

# Aufbau einer dynamischen Webseite mit Apache, PostgreSQL, Perl und Mason auf NetBSD

Stefan Schumacher

stefan@net-tex.de, PGP:0xB3FBAE33

<http://www.net-tex.de>

<http://www.cryptomancer.de>

\$Id: napp.tex,v 1.29 2006/06/27 20:34:16 stefan Exp \$

## Zusammenfassung

Dieser Artikel beschreibt den Aufbau einer komplexen Webseite mit Datenbankbindung aus NetBSD, Apache, PostgreSQL und Perl/Mason an einem konkreten Beispiel. Grundkenntnisse in NetBSD (bzw. in einem Unixsystem), Datenbanken und Perl werden hierbei als gegeben vorausgesetzt.

## Inhaltsverzeichnis

---

<b>1</b>	<b>Warum NetBSD, Apache, PostgreSQL, Perl, Mason?</b>	<b>2</b>
1.1	NetBSD . . . . .	2
1.2	Pkgsrc . . . . .	2
1.3	Apache . . . . .	3
1.4	PostgreSQL . . . . .	3
1.5	Perl . . . . .	3
1.6	Mason . . . . .	3
1.7	Zusammenspiel mit DBD/DBI und Mod_perl . . . . .	4
<b>2</b>	<b>Installation</b>	<b>4</b>
2.1	NetBSD, Pkgsrc und Pakete . . . . .	4

<b>3</b>	<b>Einrichtung</b>	<b>6</b>
3.1	Apache . . . . .	6
3.1.1	SSL für Apache . . . . .	7
3.2	PostgreSQL . . . . .	7
3.2.1	Rutinewartung . . . . .	8
3.3	Mod_perl und DBI/DBD . . . . .	9
3.4	Datenbankgestützte Authentifizierung und Autorisierung mit Apache::AuthDBI . . . . .	10
3.5	persistente Datenbankverbindungen mit Apache::DBI . . . . .	12
3.6	Apaches Logdaten an PostgreSQL verfüttern . . . . .	13
3.7	Übertragungen mit gzip komprimieren . . . . .	15
<b>4</b>	<b>Mason</b>	<b>15</b>
4.1	Perl einbetten . . . . .	15
4.2	Komponenten . . . . .	17
4.2.1	Unterkomponenten . . . . .	18
4.2.2	Daten zwischen Komponenten austauschen . . . . .	18
<b>A</b>	<b>Webskript</b>	<b>20</b>
<b>B</b>	<b>AuthDBI Beispiel</b>	<b>21</b>

## 1 Warum NetBSD, Apache, PostgreSQL, Perl, Mason?

### 1.1 NetBSD

NetBSD ist als Abkömmling der Berkeley-Linie ein waschechtes Unix und damit natürlich bestens für den Einsatz als Serversystem gerüstet. Es verfügt über alle notwendigen Programme, eine ausgereifte Entwicklungspolitik, stabile und sichere Software und mit Pkgsrc ein ausgezeichnetes Werkzeug zur Softwareverwaltung. Schließlich unterstützt NetBSD ein Vielzahl von Netzwerkprotokollen und Sicherheitsmechanismen und ist unter der äußerst flexiblen BSD-Lizenz kostenlos und frei erhältlich. Weitere Informationen zu NetBSD finden sich auf der NetBSD-Projektseite <http://www.netbsd.org>.

### 1.2 Pkgsrc

Pkgsrc<sup>1</sup> (»Package Source«) ist ein Framework um auf NetBSD Software installieren zu können. Vereinfacht gesagt handelt es sich hierbei um eine Verzeichnisstruktur die Makefiles enthält um den benötigten Quellcode herunterzuladen und kompilieren zu können. Es ist also mit einem einfachen `make install clean` möglich, komplette Softwarepakete inkl. Abhängigkeiten zu

---

<sup>1</sup><http://www.pkgsrc.org>

installieren. Pkgsrc ist auch für andere Systeme als NetBSD verfügbar, ich werde mich aber hier auf NetBSD mit Pkgsrc beschränken. Alle im weiteren besprochenen Pakete sind via Pkgsrc installierbar.

Zusätzlich gibt es noch den pkgsrc-wip<sup>2</sup> (»Work in Progress«) Zweig, der neue Pakete nach Pkgsrc bringen soll. Daher wird er noch nicht als stabil geführt, ermöglicht aber so den experimentellen Einsatz neuer Software-Versionen.

### 1.3 Apache

Apache ist der weltweit verbreitetste Webserver und in seinem Funktionsumfang kaum zu schlagen. Er unterstützt den Einsatz von Modulen und ist somit sehr leicht erweiterbar. Auch Apache ist freie Software und als solche frei und kostenlos im Quellcode erhältlich.

### 1.4 PostgreSQL

Als Forschungsprojekt in Berkeley entstanden, ist PostgreSQL heute das weltweit beste freie objektrelationale Datenbankmanagementsystem. PostgreSQL unterstützt eine Vielzahl von Datenbankmanagementsystemfunktionen, wie bspw. Transaktionen, Point-in-Time-Recovery, Trigger und Rules oder Vererbung. PostgreSQL ist über Netzwerkverbindungen kontaktierbar kann von verschiedensten Programmiersprachen angesprochen werden. Außerdem unterstützt PostgreSQL-serverseitige Programmiersprachen, unter anderem PL/PgSQL und PL/Perl.

PostgreSQL ist in der Berkeleytradition stehend selbstverständlich ebenfalls freie Software und unter der BSD-Lizenz erhältlich.

