder die gleiche) html-Seiten verzweigen. Dies können wir (u.a.) durch die Nutzung der replace-Methode des JavaScript-location-Objekts erreichen. Die zugehörige JavaScript-Funktion müssen wir aus php erzeugen. Wie so etwas gemacht wird, haben Sie ja bereits in Kapitel

¹⁷Dies deshalb, weil wir unsere Klassen in Dateien mit dem Klassennamen abspeichern werden und dem Aufruf einer Klassenmethode ja der Klassenname vorangestellt wird.

10.3 gesehen. Hier lagern wir diese Funktion allerdings in eine eigene Datei aus. Und zur Verwirrung werden wir in diesem Fall keine Klasse erzeugen, sondern diese und ähnliche Funktionen in einer eigenen Datei sammeln. In dieser Datei werden wir alle Funktionen der Anwendung sammeln, die wir sonst nirgendwo zuordnen können (wo wir also keinen Oberbegriff und demzufolge auch keinen Klassennamen für finden können). Insgesamt ergibt sich der in Abb. 10.17 dargestellte Zusammenhang.

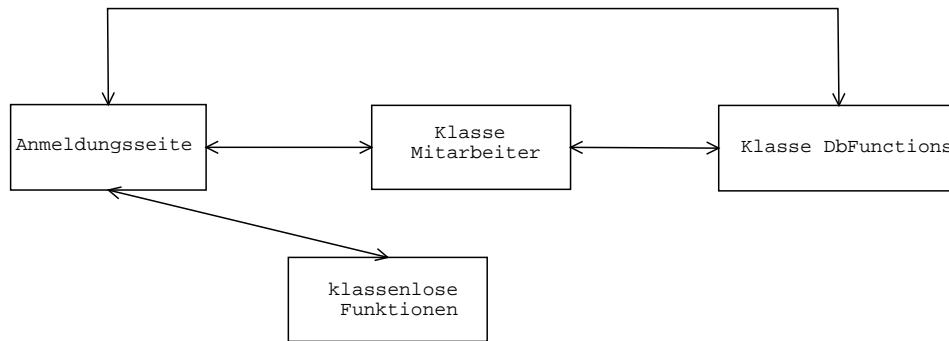


Abbildung 10.17: Architektur der Anwendung: Teil 1

Die Klasse DbFunctions

Beginnen wir mit der Klasse DbFunctions:

Beispiel 10.8 Die Klasse DbFunctions

```

<?php
class DbFunctions
{
/*
* class: DbFunctions
* Filename: DbFunctions.inc.php
* Directory: classes
* lastChanged: 07.01.2002 by bb
* authors: Bernd Bluemel, Christian Metzger
* purpose: Common usable db-Functions
*/

// no instance-Variables, only classmethods, the class-construct is used to
// create a
// separate namespace for our DbFunctions

/* boolean connect(&$link)
* connects to mysql, provides username and password
* creates the Link to the database
* the link is returned as the only parameter in
* the argument list
* sets the database to euroDollarUmrechnung
* returns true on success, false otherwise
*
*/

```

```

function connect(&$link)
{
    $link = mysql_pconnect("localhost", "root", "");
    if (!$link) return false;
    // select the database
    $query = "use euroDollarUmrechnung";
    mysql_query($query, $link);
    return true;
}
function getFirstFieldOfResult($link, $query)
{
    $result=mysql_query($query, $link);
    if(mysql_num_rows($result)==0)
    {
        return;
    }
    return(mysql_result($result,0,0));
}
function executeQuery($link, $query)
{
    $result=mysql_query($query, $link);
    return $result;
}
}
?>

```

Wir betrachten zunächst die Funktion connect():

```

function connect(&$link)
{
    $link = mysql_pconnect("localhost", "root", "");
    if (!$link) return false;
    // select the database
    $query = "use euroDollarUmrechnung";
    mysql_query($query, $link);
    return true;
}

```

connect() erwartet einen Parameter (\$link). An dem &-Zeichen erkennen wir, dass es sich um einen Referenzparameter handelt. Änderungen dieses Parameters werden also an das aufrufende Programm zurückgegeben. connect() stellt also eine Verbindung zum Datenbank-System her und wählt die Datenbank "euroDollarUmrechnung" aus. Auf dem Übergabeparameter \$link wird die Verbindung zur Datenbank an das aufrufende Programm zurückgegeben. Die Funktion gibt allerdings noch eine zweite Information an das aufrufende Programm zurück: Funktioniert der Aufbau der Verbindung zur Datenbank nicht (\$link hat dann keinen Wert), gibt die Funktion false zurück. Im Erfolgsfall wird true zurückgegeben.

