



Webdesign Guide

Programmieren einer eigenen Webanwendung mit PHP, MySQL, Smarty, Pear und ModRewrite

Inhalt:

1. Webanwendung Einführung (Seite 2)
2. die Arbeitsumgebung (Seite 3)
3. Awakening the giant (Seite 5)
4. ModRewrite und HtAccess (Seite 9)
5. User-Modul: Registrieren (Seite 12)
6. User-Modul: Login/Logout (Seite 21)
7. Modul: Kontaktformular und Dankesseite (Seite 24)
8. Eintragsformular und Webnotizen (Seite 29)
9. Ajax für das Volk (Seite 34)

Vorwort:

Die folgende Anleitung zur Programmierung einer Webanwendung zeigt nur eine Möglichkeit von vielen auf. Es gibt sicherlich andere Coder die besseren Programmcode erzeugen und sauberer programmieren, die eingesetzten Frameworks und Hilfsmittel müssen nicht zwangsweise die optimalen sein. Ich habe diesen Webdesign Guide geschrieben da man im Internet nur zu den einzelnen Teilen Tutorials und Anleitungen findet, nie aber etwas das die Programmierung des Ganzen behandelt. Darum hoffe ich das ich mit meinem Webdesign Guide diese Lücke etwas schließen konnte und andere, aus dieser umfangreichen Anleitung, einen Nutzen ziehen können.

Christian Strang, 22.01.07, <http://webdesignblog.de>

1. Webanwendung Einführung

Eine Web-Anwendung oder Web-Applikation ist ein Computer-Programm, das auf einem Webserver ausgeführt wird, wobei eine Interaktion mit dem Benutzer ausschließlich über einen Webbrowser erfolgt. Hierzu sind der Computer des Benutzer (Client) und der Server über ein Netzwerk, wie das Internet oder über ein Intranet miteinander verbunden, so dass die räumliche Entfernung zwischen Client und Server unerheblich ist.

Quelle: <http://de.wikipedia.org/wiki/Webanwendung>

Für mich ist eine Website als Webanwendung zu bezeichnen, wenn der Benutzer selbst die Möglichkeit hat Inhalte zu produzieren (und damit meine ich nicht nur das Schreiben eines Artikels), wenn der Benutzer die Möglichkeit hat mit der Website zu interagieren (Ändern des Profils, Einstellungen der Website) und wenn er generell einen Mehrwert hat, der nicht durch den Content erzeugt wird (bei Xing.com ist es das Networking, bei Netvibes.com die Organisation seiner Daten und bei Del.icio.us das abspeichern und taggen seiner Bookmarks).

Eine Webanwendung ist ein aus vielen Modulen bestehendes Ganzes, umso mehr Funktionalität die Website anbietet, desto größer der Programmieraufwand. Im Webdesign Guide möchte ich mich nicht auf genau eine Webanwendung spezialisieren, sondern vielmehr die Grundlage schaffen damit ihr viele unterschiedliche Webanwendungen und Ideen umsetzen könnt. Dazu zählt das Vermitteln um das Wissen bestimmter Techniken, ein Grundgerüst für die Webanwendung, Module für die Benutzerverwaltung (Registrieren, Login/Logout) und, damit man doch mal was sieht, das Erstellen von Content durch die Benutzer (nicht zu verwechseln mit einem CMS, die Funktionalität zeigt nur eine Möglichkeit von Vielen).

Eingesetzte Techniken

Ich verwende als Grundlage PHP und MySQL, dies hat folgende Gründe:

- PHP und MySQL sind weit verbreitet, werden häufig eingesetzt und haben sich im Bereich der Webprogrammierung etabliert
- Beide sind kostenlos und die meisten Webhoster bieten Apache Webserver mit MySQL-Datenbanken zu günstigen Preisen an
- Es gibt Unmengen an Tutorials und sehr viele hilfsbereite Forenuser, falls mal was nicht funktionieren sollte

Neben **PHP und MySQL** werden noch Teile der **PHP-PEAR**-Bibliothek (<http://pear.php.net>) eingesetzt, diese sollen uns beim Umgang mit PHP unterstützen und einiges an Arbeit abnehmen.

Template-Engine Smarty

Wir setzen als **Template-Engine Smarty** (<http://smarty.php.net>) ein, dadurch können wir den Code sauber vom Layout trennen. Smarty bietet von Haus aus ein starkes Caching-System, bevor die eigene Webanwendung unter einem Benutzeransturm zusammenbricht, kann man mit dem Cache noch einiges an Last abfangen.

Hallo Google

Und zuletzt wollen wir dann auch noch Google eine Freude machen und unsere Links suchmaschinen-optimiert ausgeben, dies erreichen wir mit **Mod Rewrite**.

Web 2.0

Als Schmankerl obendrauf zeige ich den Einsatz von Scriptaculous und Prototype um die Vorteile von Ajax nutzbar zu machen.

Quellen zur Einarbeitung

- PHP und MySQL 1. Einstiegskurs: <http://schattenbaum.net/php/>
- PHP und MySQL 2. Einstiegskurs: <http://tut.php-q.net/>
- Was ist Mod Rewrite: http://de.wikipedia.org/wiki/Mod_rewrite

- Was ist Smarty: <http://de.wikipedia.org/wiki/Smarty>
- Was ist PEAR: http://de.wikipedia.org/wiki/PHP_Extension_and_Application_Repository

2. Die Arbeitsumgebung

In diesem Teil richten wir uns unsere Arbeitsumgebung ein, wir bringen Smarty zum laufen und verschaffen uns einen Überblick über unser Datenbank-Schema (inklusive anlegen der benötigten Tabellen). Jetzt holt sich noch jeder schnell einen Kaffee (oder ein beliebiges Getränk nach eigener Wahl) und unternimmt dann folgende Schritte:

1. Vergewißert euch das euer bevorzugter Texteditor einsatzbereit ist. Wer lieber mit Dreamweaver oder GoLive arbeitet kann das gerne machen, aber es reicht ein einfacher Texteditor mit Syntax-Highlighting aus ("Notepad" tut es natürlich auch)
2. auf dem Server muss PHP mindestens in der Version 4.4.2 installiert sein (*damit hab ich es getestet, es geht sicherlich auch mit niedrigeren Versionen*) - kann man prüfen indem man die Datei "infophp.php" aufruft (wenn nicht vorhanden, vorher anlegen)
3. mit seinem Server-Administrations-Tool legt man nun eine Datenbank an und merkt sich **Benutzernamen und Passwort**
4. Ihr habt nun zwei Möglichkeiten an Smarty zu kommen: Ladet euch hier meine vorkonfigurierte Version runter http://webdesignblog.de/wp-content/uploads/2006/11/smarty_fertig.rar oder geht auf <http://smarty.php.net> unter downloads und ladet euch die aktuelle Version von Smarty herunter
5. Von PEAR verwenden wir vier Pakete, erstmal das Datenbank-Paket, dann das Autorisierungspaket, das HTML-Paket (für Quickform) und das Mail-Paket. Man kann die ganzen Funktionen auch ohne PEAR umsetzen, aber PEAR erleichtert einem die Arbeit schon erheblich, der Einarbeitungsaufwand lohnt sich also.

Die PEAR-Pakete bekommt ihr unter <http://pear.php.net>, schaut im Menü unter "**Downloads -> List Packages**":

- **Datenbank-Paket:** "Database -> DB -> Download" (ich verwende hier die Version DB-1.7.6)
- **Quickform-Paket:** "HTML -> HTML_Quickform -> Download" (ich verwende hier die Version HTML_QuickForm-3.2.7)
- **Auth-Paket:** "Authentication -> Auth -> Download" (ich verwende hier die Version Auth-1.4.2)
- **Mail-Paket:** "Mail -> Mail -> Download" (ich verwende hier die Version Mail-1.1.14)

Nun da wir alles beisammen haben (hoffentlich ist der Kaffee noch nicht kalt), entpacken wir erstmal die Rar-Dateien. Entpackt zunächst die "DB-1.7.6" als Ergebnis erhaltet ihr den Ordner "DB-1.7.6" kürzt alles weg ausser "DB". Das gleiche macht ihr mit dem Ordner "HTML_QuickForm-3.2.7" -> "HTML", "Auth-1.4.2" -> "Auth" und "Mail-1.1.14" -> "Mail". Die "package.xml" könnt ihr übrigens löschen. Ladet nun den Ordner "**DB**", den Ordner "**HTML**", den Ordner "**Auth**", den Ordner "**Mail**" und den Ordner "**smarty**" ins Root-Verzeichnis eurer Website, eure Ordnerstruktur sollte danach folgendermaßen aussehen:

Zusätzlich brauchen wir aus dem PEAR-Paket "HTML_Common" (List Packages -> HTML -> HTML_Common -> Download) noch die Datei "Common.php" und diese kopieren wir in unser HTML-Verzeichnis.

- root
- Auth
- DB

- HTML
- Common.php
- Mail
- smarty

Da wir gerade beim einrichten sind: Legt im Root noch zusätzlich eine Datei “**index.php**” und eine Datei “**.htaccess**” an. Die bleiben erstmal leer, der Inhalt kommt später.

Wechselt nun in den “smarty”-Ordner und weist den beiden Unterordnern “templates_c” und “cache” die Rechte “777” zu.

Wie ist unsere Datenbank aufgebaut

Unsere Datenbank kommt mit 8 Tabellen in ihrer Standard-Konfiguration aus (möchte man die Webanwendung später erweitern, werden wohl noch zusätzliche Tabellen hinzukommen), diese Tabellen (inklusive ihrer Spalten) sind:

- **Tabelle user** -> id_user, username, pass, email, active, regdate, activation_key
- **Tabelle post** -> id_post, title, content, xpos, ypos
- **Tabelle post_user** -> id_user, id_post

Und nun folgt der SQL-Code den ihr benötigt, um eure MySQL-Datenbank mit den richtigen Tabellen zu füllen:

/ START SQL-CODE*/*

```
CREATE TABLE `post` (
  `id_post` int(10) unsigned NOT NULL auto_increment,
  `title` varchar(255) NOT NULL default '',
  `content` text NOT NULL,
  `xpos` int(11) NOT NULL default '0',
  `ypos` int(11) NOT NULL default '0',
  PRIMARY KEY (`id_post`)
) TYPE=MyISAM;

CREATE TABLE `post_user` (
  `id_user` int(10) unsigned NOT NULL default '0',
  `id_post` int(10) unsigned NOT NULL default '0'
) TYPE=MyISAM;

CREATE TABLE `user` (
  `id_user` int(10) unsigned NOT NULL auto_increment,
  `username` varchar(100) NOT NULL default '',
  `pass` varchar(255) NOT NULL default '',
  `email` varchar(255) NOT NULL default '',
  `active` tinyint(1) NOT NULL default '0',
  `regdate` int(11) unsigned NOT NULL default '0',
  `activation_key` varchar(30) NOT NULL default '',
  PRIMARY KEY (`id_user`)
) TYPE=MyISAM;
```

/ ENDE SQL-CODE*/*

Damit sollten wir erstmal alle Tabellen, die wir benötigen, angelegt haben. Somit ist die Arbeitsumgebung fertig eingerichtet, wir haben mit den PEAR-Paketen einige Bibliotheken die unsere Arbeit gerade mit Formularen und dem Zugriff auf die Datenbank erheblich vereinfachen, die Template-Engine Smarty hilft uns dabei Code und Layout voneinander zu trennen und die eben angelegte Datenbank wird die Eingaben unserer Benutzer speichern und jederzeit wieder abrufbar machen.

Nützliche Quellen:

- **DBManager** - Der DBManager ist kostenlos und mit ihm kann man ganz einfach ER-Diagramme erstellen, mittels

eines ER-Diagramms kann man sich selbst und anderen ganz schnell die eigene Datenbankstruktur visualisieren:
<http://www.dvdsoft.info/dbmanager.htm>

- **MySQL-Performance:** http://www.schefter.net/deutsch/techtalk/mysql_performance.html

- **MySQL-Performance-TUNING:** http://blog.koehntopp.de/uploads/PT_isotopp_deutsch.pdf

3. Awakening the giant

In diesem Teil des Webdesign Guide geht es darum die erste lauffähige Version unserer Webanwendung zu programmieren. Dazu kommen wir natürlich nicht umher die Grundstruktur in unserer index.php zu definieren und zum ersten mal auf Smarty zuzugreifen.

Index.php - Struktur und Aufbau unserer Engine

Bevor wir uns nun mit dem Grundgerüst unserer Webanwendung beschäftigen brauchen wir erstmal noch zwei weitere Ordner in unserem Root-Verzeichnis, dabei handelt es sich um den Ordner **“filesystem”** und um den Ordner **“includes“**.

