

# *Webapplikationen mit PHP 4*

**Bernhard Fürst**

**Lehrgang im HRZ der TU  
Bergakademie Freiberg**

---

Dieser Lehrgang soll einen Einblick in Wege und Möglichkeiten geben, wie mit der Programmiersprache PHP komplexe Webapplikationen erstellt werden können.

## *Eine kurze Einführung in PHP*

---

### **Was ist PHP**

Seit der ersten einsetzbaren Version, die vor kaum 6 Jahren das Licht der sich damals noch im Säuglingsstadium befindlichen WWW-Welt erblickte, ist PHP von vielen Websites nicht mehr wegzudenken. Diesen Erfolg verdankt PHP einigen wenigen grundlegenden Faktoren:

- es ist schnell erlernbar, der PHP-Novize kann mit den ersten einfachen Befehlen dynamische Webseiten erzeugen,
- es existiert eine Vielzahl von Funktionen, die verschiedenste Bereiche wie Textmanipulation, Datenbankanbindung, Internetprotokolle, Dateisystemfunktionen, Sessionmanagement uvm. unkompliziert abdecken,
- PHP ist auf nahezu jedem Betriebssystem einsetzbar, daß für Webserver genutzt werden kann (selbst Windows zählt dazu) und PHP ist Open Source und damit frei, kostenlos und technologisch hochaktuell

PHP fühlt sich beim Programmieren an wie C, JavaScript oder Java, ist jedoch eine reine Scriptsprache (Bytecode-Compiler seit Version

4). Alle Datentypen, die ein Programmierer erwartet, sind vorhanden. PHP übernimmt die komplette Speicherverwaltung, Datentypen werden dynamisch erzeugt. PHP implementiert in Ansätzen Konzepte der objektorientierten Programmierung (Klassen, Vererbbarkeit), die das Programmieren komplexer Datenstrukturen vereinfachen können. Detailliertere Informationen dazu findet man im Manual<sup>1</sup>.

## *Dynamische Webseiten - was ist das?*

---

### **Statische Webseiten**

Am einfachsten werden Informationen im Web von statische Webseiten zur Verfügung gestellt. Statische Webseiten werden per Hand von einem Menschen erstellt, gespeichert und sind damit endgültig festgelegt. Jede kleinste Änderung muß manuell getätigt werden. Das ist für kleine Webangebote in Ordnung, bringt bei großen Websites jedoch schnell einen großen redaktionellen Aufwand mit sich. Natürlich kann eine statische Webseite bewegliche Inhalte wie Animationen enthalten. Dennoch fehlt die Möglichkeit, die Informationen während der „Laufzeit“ der Webseite an verschiedene Erfordernisse wie z.B. Benutzereingaben anzupassen.

### **Dynamische Webseiten**

Das ist der Punkt, an dem die Webseite von einem statischen zu einem dynamischen Gebilde wechseln muß. Dynamisch heißt hier nicht die bewegte Grafik auf der Webseite, sondern dynamisch sind die Inhalte der Webseite, die sich nach aktuellen Erfordernissen ändern. Das kann im einfachsten Fall ein Seitenzähler sein, der mit jedem Aufruf der Webseite erhöht wird. Umfangreichere Anwendungen sind Online-Shops oder Newsticker. Eines haben alle dynamischen Webseiten gemeinsam: sie greifen auf irgendeine externe Datenquelle zu, in der die anzuzeigenden Inhalte abgelegt sind. Das ist beim Seitenzähler vielleicht eine einfache Datei, bei einem Online-Shop in der Regel ein Datenbanksystem. Allerdings geben weder Dateisystem noch Datenbanksystem Information im HTML-Format aus. Es wird also eine Schnittstelle gebraucht, die Informationen aus diesen Quellen in das Format umwandelt, die von üblichen Webbrowsern verstanden werden. Außerdem sollte diese Schnittstelle in der Lage sein, auf verschiedene Anforderungen eines Webseitenbenutzers zu reagieren. Die Schnittstelle muß also logische Operationen ausführen können. Genau so eine Schnittstelle ist PHP und noch dazu eine funktionell sehr umfangreiche. Insgesamt stellen Webseite, Datenbank und ihre Schnittstelle ein System dar, daß funktionell einem Programm

---

1. Im Web: <http://www.php.net/manual/>

(Applikation) wie z.B. einer Textverarbeitung ähnelt. Nur läuft diese Applikation nicht auf einem lokalen Computer ab, sondern im Web. Man spricht deswegen von einer Webapplikation.