Die zweite Funktion der Klasse DbFunctions ist getFirstFieldOfResult():

```

function getFirstFieldOfResult($link, $query)
{
    $result=mysql_query($query, $link);

```

```

        if(mysql_num_rows($result)==0)
        {
            return;
        }
        return(mysql_result($result,0,0));
    }

```

Diese Funktion erwartet zwei Übergabeparameter, die Verbindung zur Datenbank (\$link) und eine SQL-Abfrage. Die Funktion führt die SQL-Abfrage durch und gibt entweder das erste Feld in der ersten Zeile der Resultatsmenge zurück oder nichts. `getFirstFieldOfResult()` benutzt eine neue php-Funktion: `mysql_num_rows()`. `mysql_num_rows()` benötigt die Ergebnismenge einer SQL-Abfrage als Übergabeparameter. `mysql_num_rows()` gibt¹⁸ die Anzahl der gefundenen Datensätze¹⁹ zurück.

Die letzte Funktion von `DbFunctions` ist `executeQuery()`:

```

function executeQuery($link, $query)
{
    $result=mysql_query($query, $link);
    return $result;
}

```

`executeQuery()` gibt einfach das Ergebnis einer an die MySQL-Datenbank gerichteten SQL-Abfrage zurück.

Die Klasse Mitarbeiter

In der Klasse `Mitarbeiter` benötigen wir zur Zeit nur eine Funktion, nämlich die, die überprüft, ob eine Benutzernamen/Passwort-Kombination okay ist:

Beispiel 10.9 Die Klasse Mitarbeiter

```

<?php
    // require_once("DbFunctions.inc.php");
class Mitarbeiter
{
    /*
    * class: Mitarbeiter
    * Filename: Mitarbeiter.inc.php
    * Directory: classes
    * lastChanged: 07.01.2002 by bb
    * authors: Bernd Bluemel, Christian Metzger
    * purpose: User related functions
    */
    function passwortIstOkay($link, $name, $passwort)
    {
        $query= "select * from mitarbeiter where ";
        $query.="benutzerName='$name' and ";
        $query.="passwort='$passwort' ";
        $mitarbeiter_nr=DbFunctions::getFirstFieldOfResult($link, $query);
        if(!isset($mitarbeiter_nr) || empty ($mitarbeiter_nr))
        {

```

¹⁸Wie der Name schon andeutet.

¹⁹Dies sind die Zeilen der Ergebnismenge.

```

        return false;
    }
    return true;
}
}
?>

```

passwordIstOkay() erwartet drei Übergabeparameter: Die Verbindung zur Datenbank (\$link), den Benutzernamen (\$name) und das Passwort (\$password). passwordIstOkay() erzeugt dann ein SQL-Kommando, um festzustellen, ob es die Benutzernamen/Passwort-Kombination gibt:

```

$query= "select * from mitarbeiter where ";
$query.="benutzerName='$name' and ";
$query.="password='$password' ";

```

Die Ergebnismenge zu diesem SQL-Kommando besteht entweder aus einem oder aus keinem Datensatz. Ist die Ergebnismenge leer, gibt es die Benutzernamen/Passwort-Kombination nicht und die Anmeldung ist fehlgeschlagen. passwordIstOkay() ruft dann getFirstFieldOfResult() auf. Sie sehen, dass dem Methodenaufruf bei der Programmierung mit Klassen der Klassennamen gefolgt von zwei Doppelpunkten vorangestellt werden muss. getFirstFieldOfResult() hatten wir weiter oben bereits besprochen. getFirstFieldOfResult() gibt nichts zurück, wenn die Ergebnismenge des SQL-Kommandos leer war, das erste Feld des ersten²⁰ Datensatzes der Ergebnismenge ansonsten. Wir prüfen also nur noch, ob das Ergebnis von getFirstFieldOfResult() existiert oder nicht. Existiert es, war die Anmeldung okay und es wird true zurückgegeben, existiert es nicht, wird false zurückgegeben. Zur Überprüfung benutzen wir die von php bereitgestellten Funktionen isset und empty. isset gibt true zurück, wenn eine Variable existiert, empty, wenn eine Variable keinen Wert hat.

Die von php erzeugten JavaScript-Funktionen

Beispiel 10.10 *Oft benutzte Funktionen*

```

<?php
// oft benutzte Funktionen
// in einer Datei zusammengefasst
// keine Klasse, wegen Schreibersparnis
// Dateiname: oftBenutzteFunktionen.inc.php
// Verzeichnis: includes
// Autor: Bernd Bluemel
// letzte Aenderung 07.01.2002 by bb

function loadPage($newPage)
{

echo "<script language=\"JavaScript\">";
echo "location.replace(\"$newPage\")";
echo "</script>";
}
function displayAlert ($alertMessage)
{

```

²⁰Und im Fall unserer Abfrage auch einzigen

```

echo "<script language=\"JavaScript\">";
echo "alert(\"$alertMessage\")";
echo "</script>";
}
?>

```

Diese Funktionen müssten Sie eigentlich sofort verstehen. Die Funktion `loadPage()` lädt die ihr übergebene URL in das Browser-Fenster, `displayAlert()` gibt den ihr übergebenen Text in einem Alert-Fenster auf. Beachten Sie das "Escapen" der Sonderzeichen in den beiden Funktionen.