Demnach sollte unser Server nun folgende Struktur haben:

- root
- Auth
- DB
- filesystem
- HTML
- includes
- Mail
- smarty
- .htaccess
- index.php

Im Ordner **“includes”** legen wir nun die Datei **“config.php”** an die brauchen wir später um unsere Verbindungsdaten abzulegen. Dann wechseln wir in folgenden Unterordner: **“smarty -> templates”** und erstellen darin zwei leere Dateien mit Namen:

- **header.tpl**
- **footer.tpl**

Jetzt und erst jetzt fügt ihr folgenden Code in eure Index.php ein (die Erklärung zum Code und warum dann trotzdem noch nichts angezeigt wird, erfolgt danach):

/ Start Index.php */*

```
<?php
require_once('smarty/libs/Smarty.class.php');
require_once('includes/config.php');

require_once('Auth/Auth.php');
require_once('DB/DB.php');
```

```
$smarty = new Smarty;
$smarty->template_dir = "smarty/templates/";
$smarty->config_dir = "smarty/configs/";
$smarty->cache_dir = "smarty/cache/";
$smarty->compile_dir = "smarty/templates_c/";

$_action = isset($_REQUEST['action']) ? $_REQUEST['action'] : 'index';

switch($_action) {

    case 'register':
        require_once('includes/register.php');
        break;

    case 'profile':
        require_once('includes/profile.php');
        break;

    case 'login':
    case 'logout':
        require_once('includes/login.php');
        break;

    case 'contact_mail':
        require_once('includes/mail.php');
        break;

    case 'password_lost':
        require_once('includes/password_lost.php');
        break;

    case 'thanks':
        require_once('includes/thanks.php');
        break;

    case 'index':
    default:
        break;
}

$smarty->display('header.tpl');

switch($_action) {

    case 'register':
        $smarty->display('regform.tpl');
        break;

    case 'profile':
        $smarty->display('profile.tpl');
        break;

    case 'login':
    case 'logout':
        $smarty->display('loginform.tpl');
        break;

    case 'contact_mail':
        $smarty->display('contact_mail.tpl');
        break;

    case 'password_lost':
        $smarty->display('password_lost.tpl');
        break;

    case 'thanks':
        $smarty->display('thanks.tpl');
        break;

    case 'index':
    default:
        $smarty->display('index.tpl');
        break;
}
```

```

}

$smarty->display('footer.tpl');

?>

```

/ Ende Index.php */*

Wer nun seine Domain aufruft wird feststellen, das noch nichts ausgegeben wird und das ist auch gut, denn dann wurde (hoffentlich) alles richtig eingerichtet. Sollte eine Fehlermeldung ausgegeben werden dann bekommt man meist genug Informationen über die Fehlermeldung um den Fehler selbst zu beheben, häufig liegt es daran das eine Datei vergessen oder ein Verzeichnisname falsch geschrieben wurde.

Nun gut, gehen wir mal den Quellcode durch:

```

require_once('smarty/libs/Smarty.class.php');
require_once('includes/config.php');
require_once('Auth/Auth.php');
require_once('DB/DB.php');

```

In diesem Part laden wir alle benötigten Klassen in unser Projekt, neben der Smarty-Klasse und zwei Klassen aus der PEAR-Bibliothek (Auth und DB) fügen wir zudem noch eine Konfigurationsdatei in unser Projekt ein, bislang ist die Datei noch leer, aber später werden wir darin die Verbindungsdaten zu unserer Datenbank definieren und noch andere “Kleinigkeiten” die in unserem Projekt global zum Einsatz kommen.

```

$smarty = new Smarty;
$smarty->template_dir = "smarty/templates/";
$smarty->config_dir = "smarty/configs/";
$smarty->cache_dir = "smarty/cache/";
$smarty->compile_dir = "smarty/templates_c/";

```

Um Smarty zum Laufen zu kriegen reicht es nicht aus nur die Smarty-Klasse einzubinden (wie oben geschehen), man muss zusätzlich eine Instanz der Smarty-Klasse erzeugen (mittels: “**\$smarty = new Smarty**“) und dieser Instanz sagen wo sich innerhalb unserer Struktur welcher Ordner befindet. Smarty muss wissen wo sich folgende Ordner befinden:

- **Template Ordner** - hier werden später die Templates abgelegt, die wir in der index.php dann aufrufen
- **Config Ordner** - erstmal nicht weiter von Interesse
- **Cache Ordner** - habe selbst noch nicht herausgefunden wozu dieser dient, denn die gecachten Dateien werden eigentlich im Ordner **Templates_c** abgelegt
- **Templates_c Ordner** - hier liegen die gecachten Dateien

```

$_action = isset($_REQUEST['action']) ? $_REQUEST['action'] : 'index';

```

Hiermit steuern wir unsere Webanwendung, je nachdem welcher Wert hier übergeben wird, wird dazu die betroffene Seite ausgegeben. Den Wert für die Seite (also den “Action”-Wert) erhalten wir später direkt über htaccess und unserer Mod-Rewrite-Engine.

Den restlichen Quellcode erläutere ich nun im gesamten und dieser ist essentiell um die Zusammenarbeit zwischen Smarty und PHP zu verstehen, also **aufgepasst**:

Bevor wir mit “**\$smarty->display()**” eine Datei anzeigen können, müssen wir erstmal alle benötigten PHP-Dateien eingebunden haben, darum habe ich die Engine der Webanwendung in zwei Teile gesplittet, diese werden durch die beiden “switch”-Blöcke dargestellt. Der erste “switch”-Block ist dafür da, die richtige PHP-Datei in unsere Webanwendung einzubinden und der Wert dafür steht in der Variablen “**\$_action**”. Beispielsweise: ruft man die eigene Webanwendung unter folgender URL auf “<http://domain.de/index.php?action=profil>” so erhält die Variable **\$_action** den Wert “profil”. Diese URL-Form wird allerdings später durch eine suchmaschinen-optimierte Variante ausgetauscht, doch das passiert erst im 4. Part - **ModRewrite und HtAccess**.

Bevor ich zum zweiten Teil der Engine komme, der für die eigentliche Ausgabe der Website durch Smarty zuständig

ist, noch eine kurze Erklärung für die Splittung: Bevor auch nur eine Ausgabe mittels Smarty erfolgt muss jeglicher PHP-Kram abgearbeitet sein, denn sonst kann es schnell passieren das man einen ärgerlichen “headers already sent” Fehler erhält (<http://webdesignblog.de/sonstiges/cannot-modify-header-information-headers-already-sent-gehasster-php-fehler/>).

Im zweiten “switch”-Block nutzen wir Smarty um, abhängig vom Wert der Steuerungs-Variablen “\$_action”, die benötigten Templates auszugeben. Erhält die Steuerungsvariable “\$_action” also den Wert “**profile**” dann wird im ersten “switch”-Block die Datei “profile.php” eingebunden (diese Datei existiert noch nicht und erzeugt daher auch eine Fehlermeldung bei einem manuellen Aufruf im Browser). Im zweiten “Switch”-Block wird dann das dazugehörige Template an den Ausgabe-Teil der Smarty-Engine geschickt und dieser, wenn alles glatt läuft, im Browser angezeigt (funktioniert hier noch nicht da die Dateien noch nicht vorhanden sind). So können wir beliebig viele Module für unsere Webanwendung erstellen und durch ein einfaches hinzufügen in beiden “switch”-Blöcken die Funktionalität unserer Webanwendung erweitern.

Das war es jetzt erstmal mit der Index.php, beschäftigen wir uns nun kurz mit den beiden Templates “header.tpl” und footer.tpl”.

Template-Spielereien - Modifizieren von header.tpl und footer.tpl

Um diese beiden Dateien zu modifizieren wechseln wir in das Smarty-Template-Verzeichnis “smarty -> templates” (vom Root aus). Wir schreiben in die **header.tpl** folgenden Code:

/ Start header.tpl */*

```
<html>
<head>
    <title>Meine Webanwendung</title>
</head>
<body>
```

/ Ende header.tpl */*

und in die footer.tpl diesen hier:

/ Start footer.tpl */*

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

/ Ende footer.tpl */*

Da der Code recht unspektakulär ist, werde ich nicht weiter darauf eingehen.

Rufen wir nun unsere Webanwendung auf sehen wir bis auf einen Titel keine Änderung, ein kurzer Blick in den Quellcode zeigt uns aber, das wir zumindest schonmal einen Rahmen für unsere Website haben.

Hello World

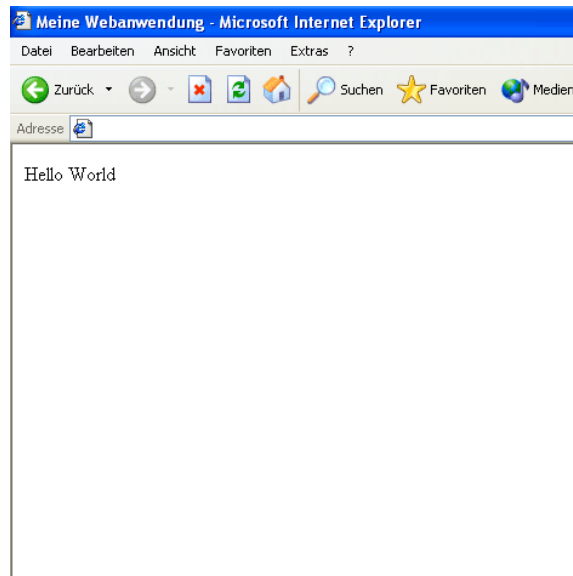
Wie schon versprochen wird unsere erste Ausgabe die in Programmiererkreisen typische “Hello World”-Ausgabe sein. Dazu wechselt nochmal schnell in die index.php und sucht im zweiten “switch”-Block nach der Zeile:

```
// $smarty->display('index.tpl');
```

Löscht die beiden “Slash”-Zeichen am anfang der Zeile die den Rest der Zeile auskommentieren, speichert die Datei und wechselt dann in den Smarty-Template-Ordner. Dort erstellt ihr nun die Datei “**index.tpl**” und fügt folgenden Inhalt ein:

```
Hello World
```

Ruft ihr nun eure Webanwendung auf, sollte euch die Kombination “Hello World” ausgegeben werden.



Das war es für diesen Part, doch was haben wir erreicht? Lediglich die Ausgabe von zwei Wörtern im Browser? Auf den ersten Blick, ja! Doch wenn man nochmal unter die Haube schaut sehen wir dort eine voll-funktionstüchtige Smarty-Engine arbeiten, ein leicht erweiterungsfähiges Grundgerüst (index.php) mit der Möglichkeit spätere Verweise/Links suchmaschinenoptimiert darzustellen. Wir haben nun alles angelegt um voll durchstarten zu können, unsere Ordnerstruktur steht und alle benötigten Klassen sind schon eingebunden. In diesem Teil haben wir unserer Webanwendung Leben eingehaucht und im übernächsten wird unser Baby seine ersten Schritte wagen. Doch bevor es soweit ist widmen wir uns htaccess und Mod Rewrite, das zögert den Spaß zwar heraus, wird uns aber im späteren Einsatz enorm nützlich sein.

4. ModRewrite und HtAccess

Warum ModRewrite?

Bevor ich darauf eingehe was ModRewrite ist, erstmal warum wir es brauchen: Wenn man mit PHP programmiert, so will man häufig die Funktion in einem Skript mit einem gewissen Wert ausführen, zum Beispiel wenn die Webanwendung ein Menü hat, dann kann man das folgendermaßen realisieren:

```
<ul>
<li><a href="index.php?menue=profile">Profil</a></li>
<li><a href="index.php?menue=send_message">Nachricht verschicken</a></li>
<li><a href="index.php?menue=user_overview">Benutzerübersicht</a></li>
</ul>
```

Wird einer dieser Links angeklickt, wird das Skript mit Namen “index.php” aufgerufen und die Variable, mit Namen “menue”, mit einem Wert belegt (entweder “profile”, “send_message” oder “user_overview”). Diesen Wert kann man nun im Zielskript (index.php) abfangen und verwerten. Sprich, wird der Menüpunkt “Profil” angeklickt, zeigt man auf der Folgeseite das Profil des angemeldeten Benutzers an usw.

Nun funktioniert das alles ganz gut, doch handelt es sich bei solchen Links (<index.php?menue=abcxyz>) nicht um

suchmaschinenoptimierte URL's. Dabei ist diese Variante noch harmlos, es geht noch krasser (<index.php?wert1=dhdh&wert2=fjjjf&wert3=3663&wert4=sjj223&wert5=...>).

Wenn diese "PHP-URL's" nicht zu empfehlen sind, was will dann Google und die anderen haben? Sowas hier:

```
<ul>
<li><a href="/profile">Profil</a></li>
<li><a href="/send_message">Nachricht verschicken</a></li>
<li><a href="/user_overview">Benutzerübersicht</a></li>
</ul>
```

Erkennt man die Änderung? Wer einen Blog laufen hat der suchmaschinenoptimierte Links unterstützt (*Permalinks*), dem kommt diese Form wahrscheinlich bekannter vor, das wollen wir auch für alle Links in unserer Webanwendung erreichen und dazu brauchen wir ModRewrite und htaccess.

Was ist ModRewrite?

Mod Rewrite ist ein Teil der Rewrite Engine vom Apache Webserver. Mit dieser ist es möglich suchmaschinenoptimierte Links, wie wir sie oben im Beispiel vorfinden, wieder nach PHP-fähige Links umzuschreiben, damit der Server auch weiß welches Skript er aufrufen muss und welche Variablen & Werte er an dieses übergeben soll. Der Apache Server braucht PHP-typische URL's und kommt mit unseren optimierten Links nicht zurecht, darum schleusen wir alle Links, bevor sie an den Webserver zur Verarbeitung kommen, erstmal durch unsere Rewrite Engine.

ModRewrite einrichten

1. **Prüfen ob Mod Rewrite aktiviert ist** - Als erstes sollten wir natürlich prüfen ob unser Webserver das überhaupt kann, manche haben dieses Modul nämlich nicht aktiviert, das kommt allerdings eher selten vor. Dazu verbinden wir uns mit unserem Webserver und legen im Root-Verzeichnis eine Datei mit dem Namen: "**infophp.php**" an. In diese fügen wir folgenden Quellcode ein:

```
<?php
PHPINFO();
?>
```

Danach speichern, die Datei wieder hochladen und im Browser diese PHP-Datei aufrufen. Nun sollte euch angezeigt werden, wie euer Apache Server konfiguriert ist und was er alles kann. Sucht nun mit dem Suchtool des Browsers nach "**mod_rewrite**", sollte dieses unter dem Punkt "Loaded Modules" aufgeführt sein, dann ist Mod Rewrite bei euch aktiviert.