### 1.5 Perl

Perl ist eine weit verbreitete und mächtige Skriptsprache, für die unzählige Module existieren. Sie kann mit Mod\_perl an Apache angebunden werden und so ein dynamisches Webinterface erzeugen. Für Perl existiert außerdem DBI/DBD, das den Zugriff auf Datenbanken wie PostgreSQL ermöglicht. Perl ist also hervorragend geeignet Daten aus PostgreSQL abzufragen, aufzubereiten und via Apache und Mod\_perl als HTML auszuliefern.

### 1.6 Mason

Mason ist ein Framework für Perl, das speziell dazu geschrieben wurde Webseiten zu erstellen. Es bettet dazu Perl in HTML ein, und nicht umgekehrt, wie man es mit CGI.pm oder einem einfachen Mod\_perl-Skript machen muss. Mason ist wesentlich mehr als eine einfache Template-Maschine, es ist in der

---

<sup>2</sup><http://pkgsrc-wip.sourceforge.net/>

Lage äußerst komplexe Webseiten modular zu entwickeln. Dabei ist natürlich eine strikte Trennung von Perlcode und HTML sehr einfach zu erreichen. Damit lässt sich die Entwicklung einer Webseite zwischen verschiedenen Personen<sup>3</sup> aufteilen.

Mason vereinfacht die Seitenerstellung, da man sich nicht durch Escapekaskaden (»Backslashitis«) quälen muss, um bspw. in einem `print`-Statement die Gänsefüßchen (") als Backslash + Gänsefüßchen (\") auszugeben.

Weiterhin bietet Mason einige sehr interessante Funktionen um komplexe Webseiten zu verwalten und Perl vom HTML-Code zu trennen. Damit können auch Nichtprogrammierer ihren HTML-Code einpflegen, ohne sich um Perl-Funktionen kümmern zu müssen.

Mason kann Seiten in sogenannte Komponenten zerlegen und diese einladen, so kann man bspw. den immer gleich aufgebauten Kopf einer Seite in je einer Datei anlegen und diese von allen Unterseiten einlesen lassen. Außerdem kann sich Mason um Caching und Variablenübergabe kümmern.

Mason läuft zwar auch ohne `Mod_perl`, mit dem Paket aber wesentlich schneller, so das man es auf jeden Fall installieren sollte.

## 1.7 Zusammenspiel mit DBD/DBI und Mod\_perl

Die Datenhaltung und -abfrage erfolgt in PostgreSQL, Perl bereitet die Daten auf und lässt sie von Apache als Webseite ausliefern. Für die Verbindung von Perl mit PostgreSQL bzw. Apache sorgen hier die Perl-Module DBI und DBD-PostgreSQL, bzw. `Mod_perl` (auch `ap_perl` in `Pkgsrc` genannt).

DBI ist das »DataBase Interface« und stellt für Perl generische Datenbankzugriffsmethoden dar. Der »DataBase Driver« DBD setzt auf DBI auf und implementiert die PostgreSQL-spezifischen Perlmethoden.

`Mod_perl` ist ein Modul, das den Perlinterpret in den Webserver implementiert, wodurch Perlskripte direkt von Apache ausgeführt werden (lassen) können. Außerdem werden diese Perlskripte im RAM des Server vorgehalten und können so ohne einen neuen Prozess gestartet werden.

## 2 Installation

### 2.1 NetBSD, Pkgsrc und Pakete

Die Installation von NetBSD geschieht am einfachsten über CD-ROM oder FTP. Der Installationsprozess wird auf der jeweiligen Portseite beschrieben. Die Einrichtung des Systems wird im NetBSD-Shortguide<sup>4</sup> beschrieben. Es sollten das Basissystem (Benutzer, Netzwerk, IPFilter) konfiguriert werden. Anschließend installiert man `Pkgsrc` via `cvs`:

---

<sup>3</sup>z.B. Perlentwickler, Datenbanker, Snowboarder ...

<sup>4</sup><http://www.lindloff.com/netbsd/handbuchmap.html>

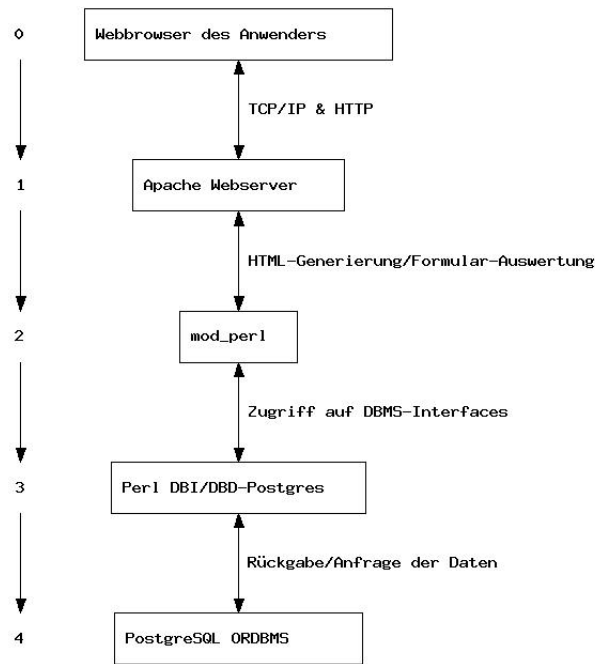


Abbildung 1.7.1: Schema des Serversystems

```
# cd /usr/  
# setenv CVS_RSH ssh  
# setenv CVSRROOT anoncvs@anoncvs.netbsd.org:/cvsroot  
# cd /usr  
# cvs checkout -PA pkgsrc
```

### Beispiel: 1: Pkgsrc via CVS installieren

In `/usr/pkgsrc` liegt nun das Pkgsrc-Framework, mithilfe dessen man Software installieren kann. Pkgsrc lädt automatisch alle benötigten Quellen herunter und kompiliert sowie installiert diese samt Abhängigkeiten.