Die Datei `anmeldung.php`

Implementieren wir nun die Anmeldeseite:

Beispiel 10.11 Die Anmeldeseite

```

<?php
    session_start();
    require_once("../classes/Mitarbeiter.inc.php");
    require_once("../classes/DbFunctions.inc.php");
    require_once("../includes/oftBenutzteFunktionen.inc.php");
?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<!-- Programm zur Euro-Dollar Umrechnung
    Teil Das grosse Ganze
    anmeldung
    Dateiname: grosseGanze2/anmeldung.php //-->
<html>
<head>
    <title> Anmeldung </title>
    <script language = "JavaScript"
        src = "../javascript/feldkontrolle.js">
    </script>
    <script language = "JavaScript">
        function teste()
        {
            if(istLeer(document.anmeldung.name,
                "Kein Name eingegeben"))
            {
                return;
            }
            if(istLeer(document.anmeldung.passwort,
                "Kein Passwort eingegeben"))
            {
                return;
            }
            document.anmeldung.method="post";
            document.anmeldung.action=document.anmeldung.action.value;
            document.anmeldung.submit();
        }
    </script>

```

```

</head>
<body>
  <table border=2 align="center">
    <tr>
      <td>
        
      </td>
      <td>
        Die Schwanen Gesellschaft
      </td>
    </tr>
  </table>
  <?php
  if ($REQUEST_METHOD!="POST" )
  {
?>
    <h2 align="center">
      Anmeldung
    </h2>
    <hr>
    <form name="anmeldung">
      <input type="hidden" name="action"
        value="<?php echo $PHP_SELF ?>" >
      <table border=2 align="center">
        <tr>
          <td>
            Benutzername:
          </td>
          <td>
            <input type="text" name="name" size="12">
          </td>
        </tr>
        <tr>
          <td>
            Passwort:
          </td>
          <td>
            <input type="text" name="passwort" size="12">
          </td>
        </tr>
        <tr>
          <td colspan="2" align="center">
            <input type="button" value="Anmelden" onClick="teste()">
          </td>
        </tr>
      </table>
    </form>
  <?php
  }
  else
  {
    if (!DbFunctions::connect($link))
    {
      echo "Fehler";
    }
  }
}

```

```

        echo "</body>";
        echo "</html>";
        exit();
    }
    if(Mitarbeiter::passwordIstOkay($link, $name, $password))
    {
        $benutzerName=$name;
        session_register("benutzerName");
        loadPage("anwendungen.php");
    }
    else
    {
        displayAlert("Passwort-Benutzernamen-Kombination stimmt nicht!");
        loadPage($PHP_SELF);
        exit;
    }
}
?>
</body>
</html>

```

Als erste Anweisung starten wir eine Session. Danach importieren wir die von anmeldung.php benötigten Dateien:

```

<?php
    session_start();
    require_once("../classes/Mitarbeiter.inc.php");
    require_once("../classes/DbFunctions.inc.php");
    require_once("../includes/oftBenutzteFunktionen.inc.php");
?>

```

Den Aufbau des Formulars, die JavaScript-Eingabeüberprüfungen und die Methode festzustellen, ob die php-Datei das erste Mal aufgerufen wurde oder als Folge des Abschickens des Formulars, dies alles kennen Sie bereits, so dass ich auf eine erneute Diskussion verzichte.

Wenn anmeldung.php als Folge der Übertragung des Formulars aufgerufen wird, wird zunächst die Verbindung zur Datenbank aufgebaut:

```

    if(!DbFunctions::connect($link))
    {
        echo "Fehler";
        echo "</body>";
        echo "</html>";
        exit();
    }

```

Funktioniert das, ist die Verbindung auf der Variablen \$link abgespeichert, funktioniert das nicht, gibt es eine Fehlermeldung und die Anwendung beendet sich.

Nach der Erstellung der Verbindung zur Datenbank wird die Methode passwordIstOkay() der Mitarbeiterklasse genutzt.

```

if(Mitarbeiter::passwortIstOkay($link, $name, $passwort))
{
    $benutzerName=$name;
    session_register("benutzerName");
    loadPage("anwendungen.php");
}
else
{
    displayAlert("Passwort-Benutzernamen-Kombination stimmt nicht!");
    loadPage($PHP_SELF);
    exit;
}

```

War die Anmeldung erfolgreich, wird der Benutzername als Session-Variable registriert:

```
session\_register("benutzerName")
```

Die Seite mit den erlaubten Anwendungen wird geladen (loadPage("anwendungen.php")). Bei erfolgloser Anmeldung wird die dementsprechende Fehlermeldung dargestellt und die Seite ruft sich selber wieder auf, um eine Neueingabe der Anmelde Daten zu gestatten (loadPage(\$PHP_SELF)).