2. **.htaccess-Datei anlegen** - einfach im root-Verzeichnis eures Servers eine Datei mit folgendem Namen anlegen "**.htaccess**". An dem führenden Punkt vor dem Dateinamen erkennt man das es sich hierbei nicht um eine gewöhnliche PHP- oder HTML-Datei handelt, sondern um eine Datei die vom Apacher-Server gelesen und verwertet wird (bzw. von der Rewrite-Engine).
3. **.htaccess mit Inhalt füllen** - bevor wir die eigentlichen Rewrite-Regeln in unsere htAccess-Datei packen müssen wir erstmal wissen wie unsere Seitenstruktur aussehen wird. Wir haben uns schon in der index.php entschieden das alle wichtigen Module unmittelbar nach der Domain aufgerufen werden, also:
 - <http://domain.de/index.php?action=register> => **Modul-Registrieren**
 - <http://domain.de/index.php?action=profile> => **Modul-Profil**
 - usw.

Wir wollen also das unsere Links folgendermaßen aussehen:

<http://domain.de/profile>,

aber von der Rewrite-Engine ins folgendes Format umgeschrieben werden:

<http://domain.de/index.php?action=profile>

und wie das geht, steht man im folgenden Part.

Erste Mod_Rewrite-Regeln

Der Besucher unserer Webanwendung findet also folgenden Link vor:

```
<a href="/profile">Profil</a>
```

und erwartet das er nach dem anklicken auf der Profilseite landet. Damit das geschieht muss unsere Datei "**index.php**" für die Variable "**\$action**" den Wert "profile" erhalten, damit das so ist schreiben wir ans Ende unserer .htaccess-Datei folgenden Code:

```
RewriteBase /
RewriteRule ^([0-9a-zA-Z]+)$ index.php?action=$1
```

Zur Regel:

- **RewriteBase** sagt wo die URL-Umschreibung starten soll, hier kann man auch einen Unterordner angeben, wir wollen aber das ausgehend vom Root die URL's umgeschrieben werden
- **RewriteRule** sagt das unsere Regel jetzt startet
- **^** hier beginnt die Regel
- **\$** hier endet die Regel
- **([0-9a-zA-Z]+)** diese Zeichenkette sagt aus: akzeptiere jegliche Zeichenkombination bestehend aus: Zahlen (0-9), Kleinbuchstaben (a-z) und Großbuchstaben (A-Z). Das "+"-Zeichen dahinter besagt: akzeptiere einen oder mehrere Zeichen die der vorher angegebenen Zeichenkombination entsprechen. Mit dieser Regel wären folgende URL's möglich (aber nicht zwangsweise sinnvoll):
 - http://domain.de/asdaskjd
 - http://domain.de/ZEW RHWE
 - http://domain.de/838293
 - http://domain.de/fhjdDFsjdfDDF
 - http://domain.de/87SDsdsjkd83
- **index.php?action=\$1** dieser Bereich sagt aus: rufe die Datei "index.php" auf und übergib der Variablen "action" den Wert, welcher den Regeln entsprochen hat. Damit wird die URL erfolgreich umgeschrieben und auch der Wert an das Skript gesendet ohne das der Besucher davon etwas mitbekommt

Dies ist erst eine Rewrite-Regel mit der wir aber schon einen großen Effekt erzielen, alle Hauptmodule lassen sich nun, unter suchmaschinenoptimierten Bedingungen, aufrufen.

Kleiner Eingriff gegen Duplicate Content

Da wir schon gerade in der htaccess-Datei rumfuschen, können wir auch noch schnell einen Handgriff tätigen um "duplicate Content" innerhalb unserer eigenen Webanwendung zu verhindern. Den Code einfach an den Anfang der htaccess-Datei einfügen und "domain" durch euren Domainnamen ersetzen:

```
Options +FollowSymLinks
RewriteEngine on
RewriteCond %{HTTP_HOST} ^www.domain.de$ [NC]
RewriteRule ^(.*) http://domain.de/$1 [L,R=301]
```

Ohne diesen Code läßt sich die eigene Webanwendung mittels:

- **http://www.domain.de**

und

- **http://domain.de**

was dazu führen kann das auch der Google-Bot die Domain doppelt besucht und den Inhalt doppelt indexiert, was zwangsläufig zu duplicate Content führt und das wiederum zu einer eventuell schlechteren Suchmaschinen-Platzierung.

weiterführende Links

- http://www.webmaster-toolkit.com/mod_rewrite-rewriterule-generator.shtml - automatisch die Rewrite Regeln erzeugen lassen
- http://www.ilovejackdaniels.com/apache/mod_rewrite-cheat-sheet/ - dies ist das Mod Rewrite Cheat Sheet und enthält nahezu alles was mit Mod Rewrite möglich ist in einer geordneten übersichtlichen Form
- <http://httpd.apache.org/docs/2.0/misc/rewriteguide.html#ToC4> - Mod-Rewrite Guide mit vielen Beispielen und guten Erklärungen
- <http://www.modrewrite.de/> - Last but not least das **deutsche Mod_Rewrite-Forum**

5. User-Modul: Registrieren

Da bei einer Webanwendung meist der Benutzer im Vordergrund steht, wollen wir uns auch extra mit Modulen zum Anlegen und Verwalten von Benutzern beschäftigen. Folgende Module werden in den nächsten Teilen realisiert:

- **Registrieren** - Ohne Registrierung, keine Benutzer, zumindest keine validen. Wir wollen aber valide Benutzer, darum verlangen wir vom User eine gültige Emailadresse (an die ein Validierungskey geschickt wird). Wir werden zum ersten mal mit HTML-Quickform arbeiten (und sehen wie mächtig diese PEAR-Klasse ist) und Gäste unserer Webanwendung werden nun die Möglichkeit haben, sich zu registrieren und einen Account anzulegen
- **Login/Logout** - Ist der Account erstmal angelegt und aktiviert muss der User auch die Möglichkeit haben sich in diesen einzuloggen und die speziellen Features nutzen können, die ihm von unserer Webanwendung geboten werden. In diesem Modul verwenden wir die PEAR-Klasse “Auth“, mit ihr können wir einfach Sessions erzeugen und auf jeder Seite identifizieren ob der User noch eingeloggt ist. Zu jedem vernünftigen Login gehört selbstverständlich auch ein Logout.
- **Kontaktformular** - das hat nur sekundär etwas mit dem User zu tun, aber bislang bieten wir diesem noch keine Möglichkeit mit uns, dem Webmaster, Kontakt aufzunehmen. Über ein Kontaktformular werden unsere registrierten Besucher, aber auch alle anderen Gäste unserer Webanwendung, die Möglichkeit haben, ohne ihr eigenes Email-Programm zu öffnen, mit uns in Kontakt zu treten und dafür müssen wir unsere eigene Emailadresse nicht mal öffentlich machen (*Hinweis:* die eigene Emailadresse muss natürlich im Impressum stehen, aber dort kann man sie dann leicht vor Spammern schützen, indem man sie als Bild ausgibt)

Die Datenbankverbindung abspeichern

Bevor wir uns dem “Registrieren“-Modul widmen, müssen wir erst noch etwas Code in die Datei “config.php” packen.

```
<?php
define('DB_SYSTEM_1', 'mysql://username:password@localhost/datenbankname');
define('EMAIL_CONTACT', 'kontakt@domain.de');
?>
```

Was macht der Code? Ganz einfach: Bei “define” erzeugen wir eine Konstante, die wir in all unserem Quellcode verwenden können und PHP ersetzt dann überall wo wir die Konstante verwenden, diese mit ihrem Inhalt. In unserem Fall erzeugen wir als erstes die Konstante “DB_SYSTEM_1” und speichern dort unsere Datenbankverbindung rein. Nun können wir in all unseren Dateien die Konstante “DB_SYSTEM_1” verwenden und sollten wir irgendwann mal entweder die Datenbank wechseln oder das Passwort unserer Datenbank ändern, können wir das an **einer** Stelle machen und für die gesamte Webanwendung übernehmen. Nun aber zu den Verbindungseinstellungen:

mysql://username:password@localhost/datenbankname

- **mysql** - dies steht für den Typ der Datenbank, da wir die DB-Klasse von PEAR verwenden wird es uns ermöglicht auf **jeden** Typ von Datenbank zuzugreifen. Wir können also Problemlos später noch auf eine Oracle-Datenbank wechseln und müssen dies nur an dieser Stelle anpassen. Da wir hier aber von einer

- MySQL-Datenbank ausgehen, kann man dies so stehen lassen
- **username** - dies ersetzt ihr bitte durch euren Loginnamen, mit dem ihr euch in eure Datenbank einloggt
 - **password** - dürfte klar sein, das Passwort mit dem ihr euch zu eurer Datenbank verbindet
 - **localhost** - meist könnt ihr dies unverändert stehen lassen, nämlich dann wenn sich die Datenbank, die ihr verwendet, direkt auf eurem Server befindet. Ihr fragt euch ob sich die Datenbank auf eurem Server befindet? In diesem Fall könnt ihr getrost "localhost" stehen lassen ;)
 - **datenbankname** - entweder habt ihr beim anlegen eurer Datenbank selbst einen Namen für diese vergeben, oder euer Hoster hat das automatisch gemacht. Hier soll nicht der Loginname hin, aber oft sind Loginname und Datenbankname identisch

Der Inhalt der Konstanten "EMAIL_CONTACT" muss auch noch angepasst werden, ersetzt "domain" einfach durch euren jeweiligen Domainnamen, denkt aber auch daran das ihr die Emailadresse "kontakt@euer_domainname.de" auf eurem Server erzeugt.

Programmierung der Module

Ich teile jedes Modul in zwei Bereiche: PHP und Smarty. Erst werde ich den kompletten Quellcode ausgeben und ihn dann erläutern.

Modul - Registrieren - PHP

Als erstes wechseln wir in den Ordner "includes" und erzeugen dort die Datei "**register.php**". In diese packen wir dann den folgenden Code:

/ START - Quellcode register.php */*

```
<?php
require_once 'HTML/QuickForm.php';
require_once 'HTML/QuickForm/Renderer/ArraySmarty.php';

require_once 'smarty/libs/Smarty.class.php';

require_once 'DB/DB.php';
require_once 'includes/config.php';

//Prüfe ob Username schon existiert
function check_db_user($username)
{
    $dbh = DB::connect(DB_SYSTEM_1);
    $res =& $dbh->query("SELECT username FROM user WHERE username = '".$username.'");
    $dbh->disconnect();

    if($res->numRows() >= 1){
        return false;
    }else{
        return true;
    }
}

//Prüfe ob Email schon existiert
function check_db_email($email)
{
    $dbh = DB::connect(DB_SYSTEM_1);
    $res =& $dbh->query("SELECT email FROM user WHERE email = '".$email.'");
    $dbh->disconnect();

    if($res->numRows() >= 1){
        return false;
    }else{
        return true;
    }
}

//Erzeuge den Validierungscode
function validation_key($length=20){
    $randstr='';
    srand((double)microtime()*1000000);
```

```

    $chars = array ( 'a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q',
'r','s','t','u','v','w','x','y','z','1','2','3','4','5','6','7','8','9','0');
    for ($rand = 0; $rand <= $length; $rand++)
    {
        $random = rand(0, count($chars) -1);
        $randstr .= $chars[$random];
    }
    return $randstr;
}

//Prüfe die Registrierungsdaten und erzeuge den User in der Datenbank
function process_data_regForm($values) {

    $val_key = validation_key();
    $recipients = $values[email];
    $md5_user_pass = md5($values[pass]);

    $dbh = DB::connect(DB_SYSTEM_1);
    $time_value = time();
    $res =& $dbh->query("INSERT INTO user (username, pass, email, regdate, activation_key)
VALUES ('$values[username]', '$md5_user_pass', '$values[email]', '$time_value', '$val_key')");
    $dbh->disconnect();

    require_once 'Mail/Mail.php';

    $activation_link = 'http://domain.de/index.php?action=register&activate_user='.$val_key;

    $headers['From']    = EMAIL_CONTACT;
    $headers['To']      = $recipients;
    $headers['Subject'] = 'Hier steht der Titel mit dem die Nachricht des Nutzers verschickt
wird';

    $body = 'Dies ist dein Aktivierungslink: '.$activation_link;

    // Create the mail object using the Mail::factory method and send it
    $mail_object =& Mail::factory('mail');
    $mail_object->send($recipients, $headers, $body);

    //this has to be changed
    header("Location: /thanks/reg");
} //ENDE - function "process_data_regForm"

//Prüfe ob der User seinen Account aktivieren möchte
if(isset($_GET['activate_user'])){
    $user_activation_key = $_GET[activate_user];

    $sql = 'SELECT * FROM user WHERE activation_key = ?';

    $dbh = DB::connect(DB_SYSTEM_1);
    $res =& $dbh->getOne($sql, $user_activation_key);

    //check if activation key is in database
    //user_activation_key is not allowed to be 0, or all users get deleted
    if(($res != '') AND ($user_activation_key != '0')){
        $user_activation =& $dbh->query("UPDATE user SET activation_key = '0', active = '1'
WHERE id_user = $res");

        //thank user for his/her registration
        header("Location: /thanks/reg_thx");
    }
    else {
        //give user notice about that he is allready activated
        header("Location: /thanks/reg_thx_stop");
    }
    $dbh->disconnect();
}

else {

//Hier erzeugen wir die Elemente für das Registrierungsformular
$regForm = new HTML_QuickForm('regForm','post','http://domain.de/register','','',true);