Um alle benötigten Pakete zu installieren reichen folgende Befehle:

```
# cd /usr/pkgsrc/databases/postgresql82
# make install clean
# cd /usr/pkgsrc/www/apache
# make install clean
# cd /usr/pkgsrc/databases/p5-DBD-postgresql/
# make install clean
# cd /usr/pkgsrc/www/ap-perl/
# make install clean
# cd /usr/pkgsrc/www/p5-HTML-Mason/
#make install clean
```

### Beispiel: 2: Pakete via Pkgsrc installieren

Um den Server abzusichern sollte man einige Sicherheitssoftware einrichten, bspw. AIDE<sup>5</sup>, welches die Integrität des Dateisystems überwacht, Audit-packages, das die Installierten Pakete auf Sicherheitsrisiken hin überprüft oder Portsentry, der die TCP/IP-Ports auf Scans überwacht.

## 3 Einrichtung

### 3.1 Apache

Apache wird in `/usr/pkg/etc/httpd/httpd.conf` konfiguriert. Anzupassen sind hierbei vor allem:

```
# Adresse des Administrators
ServerAdmin you@your.address
# Pfad in dem die HTML Seiten liegen sollen
DocumentRoot "/home/www"

#mod_perl einbinden um Perl anwenden zu können
LoadModule perl_module lib/httpd/mod_perl.so
<Files ~ "\.pl$" >
    PerlHandler          Apache::Registry
    PerlSendHeader       On
    SetHandler           perl-script
    Options               +ExecCGI
</Files>

#mason einbinden
<FilesMatch ".mhtml">
SetHandler perl-script
PerlHandler HTML::Mason::ApacheHandler
</FilesMatch>
```

<sup>5</sup><http://www.net-tex.de/unix/aide.html>

### Beispiel: 3: Apache in der httpd.conf konfigurieren

Man kann nun `/home/www` erstellen und als Homeverzeichnis des Users `www` einrichten. Anschließend erstellt man eine einfache `/home/www/index.html`:

```
<html>
<body>
Hallo WWW-Welt!
</body>
</html>
```

### Beispiel: 4: einfaches HTML-Beispiel

und startet Apache mit `apachectl start` als `root`. In einem beliebigen Browser kann man die IP-Adresse des Apache öffnen und sollte so unsere »Hallo Welt«-Nachricht lesen können.

#### 3.1.1 SSL für Apache

Apache unterstützt selbstverständlich auch SSL-Verschlüsselung für die laufenden Server. Apache 2.0 hat SSL-Unterstützung bereits in der Basisversion implementiert, Apache 1.3 benötigt dazu `pkgsrc/www/ap-ssl/`.

Nachdem man `Ap-ssl` installiert hat, muss mit OpenSSL der Serverschlüssel und das Zertifikat für Apache erstellt werden:

```
{1} openssl genrsa -des3 -out server.key 1024
{2} openssl req -new -key server.key -out server.csr
{3} /usr/bin/openssl x509 -req -days 365 -in \
    server.csr -signkey server.key -out server.crt
{4} cp server.key /usr/pkg/etc/httpd/ssl.key/
{5} cp server.crt /usr/pkg/etc/httpd/ssl.crt/
{6} chmod 0400 /usr/pkg/etc/httpd/ssl.crt/server.crt \
    /usr/pkg/etc/httpd/ssl.key/server.key
```

### Beispiel: 5: OpenSSL Schlüssel und Zertifikat generieren

unter 3. wird der »Common Name« verlangt, das ist der FQDN (Hostname) des Servers.

## 3.2 PostgreSQL

PostgreSQL wurde bereits aus `Pkgsrc` heraus installiert und muss nun konfiguriert werden. Zuerst benötigt man ein Verzeichnis für die Datenhaltung (»Cluster«), welches mit entsprechenden Zugriffsrechten versehen und initiiert werden muss.

```
# mkdir /usr/pkg/pgsql/  
# chown pgsql.pgsql /usr/pkg/pgsql/  
# chmod 700 /usr/pkg/pgsql/  
# su -m pgsql -c 'initdb -D /usr/pkg/pgsql/'
```

**Beispiel: 6:** PostgreSQL einrichten

Nun können wir den PostgreSQL Server (`postmaster`) starten und einen Datenbankbenutzer sowie eine Datenbank anlegen.

```
# su pgsql  
$ postmaster -D /usr/pkg/pgsql/  
$ createuser stefan  
$ createdb mitarbeiter
```

**Beispiel: 7:** PostgreSQL Cluster einrichten

PostgreSQL kann besonders komfortabel über das Terminal `psql` bedient werden. Hierzu muss es lediglich mit der gewünschten Datenbank als Parameter aufgerufen werden:

```
$ psql -Udbnutzer datenbank
```