Abb. 10.18 zeigt dies anhand der Abläufe im Browser. Beachten Sie die Ähnlichkeit mit Abb. 10.16.

Die Datei anwendungen.php

Diskutieren wir nun kurz die Seite anwendungen.php:

Beispiel 10.12 Die Seite Anwendungen

```

<?php
    session_start();
    require_once("../classes/Mitarbeiter.inc.php");
    require_once("../classes/DbFunctions.inc.php");
    require_once("../includes/oftBenutzteFunktionen.inc.php");
?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<!-- Programm zur Euro-Dollar Umrechnung
    Teil Das grosse Ganze
    anwendungen
    Dateiname: grosseGanze2/anwendungen.php //-->
<html>
<head>
    <title> Anwendungen </title>
</head>
<body>
    <table border=2 align="center">
        <tr>
            <td>
                
            </td>
            <td>
                Die Schwanen Gesellschaft
            </td>
        </tr>
    </table>

```

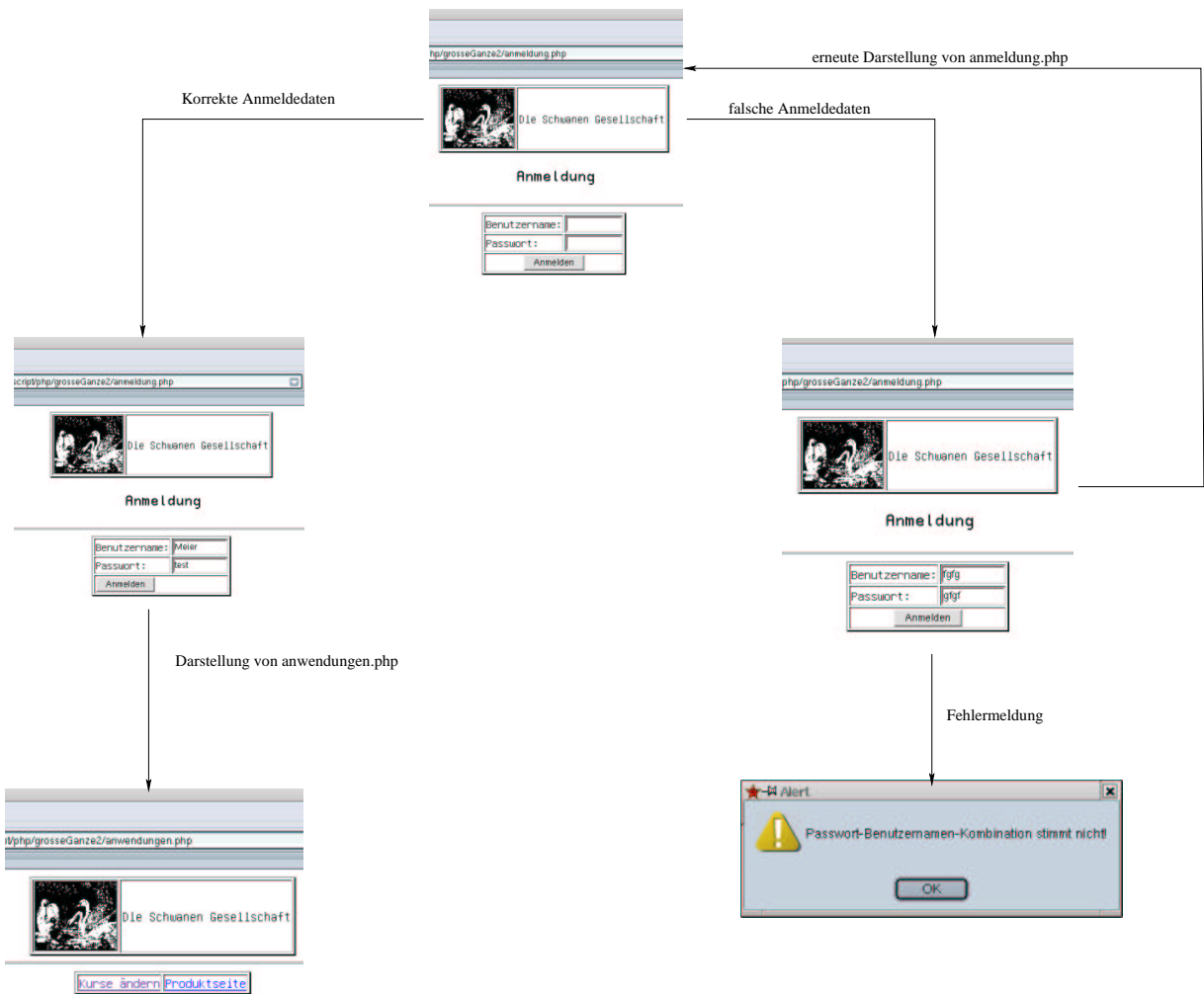


Abbildung 10.18: Ablauf bei der Anmeldung im Browser

```

        </td>
      </tr>
    </table>
    <hr>
<?php
    if(!session_is_registered("benutzerName"))
    {
        displayAlert("Nicht angemeldet");
        loadPage("anmeldung.php");
        exit();
    }
?>
<table border=2 align="center">
<tr>
    <td> <a href= "./kurspflege.php"> Kurse &auml;ndern </a> </td>
    <td> <a href= "./produkte.php"> Produktseite </a> </td>
</tr>