```

```

$regForm->applyFilter('__ALL__', 'trim');

$regForm->addElement('text', 'username', 'Benutzername');
$regForm->addElement('password', 'pass', 'Passwort eingeben');
$regForm->addElement('password', 'pass2', 'Passwort wiederholen');
$regForm->addElement('text', 'email', 'Email');
$regForm->addElement('submit', 'btn_registration', 'Registrieren');

$regForm->addRule('username', 'Bitte Benutzername eingeben', 'required');
$regForm->addRule('pass', 'Bitte Passwort eingeben', 'required');
$regForm->addRule('pass2', 'Bitte Passwort eingeben', 'required');
$regForm->addRule('email', 'Bitte Emailadresse eingeben', 'required');

$regForm->addRule('username', 'Mindestlänge beträgt 3 Zeichen', 'minlength', 3);
$regForm->addRule('username', 'Maximale Länge beträgt 40 Zeichen', 'maxlength', 40);
$regForm->addRule('pass', 'Mindestlänge beträgt 5 Zeichen', 'minlength', 5);
$regForm->addRule('pass', 'Maximale Länge beträgt 20 Zeichen', 'maxlength', 20);
$regForm->addRule('email', 'Maximale Länge beträgt 60 Zeichen', 'maxlength', 60);

//die beiden Passwörter müssen übereinstimmen
$regForm->addRule(array('pass', 'pass2'), 'Die eingegebenen Passwörter müssen übereinstimmen',
'compare', null);
$regForm->addRule('email', 'Dies ist keine gültige Emailadresse', 'email');

//Hier werden die Funktionen aufgerufen und geprüft ob das eingebene Passwort oder der Benutzername
schon existieren
$regForm->registerRule('check_db_user', 'callback', 'check_db_user');
$regForm->addRule('username', 'Dieser Benutzername wird schon verwendet', 'check_db_user');
$regForm->registerRule('check_db_email', 'callback', 'check_db_email');
$regForm->addRule('email', 'Diese Emailadresse existiert schon', 'check_db_email');

//Validiere das Formular
if ($regForm->validate()) {
    $regForm->freeze();
    $regForm->process('process_data_regForm');
}

}

// DIESE PART GIBT DIE WERTE AN DIE SMARTY-ENGINE WEITER
// Erzeuge das "Renderer"-Objekt

$regFormRenderer =& new HTML_QuickForm_Renderer_ArraySmarty($smarty);

$regFormRenderer->setErrorTemplate(
    '{if $error}
    <span style="color: red; font-size: 0.7em;">{$error}</span><br />
    {/if}
    {$html}'
);

$regForm->setRequiredNote(
    '<font color="red" size="2">
    *&nbsp;
    </font>
    <font color="blue" size="2">
    Du hast ein Pflichtfeld vergessen auszufüllen
    </font>');

$regForm->accept($regFormRenderer);

$smarty->assign('regForm_data', $regFormRenderer->toArray());

?>

```

/ ENDE - Quellcode register.php */*

Register.php - Codeerklärung

Unsere Register.php erfüllt folgende Dinge:

- User kann sich registrieren und damit einen Account erstellen
- Register.php schickt an registrierten User eine Email mit einem Aktivierungslink
- Wenn der User den Aktivierungslink anklickt wird sein Account aktiviert (läuft auch über die register.php)
- zudem wird noch geprüft ob der eingetragene Username oder die eingetragene Email schon verwendet wird

```
//Prüfe ob Username schon existiert
function check_db_user($username)
{
    $dbh = DB::connect(DB_SYSTEM_1);
    $res =& $dbh->query("SELECT username FROM user WHERE username = '". $username. "'");
    $dbh->disconnect();

    if($res->numRows() >= 1){
        return false;
    }else{
        return true;
    }
}

//Prüfe ob Email schon existiert
function check_db_email($email)
{
    $dbh = DB::connect(DB_SYSTEM_1);
    $res =& $dbh->query("SELECT email FROM user WHERE email = '". $email. "'");
    $dbh->disconnect();

    if($res->numRows() >= 1){
        return false;
    }else{
        return true;
    }
}

//Erzeuge den Validierungscode
function validation_key($length=20) {
    $randstr='';
    srand((double)microtime()*1000000);

    $chars = array
    ( 'a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z','1','2','3','4','5','6','7','8','9','0');
    for ($rand = 0; $rand <= $length; $rand++)
    {
        $random = rand(0, count($chars) -1);
        $randstr .= $chars[$random];
    }
    return $randstr;
}
```

Erklärung - Verarbeitung des Registrierungs-Formular

Hier wird einfach nur geprüft ob der eingegebene Benutzername oder die Emailadresse schon existieren, ist dies der Fall wird "false" als Wert zurückgegeben und unser Formular kann dann mit einem Hinweis an den Besucher reagieren. Die Funktion "validation_key" macht nichts anderes als im default-Modus einen Zufallsstring mit 20 Zeichen zu erzeugen, es gibt hier sicherlich geschicktere Möglichkeiten einen Zufallsstring zu erzeugen, aber man sollte es mit der Optimierung auch nicht übertreiben und die Zeit in die wichtigen Parts stecken.

```
//Prüfe die Registrierungsdaten und erzeuge den User in der Datenbank
function process_data_regForm($values) {

    $val_key = validation_key();
    $recipients = $values[email];
    $md5_user_pass = md5($values[pass]);
```

```

$dbh = DB::connect(DB_SYSTEM_1);
$time_value = time();
$res =& $dbh->query("INSERT INTO user (username, pass, email, regdate, activation_key) VALUES
('$values[username]', '$md5_user_pass', '$values[email]', '$time_value', '$val_key')");
$dbh->disconnect();

require_once 'Mail/Mail.php';

$activation_link = 'http://domain.de/index.php?action=register&activate_user='.$val_key;

$headers['From'] = EMAIL_CONTACT;
$headers['To'] = $recipients;
$headers['Subject'] = 'Hier steht der Titel mit dem die Nachricht des Nutzers verschickt wird';

$body = 'Dies ist dein Aktivierungslink: '.$activation_link;

// Create the mail object using the Mail::factory method and send it
$mail_object =& Mail::factory('mail');
$mail_object->send($recipients, $headers, $body);

//this has to be changed
header("Location: /thanks/reg");

} //ENDE - function "process_data_regForm"

```

Erklärung - Verarbeitung des Registrierungs-Formular

Was geschieht hier? Diese Funktion bekommt die Angaben des Benutzers zugeschickt und validiert dann die eingegebenen Werte und bei erfolgreicher Prüfung wird der neue Benutzer angelegt. Dann schicken wir dem gerade erzeugten Benutzer an die Emailadresse, die er bei seiner Registrierung angegeben hat, einen Aktivierungslink um die Echtheit der Emailadresse zu validieren. Ist dies alles erfolgt werden wir der Benutzer mit *header("Location: /thanks/reg")*; auf eine Dankesseite weitergeleitet (die aber bislang noch nicht existiert und einen Fehler verursacht. Beim Aktivierungslink muss natürlich noch "domain.de" durch die eigene Domain ersetzt werden.

```

//Prüfe ob der User seinen Account aktivieren möchte
if(isset($_GET['activate_user'])){
$user_activation_key = $_GET[activate_user];

$sql = 'SELECT * FROM user WHERE activation_key = ?';

$dbh = DB::connect(DB_SYSTEM_1);
$res =& $dbh->getOne($sql, $user_activation_key);

//check if activation key is in database
//user_activation_key is not allowed to be 0, or all users get deleted
if(($res != '') AND ($user_activation_key != '0')){
$user_activation =& $dbh->query("UPDATE user SET activation_key = '0', active = '1' WHERE id_user = $res");

//thank user for his/her registration
header("Location: /thanks/reg_thx");
}
else {
//give user notice about that he is allready activated
header("Location: /thanks/reg_thx_stop");
}
$dbh->disconnect();
}

else {

```

Erklärung - User aktiviert seinen Account

Zugegeben, diesen Part hätte ich deutlicher und verständlicher abtrennen können, ich hoffe man kann ihn trotzdem

erschließen. Was hier passiert ist eine einfache If-Abfrage die prüft, wurde das Flag “activate_user” gesetzt, also will sich der User aktivieren, oder ist dies nicht der Fall, dann heißt es einfach “zeige dem Besucher das Registrierungsformular an” (dieser Code-Part kommt gleich). Was geschieht bei der Aktivierung? Der User klickt den Aktivierungslink in seiner EMail an und übermittelt diesen an das Script. Mit diesem Wert wird dann geprüft ob überhaupt ein User in der Datenbank existiert, der aktiviert werden muss. Ist dies der Fall wird er aktiviert und der Besucher wird auf die Dankeseite verwiesen, wurde der User allerdings schon aktiviert wird der Besucher auf eine Webseite weitergeleitet die ihn auch darauf aufmerksam macht.

Code - Formular erzeugen und Regeln festlegen

```
//Hier erzeugen wir die Elemente für das Registrierungsformular
$regForm = new HTML_QuickForm('regForm','post','http://domain.de/register','','',true);

$regForm->applyFilter('__ALL__','trim');

$regForm->addElement('text','username','Benutzername');
$regForm->addElement('password','pass','Passwort eingeben');
$regForm->addElement('password','pass2','Passwort wiederholen');
$regForm->addElement('text','email','Email');
$regForm->addElement('submit','btn_registration','Registrieren');

$regForm->addRule('username','Bitte Benutzername eingeben','required');
$regForm->addRule('pass','Bitte Passwort eingeben','required');
$regForm->addRule('pass2','Bitte Passwort eingeben','required');
$regForm->addRule('email','Bitte Emailadresse eingeben','required');

$regForm->addRule('username','Mindestlänge beträgt 3 Zeichen','minlength',3);
$regForm->addRule('username','Maximale Länge beträgt 40 Zeichen','maxlength',40);
$regForm->addRule('pass','Mindestlänge beträgt 5 Zeichen','minlength',5);
$regForm->addRule('pass','Maximale Länge beträgt 20 Zeichen','maxlength',20);
$regForm->addRule('email','Maximale Länge beträgt 60 Zeichen','maxlength',60);

//die beiden Passwörter müssen übereinstimmen
$regForm->addRule(array('pass','pass2'),'Die eingegebenen Passwörter müssen übereinstimmen','compare',null);
$regForm->addRule('email','Dies ist keine gültige Emailadresse','email');

//Hier werden die Funktionen aufgerufen und geprüft ob das eingebene Passwort oder der Benutzername schon existieren
$regForm->registerRule('check_db_user','callback','check_db_user');
$regForm->addRule('username','Dieser Benutzername wird schon verwendet','check_db_user');
$regForm->registerRule('check_db_email','callback','check_db_email');
$regForm->addRule('email','Diese Emailadresse existiert schon','check_db_email');

//Validiere das Formular
if ($regForm->validate()) {
    $regForm->freeze();
    $regForm->process('process_data_regForm');
}
}
```

Erklärung - Formular erzeugen und Regeln festlegen

Endlich der Code für das eigentliche Formular! Und dieses hat es in sich, dank PEAR. Dabei ist es wirklich einfach zu verstehen und einzusetzen, wenn man erstmal verstanden hat wie die Arbeitsweise von Quickform ist. Als erstes wird eine Instanz der Klasse “HTML_QuickForm” angelegt, quasi ein Behälter in den wir nachher alle unsere Einstellungen, Werte, Bezeichnungen und Regeln für unser Formular packen (ersetzt “domain.de” durch eure eigene Domain). Dann geht es auch schon los:

- **\$regForm->applyFilter('__ALL__', 'trim');** - applyFilter ordnet einen Filter auf ein Formular-Element zu, da wir hier die Einstellung “__ALL__” verwenden, gilt der Filter für alle Elemente des Formulars. “Trim” ist eine

PHP-Funktion die führende Leerzeichen rauswirft, das sollte man in der Regel machen um keine Konflikte beim Login zu erhalten (wenn versehentlich der Benutzername bei der Registrierung mit führendem Leerzeichen eingegeben wird)

- **\$regForm->addElement('text', 'username', 'Benutzername');** - mit addElement weist man dem Formular ein weiteres Element hinzu, um welchen Typ es sich dabei handelt, zeigt jeweils der erste Parameter, der zweite Parameter ist später der Name mit dem das Element in Smarty angesprochen werden kann und der dritte ist dann noch lediglich die Bezeichnung für das Element (die Beschriftung)
- **\$regForm->addRule('username', 'Bitte Benutzername eingeben', 'required');** - mit "addRule" lassen sich Regeln für einzelne Elemente festlegen, in diesem Fall sagt man das das Feld "username" auf jedenfall ausgefüllt sein muss. Zusätzlich sollte man die Länge der Eingabe einschränken, also das der Benutzername nicht zu kurz und nicht zu lang ist.
- **\$regForm->addRule(array('pass', 'pass2'), 'Die eingegebenen Passwörter müssen übereinstimmen', 'compare', null);** - Prüft ob zwei Felder den gleichen Wert haben, wichtig für die Passwort-Eingabe
- **\$regForm->addRule('email', 'Dies ist keine gültige Emailadresse', 'email');** - mir ist bislang keine Methode begegnet mit der es einfacher ist, die Struktur einer Emailadresse zu prüfen
- **\$regForm->registerRule('check_db_user', 'callback', 'check_db_user');**
\$regForm->addRule('username', 'Dieser Benutzername wird schon verwendet', 'check_db_user'); - weiter oben wurde darauf hingewiesen, das der eingegebene Benutzername noch nicht existieren darf, dies wird hiermit geprüft. Die Kombination aus "registerRule" (lege eine Regel fest) und "addRule" (verwende die gerade festgelegte Regel) erledigt dies (vorausgesetzt wird haben auch eine Funktion geschrieben (die wir weiter oben schon behandelt haben)

```
//Validiere das Formular
if ($regForm->validate()) {
    $regForm->freeze();
    $regForm->process('process_data_regForm');
}
```

Ist alles erledigt können wir beim Abschicken des Formulars die Eingaben validieren und bei Korrektheit an die Verarbeitungsfunktion schicken.