**Beispiel: 8:** psql starten

Jetzt kann im Terminal mit den gängigen SQL- bzw. Datenbankbefehlen gearbeitet werden:

```
=> create table mitarbeiter(name varchar(150),mtime timestamp,  
    ident serial unique primary key);  
=> copy bsdnode from '/home/daten/mitarbeiter.csv'  
    with delimiter ',';  
=> create table rechner(inventarnummer int;  
    hersteller varchar(250));  
=> insert into rechner values(165829,  
    'Digital Equipment Corporation');
```

**Beispiel: 9:** Tabellen erstellen und befüllen

### 3.2.1 Routinewartung

PostgreSQL muss regelmäßig verschiedene Wartungsarbeiten durchführen, u. a. VACUUM und ANALYZE,

- um den von aktualisierten oder gelöschten Zeilen belegten Speicherplatz wiederzugewinnen
- um die vom PostgreSQL-Anfrageplaner verwendeten Datenstatistiken zu aktualisieren

- um sich gegen den Verlust sehr alter Daten durch Überlauf der Transaktionsnummern zu schützen.

Um den Speicherplatz nicht mehr verwendeter Zeilen wiederzugewinnen und die Statistik für den Anfrageplaner zu aktualisieren, ist ein Durchlauf von `vacuumdb` notwendig, dazu kann von einem cronjob das Systemprogramm `vacuumdb(1)` aufgerufen werden:

```
$ vacuumdb -UDBNUTZER -a -f -z
```

#### Beispiel: 10: Datenbanken pflegen

Hierbei werden alle Datenbanken (-a) voll (-f) mit VACUUM behandelt und ANALYZE (-z) ausgeführt. Ab PostgreSQL 7.4 kann VACUUM parallel zum Nutzerbetrieb ausgeführt werden, trotzdem erzeugt ein volles VACUUM eine signifikante Systemlast, daher sollte man den cronjob des nächstens laufen lassen.

Selbstverständlich ist eine regelmäßige Sicherung der Daten erforderlich. Der effektivste Weg, die Anwendungsdaten zu sichern, ist das Erzeugen eines Backups mit `dump` und die Sicherung dessen auf Wechselmedien.

Die zu sichernden Daten beschränken sich auf Anwendungsdaten der Datenbank. Der Datenbankinhalt wird vor der Sicherung mit `pg_dump` in eine Datei gedumpt, die im zu sichernden Verzeichnis liegt.

Die Befehle zur Sicherung der Daten:

```
pg_dump -USICHERUNGSNUTZER rdw | bzip2 -9 \  
> pgdump-'date +%y%m%d'.bz2  
cp pgdump-'date +%y%m%d'.bz2 /mnt/floppy/
```

#### Beispiel: 11: Datenbanken sichern

anschließend kann man den dump auf ein Wechselmedium sichern und vom Server löschen.

### 3.3 Mod\_perl und DBI/DBD

Apache ist nun angepasst um Perlskripte auszuführen und als simples HTML auszuliefern. Dazu muss das Skript lediglich mit der Endung `.pl` im Datenverzeichnis (DocumentRoot) abgelegt und aufgerufen werden. Ein einfaches Perlskript sieht so aus:

```
#!/usr/pkg/bin/perl  
print "Content-type: text/plain\n\n";  
print "Dies ist test.pl auf \n"; print 'hostname';  
print "mod_perl ist geil\n";
```

**Beispiel: 12:** einfaches Mod\_perl Beispiel

und kann ganz normal über `http://127.0.0.1/index.pl` aufgerufen werden. Selbstverständlich können mit diesen Fähigkeiten auch äußerst komplexe Skripte erstellt werden, die mit bereits vorhandenen Modulen erweitert werden können.

Der Zugriff auf Datenbanken erfolgt mithilfe der bereits installierten Perlmodule DBI/DBD und gestaltet sich in der Regel sehr einfach:

```
#Datenbankbenutzer, -passwort, -name
$user = 'stefan';
$pw = '' ;
$db='mitarbeiter';
#Verbindung oeffnen
my $dbh = DBI->connect("dbi:Pg:dbname=$db","$user","$pw");

### Anfrage stellen und fehlerbehandeln
$query='select name from mitarbeiter order by name;';
my $sth = $dbh->prepare("$query");
if (!$sth) {die "Error:" . $dbh->errstr . "\n";}
if (!$sth->execute) {die "Error:" . $sth->errstr . "\n";}
my $names = $sth->{'NAME'};
my $numFields = $sth->{'NUM_OF_FIELDS'};

### Ausgabe der Daten
while (my $ref = $sth->fetchrow_arrayref){
print "Mitarbeiter ist: $$ref[0]\n";
}
```

**Beispiel: 13:** Datenbankzugriff mit Perl

Anhang 1 zeigt og. Beispiel als Perlskript für Apache/Mod\_perl.

### 3.4 Datenbankgestützte Authentifizierung und Authorisierung mit Apache::AuthDBI

`AuthDBI.pm` ermöglicht es Apache die Benutzerverwaltung mit einer beliebigen DBI-Datenbank abzugleichen. Dies ist insbesondere dann von Vorteil wenn viele verschiedene Benutzer auf die Webverzeichnisse zugreifen sollen oder die eigentliche Benutzerverwaltung über die Webseiten selbst erfolgen soll.