```

```
</table>
</body>
</html>
```

Da diese Seite vor unbefugten Benutzern geschützt werden soll, wird auch hier zunächst eine Session gestartet:

```
<?php
    session_start();
```

Die Überprüfung, ob ein Benutzer angemeldet ist, und damit die Seite benutzen darf, findet durch folgende Zeilen statt:

```
    if(!session_is_registered("benutzerName"))
    {
        displayAlert("Nicht angemeldet");
        loadPage("anmeldung.php");
        exit();
    }
```

Wenn die Variable `benutzerName` innerhalb der Session registriert ist, werden obige Zeilen nicht ausgeführt. Dies ist der Fall, wenn der Anwender sich erfolgreich angemeldet hat, denn dann wurde ja durch die Seite `anwendungen.php` eben diese Variable für die Session registriert. Ist die Variable jedoch nicht registriert (und das bedeutet, der Benutzer ist nicht angemeldet), wird in das `if`-Konstrukt verzweigt (beachten Sie das Ausrufungszeichen in der Bedingung des `if`-Konstrukts). Dort wird zunächst eine Fehlermeldung ausgegeben (`displayAlert("Nicht angemeldet")`), dann wird die Anmeldeseite geladen (`loadPage("anmeldung.php")`), um dem Benutzer die Chance zu geben, sich anzumelden.

Die Datei `kurspflege.php` und die Währungsdatei

Bevor wir uns die Seite `kurspflege.php` ansehen, müssen wir uns vorab einige Gedanken machen. Da wir die Anforderung (4) –alle Währungsfunktionen in einer eigenen Datei– realisieren müssen, müssen wir, bevor wir `kurspflege.php` selber realisieren, die Datei `euroDollarUmrechnung.inc.php` ändern. Denn dort müssen wir die Kursänderung (die diesbezüglichen Anweisungen stehen zur Zeit ja noch in `kurspflege.php`) realisieren. Darüber hinaus benennen wir die Datei um: Da wir hier alle unsere Währungsfunktionen sammeln, erzeugen wir eine neue Klasse und nennen diese Klasse `Wahrung`. Das macht es uns später auch vom Namen her einfacher, weitere Währungen zu integrieren. Desweiteren nehmen wir den Verbindungsaufbau zur Datenbank aus dieser Klasse heraus, dafür haben wir die Klasse `DbFunctions` erzeugt.

Damit sieht die neue Klasse `Wahrung.inc.php` folgendermaßen aus:

Beispiel 10.13 Die Klasse `Wahrung`

```
<?php
require_once("./classes/DbFunctions.inc.php");
class Wahrung
{
    /*
    * class: Wahrung
```

```

* Filename: Waehrung.inc.php
* Directory: classes
* lastChanged: 07.01.2002 by bb
* authors: Bernd Bluemel
* purpose: Funktionen zur Dollar-Euro
*          oder Euro-Dollar Umrechnung
*/

function holeDollarKurs($link)
{
    $query="Select kurs from waehrung where name='Dollar' ";
    $kurs=DbFunctions::getFirstFieldOfResult($link, $query);
    return $kurs;
}
function euroDollarUmrechnung($link, $zielwaehrung, $betrag)
{
    $kurs=Waehrung::holeDollarKurs($link);
    if(($zielwaehrung=="Dollar")||($zielwaehrung=="dollar"))
    {
        $dollarbetrag=$kurs*$betrag;
        return $dollarbetrag;
    }
    if(($zielwaehrung=="Euro")||($zielwaehrung=="euro"))
    {
        $eurobetrag=(1/$kurs)*$betrag;
        return $eurobetrag;
    }
}
function erzeugeEuroDollarUmrechnungsScript($link)
{
    $kurs=Waehrung::holeDollarKurs($link);
    echo "<script language=\"JavaScript\">\n";
    echo "function euroDollarUmrechnung(zielwaehrung, betrag)\n";
    echo "{\n";
    echo "    var kurs=$kurs;\n";
    echo "    if((zielwaehrung==\"Dollar\")||(zielwaehrung==\"dollar\"))\n";
    echo "{\n";
    echo "        dollarbetrag=kurs*betrag;\n";
    echo "        return(dollarbetrag);\n";
    echo "    }\n";
    echo "    if((zielwaehrung==\"Euro\")||(zielwaehrung==\"euro\"))\n";
    echo "{\n";
    echo "        eurobetrag=(1/kurs)*betrag;\n";
    echo "        return(eurobetrag);\n";
    echo "    }\n";
    echo "}\n";
    echo "</script>\n";
}

function aendereDollarKurs($link, $kurs)
{
    $updateQuery ="update waehrung set kurs='$kurs' ";
    $updateQuery.="where name='Dollar' ";
    DbFunctions::executeQuery($link, $updateQuery);
    return;
}