Code - Smarty und Quickform

```
$regFormRenderer =& new HTML_QuickForm_Renderer_ArraySmarty($smarty);

$regFormRenderer->setErrorTemplate(
    '{if $error}
    <span style="color: red; font-size: 0.7em;">{$error}</span><br />
    {/if}
    {html}'
);

$regForm->setRequiredNote(
    '<font color="red" size="2">
    *&nbsp;
    </font>
    <font color="blue" size="2">
    Du hast ein Pflichtfeld vergessen auszufüllen
    </font>'
);

$regForm->accept($regFormRenderer);

$smarty->assign('regForm_data', $regFormRenderer->toArray());
```

Erklärung - Smarty und Quickform

Brav haben wir unser Template mit Smarty und unser Formular mit PEAR angelegt, doch wie bringt man die beiden

ein Input-Field oder ein Submit-Button).

Registrieren

Nun rufen wir diese URL auf ("domain.de" durch eigene URL ersetzen): "http://domain.de/register" und sollten folgende Ausgabe erhalten:

The image shows a registration form with the following fields and a button:

- Benutzername
- Passwort eingeben
- Passwort wiederholen
- Email
- Registrieren

Die anderen Module werden jeweils in einem eigenen Part behandelt, da es sonst den Umfang dieses Artikel sprengen würde.

Weitere Quellen:

- Quickform-Dokumentation (<http://pear.php.net/manual/en/package.html.html-quickform.php>) - es dauert etwas bis man damit umgehen kann, aber dann sind auch die anderen Dokumentationen von PEAR wirklich hilfreich
- Schneller coden mit PEAR-Quickform (<http://lurki.de/internet/webdesign/86-schneller-coden-mit-pear-quickform/>)

6. User-Modul: Login/Logout

Seit dem letzten Teil des Webdesign-Guide können sich unsere Besucher registrieren, einen Account anlegen und ihren Account aktivieren. Es werden bislang zwar noch Fehlermeldungen nach der Registrierung und nach dem Aktivieren ausgegeben, aber darum kümmern wir uns später. In diesem Teil kümmern wir uns nun darum das sich unsere registrierten Besucher einloggen können.

Im Ordner "includes" legen wir dafür die Datei "**login.php**" an und im Smarty-Template Ordner die Datei "**loginform.tpl**".

/ START - Quellcode login.php */*

```
<?php
require_once 'HTML/QuickForm.php';
require_once 'HTML/QuickForm/Renderer/ArraySmarty.php';
require_once 'Auth/Auth.php';
require_once 'smarty/libs/Smarty.class.php';
require_once 'includes/config.php';

function controlLogin()
{
    global $myauth;
```

```

    return $myauth->getAuth();
}

$loginForm = new HTML_QuickForm('loginForm','post','/login','','',true);

$authParams =
array('dsn'=>DB_SYSTEM_1,'table'=>'user','usernamecol'=>'username','passwordcol'=>'pass','cryptType'
=>'md5','db_fields'=>'*');

$myauth = new Auth ('DB', $authParams, 'controlLogin()', false);

$myauth->start();

//wenn Account nicht existiert, logge den User wieder aus
if($_SESSION[_authsession][data][active] == 0)
    $myauth->logout();

//prüfe ob der eingeloggte User den Logout-button betätigt hat
if($myauth->checkAuth() && isset($_GET['logout']) && $_GET['logout'] == '1') {
    $myauth->logout();
    $myauth->start();
}

//prüfe ob der user schon eingeloggt ist und den logout button gedrückt hat
if($myauth->checkAuth()) {
    $smarty->assign('loginForm_logout', '<a
href="index.php?action=login&logout=1">logout</a>');
}

//zeige das loginformular an falls der user noch nicht eingeloggt ist
if(!$myauth->checkAuth()){
    $loginForm->applyFilter('__ALL__', 'trim');

    $loginForm->addElement('header', null, 'Login');
    $loginForm->addElement('text','username', 'Benutzername',
array('size'=>12,'maxlength'=>40,'class'=>'login_field'));
    $loginForm->addElement('password','password', 'Passwort',
array('size'=>12,'maxlength'=>20,'class'=>'login_field'));
    $loginForm->addElement('submit', btn_login, 'Einloggen', array('id'=>'login_button'));

    $loginForm->addRule('username', 'Bitte den Benutzernamen eingeben', 'required');
    $loginForm->addRule('password', 'Bitte das Passwort eingeben', 'required');

    $loginForm->registerRule('controlLogin', 'function', 'controlLogin');

if ($loginForm->isSubmitted() && $loginForm->validate()) {
    $loginForm->freeze();
}
else if (($loginForm->isSubmitted() && !($loginForm->validate())) && !($_GET['logout'] == '1')){
    $smarty->assign('login_error', 'Login fehlgeschlagen');
}
}

$loginFormRenderer =& new HTML_QuickForm_Renderer_ArraySmarty($smarty);

$loginForm->accept($loginFormRenderer);

$smarty->assign('loginForm_data', $loginFormRenderer->toArray());

?>

```

/* ENDE - Quellcode login.php */

Login.php - Codeerklärung

```
function controlLogin()
{
    global $myauth;
    return $myauth->getAuth();
}

$loginForm = new HTML_QuickForm('loginForm','post','/login','','',true);

$authParams =
array('dsn'=>DB_SYSTEM_1, 'table'=>'user', 'usernamecol'=>'username', 'passwordcol'=>'pass', 'cryptType'
=>'md5', 'db_fields'=>'*');

$myauth = new Auth ('DB', $authParams, 'controlLogin()', false);

$myauth->start();
```

Dieser Abschnitt ist für die Authentifizierung unserer Benutzer zuständig. In “authParams” sagen wir der Klasse Auth wie unsere Datenbankstruktur mit den Userinformationen aussieht, haben wir das einmal abgebildet, weiß Auth fortan wie es auf die Benutzerinformationen zugreifen kann. Die Parameter im einzelnen:

- **dsn** - die Verbindungsdaten zur Datenbank damit Auth auch die Daten abfragen kann
- **table** - wie heißt die Tabelle in der die Benutzerinformationen gespeichert sind
- **usernamecol** - wie heißt die Spalte in der der Benutzername gespeichert ist
- **passwordcol** - das gleiche für das Passwort
- **cryptType** - wie wurde das Passwort abgespeichert. Da wir die Benutzerpasswörter mit “MD5” codieren, müssen wir das hier auch angeben
- **db_fields** - wenn sich der Benutzer erfolgreich authentifiziert hat, welche Daten sollen dann aus der Tabelle selektiert werden

Als nächstes erzeugen wir dann die Instanz der Klasse Auth und dort verwenden wir auch die gerade angegebenen Parameter. Mit “controlLogin” wird die Auth-Instanz initialisiert.

```
//wenn Account nicht existiert, logge den User wieder aus
if($_SESSION[_authsession][data][active] == 0)
    $myauth->logout();

//prüfe ob der eingeloggte User den Logout-button betätigt hat
if($myauth->checkAuth() && isset($_GET['logout']) && $_GET['logout'] == '1') {
    $myauth->logout();
    $myauth->start();
}

//prüfe ob der user schon eingeloggt ist und den logout button gedrückt hat
if($myauth->checkAuth()) {
    $smarty->assign('loginForm_logout', '<a
href="index.php?action=login&logout=1">logout</a>');
}
```

Diese drei Abfragen prüfen folgendes:

1. Wurde der Useraccount schon aktiviert, falls nein, logge den Benutzer wieder aus
2. Prüfe ob der eingeloggte Benutzer den Logout button gedrückt hat, dann logge ihn aus
3. Wenn der Benutzer eingeloggt ist, dann zeige ihm den Logout button an (wird einer Smarty-Variablen zugewiesen)

Hier sollte ich vielleicht noch erwähnen das, sobald ein User eingeloggt wurde, eine Session für ihn angelegt wird, auf die wir mit **\$_SESSION[_authsession][data][*]** zugreifen können. Das “*” steht dabei für eine beliebige Spalte aus der Usertabelle der Datenbank, da wir alle selektiert haben, können wir auf jede Spalte zugreifen.

Der restliche Code ist nur dafür da das Loginformular zu erzeugen und die Werte an Smarty zu übermitteln, das wurde in den vorigen Teilen schon ausführlich besprochen.

/ START - Quellcode loginform.tpl */*

```
<span id="loginForm">

{if !$loginForm_logout}
  <form {$loginForm_data.attributes}>
    {$loginForm_data.hidden}
    <!-- Display the fields -->
    <table>
      <tr>
        <th class="form_login_label">{$loginForm_data.username.label}</th>
        <th class="form_login_label">{$loginForm_data.password.label}</th>
        <th>&nbsp;</th>
        <td style="font-size: 0.8em; width: 250px; color: red;">{if
$login_error}{$login_error}{/if}</td>
      </tr>
      <tr>
        <td>{$loginForm_data.username.html}</td>
        <td>{$loginForm_data.password.html}</td>
        <td>{$loginForm_data.btn_login.html}</td>
        <td style="font-size: 0.7em; width: 250px;"><a
href="http://domain.de/index.php?action=password_lost">Passwort vergessen</a></td>
      </tr>
    </table>
  </form>

  {else}
    <br />
    eingeloggt
    <br />
    {$loginForm_logout}
  {/if}

</span>
```

/ ENDE - Quellcode loginform.tpl */*

loginform.tpl - Codeerklärung

Im Smarty-Template geben wir nun unser Loginformular aus, vorausgesetzt der User ist nicht schon eingeloggt (was wir mittels **\$loginForm_logout** prüfen), denn ist das der Fall wird stattdessen die Mitteilung ausgegeben das der User schon eingeloggt ist und einen Link mit dem er sich ausloggen kann. Sollte dieser Code nicht verständlich genug sein empfiehlt sich nochmal ein Blick in die vorigen Teile in denen dieser Part ausführlich erläutert wurde.

7. Modul: Kontaktformular und Dankeseite

Nun kann sich der User registrieren und einloggen, doch bevor er mit seinem Account wirklich etwas anfangen kann und unsere Webanwendung einen Nutzen bekommt, ermöglichen wir ihm noch schnell uns per Kontaktformular Nachrichten, Hinweise und Fehlermeldungen zukommen zu lassen.

Doch vorerst noch eine kleine Korrektur an der Datei ".htaccess":

Aus

RewriteRule ^([0-9a-zA-Z]+)\$ index.php?action=\$1

wird

```
RewriteRule ^([0-9a-zA-Z_]+)$ index.php?action=$1
```

Bislang haben wir nur Zahlen und klein/groß-Buchstaben als regulären Ausdruck erlaubt, nun kommt noch der Unterstrich hinzu, so können wir ab jetzt auch Strings in folgendem Format auswerten:

- dies_ist_ein_string
- _string_
- eine_datei

Ein Kontaktformular mittels Smarty und PEAR zu erstellen ist recht einfach, wenn man sich schon die vorigen Kapitel angeschaut hat. Wie immer folgt erst der Code und danach die Erklärung.

Im Ordner “includes” legen wir die Datei “**mail.php**” an und im Smarty-Template Ordner die Datei “**contact_mail.tpl**”.