Das Modul muss dazu wie üblich via `httpd.conf` eingebunden werden:

```
PerlModule Apache::AuthDBI
```

**Beispiel: 14:** Apache::AuthDBI in `httpd.conf` einbinden

Nun muss in der jeweiligen `.htaccess` im entsprechenden Verzeichnis das Modul explizit benutzt werden:

```
AuthName "Geben Sie Benutzernamen und Passwort an:"
AuthType Basic

PerlAuthenHandler Apache::AuthDBI::authen
PerlAuthzHandler Apache::AuthDBI::authz
PerlSetVar Auth_DBI_encrypted on

PerlSetVar Auth_DBI_data_source dbi:Pg:dbname=apacheuser
PerlSetVar Auth_DBI_username dbnutzer
PerlSetVar Auth_DBI_password geheim

PerlSetVar Auth_DBI_pwd_table wwwuser
PerlSetVar Auth_DBI_uid_field uid
PerlSetVar Auth_DBI_pwd_field passwd
PerlSetVar Auth_DBI_grp_field gid

require group gruppenname user
```

**Beispiel: 15:** Apache::AuthDBI über `.htaccess` aktivieren

Zuerst wird Apache angewiesen das AuthDBI-Modul zur Authentifizierung und Authorisierung zu verwenden. Das in der Datenbank abgelegte Passwort wird als verschlüsselt deklariert. Die Benutzereingabe im Passwortfeld wird dazu erst mit `crypt()`<sup>6</sup> verschlüsselt und dann mit dem Datenbankinhalt verglichen. Sollte also jemand unerlaubterweise Zugriff auf die Datenbank bekommen, hat er nur Zugriff auf die verschlüsselten Passwörter.

Im dritten Block werden die Daten zur PostgreSQL-Datenbank (`apacheuser`) angegeben.

Abschließend wird die Struktur der Tabelle `wwwuser` angegeben. Notwendig sind mit mindestens die beiden Spalten `uid` und `passwd` für Name und Passwort, eine Spalte für die Gruppe ist optional, für große Seiten aber empfohlen.

Abschließend wird wie gewohnt mit der `require`-Direktive Apache angewiesen die Gruppe `gruppenname` anzufordern und jedes Mitglied zuzulassen.

Öffnet nun ein Benutzer ein Dokument das per Passwort geschützt ist, gibt er wie üblich seinen Benutzernamen und das passende Passwort im Browser ein. Diese beiden Daten werden von Apache aus der Datenbank abgefragt und mit den Eingaben verglichen. Stimmen sie überein, erhält der Benutzer Zugriff.

Über ein einfaches HTML-Formular oder automatisierte Perlroutinen können Benutzer in der Tabelle manipuliert werden.

<sup>6</sup>Zur Zeit (03.07.05) wird gerade an SHA1/MD5 gearbeitet

Wenn `Apache::AuthDBI` eingesetzt wird, sollte unbedingt ebenfalls `Apache::DBI` aktiviert werden, um persistente Datenbankverbindungen zu etablieren. Siehe dazu Kap. 3.5.

### 3.5 persistente Datenbankverbindungen mit `Apache::DBI`

Apache führt jedes Perlskript als eigenen Prozess aus, daher ist es nicht möglich eine persistente Datenbankverbindung aufzubauen, d. h. es ist nicht möglich in einem Skript eine Datenbankverbindung aufzubauen und diese in einem anderen Skript weiterzubenutzen. Dies ist problematisch, da insbesondere der Verbindungsauf- und -abbau zur Datenbank eine recht hohe Last erzeugt die die der eigentlichen

Abhilfe schafft `Apache::DBI`<sup>7</sup>, indem sich Apache selbst an der Datenbank anmeldet und so eine persistente Datenbankverbindung ermöglicht. Seiten, die recht häufig Datenbankverbindungen erzeugen, können so die Last enorm verringern und an Geschwindigkeit gewinnen, da in der Regel der Aufwand zum nackten Verbindungsauf/-abbau den der eigentlichen Datenbankabfrage weit übersteigt.

```
PerlModule Apache::DBI
... andere DBI Module ...
```

#### Beispiel: 16: `Apache::DBI` einbinden

`Apache::DBI` muss vor allen anderen DBI-Modulen geladen werden, da normale DBI-Module beim Verbindungsaufbau die Kontrolle an `Apache::DBI` delegieren. `Apache::DBI` speichert das erzeugte Datenbankhandle in einem globalen Hash und überläd die normale `DBI-disconnect`-Methode mit einem Plazebo. Dieses Verfahren hat auch den Vorteil, das vorhandene Perlskripte nicht verändert werden müssen.

Um die persistente Verbindung zu testen, kann in einem beliebigen Perlskript

```
$Apache::DBI::DEBUG = 2;
$Apache::AuthDBI::DEBUG = 2;
```

#### Beispiel: 17: `Apache::DBI` Debuglevel setzen

benutzt werden. Erscheint dann im Apache ErrorLog folgendes:

```
99 Apache::DBI need ping: yes
99 Apache::DBI already connected to
  'dbname=apacheuser^\\XXXXXX^\\AutoCommit=1^\\
  PrintError=1^\\Username=XXXXXX'
99 Apache::DBI disconnect (overloaded)
```

<sup>7</sup>[pkgsrc/databases/p5-Apache-DBI](#)

**Beispiel: 18:** Apache ErrorLog Eintrag zu Apache::DBI

ist die Verbindung persistent.

Alternativ lässt sich das Modul auch über ein startup-Perlskript starten, dann werden bereits beim Start von Apache alle notwendigen Verbindungen aufgebaut und zusätzlich kann Cacheing für Apache::AuthDBI aktiviert werden, so dass nicht jedes Passwort in der Datenbank abgefragt werden muss, sondern im Apachespeicher vorgehalten wird.