```

```

    }
}
?>

```

Hier ist eigentlich nur die Funktion `aendereDollarKurs()` neu. Sie erwartet zwei Parameter: `$link` und `$kurs`. Auf `$link` ist, wie immer, die Verbindung zur Datenbank abgespeichert, `$kurs` enthält den neuen Dollarkurs. Die Funktion an sich enthält nichts Neues. Wir haben nur die bislang in `kurspflege.php` beheimateten Befehle in unsere Währungsklasse ausgelagert.

Die weitere Veränderung gegenüber der alten Datei `euroDollarUmrechnung.inc.php` besteht nun darin, dass wir eine Klasse erzeugt haben und jeder Funktion dieser neuen Klasse die Verbindung zur Datenbank (`$link`) zusätzlich übergeben. Die Nutzung von `Waehrung.inc.php` setzt also eine bestehende Verbindung zur Datenbank voraus.

Betrachten wir nun die Seite `kurspflege.php`:

Beispiel 10.14 Die Seite Kurspflege

```

<?php
    session_start();
    require_once("../classes/DbFunctions.inc.php");
    require_once("../classes/Waehrung.inc.php");
    require_once("../includes/oftBenutzteFunktionen.inc.php");
?>
<!-- Programm zur Euro-Dollar Umrechnung
    Teil Das grosse Ganze
    Kurspflege
    Dateiname: grosseGanze2/kurspflege.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
    <title> Euro-Dollar Umrechnung Kurspflege</title>
    <script language = "JavaScript"
        src = "../javascript/feldkontrolle.js">
    </script>
    <script language = "JavaScript"
        src = "../javascript/stringbearbeitung.js">
    </script>

    <script language = "JavaScript">
        function teste()
        {
            document.kurspflege.kurs.value=komma2punkt(
                document.kurspflege.kurs.value);

            if(istKeineZahl(document.kurspflege.kurs,
                "Kurs ist keine Zahl"))
            {
                return false;
            }
            document.kurspflege.method="post";
            document.kurspflege.action=document.kurspflege.action.value;
            document.kurspflege.submit();
        }
    </script>

```

```

    </script>
</head>
<body>
<?php
    if(!session_is_registered("benutzerName"))
    {
        displayAlert("Nicht angemeldet");
        loadPage("anmeldung.php");
        exit();
    }
?>
    <table border=2 align="center">
        <tr>
            <td>
                
            </td>
            <td>
                Die Schwanen Gesellschaft
            </td>
        </tr>
    </table>
    <h2 align="center">
        Kurspflege
    </h2>
    <hr>
<?php
    // Wir pruefen ob die Anfrage ueber get oder post erfolgte
    if($REQUEST_METHOD!="POST")
    {
        // erster Aufruf, das Formular muss praesentiert werden
    ?>
        <form name='kurspflege'>
            <input type="hidden" name="action"
                value="<?php echo $PHP_SELF ?>" >
            <table border align="center">
                <tr>
                    <td>
                        Neuer Kurs
                    </td>
                    <td>
                        <input type="text" name="kurs" size=12>
                    </td>
                </tr>
                <tr>
                    <td colspan="2" align="center">
                        <input type="button" name="Button1" onClick="teste()"
                            value="Abschicken">
                    </td>
                </tr>
            </table>
        </form>
<?php
    }

```

```

else
{
    if(!DbFunctions::connect($link))
    {
        echo "Fehler";
        echo "</body>";
        echo "</html>";
        exit();
    }
    // update sql-Kommando
    Waehrung::aendereDollarKurs($link, $kurs);
    echo "<p align=\"center\">";
    echo "Kurs ge&auml;ndert auf $kurs.";
    echo "</p>";
}
?>
</body>
</html>

```

Neu ist zunächst das Starten der Session und der Import der Dateien, deren Funktionen wir nutzen wollen:

```

<?php
    session_start();
    require_once("../classes/DbFunctions.inc.php");
    require_once("../classes/Waehrung.inc.php");
    require_once("../includes/oftBenutzteFunktionen.inc.php");
?>

```

Wie bei `anwendungen.php` überprüfen wir dann im Body als Erstes, ob die Seite von einem angemeldeten Benutzer aufgerufen wurde:

```

<?php
    if(!session_is_registered("benutzerName"))
    {
        displayAlert("Nicht angemeldet");
        loadPage("anmeldung.php");
        exit();
    }
?>

```

Auch hier wird verhindert, dass nicht angemeldete Benutzer die Seite nutzen dürfen. Nicht angemeldete Benutzer werden auf die Anmeldeseite zurückgeleitet.