/ START - Quellcode mail.php */*

```
<?php
require_once 'HTML/QuickForm.php';
require_once 'HTML/QuickForm/Renderer/ArraySmarty.php';
require_once 'smarty/libs/Smarty.class.php';
require_once 'includes/config.php';

function process_data($values) {
    require_once 'Mail/Mail.php';

    //die ziel-email-adresse läßt sich in der config-datei ändern
    $recipients = EMAIL_CONTACT;

    $headers['From']    = $values[email];
    $headers['To']      = $recipients;
    $headers['Subject'] = $values[mailTitle];

    $body = $values[message];

    $mail_object =& Mail::factory('mail');

    $mail_object->send($recipients, $headers, $body);

    header("Location: index.php?action=thanks&thx_value=mail");
}

$mailForm = new HTML_QuickForm('mailForm', 'post', 'index.php?action=contact_mail');

$mailForm->applyFilter('__ALL__', 'trim');

$mailForm->addElement('text', 'name', 'Name', array('size' => 50));
$mailForm->addElement('text', 'email', 'Email', array('size' => 50));
$mailForm->addElement('text', 'mailTitle', 'Titel der Nachricht', array('size' => 50));
$mailForm->addElement('textarea', 'message', 'Ihre Nachricht', array('cols' => 37, 'rows' => 15));

$mailForm->addElement('submit', btn_send_mail, 'Nachricht senden');

$mailForm->addRule('name', 'Bitte einen Namen eingeben', 'required');
$mailForm->addRule('email', 'Bitte gib deine Emailadresse ein', 'required');
$mailForm->addRule('mailTitle', 'Du hast den Titel vergessen', 'required');
$mailForm->addRule('message', 'Die Nachricht darf nicht leer sein', 'required');

$mailForm->addRule('name', 'min 3 Zeichen', 'minlength', 3);
$mailForm->addRule('name', 'max 40 Zeichen', 'maxlength', 40);
$mailForm->addRule('mailTitle', 'min 5 Zeichen', 'minlength', 5);
$mailForm->addRule('mailTitle', 'max 250 Zeichen', 'maxlength', 250);
$mailForm->addRule('message', 'min 20 Zeichen', 'minlength', 20);
```

```

$mailForm->addRule('message', 'max 10000 Zeichen', 'maxlength', 10000);

$mailForm->addRule('email', 'die Struktur der Email ist ungültig', 'email');

if ($mailForm->validate())
$mailForm->process('process_data', false);

$mailFormRenderer =& new HTML_QuickForm_Renderer_ArraySmarty($smarty);

$mailFormRenderer->setRequiredTemplate(
    '{if $error}
    <font color="red">{$label}</font>
    {else}
        {if $required}
            <font color="red" size="3">*</font>
            {/if}
            {$label}
        {/if}'
);

$mailFormRenderer->setErrorTemplate(
    '{if $error}
    <font color="orange" size="1">{$error}</font><br />
    {/if}
    {$html}'
);

$mailForm->setRequiredNote(
    '<font color="red" size="2">
    *&nbsp;
    </font>
    <font color="blue" size="2">
    Diese Felder werden benötigt
    </font>');

// build the HTML for the form
$mailForm->accept($mailFormRenderer);

// assign array with form data
$smarty->assign('mailForm_data', $mailFormRenderer->toArray());
$smarty->assign('mailForm_header_label', 'Kontaktformular');

?>

```

/ ENDE - Quellcode mail.php */*

mail.php - Codeerklärung

Ich verzichte darauf den Code zu erklären der dafür zuständig ist das Formular mittels Pear zu erzeugen und die Werte an Smarty zu schicken und stürze mich direkt auf die Funktion **“process_data“**:

```

function process_data($values) {

    require_once 'Mail/Mail.php';

    //die ziel-email-adresse läßt sich in der config-datei ändern
    $recipients = EMAIL_CONTACT;

    $headers['From']    = $values[email];
    $headers['To']     = $recipients;
    $headers['Subject'] = $values[mailTitle];

    $body = $values[message];

    $mail_object =& Mail::factory('mail');

    $mail_object->send($recipients, $headers, $body);
}

```

```
header("Location: index.php?action=thanks&thx_value=mail");
}
```

In der `Process_data` Funktion wird nach Abschicken des Kontakt-Formulars und nach erfolgreicher Validierung der Eingaben eine Email zusammengebastelt und an den Empfänger geschickt. Der Empfänger wird durch die Konstante `"EMAIL_CONTACT"` repräsentiert und kann jederzeit in der `"config.php"` geändert werden. Um eine Email zu verschicken benötigen wir das PEAR-Paket `"Mail"` welches wir schon eingebunden haben. Dann füllen wir das `"headers"`-array mit den Daten die wir für die Mail brauchen, das ist:

- die Mail den Versenders (`$headers['From']`)
- der Empfänger (`$headers['To']`)
- und der Titel der Nachricht (`$headers['Subject']`)

Die Nachricht selbst wird in der Variablen `"$body"` gespeichert. Nun haben wir alle Daten zusammen und können uns mit der `"factory"`-Methode der Klasse `Mail` ein Nachrichten-Objekt anlegen, einmal angelegt können wir nun mittels der Methode `"send"` und den vorher angelegten Parametern die Nachricht an uns selbst, bzw an den Empfänger schicken. Nachdem die Nachricht verschickt wurde lassen wir den User auf die Dankesseite leiten. Die Dankesseite existiert noch nicht und erzeugt daher eine Fehlermeldung, darum kümmern wir uns nach dem Code für die Datei `"contact_mail.tpl"` der selbsterklärend ist (ansonsten nochmal die vorigen Kapitel durchlesen, dort stehen die Basics zu `Smarty`, `Pear` und `Quickform` drin).

/* START - Quellcode contact_mail.tpl */

```
<div id="mailForm_box">
<fieldset><legend>{$mailForm_header_label}</legend>
  <form {$mailForm_data.attributes}>
    {$mailForm_data.hidden}
    <!-- Display the fields -->
    <table>
      <tr>
        <th>{$mailForm_data.name.label}</th>
        <td class="field">{$mailForm_data.name.html}</td>
      </tr>
      <tr>
        <th>{$mailForm_data.email.label}</th>
        <td class="field">{$mailForm_data.email.html}</td>
      </tr>
      <tr>
        <th>{$mailForm_data.mailTitle.label}</th>
        <td class="field">{$mailForm_data.mailTitle.html}</td>
      </tr>
      <tr>
        <th>{$mailForm_data.message.label}</th>
        <td></td>
      </tr>
      <tr>
        <td colspan=2>{$mailForm_data.message.html}</td>
      </tr>

      <!-- Display the buttons -->
      {if not $mailForm_data.frozen}
      <tr>
        <td colspan=2>{$mailForm_data.btn_send_mail.html}</td>
      </tr>
      {/if}
      {if $mailForm_data.requirednote and not $mailForm_data.frozen}
      <tr>
        <td colspan=2 valign="top">{$mailForm_data.requirednote}</td>
      </tr>
      {/if}
    </table>
```

```

</form>
</fieldset>
</div>

```

/ ENDE - Quellcode contact_mail.tpl */*

Vergesst nicht die Konstante "EMAIL_CONTACT" in der Datei "config.php" anzupassen, denn sonst bekommt ihre die verschickte Email nicht zugesendet.

Die Dankesseite

Eine Dankesseite gibt dem User bei jedem seiner Handlungen Feedback, sei es nun die Registrierung, das Verschicken einer Email, das Aktivieren seines Accounts usw., da kann sich schon einiges mit der Zeit ansammeln. Darum legen wir jetzt für die Dankesseite eine .php-Datei im Ordner includes an mit dem Namen "thanks.php" und eine .tpl-Datei im Smarty-Template-Ordner mit Namen: "thanks.tpl".

/ START - Quellcode thanks.php */*

```

<?php
    require_once 'smarty/libs/Smarty.class.php';

    function output_thanks($value){
        global $smarty;

        switch($value){
            case 'password_lost':
                $smarty->assign('thanks_legend', 'Neues Passwort');
                $smarty->assign('thanks_output', 'Dir wurde eine Email mit dem
neuen Passwort zugeschickt. Du musst es nur noch aktivieren. ');
                break;

            case 'new_pass_activated':
                $smarty->assign('thanks_legend', 'Neues Passwort aktiviert');
                $smarty->assign('thanks_output', 'Dein neues Passwort wurde
aktiviert und du kannst dich nun damit einloggen');
                break;

            case 'new_pass_not_activated':
                $smarty->assign('thanks_legend', 'Passwortaktivierung
fehlgeschlagen');
                $smarty->assign('thanks_output', 'Fehler - Das neue Passwort konnte
nicht aktiviert werden');
                break;

            case 'mail':
                $smarty->assign('thanks_legend', 'Nachricht verschickt');
                $smarty->assign('thanks_output', 'Danke für deine Nachricht. Ich
werde mich schnellstmöglich um eine Antwort bemühen. ');
                break;

            case 'reg':
                $smarty->assign('thanks_legend', 'Registrierung abschliessen');
                $smarty->assign('thanks_output', 'Danke für deine Registrierung. Du
bekommst nun eine Email zugeschickt und darin befindet sich ein Link zur Aktivierung. Nach der
Aktivierung kannst du direkt mit dem Sprachen lernen beginnen. ');
                break;

            case 'reg_thx':
                $smarty->assign('thanks_legend', 'Registrierung abgeschlossen');
                $smarty->assign('thanks_output', 'Deine Registrierung wurde
bestätigt und du kannst dich nun einloggen. Viel Spass beim Sprachen lernen! ');
                break;

            case 'reg_thx_stop':
                $smarty->assign('thanks_legend', 'Es kann losgehen! ');

```

```

        $smarty->assign('thanks_output', 'Du hast deinen Account schon
aktiviert und kannst dich nun einloggen. Viel Spass beim Sprachen lernen!');
        break;

        default:
            break;
    }
}

$value = $_GET[thx_value];
output_thanks($value);

?>

```

/ ENDE - Quellcode thanks.php */*

thanks.php - Codeerklärung

Wieder einmal ganz einfacher Code der auch schnell erweiterbar ist: Wir holen uns den aktuellen "thx_value", falls vorhanden, und arbeiten uns mit diesem Wert durch die Switch-Anweisung. Jenachdem welcher Wert für "thx_value" gesetzt ist, führen wir eines der "Case"-Anweisungen aus. In den Case-Anweisungen wird einfach nur Text einer Smarty-Variablen zugewiesen, keine große Magie. Der zugewiesene Wert wird dann von der Smarty-Variablen im Smarty-Template ausgespuckt, der Code des Templates ist selbsterklärend:

/ START - Quellcode thanks.tpl */*

```

<div id="thanks_box">
<fieldset><legend>{$thanks_legend}</legend>
<p>{$thanks_output}</p>
</fieldset>
</div>

```

/ ENDE - Quellcode thanks.tpl */*

Im nächsten Part geht es daran eine wirkliche Funktion für den User zu erstellen, mittels eines Eintragsformulars kann der User Webnotizen erstellen und diese bei jedem Login einsehen. Um das ganze etwas mehr "Web2.0" zu machen und häßliche Seitenreloads beim erstellen und löschen der Webnotizen zu vermeiden, wird im darauffolgenden Part die Ajax-Engine mittels Prototype angeschmissen und schöne Effekte mit Scriptaculous erzeugt.

8. Eintragsformular und Webnotizen

Nun haben wir unsere Webanwendung soweit lauffähig, fehlt es nur noch an der Funktionalität. Mit meiner Idee lassen sich sicher keine Preise gewinnen, noch ist sie neu, aber zumindest schnell umsetzbar und erweiterungsfähig. Wer seine eigenen Ideen umsetzen möchte kann ab hier auch alleine losgehen, für alle anderen werde ich erstmal erklären worum es geht.

Webnotizen

Mittels eines Eintragsformulars soll der eingeloggte User die Möglichkeit haben, Notizen online abzulegen und auch wieder zu löschen. Dies wird erstmal nicht sonderlich spektakulär aussehen aber im nächsten Teil werden wir das ganze

dann noch mit Ajax aufpeppen.

Das Eintragsformular

Als erstes passen wir die Datei "index.php" an, wir fügen einen zusätzlichen Fall (case ein). Im ersten Switch-Block verwenden wir:

```
case 'insert':
require_once('includes/insert.php');
break;
```

Der zweite Switch-Block, welcher dafür zuständig ist die Smarty-Templates zu laden, verwenden wir diese Anweisung:

```
case 'insert':
$smarty->display('insert.tpl');
break;
```

Diese beiden Dateien müssen wir natürlich auch noch anlegen.

Was soll unser Formular leisten können? Nun, nicht viel, es besitzt lediglich ein Textfeld, in welches man seinen Text eintragen kann und ein Titelfeld, in welches man den Titel für eine Webnotiz notieren kann, da es aber mit Pear und Smarty entwickelt wird, dürfte es kein Problem sein die Funktionalität zu erweitern.

Nun der Code für das Eintragsformular:

/ START - Quellcode insert.php */*

```
<?php
require_once 'HTML/QuickForm.php';
require_once 'HTML/QuickForm/Renderer/ArraySmarty.php';
require_once 'smarty/libs/Smarty.class.php';
require_once 'includes/config.php';
require_once ($_SERVER['DOCUMENT_ROOT'].'/DB/DB.php');

session_start();

//wenn der Benutzer nicht eingeloggt ist, schicke ihn zu der Loginseite
if(!isset($_SESSION[_authsession][data][id_user]))
    header("Location: /login");

function insertprocess_data($values) {
    $dbh = DB::connect(DB_SYSTEM_1);
    $res =& $dbh->query("INSERT INTO post (title, content) VALUES ('".$values[title]."',
    '".$values[content]."'");
    $id_webnotiz = mysql_insert_id();
    $res =& $dbh->query("INSERT INTO post_user (id_user, id_post) VALUES
    ('".$_SESSION[_authsession][data][id_user]."', '".$id_webnotiz."");
    $dbh->disconnect();

    header("Location: /");
}

$insertForm = new HTML_QuickForm('insertForm', 'post', 'index.php?action=insert');

$insertForm->applyFilter('__ALL__', 'trim');

$insertForm->addElement('text', 'title', 'Titel der Webnotiz', array('size' => 50));
$insertForm->addElement('textarea', 'note', 'Ihre Notiz', array('cols' => 37, 'rows' => 15));
$insertForm->addElement('submit', 'btn_send_note', 'Webnotiz anlegen');
$insertForm->addRule('title', 'Bitte vergeben Sie einen Titel für Ihre Nachricht', 'required');
$insertForm->addRule('note', 'Bitte tragen Sie ihre Webnotiz eintragen', 'required');

if ($insertForm->validate())
    $insertForm->process('insertprocess_data', false);
```

```

$insertFormRenderer =& new HTML_QuickForm_Renderer_ArraySmarty($smarty);

$insertFormRenderer->setRequiredTemplate(
    '{if $error}
    <font color="red">{$label}</font>
    {else}
        {if $required}
            <font color="red" size="3">*</font>
            {/if}
            {$label}
        {/if}'
);

$insertFormRenderer->setErrorTemplate(
    '{if $error}
    <font color="orange" size="1">{$error}</font><br />
    {/if}
    {$html}'
);

$insertForm->setRequiredNote(
    '<font color="red" size="2">
    *&nbsp;
    </font>
    <font color="blue" size="2">
    Diese Felder werden benötigt
    </font>');

$insertForm->accept($insertFormRenderer);

$smarty->assign('insertForm_data', $insertFormRenderer->toArray());
$smarty->assign('insertForm_header_label', 'Eintragsformular');

?>

```

/ ENDE - Quellcode insert.php */*

Sollte sich mittlerweile von selbst erklären: Zuerst prüfen wir ob der User auch eingeloggt ist (also die Session Variable eine id hat, diese Id brauchen wir auch um einzelne Notizen dem jeweiligen Benutzer beim speichern zuzuordnen). Danach erzeugen wir ein Formular mit zwei Feldern, das Titelfeld und das Feld für die Notiz, wenn das Formular abgeschickt und erfolgreich verarbeitet wurde, wird eine neue Notiz in der Datenbank abgelegt. Mit “mysql_insert_id();” holen wir uns nach dem insert in der Datenbank die ID des zuletzt eingetragenen Datensatzes (also die ID der Notiz) und verwenden diese ID, mit der ID des eingeloggten Benutzers um in der Tabelle “post_user” eine Assoziation dieser beiden zu erstellen.