Eine Beispiel-startup.pl aus Apache::AuthDBI gibt es im Anhang B.

**3.6 Apaches Logdaten an PostgreSQL verfüttern**

Da man auf diesem Webserver schon PostgreSQL installiert hat, kann man natürlich auch dafür sorgen dass Apache seine Logdaten nicht nur in die Logdateien schreibt, sondern auch in eine PostgreSQL Datenbank.

Dazu existieren p5-Apache-DBILogger und p5-Apache-DBILogConfig. Ersteres lässt Apache an eine beliebige DBI-Datenbankquelle loggen, letzteres konfiguriert das ganze via httpd.conf.

Man kann beide Pakete wieder via PostgreSQL installieren und dann per httpd.conf folgendermaßen einbinden:

```
PerlLogHandler Apache::DBILogConfig
PerlSetVar DBILogConfig_data_source DBI:Pg:apache
PerlSetVar DBILogConfig_username PGuser
PerlSetVar DBILogConfig_password PGpasswd
PerlSetVar DBILogConfig_table log
PerlSetVar DBILogConfig_log_format "%a=ip %t=time
    %U=requrl %{Referer}i=referer %b=bytes_sent
    %f=filename %h=remote_host %r=request %s=status"
```

**Beispiel: 19:** Apache::DBILogConfig in httpd.conf einbinden

Wobei DBILogConfig\_data\_source den DBI-Datenbanktreiber und die Datenbank angeben, DBILogConfig\_username und DBILogConfig\_password geben den PostgreSQL Benutzer und dessen Passwort an, DBILogConfig\_table die entsprechende Tabelle in der Datenbank apache. DBILogConfig\_log\_format schlussendlich definiert das Logformat und somit auch das Format der eingesetzten Tabelle. Die Notation entspricht der aus httpd.conf bekannten.

Die passende PostgreSQL-Tabelle sieht so aus:

Table "public.log"		
Column	Type	Modifiers
ip	character varying(20)	
time	character varying(30)	
requrl	text	
referer	text	
bytes_sent	character varying(10)	
filename	character varying(300)	
remote_host	character varying(300)	
request	character varying(300)	
status	character varying(300)	
zeitstempel	timestamp without time zone	

Indexes:

- "ip" btree (ip)
- "request" btree (requrl)

**Beispiel: 20:** Die public.log-Tabelle

```
CREATE TABLE log (
  ip character varying(20),
  "time" character varying(30),
  requrl text,
  referer text,
  bytes_sent character varying(10),
  filename character varying(300),
  remote_host character varying(300),
  request character varying(300),
  status character varying(300),
  zeitstempel timestamp
);
```

**Beispiel: 21:** Erzeugung der Tabelle log

Standardmäßig fügt DBI-LogConfig in die Spalte `time` den Apache Zeitstempel ein. Dieser ist ein normales Textfeld und kann daher nicht mit der integrierten Zeitstempelarithmetik von PostgreSQL bearbeitet werden. Um dieses Problem zu umgehen habe ich das Feld `zeitstempel` als »timestamp« angelegt und in `DBILogConfig.pm`<sup>8</sup> Die SQL-Anfrage `my $statement = qq(INSERT INTO $table ($fields, zeitstempel) VALUES ($values, now())); $r->warn('DBILogConfig: statement=$statement');` um »`zeitstempel`« und »`now()`« erweitert. Somit fügt PostgreSQL bei jedem INSERT einen Zeitstempel ein.

<sup>8</sup>/usr/pkg/lib/perl5/site\_perl/5.8.6/Apache/DBILogConfig.pm

Nun kann man mit den PostgreSQL-Fähigkeiten die Logdaten analysieren, bspw. eine Sicht erstellen die nur einmal jede vorkommende Benutzer-IP anzeigt: `CREATE VIEW uniqip AS SELECT DISTINCT ip FROM log ORDER BY ip`; oder alle heutigen Zugriffe anzeigen: `CREATE VIEW heute AS SELECT * FROM log WHERE zeitstempel > 'today' ORDER BY zeitstempel`;

Um die Abfragegeschwindigkeit in der Datenbank zu erhöhen, kann man Indizes anlegen. Der Index reduziert die Kosten für Suchen und Sortieren, in dem die Spalte intern in einem B-Baum sortiert wird. Ein Index erzeugt aber eine interne Überlast, so daß er nur auf notwendigen Spalten (also häufig durchsuchten) angelegt werden sollte. Um einen Index über der Spalte `referer` anzulegen, verwendet man `CREATE INDEX meinindex ON log (referer)`;

### 3.7 Übertragungen mit gzip komprimieren

Das Modul `mod_gzip` bzw. `pkgsrc/www/ap-gzip/` ermöglicht es Apache auszuliefernde Daten während der Übertragung mit `gzip` zu komprimieren. Ist das Modul installiert und aktiviert wird es von Apache selbständig verwendet, wenn der Benutzerclient `gzip`-Kompression unterstützt.

Die Installation kann wieder sehr einfach via `pkgsrc` geschehen, anschließend muss lediglich noch

```
LoadModule gzip_module lib/httpd/mod_gzip.so
```

**Beispiel: 22:** `mod_gzip` einbinden

in die `httpd.conf` eingetragen werden.

## 4 Mason

Wie bereits eingangs erwähnt, ist Mason ein System das der Erstellung von Webseiten dient und in diese Perl einbetten kann. Es sollte unbedingt zusammen mit `Mod_perl` betrieben werden, da es so um den Faktor 10-20 schneller läuft.