Die nächste Änderung ergibt sich im `else`-Teil. Da die neue Klasse `Waehrung` eine Verbindung zur Datenbank voraussetzt, schaffen wir diese:

```

if(!DbFunctions::connect($link))
{
    echo "Fehler";
    echo "</body>";
}

```

```

        echo "</html>";
        exit();
    }

```

Diese Zeilen sind uns ja bereits bekannt. Wir bauen mit der Methode connect() der Datenbankklasse DbFunctions eine Verbindung zur Datenbank auf. Funktioniert dies, ist die Verbindung auf \$link abgespeichert, ansonsten gibt es eine Fehlermeldung und das Programm wird beendet.

Nun bleibt nur noch die in Waehrung implementierte Methode aendereDollarKurs() aufzurufen:

```

    Waehrung::aendereDollarKurs($link, $kurs);

```

Ein Screenshot dieser Seite erübrigt sich, da sich in der Bildschirmdarstellung gegenüber Kapitel 10.3 keine Änderungen ergeben haben.

Die Datei produkte.php

Hier gibt es die wenigsten Änderungen. Diese Seite darf nicht vor Zugriffen der Benutzer geschützt werden. Schließlich sollen die ja was bestellen und da ist es eigentlich unsinnig, den Zugriff auf diese Seite zu reglementieren. Der einzige Unterschied zur Implementierung in Kapitel 10.3 besteht darin, dass wir die Verbindung zur Datenbank unter Zuhilfenahme von DbFunctions in produkte.php selber aufbauen und dann die Verbindung mit Hilfe der Variablen \$link an die aufgerufenen Methoden in der Waehrungsklasse übergeben. darüber hinaus müssen wir den Klassennamen (Waehrung) gefolgt von zwei Doppelpunkten vor den Methodenaufruf schreiben. Aber das kennen wir ja schon von den anderen Beispielen in diesem Kapitel. Damit ergibt sich als neue Implementierung von produkte.php:

Beispiel 10.15 Die Seite Produkte

```

<!-- Programm zur Euro-Dollar Umrechnung
    Teil Das grosse Ganze
    Produktseite mit Euro-Rechner
    Dateiname: grosseGanze2/produkte.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
    <title> Euro-Dollar Umrechnung: Produkte</title>
<?php
    require_once("./classes/Waehrung.inc.php");
    require_once("./classes/DbFunctions.inc.php");
    if(!DbFunctions::connect($link))
    {
        echo "Fehler";
        echo "</body>";
        echo "</html>";
        exit();
    }
    Waehrung::erzeugeEuroDollarUmrechnungsScript($link);
?>
<script language = "JavaScript"
    src = "./javascript/feldkontrolle.js">
</script>
<script language = "JavaScript"
    src = "./javascript/stringbearbeitung2.js">
</script>

```

```
<script language = "JavaScript">
function dollar2euro()
{
    document.euro6.dollar.value=komma2punkt
        (document.euro6.dollar.value);
    if(istKeineZahl(document.euro6.dollar,
        "Keine Zahl eingegeben!"))
    {
        return;
    }
    else
    {
        document.euro6.euro.value=
            schoeneAusgabe(euroDollarUmrechnung(
                "euro", document.euro6.dollar.value).toString());
    }
}
function euro2dollar()
{
    document.euro6.euro.value=komma2punkt
        (document.euro6.euro.value);
    if(istKeineZahl(document.euro6.euro,
        "Keine Zahl eingegeben!"))
    {
        return;
    }
    else
    {
        document.euro6.dollar.value=
            schoeneAusgabe(euroDollarUmrechnung(
                "dollar", document.euro6.euro.value).toString());
    }
}
</script>
</head>
<body>
<table border=2 align="center">
<tr>
<td>

</td>
<td valign="bottom">
<form name='euro6'>
<table border>
<tr>
<td>
        Dollar
    </td>
<td>
        <input type="text" name="dollar"
            onChange="dollar2euro()" size=12>
    </td>
</tr>
</table>
</td>
</tr>
</table>
```

```

        <tr>
            <td>
                Euro
            </td>
            <td>
                <input type="text" name="euro"
                    onChange="euro2dollar()" size=12>
            </td>
        </tr>
    </table>
</form>
</td>
</tr>
</table>
<h2 align="center">
    Die Schwanen Gesellschaft: Unsere Produkte
</h2>
<hr>
<table border="2" align="center">
<tr>
    <th> Produktname </th>
    <th> Preis in Euro </th>
    <th> Preis in Dollar </th>
</tr>
<tr>
    <td> Tolle Seife </td>
    <td align="center"> 2,00 </td>
    <td align="center">
        <?php
            echo(number_format(Waehrung::euroDollarUmrechnung($link,"Dollar",
                2),2," "," "));
        ?>
    </td>
</tr>
<tr>
    <td> Tolle Zahnpasta </td>
    <td align="center"> 3,50 </td>
    <td align="center">
        <?php
            echo(number_format(Waehrung::euroDollarUmrechnung($link,"Dollar",
                3.5),2," "," "));
        ?>
    </td>
</tr>
<tr>
    <td> Zahnseide </td>
    <td align="center"> 4,50</td>
    <td align="center">
        <?php
            echo(number_format(Waehrung::euroDollarUmrechnung($link,"Dollar",
                4.5),2," "," "));
        ?>
    </td>
</tr>