Die Template-Datei ist noch kürzer gehalten:

/ START - Quellcode insert.tpl */*

```

<div id="insertForm_box">
<fieldset><legend>{$insertForm_header_label}</legend>
<form {$insertForm_data.attributes}>
    {$insertForm_data.hidden}
    <!-- Display the fields -->
    <table>
        </tr>
        <tr>
            <td colspan=2>{$insertForm_data.title.label}</td>
        </tr>
        <tr>
            <td colspan=2>{$insertForm_data.title.html}</td>
        </tr>
        <tr>
            <td colspan=2>{$insertForm_data.note.label}</td>

```

```

</tr>
<tr>
  <td colspan=2>{$insertForm_data.note.html}</td>
</tr>

<!-- Display the buttons -->
      {if not $insertForm_data.frozen}
      <tr>
        <td colspan=2>{$insertForm_data.btn_send_note.html}</td>
      </tr>
      {/if}

      {if $insertForm_data.errors}
      <tr>
        <td colspan="2">{$insertForm_data.requirednote}</td>
      </tr>
      {/if}

</table>
</form>
</fieldset>
</div>

```

/ ENDE - Quellcode insert.tpl */*

Steht natürlich jedem frei das Formular an die eigenen optischen Anforderungen anzupassen, das ist ja auch das tolle an einer Template-Engine, man platziert die Elemente welche man benötigt dort wo sie gebraucht werden, der Rest wird dann mittels CSS gelöst.

Bislang haben wir uns noch kein Stylesheet angelegt, das soll nun folgen, den schließlich wollen wir unsere Notizzettel auch mal ausgeben, am besten auf der Startseite und nur wenn der User eingeloggt ist. Also wir kümmern uns nun um folgendes:

- Anlegen eines Stylesheets
- Prüfen auf der Startseite ob der Benutzer eingeloggt ist und wenn ja, ausgeben seiner Notizen, wenn nein, weiterleiten zum Loginformular

Anlegen eines Stylesheets

Geht in euer Root-Verzeichnis (dort wo auch die index.php liegt) und legt eine Datei “style.css” an. Diese bekommt folgenden Inhalt:

/ START - Quellcode style.css */*

```

.note {border: 1px solid #bbbbbb; width: 400px; background: #eeeeee; margin: 10px;}
.notetitle { font-weight: bold; margin: 5px;}
.notetext{ margin: 5px; border: 1px solid #cccccc; background: white; padding: 5px;}

```

/ ENDE - Quellcode style.css */*

Für jede Notiz wird ein eigener Container erstellt, dieser besitzt die Klasselemente von “note”. In dem Container Note befinden sich zwei weitere Container, title und text, diese werden über die Klassen “notetitle” und “notetext” angesprochen.

Jetzt binden wir als erstes das Stylesheet in unsere Webanwendung ein, dazu müssen wir die Datei “header.tpl” um folgende Codezeile erweitern:

```
<link rel="stylesheet" href="style.css" type="text/css" media="screen" />
```

Den Code einfach in den “head”-Block packen.

So, jetzt müssen wir unsere Notizen nur noch irgendwo ausgeben, wie oben angedeutet machen wir das auf der Startseite. Folgendes muss dafür geschehen:

- wir erzeugen die Datei “output.php”, in dieser wird eine Smarty-Variable mit Werten aus der Datenbank befüllt
- in der index.tpl wird die Variable dann einfach ausgegeben.

Ab hier wird es etwas “schmutzig” denn normalerweise sollte man eine Ergebnismenge in ein Array packen und dieses Array dann in der Template Datei durchlaufen. Da ich mich aber nie wirklich mit den Smarty-Schleifen anfreunden konnte, mache ich die “Formatierung” in der Output.php und packe alles in eine Smarty-Variable die ich dann in der index.tpl ausgabe. Wir legen also im Ordner “includes” die Datei “output.php” an und befüllen diese mit folgendem Code:

/ START - Quellcode output.php */*

```
<?php
require_once ($_SERVER['DOCUMENT_ROOT'].'/DB/DB.php');
require_once 'includes/config.php';
require_once 'smarty/libs/Smarty.class.php';

$dbh = DB::connect(DB_SYSTEM_1);
$res =& $dbh->query("SELECT post.id_post AS id, post.title AS title, post.content AS content FROM
post, post_user WHERE post_user.id_user = '". $_SESSION[_authsession][data][id_user]."' AND
post_user.id_post = post.id_post ORDER BY id DESC");
$dbh->disconnect();

$webnotizen = "";
while ($row =& $res->fetchRow(DB_FETCHMODE_ASSOC)) {
    $webnotizen .= '<div class="note"><div class="notetitle">';
    $webnotizen .= $row[title];
    $webnotizen .= '</div><div class="notetext">';
    $webnotizen .= $row[content];
    $webnotizen .= '</div></div>';
}

$smarty->assign('webnotizen', $webnotizen);

?>
```

/ ENDE - Quellcode output.php */*

Der MySQL-Query macht nichts anderes als alle Webnotizen, für den jeweils eingeloggt User, aus der Datenbank zu holen. Dann durchlaufen wir die Ergebnismenge und speichern die selektierten Werte zwischen den Div-Containern, die wir mittels CSS formatieren. Als letztes weisen wir der Smarty-Variablen “webnotizen” unsere (“schmutzig”) zusammengebastelte Variable zu, damit wir in der index.tpl diese ausgeben können.

Jetzt existiert aber noch ein Problem, was ist wenn der User noch garnicht eingeloggt ist, welche Daten werden ihm dann ausgegeben? Am besten keine und deshalb schicken wir einen nicht eingeloggt User direkt auf die Loginseite, wir ersetzen also in der index.php folgenden Code:

```
case 'index':
default:
break;
```

durch diesen (zusätzlich ist auch die include-Anweisung für die output.php mit drin):

```
case 'index':
default:
if(!isset($_SESSION[_authsession][data][id_user]))
header("Location: /login");
else
require_once('includes/output.php');
break;
```

und am anfang der Datei, nachdem die Dateien mit “require_once” eingebunden werden, muss noch diese Anweisung hinzugefügt werden:

```
session_start();
```

Als letzten Schritt schreibt in eure “index.tpl” folgenden Code, damit eure Webnotizen auch ausgegeben werden:

```
{$webnotizen}
```

Spass mit Ajax

Im nächsten Teil peppen wir das ganze mit Ajax auf, dann lassen sich Elemente löschen ohne Seitenreload und (hoffentlich) auch frei auf der Fläche platzieren, mal sehen was mir da so einfällt...

Quellen

Mit Smarty ein Array durchlaufen: <http://smarty.php.net/manual/de/language.function.foreach.php>

9. Eintragsformular und Webnotizen

Wir nähern uns dem Ende vom Webdesign Guide und bislang haben wir folgendes erreicht:

- eine lauffähige Anwendung die auf Klassenbibliotheken von PEAR zurückgreift
- als Template-Engine läuft unter der Haube Smarty
- mittels HtAccess haben wir uns suchmaschinen-optimierte Links gebastelt
- eine Datenbank liegt bereit um unsere Eingaben zu speichern
- User können sich Registrieren, Einloggen und Webnotizen (die überaus “tolle” Funktionalität unserer Webanwendung) erstellen

Nun wollen wir mittels Ajax unsere Webanwendung ein wenig aufmotzen, der Nutzen wird der gleiche bleiben, aber es wird mehr Spass machen Notizen anzulegen und zu löschen ;)

Um Ajax einfach und schnell verwenden zu können, verwenden wir Scriptaculous: <http://script.aculo.us/downloads>

Wir erzeugen im Root-Verzeichnis unseres Webservers einen Ordner namens “ajaxstuff” und dort fügen wir folgende Dateien (welche alle im entpackten Scriptaculous-Order zu finden sind) ein:

- dragdrop.js
- effects.js
- prototype.js
- scriptaculous.js

Um Prototype und die Funktionalität von Scriptaculous in unserem Projekt bekanntzumachen, müssen wir noch zwei Zeilen in unserer header.tpl hinzufügen, so sollte das Ergebnis dann aussehen:

```
<html>
<head>
<title>Meine Webanwendung</title>
<link rel="stylesheet" href="style.css" type="text/css" media="screen" />
<script src="ajaxstuff/prototype.js" type="text/javascript"></script>
<script src="ajaxstuff/scriptaculous.js" type="text/javascript"></script>
</head>
<body>
```

Als erstes wollen wir eingetragene Webnotizen auf unserer Übersichtsseite schnell löschen können, natürlich ohne Reload sondern mittels Ajax. Dazu sind folgende Schritte nötig:

1. **Anpassen der Output.php** - jede Webnotiz soll ein kleines "x"-Zeichen tragen, ein Klick auf dieses Zeichen soll die Webnotiz löschen
2. Ebenfalls in der output.php werden wir dem "x"-zeichen eine Javascript-onclick Methode verpassen und sobald das "x" geklickt wird, wird eine PHP-Datei mittels Ajax aufgerufen
3. die PHP-Datei löscht dann die jeweilige Webnotiz
4. ein Scriptaculous-Effekt wird dafür zuständig sein, dem User ein Feedback darüber zu geben das die Notiz gelöscht wurde

Löschen der Notiz mit Ajax

In der Output.php werden die folgenden zwei Zeilen aktualisiert / eingefügt:

```
$webnotizen .= '<div class="note" id="note_'. $row[id]. ' "><div class="notetitle">';
$webnotizen .= '<a class="delete" href="#" onclick="new Effect.Fade(note_'. $row[id]. ') ">x</a>';
```

das ganze sollte dann so aussehen:

```
<?php
require_once ($_SERVER['DOCUMENT_ROOT'].'/DB/DB.php');
require_once 'includes/config.php';
require_once 'smarty/libs/Smarty.class.php';

$dbh = DB::connect(DB_SYSTEM_1);
$res =& $dbh->query("SELECT post.id_post AS id, post.title AS title, post.content AS content FROM
post, post_user WHERE post_user.id_user = '".$_SESSION[_authsession][data][id_user]."' AND
post_user.id_post = post.id_post");
$dbh->disconnect();

$webnotizen = "";
while ($row =& $res->fetchRow(DB_FETCHMODE_ASSOC)) {
    $webnotizen .= '<div class="note" id="note_'. $row[id]. ' "><div class="notetitle">';
    $webnotizen .= '<a class="delete" href="#" onclick="new
Effect.Fade(note_'. $row[id]. ') ">x</a>';
    $webnotizen .= $row[title];
    $webnotizen .= '</div><div class="notetext">';
    $webnotizen .= $row[content];
    $webnotizen .= '</div></div>';
}

$smarty->assign('webnotizen', $webnotizen);

?>
```

Zur Erläuterung:

Wir belegen unseren Link mit einer "onclick" Funktion, wird der Link geklickt soll keine neue Seite aufgerufen werden (darum auch die "#"), sondern stattdessen ein Instanz der Klasse "Effect" von Scriptaculous erzeugt werden, in unserem speziellen Fall "Fade". Wer Lust hat mit den Effekten herumzuspielen, kann "Fade" auch einfach mal durch folgendes ersetzen:

- SwitchOff

- Puff
- BlindUp
- Shrink

und hier finden sich alle Effekte: <http://wiki.script.aculo.us/scriptaculous/show/CombinationEffectsDemo>

Ich pflücke den Code nochmal etwas auseinander:

```
while ($row =& $res->fetchRow(DB_FETCHMODE_ASSOC)) {
$webnotizen .= '<div class="note" id="note_'. $row[id]. '"><div class="notetitle">';
$webnotizen .= '<a class="delete" href="#" onclick="new Effect.Fade(note_'. $row[id]. '">x</a>';
$webnotizen .= $row[title];
$webnotizen .= '</div><div class="notetext">';
$webnotizen .= $row[content];
$webnotizen .= '</div></div>';
}
```

Wir durchlaufen in diesem Block unsere Ergebnis-Werte die wir durch unseren Datenbank-Query erhalten haben. Unser Ergebnis sind alle Notizen die wir für unseren Benutzer angelegt haben. Mittels “\$row” können wir für jeden Schleifendurchlauf auf die jeweilige Ergebnis-Zeile zugreifen und in jeder Zeile haben wir Zugriff auf die Werte “id”, “title”, und “content”. Wenn ich also innerhalb der Schleife folgendes aufrufe:

```
$row[id]
```

dann erhalte ich die ID der gerade ausgewählten Notiz, genauso bekomme ich dann mit \$row[content] denn Inhalt oder mittels “\$row[title]” den Titel der Notiz.