## *Webapplikationen*

---

### **Was zeichnet eine Webapplikation aus?**

Wie im vorigen Abschnitt erwähnt, sind Webapplikationen in der Lage, aus dynamischen Inhalten Webseiten zu erzeugen. Dabei sollten einige grundlegende Dinge beachtet werden:

1. Sicherheit für Benutzer und Betreiber
2. Handling mehrerer Benutzer gleichzeitig
3. Benutzbarkeit der Webapplikation

### **Sicherheit für Benutzer und Betreiber**

Eherne Grundregel, die allseits und immer gilt: *Don't trust the Web*. Etwas ausführlicher: alle Daten, die eine Webapplikation aus dem Internet erhält, sind a priori als unsicher einzustufen. In den letzten Jahren hat es genügend Beispiele für gehackte Webserver gegeben, die unter anderem durch manipulierte Daten möglich waren. Trotzdem gehört dieser Punkt wahrscheinlich zu den stiefmütterlich beachteten Punkten der Webprogrammierung (ähnlich wie die Dokumentation des Quellcodes, die gerne vernachlässigt wird).

### *Überprüfung der Daten*

Die Überprüfung übermittelter Daten ist eine wichtige Aufgabe, die einfach zu implementieren ist. So kann beispielsweise das Eingabefeld für eine Postleitzahl von vornherein auf 5 Stellen begrenzt werden und sollte darauf überprüft werden, ob nur Zahlen eingegeben wurden (gilt mindestens in Deutschland). Trifft das nicht zu, werden die eingegebenen Daten nicht weiterverarbeitet und mit einem entsprechenden Hinweis erneut zur Korrektur präsentiert. Erst, wenn alle Daten den Vorstellungen der Webapplikationen entsprechen, dürfen sie von ihr weiterverarbeitet und gespeichert werden.

### *Herkunft der Daten*

Wichtig ist auch, auf die Herkunft der Daten zu achten. Das HTTP-Protokoll, welches für die Datenübermittlung zwischen Benutzer und Webapplikation zuständig ist, kennt dazu verschiedene Wege. Daten aus einem HTML-Formular beispielsweise können via POST- oder GET-Methode übertragen werden. PHP ist in seiner Voreinstellung so konfiguriert, daß das Eingabefeld eines HTML-Forms in PHP als globale Variable erscheint, unabhängig von POST- oder GET-Methode.

Folgendes Szenario stellt das damit verbundene Sicherheitsproblem kurz dar: Ein HTML-Formular mit einem Texteingabefeld

```
<input name="sicherheit" type="text">
```

wird via GET-Methode zum Webserver abgesendet. Die im Feld eingegebenen Daten stehen jetzt in PHP in der Variable `$sicherheit` und in dem Array `$HTTP_GET_VARS["sicherheit"]` zur Verfügung. PHP kopiert also die Daten automatisch in den globalen Namensraum und stellt parallel dazu ein Array bereit, mit dem gezielt nur auf GET-Variablen zugegriffen werden kann. Böswillige Websurfer können allerdings zusätzlich zu den GET-Variablen modifizierte Daten gleichen Namens via POST übertragen. Beim Kopieren der Daten in die globale Variable `$sicherheit` muß PHP sich nun zwischen GET- und POST-Wert entscheiden und gibt in standardgemäß dem POST-Wert den Vorzug. Somit hätte der Hacker seine Daten in die Webapplikation geschmuggelt. Greift die Webapplikation allerdings über das Array `$HTTP_GET_VARS["sicherheit"]` auf die Daten zu, gibt es kein Problem. Folgende `HTTP_*` Arrays gibt es:

- `HTTP_GET_VARS` (Variablen via POST-Methode)
- `HTTP_POST_VARS` (Variablen via POST-Methode)
- `HTTP_COOKIE_VARS` (via Cookie übertragene Variablen)
- `HTTP_POST_FILES` (Informationen zu HTTP File Uploads)
- `HTTP_SERVER_VARS` (auf den Webserver bezogene Variablen)
- `HTTP_ENV_VARS` (Umgebungsvariablen des Betriebssystems)

Auch wenn es komplizierter erscheint - diesen Arrays sollte in sicherheitsrelevanten Umgebungen immer der Vorzug vor der in den globalen Namensraum kopierten Variable gegeben werden. Sicherheitshalber konfiguriert man PHP so, daß keine Variable in den globalen Namensraum kopiert wird (`register_globals = off` in der `php.ini`).