Man sollte für Masondateien unbedingt eine eigene Dateiendung etablieren und in der `httpd.conf` (siehe 3.1) konfigurieren, meist verwendet man hierzu `.mhtml`, da es so nicht zu Problemen mit normalen `.html` kommt.

### 4.1 Perl einbetten

Eine einfache Masondatei kann aufgebaut sein wie eine einfache HTML-Datei, da Mason HTML-Befehle nicht beachtet und direkt an den Apache weiterleitet.

Perlcode kann hingegen in drei verschiedenen Varianten eingebettet werden:

```

1 <html><body>
2 Hallo HTML-Welt!<br>

3 %my $text="Hallo Masonwelt!";
4 <% $text %>

5 <%perl>
6 print "Hallo ";
7 print "Masonwelt!";
8 print "<br>";
9 my $sum=9+8;
10 print $sum;
11 </%perl>

12 </body></html>

```

**Beispiel: 23:** Perl in Masoncode einbinden

Die Zeilen 1, 2 und 12 geben hier wie gewohnt normalen HTML-Code aus.

Die Zeile 3 verwendet das Prozentzeichen, um bis zum Zeilenende Perlcode auszuführen. Hier wird der Variable `$text` ein String zugewiesen.

Zeile 4 verwendet die Masonnotation `<% $VARIABLE %>` um eine Variable auszugeben. Die `<% $VARIABLE %>`-Notation ermöglicht auch die automatische Ersetzung von HTML-/URL-Schlüsselwörtern. Die Ersetzung wird mit einer Pipe (—) angeschaltet und danach mit `h` für HTML Code (ö wird zu `&ouml`);, `u` für URL (ö wird zu `%C3%B6`) bzw. `n`

Dieser Code lässt sich auch beliebig in ganz normale HTML-Schnipsel einbinden.

Die Zeilen 5 und 11 erzeugen einen ganzen Perlblock, zwischen den beiden Tags kann ein beliebig langes Perlprogramm eingebaut werden, das prinzipiell alle Perlfunktionen nutzen kann.

Mason verwendet grundsätzlich `use strict;`, so daß Variablen immer mit `my` deklariert werden müssen und Referenzierung über Namen eingeschränkt ist. Mehr zu `strict` findet sich in `perldoc strict`.

Das ganze kann auch munter in einer Komponente gemischt werden:

```

<html><body>
Hallo HTML-Welt!<br>

<%perl>
my $var='pwd';
for my $i (1..99)
{
</%perl>

<% $i %> <br>

% }

<br>Wir sind in: <i><% $var %></i><br>
</body></html>

```

**Beispiel: 24:** Masonkomponente mit Perl und HTML

Innerhalb von Perlblöcken fungiert die Raute (#) als Kommentarzeichen, außerhalb davon gibt es die `<%doc>`Kommentar`</%doc>` Blöcke.

## 4.2 Komponenten

Mason verwendet Komponenten um eine Webseite zusammenzusetzen. Die Komponenten werden hierbei als anonyme Subroutine kompiliert, wobei `parser.pl` die Objekte `ApacheHandler`, `Interpreter` und `Parser` erzeugt. Normalerweise entscheidet ein Zeitstempel über die Neukompilierung einer Komponente, dies kann aber abgeschaltet werden.

```

1 <& ../../header.html &>

2 $m->comp("/priv/comp2uri", $comp1, $comp2, ...);

3 <& ../../footer.html, autor=>'Stefan' &>

```

**Beispiel: 25:** Einbindung von Komponenten

Zeile 1 und 3 binden je eine Komponenten, die lediglich ausgegeben werden soll, d.h. die Komponente kann keinen Rückgabewert zurückgeben. Man kann aber problemlos, wie in Zeile 3 gezeigt, eine Variable übergeben und bearbeiten lassen.

Zeile 2 hingegen ruft eine Komponente auf, deren Rückgabewert weiterverarbeitet werden soll. Die Komponente wird nicht ausgegeben, sondern ausgeführt und deren Ergebnis übergeben.

Mit dem Komponentensystem kann man Teile des Codes bzw. der Seite wiederverwenden. So habe ich z.B. für ein Videotheksprojekt den Kopf und

Fuß der Seite als Komponenten erstellt und in jeder anderen Seite entsprechend eingebunden. Für den Verwalter des Videothekenbestands existiert eine Seite um neue Videos zu erfassen. Dazu wird ein HTML-Formular angezeigt, das entsprechende Daten erfasst und an eine folgende »FORM ACTION«-Seite übergibt, welche die Daten wiederum in die Datenbank einträgt. Da die Kunden auch nach Videos suchen können, verwende ich dafür das selbe Formular. Über eine POST-Variable<sup>9</sup> wird festgestellt ob das Formular hinzufügen oder suchen soll und entsprechend die Datenbankenkomponente ausgewählt.