Eine wichtige Veränderung die ich durchgeführt habe findet in folgender Zeile statt:

```
$webnotizen .= '<div class="note" id="note_'. $row[id]. '"><div class="notetitle">';
```

Nun bekommt jeder Notiz-Div-Kalender eine eindeutige ID zugewiesen, also “note_1”, “note_2”, “note_15” usw. jenachdem welche ID in der Datenbank selektiert wurde für die Notiz. Dies ist wichtig damit wir in der darauffolgenden Zeile der Scriptaculous Klasse die ID übergeben können, für welche der Effekt “Fade” ausgeführt werden soll. An Scriptaculous wird bei einem Klick auf einen Link dann nur noch die ID übergeben und Scriptaculous selbst übernimmt dann das Ausfaden, indem es sich das Div-Element im DOM sucht, für welches die ID zutrifft.

Mehr zum DOM (Document Object Model):

- http://de.wikipedia.org/wiki/Document_Object_Model
- <http://www.webkrauts.de/2006/12/02/was-ist-dom-scripting/>

Nun können wir bei einem Klick auf das “x”-Zeichen unsere Notizen löschen...nein, ganz so einfach ist es nicht, mittels Scriptaculous werden nun zwar bei einem Klick die Div-Container ausgeblendet, gelöscht wird aber noch nichts, denn dafür braucht es noch ein PHP-Script welches aufgerufen werden muss und in dem dann mittels eines Datenbank-Queries der jeweilige Datensatz entfernt wird, doch dazu später mehr, erstmal soll nun der “Löschen”-Link etwas aufpoliert werden, dazu muss unser Stylesheet durch folgende Zeile erweitert werden:

```
.delete {color: red; text-decoration: none; margin-right: 15px; font-size: 1.5em;}
```

So, optisch ist die Funktionalität nun da (wenn auch unter Usability-Gesichtspunkten missverständlich), jetzt wollen wir das bei einem Klick auf das “x” die jeweilige Notiz auch aus der Datenbank gelöscht wird. Dazu legen wir als erstes einen Ordner namens “ajaxscripts” im Ordner “filesystem” an. Darin erstellen wir die Datei “delete_note.php”, diese Datei wird durch einen Ajax-Request aufgerufen und darin löschen wir die jeweilige Notiz.

Löschen von Notizen mit Ajax

Der Code für die Datei “delete_note.php” ist folgender:

```
<?php
    require_once ($_SERVER['DOCUMENT_ROOT'].'/DB/DB.php');
    require_once ($_SERVER['DOCUMENT_ROOT'].'/includes/config.php');

    $note_id = $_GET['note_id'];

    $dbh = DB::connect(DB_SYSTEM_1);
    $res =& $dbh->query("DELETE FROM post WHERE id_post = '". $note_id. "'");
    $dbh->disconnect();

?>
```

Codeerklärung “delete_note.php”

Die delete_note.php ist dafür da, von der Ajax-Engine angesprochen zu werden, eine ID zu erhalten und mittels dieser ID einen Wert aus der Datenbank zu löschen. Der Aufruf des PHP-Scripts erfolgt asynchron, doch dafür müssen wir nochmal in die output.php und dort den “Löschen”-Link anpassen, denn zusätzlich zum Scriptaculous Effekt soll nun ja auch noch etwas passieren. Dafür brauchen wir einen Ajax-Request, den wir mit dem Scriptaculous Effekt verschmelzen:

```
$webnotizen .= '<a class="delete" href="#" onclick="new Effect.Fade(\'note_\' . $row[id] . \''); new
Ajax.Request(\'/filesystem/ajaxscripts/delete_note.php\', {method: \'post\',
parameters: \'note_id=\' . $row[id] . \'});">x</a>';
```

Das “onclick”-Ereignis ist nun ganz schön vollgepackt, neben dem “Ausfaden” haben wir nun auch noch den Ajax Request da drin. Der Ajax.Request unterteilt sich in folgende zwei Bereiche:

```
new Ajax.Request(url, options);
```

Die URL gibt den Ort an, wo das PHP Script zu finden ist und in options können wir allerhand zusätzlichen Kram packen, wir begnügen uns an dieser Stelle mit der Methode “post” und den Parametern die wir dem Script mitschicken. Benötigt wird die ID einer Notiz, und diese soll das Script auch bekommen.

Notizen platzieren mit “Draggable”

Um noch ein wenig die eigene Kreativität anzuregen und zu zeigen was Ajax sonst noch kann, wollen wir nun noch die Möglichkeit bieten das man seine Notizzettel frei auf dem Bildschirm platzieren kann. Als erstes machen wir unsere Notizen “frei platzierbar” und in einem weiteren Schritt sollen die Koordinaten auch in der Datenbank mittels Ajax abgespeichert werden um bei einem Reload der Webanwendung (oder nach einem Logout/Login) die Notizen an der gleichen Stelle wiederzufinden, wo man sie abgelegt hat.

Wir nutzen die Klasse “Draggable” zum freien platzieren unserer Notizen. Die Datei output.php erweitern wir um folgenden Code:

```
$note_drag .= '
<script type="text/javascript">
    var messages = document.getElementsByClassName('note');
    for (var i = 0; i < messages.length; i++) {
        new Draggable(messages[i].id, {ghosting:true, revert:false})
    }
</script>
';

$smarty->assign('note_drag', $note_drag);
```

Dieser Code muss nach der Ausgabe unserer Webnotizen erfolgen, sprich nach der “While”-Schleife in der wir der

Variablen “\$webnotizen” die Werte zugewiesen haben. Der Javascript-Code macht folgendes:

- Er holt sich alle Elemente die die Klasse “note” tragen (alle unsere Notizen tragen diese Klasse)
- Dann wird eine Schleife erstellt die solange läuft bis alle Notizen erfasst wurden
- Jede Notiz wird “draggable” gemacht, “ghosting” bedeutet das wenn man ein Element bewegt, dass das Element vorerst an der Ursprungsposition bleibt und man nur eine Kopie bewegt und “revert: false” sagt aus das Elemente, welche bewegt wurden, nicht wieder zu ihrer Startposition zurückspringen

Wir speichern das Javascript in der Smarty-Variablen “note_drag” und müssen, um den Inhalt der Variablen auch auszugeben, in der Datei **index.tpl** folgendes nach “{\$webnotizen}” ausgeben:

```
{note_drag}
```

Jetzt lassen sich unsere Notizen schon lustig auf dem Bildschirm platzieren, als nächstes werden die Koordinaten abgespeichert, dazu nutzen wir die Spalten „xpos“ und „ypos“ die uns für jede Notiz in der Datenbank zur Verfügung steht:

Dadurch legen wir zwei weitere Spalten an, so können wir für jede Notiz individuell die X- und die X-Position abspeichern. Standardmäßig sollten alle schon nun angelegten Notizen mit der X- und Y-Pos “0” versorgt sein. Um diese Koordinaten auch verwenden zu können, müssen wir einen weiteren Eingriff in die **output.php** wagen.

Die Zeile mit dem Datenbankquery muss ersetzt werden durch:

```
$res =& $dbh->query("SELECT post.id_post AS id, post.title AS title, post.content AS content, post.xpos AS xpos, post.ypos AS ypos FROM post, post_user WHERE post_user.id_user = '".$_SESSION[_authsession][data][id_user]."' AND post_user.id_post = post.id_post");
```

Nun selektieren wir noch zusätzlich die xpos und ypos-Spalte, welche wir gerade eben angelegt haben. Mit diesen Werten können wir jedem Element seine eigene Position geben, dazu ändern wir die erste Zeile in der While-Schleife zu:

```
$webnotizen .= '<div class="note" id="note_'.$row[id].'" style="position: absolute; left: '.$row[xpos].'; top: '.$row[ypos].';"><div class="notetitle">';
```

Die Elemente bekommen eine absolute Position damit wir ihnen ihre x- und y-Koordinate geben können. Nachdem man eine Notiz gedragged und dropped hat, soll die neue Koordinate ermittelt und mittels Ajax an ein Script geschickt werden, dieses Script speichert dann für die jeweilige Notiz den x- und y-wert in der Datenbank ab. Der folgende Code könnte für Scriptaculous Neulinge recht komplex wirken, ich musste mich auch erst durch einige Foren kämpfen bevor ich einen Lösungsweg fand und zu dem lauffähigen Ergebnis kam. Probiert es einfach mal aus und erfreut euch an dem Effekt ;)

Freie Platzierung, wie auf dem Desktop

Folgende Schritte sind zum Ziel nötig:

- Anlegen einer Datei “save_position.php”
- Anpassen des Konstruktors “Draggable”

Als erstes legen wir die Datei save_position.php im Ordner “/filesystem/ajaxscripts/” an. Diese Datei erhält von uns eine ID, eine Xposition und eine YPosition und updated für die jeweilige ID die Position in der Datenbank.

saveposition.php

```

<?php
    require_once ($_SERVER['DOCUMENT_ROOT'].'/DB/DB.php');
    require_once ($_SERVER['DOCUMENT_ROOT'].'/includes/config.php');

    $note_id = $_POST['note_id'];
    $xpos = $_POST['xpos'];
    $ypos = $_POST['ypos'];

    $note_id = str_replace("note_", "", $note_id);

    $dbh = DB::connect(DB_SYSTEM_1);
    $res =& $dbh->query("UPDATE post SET xpos = '". $xpos."', ypos = '". $ypos.'" WHERE id_post =
    '". $note_id."'");
    $dbh->disconnect();
?>

```

Hier geschieht nichts spektakuläres, wir erhalten xpos, ypos und den ID-Namen des Elements, welches verändert werden soll. Da wir aber nur den Namen bekommen, also zum Beispiel “note_21”, “note_52” oder “note_12232” und wir uns erst die ID aus diesem Konstrukt basteln müssen, löschen wir den Anfangspart mit str_replace. Also “note_” fliegt raus und wir erhalten unsere ID.

Und nun kommt der Brocken:

Ersetzt in der output.php den Teil, wo die Variable “\$note_drag” den Javascript Part zugewiesen bekommt durch folgenden:

```

$note_drag .= '
<script type="text/javascript">
    var messages = document.getElementsByClassName('note');
    for (var i = 0; i < messages.length; i++) {
        new Draggable(messages[i].id, {ghosting:false, revert:false, endeffect:
function(element) {
    var toOpacity = typeof element._opacity == 'number' ? element._opacity : 1.0;
    new Effect.Opacity(element, {duration:0.2, from:0.7, to:toOpacity,
    queue: {scope:'_draggable', position:'end'},
    afterFinish: function(){
        Draggable._dragging[element] = false
    }
    });
    new Ajax.Request('/filesystem/ajaxscripts/save_position.php',{method: 'post',
parameters:'note_id='+element.id+'&xpos='+document.getElementById(element.id).style.left+'&ypos='+d
ocument.getElementById(element.id).style.top});
    });
}
}
</script>
';

```

Uff, was ist denn hier geschehen? Vorher hatte der Konstruktor “Draggable” lediglich die Optionen “ghosting:false” und “revert:false” nun ist aber noch eine weitere hinzugekommen, “**endeffect**”.

Wir schauen uns den Inhalt von “endeffect” im Detail an:

```

endeffect: function(element) {
    var toOpacity = typeof element._opacity == 'number' ? element._opacity : 1.0;
    new Effect.Opacity(element, {duration:0.2, from:0.7, to:toOpacity,
    queue: {scope:'_draggable', position:'end'},
    afterFinish: function(){
        Draggable._dragging[element] = false
    }
    });
}

```

```
parameters:'note_id='+element.id+'&xpos='+document.getElementById(element.id).style.left+'&ypos='+document.getElementById(element.id).style.top);
}
```

Es geschieht folgendes: Wenn endeffect eintritt (also wenn der drag beendet ist, der User das Element also losgelassen hat) dann starte die Funktion und überbe der Funktion den aktuellen ID-Namen des Elements (also z.b.: note_12, note_34, note_1232, usw). Die folgenden 7 Zeilen sind entnommen aus "draganddrop.js", dies sind die Standardaktionen von Scriptaculous für die Drag-Funktionalität.

Nun wird es interessant: Wenn der Drag erfolgreich abgeschlossen wurde, führen wir mittels Ajax.Request eine Anfrage an eine PHP-Datei aus. Wir rufen also die Datei "save_position.php" und übergeben dieser die aktuelle ID des Elements welches gedragged und gedropped wurde (schlimmes D-Englisch) und die aktuelle X- und Y-Position für das Element mittels

```
document.getElementById(element.id).style.left
document.getElementById(element.id).style.top
```

Nun sollten sich die Notizen frei auf dem Bildschirm platzieren lassen und auch nach einem Logout/Login an der gewünschten Position verweilen. Probiert es einfach mal aus, es ist wirklich beeindruckend was mit Prototype und Scriptaculous möglich ist.

Weiterführende Quellen:

- Scriptaculous-Einstieg: <http://wiki.script.aculo.us/scriptaculous/show/Usage>
- Diskussion zu Draggables: <http://wiki.script.aculo.us/scriptaculous/discuss/Draggables>
- Ajax-Request: <http://wiki.script.aculo.us/scriptaculous/show/Ajax.Request>

Zum Schluß

So, hiermit geht der Webdesign Guide zuende und ich hoffe ihr konntet einen Nutzen daraus ziehen. Auch wenn es lange gedauert hat, mir hat es wirklich Spass gemacht diesen Guide zu schreiben und ich denke es werden weitere folgen.