### *Handling mehrer Benutzer gleichzeitig*

Im Gegensatz zu einer Textverarbeitung auf einem lokalen Arbeitsplatzrechner hat eine Webapplikation immer mit mehreren Benutzern gleichzeitig zu tun. Es wäre fatal, wenn die Online-Bestellung eines Autos für Benutzer A durch Benutzer B bezahlt wird, der eigentlich nur ein Tafel Schokolade haben wollte, aber kurz vor Benutzer A auf den Knopf „Bestellung absenden“ gedrückt hatte. Im HTTP-Protokoll ist leider keine dauerhafte Verbindung - möglicherweise noch verschlüsselt - vorgesehen. HTTP ist ein sogenanntes *stateless protocol*. Das bedeutet, daß nach der Übermittlung von Daten die Verbindung zwischen Webbrowser und Webserver beendet wird und beim nächsten Aufruf der Webseite keine Erinnerung mehr an Variablenzustände vom vorherigen Aufruf existieren. Das würde eine Programmierung von Webapplikationen, die über eine Webseite gehen, unmöglich machen. Dieses Dilemma löst man im einfachsten Fall, indem Variableninhalte von Seite zu

Seite in der URI oder in der Webseite selbst mit versteckten Form-Feldern (`<input type="hidden">`) übertragen werden. Damit entstehen gleich mehrere Probleme, nämlich muß man jedesmal alle übertragenen Daten erneut auf Richtigkeit überprüfen, da sie ja wieder aus dem potentiell unsicheren Internet kommen. Und das Programmieren selbst wird erschwert, da jeder kleinste Link einen kompletten Rattenschwanz von Variablen erhalten muß bzw. zwischen Seiten nur noch mittels HTML-Forms navigiert werden kann, um die versteckten Form-Felder übertragen zu können.

### *Sessionmanagement*

Wesentlich eleganter wird dieses Problem mit einem Sessionmanagement gelöst, über das PHP seit Version 4 verfügt. Dabei werden alle relevanten Daten nur einmal vom Benutzer an den Webserver gesendet, dort von der Webapplikation geprüft und auf dem Server temporär zwischengespeichert. Der Benutzer erhält eine eindeutige Session ID, durch welche die Webapplikation den Benutzer identifizieren und mit den zu ihm gehörenden Daten verbinden kann. Diese Session ID muß natürlich von Seite zu Seite weitergegeben werden, der Aufwand für diese eine Variable ist aber wesentlich geringer, als z.B. für ein komplettes großes Formular. Bei PHP werden Session IDs via Cookies weitergegeben, oder automatisch an jeden Link angehängt. (Weitere - mit etwas Aufwand verbundene Methoden - sind die Weitergabe der Session ID im Pfad oder im DNS-Namen des Webservers.)

### **Benutzbarkeit der Webapplikation**

Wie jedes andere Computer-Programm soll auch eine Webapplikation einfach anzuwenden sein. Das schnelle Umsetzen von Ideen mit PHP und HTML ist ein unbestrittener Vorteil, der komplexen Applikationen jedoch schnell in einem Bedien-Chaos enden läßt.

Hier in kurzer Folge ein paar Randpunkte:

- Sollte die Erwartungen der Benutzer bezüglich Benutzung erfüllen
- Selbsterklärend
- Der Benutzer will schnell und einfach sein Ziel erreichen
- Kein Informations-Overload, nur die gerade benötigten Daten anbieten
- Voreinstellungen bei Eingaben anbieten (vom Benutzer änderbar)
- Keine unnötigen Schritte vom Benutzer verlangen (wie z.B. Fehlermeldungen, die auf Falscheingaben hinweisen ohne die direkte Möglichkeit der Korrektur)
- Hat der Benutzer eine falsche Funktion gewählt, braucht er einen sicheren Rückweg, ohne Daten zu verlieren oder neu eingeben zu müssen

- Auf die verschiedenen Erfahrungs- und Wissensstufen der Benutzer einstellen (Hilfestellungen für Anfänger, direkte Wege für erfahrene Benutzer)
- Den Benutzer über den Status der Webapplikation informiert halten
- Konsistenz in der Benutzerführung in allen Bereichen der Webapplikation
- Vom Betriebssystem angebotene Schaltknöpfe und andere Elemente der Benutzeroberfläche (GUI) sollten genutzt werden, wenn möglich. Kein eigenes GUI entwerfen.

Viele nützliche Tips zur Benutzbarkeit von Softwaresystem findet man bei einem der bekanntesten Gurus, Jacob Nielsen<sup>1</sup>. Seine zweiwöchentlich per Mail verschickte *Alertbox* ist ein Muß für Systemdesigner.

---

1. <http://www.useit.com>