```

```

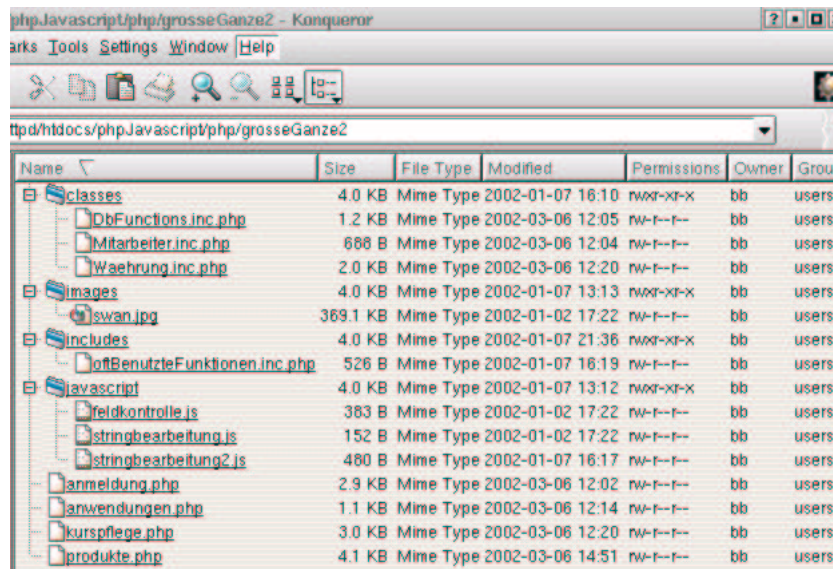
</table>
</body>
</html>

```

Ein Screenshot dieser Seite erübrigt sich, da sich in der Bildschirmdarstellung gegenüber Kapitel 10.3 keine Änderungen ergeben haben.

Zusammenfassung

Auch hier zeigt sich, dass Internet-Anwendungen nicht ganz einfach zu realisieren sind, sondern Nachdenken und Grundkenntnisse in Programmierung erfordern. Moderne, auf Inter- oder Intranettechnologien basierende Informationssysteme sind nicht "einfach eben so" realisierbar, sondern erfordern Sachverstand und auch Entwicklungszeit. Abb. ??eigt die die Dateistruktur der neuen Anwendung.



Name	Size	File Type	Modified	Permissions	Owner	Group
classes	4.0 KB	Mime Type	2002-01-07 16:10	rw-r-xr-x	bb	users
DbFunctions.inc.php	1.2 KB	Mime Type	2002-03-06 12:05	rw-r--r--	bb	users
Mitarbeiter.inc.php	688 B	Mime Type	2002-03-06 12:04	rw-r--r--	bb	users
Waehrung.inc.php	2.0 KB	Mime Type	2002-03-06 12:20	rw-r--r--	bb	users
images	4.0 KB	Mime Type	2002-01-07 13:13	rw-r-xr-x	bb	users
swan.jpg	369.1 KB	Mime Type	2002-01-02 17:22	rw-r--r--	bb	users
includes	4.0 KB	Mime Type	2002-01-07 21:36	rw-r-xr-x	bb	users
oftBenutzteFunktionen.inc.php	526 B	Mime Type	2002-01-07 16:19	rw-r--r--	bb	users
javascript	4.0 KB	Mime Type	2002-01-07 13:12	rw-r-xr-x	bb	users
feldkontrolle.js	383 B	Mime Type	2002-01-02 17:22	rw-r--r--	bb	users
stringbearbeitung.js	152 B	Mime Type	2002-01-02 17:22	rw-r--r--	bb	users
stringbearbeitung2.js	480 B	Mime Type	2002-01-07 16:17	rw-r--r--	bb	users
anmeldung.php	2.9 KB	Mime Type	2002-03-06 12:02	rw-r--r--	bb	users
anwendungen.php	1.1 KB	Mime Type	2002-03-06 12:14	rw-r--r--	bb	users
kurspflege.php	3.0 KB	Mime Type	2002-03-06 12:20	rw-r--r--	bb	users
produkte.php	4.1 KB	Mime Type	2002-03-06 14:51	rw-r--r--	bb	users

Abbildung 10.19: Dateistruktur der geänderten Anwendung