#### 4.2.1 Unterkomponenten

Eine Masonkomponente kann auch Unterkomponenten beinhalten. Die Unterkomponente selbst ist nur von ihrer jeweiligen Überkomponente aus erreichbar. Unterkomponenten bieten sich insbesondere dann an, wenn spezielle Funktionen in bestimmten Situationen mehrfach ausgeführt werden sollen.

```
<h1>Über- und Unterkomponenten</h1>
% foreach my $i (@mitarbeiter){
  <& .meine_uk, name => $suchname' &>
% }

<%def .meine_uk>
$query='select * from mitarbeiter where name ~*
        '%> $suchname %>'order by name;';
my $sth = $dbh->prepare("$query");
if (!$sth) {die "Error:" . $dbh->errstr . "\n";}
if (!$sth->execute) {die "Error:" . $sth->errstr . "\n";}
my $names = $sth->{'NAME'};
my $numFields = $sth->{'NUM_OF_FIELDS'};

while (my $ref = $sth->fetchrow_arrayref){
  <tr><td><% $$ref[0] %></td> <td><% $$ref[1] %></td></tr>
}
</%def>
```

Beispiel: 26: Unterkomponente in einer Überkomponente

#### 4.2.2 Daten zwischen Komponenten austauschen

Weiterführende Literatur zu Mason:

<sup>9</sup>

- *Dynamische Webseiten mit Mason erzeugen*  
Peter Dintelmann, iX 02/2002  
<http://www.heise.de/ix/artikel/2002/02/128/>
- *Perl in Webseiten einbetten mit HTML::Mason*  
Michael Schilli, Linux-Magazin 02/2001  
<http://www.linux-magazin.de/Artikel/ausgabe/2001/02/Perl/perl.html>
- Mason HQ  
<http://www.masonhq.com/>
- *Embedding Perl in HTML with Mason*  
Dave Rolsky & Ken Williams  
<http://www.masonbook.com/book/> (Online available)

## A Webskript

Listing 1: Perl Webskript

```

1 #!/usr/pkg/bin/perl -wT
2 ###CGI
3 use CGI;
4 $cgi = CGI->new;
5 print $cgi->header;      # function call: $obj->function
6 print $cgi->start_html("PostgreSQL_Database_System");
7 # Postgres
8 use DBI();
9
10 ### Metadata
11 my $dbh = DBI->connect("dbi:Pg:dbname=mitarbeiter","DB","XXX");
12 print "Mitarbeiter anzeigen:<br>";
13
14 ### Datenbank
15 $query='select name,vname,ident from mitarbeiter;';
16
17 my $sth = $dbh->prepare("$query");
18 if (!$sth) {die "Error:" . $dbh->errstr . "\n";}
19 if (!$sth->execute) {die "Error:" . $sth->errstr . "\n";}
20 my $names = $sth->{'NAME'};
21 my $numFields = $sth->{'NUM_OF_FIELDS'};
22
23 ### Mitarbeiterliste ausgeben
24 print "Mitarbeiterliste:<br>";
25
26 while (my $ref = $sth->fetchrow_arrayref){
27   print "Name: _$$ref[0] _Vorname: _$$ref[1]<br>";
28 }
29
30 print "<br><br><a href=\"../..\">Startseite</a><br>"

```

## B AuthDBI Beispiel

Listing 2: AuthDBI Beispiel

```

1 #!/usr/pkg/bin/perl -w
2
3 # um diese Datei beim Start einzubinden,
4 # muss sie in der httpd.conf angefordert werden:
5 PerlRequire /path/to/startup.pl
6
7 # Auf mod_perl pruefen:.
8 $ENV{MOD_PERL} or die "GATEWAY_INTERFACE_not_Perl!";
9
10 # notwendige Pakete einbinden:
11 use Apache::Registry;
12 use Apache::DBI;
13 use Apache::AuthDBI;
14 use strict;
15
16 #####
17 # Apache::AuthDBI.pm:
18 ###
19
20 # Debuglevel: 0 = aus, 1 = wenig, 2 = sehr viel
21 $Apache::DBI::DEBUG = 0;
22
23 # Hier werden Datenbankverbindung sofort beim Start von Apache
24 # etabliert. Achtung: Haengt die Verbindung, startet Apache nicht
25 # bis zum Timeout der Verbindung!
26 #Apache::DBI->connect_on_init
27 ("dbi:driver(AutoCommit=>1):database", "userid", "passwd");
28
29 # Pingverhalten der Datenbankverbindung:
30 # $timeout = 0 -> immer pingen (voreingestellt)
31 # $timeout < 0 -> niemals pingen
32 $timeout > 0 # -> nur pingen wenn die letzte DB-Verbindung
33             # laenger als der Timeoutintervall her ist
34 Apache::DBI->setPingTimeOut("dbi:driver:database", $timeout);
35
36 #####
37 # Apache::AuthDBI.pm
38 #####
39
40 # Debuglevel: 0 = aus, 1 = wenig, 2 = sehr viel

```

```
41 $Apache::AuthDBI::DEBUG = 0;
42
43 # Cacheverhalten, wenn aktiviert, werden die Benutzerdaten im
44 # Cache zwischengespeichert
45 # 0 = kein Cache, n > 0 ist der Zeitintervall in Sekunden,
46 # die ein zugriffener Eintrag im Cache verbleiben soll
47 Apache::AuthDBI->setCacheTime(15);
48
49 # Zeit in Sekunden die der GarbageCollector warten soll
50 # der GarbageCollector entfernt Eintraege die aelter als
51 # die og. n Sekunden sind
52 # -1 = GarbageColector deaktivieren
53 # 0 = nach jeder Anfrage laufen lassen
54 # n > 0 = nach n Sekunden laufen lassen, muss statistisch
55 # angepasst werden, welcher Intervall effizient ist
56
57 Apache::AuthDBI->setCleanupTime(-1);
58
59 # Shared Memory aktivieren, d.h. mehrere Apache teilen sich einen
60 # Speicherbereich fuer den Cache statt eines jeweils eigenen Caches
61 Apache::AuthDBI->initIPC(50000);
62
63
64 1;
```