Der allgemeine Listenserver

Codename: Curriculum Mundi

Ein Grobkonzept

von Thomas Blunck
(tblunck@gmx.de)

letzte Version 0.99 vom 19.11.2019
(entstanden 2001 – 2006)

Möge ein Anderer dieses Konzept realisieren!

Inhaltsverzeichnis

Einleitung

Der **allgemeine Listenserver** dient zur Erstellung, Bearbeitung und Darstellung von beliebigen Listen im Internet. Für jeden dieser Vorgänge ist eine Zugangskontrolle, d.h. die Einschränkung auf einen bestimmten Benutzerkreis möglich. Eine Liste besteht aus (in gewissen Grenzen) frei definierbaren Spalten verschiedener Datentypen und aus beliebig vielen Zeilen (= Datensätzen), wobei die Zugangskontrolle bis auf die Satzebene heruntergebrochen werden kann.
Wie später im Einzelnen dargestellt wird, kann man den allgemeinen Listenserver als eine Art Werkzeugkasten ansehen, der es Benutzern ohne Programmierkenntnisse erlaubt, einfache webbasierte Anwendungen zum sicheren Austausch strukturierter Daten selbst zu erstellen. In einer Ausbaustufe bietet er dem Benutzer sogar die Funktionalität einer hierarchischen Datenbank.

Im folgenden ersten Teil wird die Funktionalität des allgemeinen Listenservers genauer beschrieben. Im zweiten Teil werden einige Anwendungsszenarien beschrieben. In einem dritten Teil werden einige erste Überlegungen zur Realisierung, besonders hinsichtlich einer guten Skalierung bei wachsenden Anforderungen, dargestellt. Es folgen noch einige Gedanken zur Kommerzialisierung.
In einem Anhang befinden sich allererste Entwürfe zum Datenmodell und zur Oberfläche. Außerdem gibt es (aus historischen Gründen) noch einen Anhang, in dem sich der Sourcecode (in Perl) der allerersten Testversion zur Realisierung des Listenservers im Jahr 2006 befindet – noch ohne Datenbank und Rechteverwaltung.

Funktionalität

Erstellung von Listen

Ein Benutzer mit den dafür nötigen Rechten (im Folgenden ‚Owner' genannt) kann über einen webbasierte Oberfläche einen Liste erstellen. Jeder Liste gehört genau einem Owner und hat einen eindeutigen Namen innerhalb aller Listen desselben Owners. Nur der Owner darf die Struktur oder den Namen einer Liste ändern, beliebige Einträge ändern oder löschen, sowie die gesamte Liste löschen.
Eine Liste wird definiert durch eine geordnete Folge von Spalten, die jeweils einen Namen und einen Datentypen haben. Datentypen sind etwa (noch unvollständig, orientiert sich natürlich an Datentypen der Datenbank, ist aber nicht identisch damit – Einzelheiten siehe Realisierung):

- Datum (eventuell noch mit Parameter für Art der Darstellung, eventuell auch verbunden mit Uhrzeit)
- User (ein gültiger User im System)
- Text kurz (z.B. maximal 30 Zeichen; eventuell zusätzlicher Parameter, der die Länge in der Listendarstellung angibt)
- Text lang (z.B. maximal 256 Zeichen)
- Text mit Auswahl aus erlaubten Zeichenketten (zus. Parameter etwa ‚Text1/Text2/...')
- Zahl (mit Parameter über Anzahl Nachkommastellen)
- Email-Adresse (Textfeld, es wird zusätzlich geprüft, ob gültige Syntax)
- Web-Adresse (Textfeld, es wird zusätzlich geprüft, ob gültige Syntax)
- …

Für eine erste Version dürfte dies bereits reichen. Denkbar und wünschenswert wären in einer weiteren Ausbaustufe auch die beiden folgenden Datentypen:

- Anhang (eine oder mehrere Dateien, wie Email-Anhang)

- Liste (definiert eine abhängige (Teil-) Liste zu jedem einzelnen Datensatz; hier sollte man sich vermutlich auf eine einzige Liste beschränken, wobei dieser natürlich rekursiv weitere Listen enthalten kann – dies liefert dann die Funktionalität einer hierarchischen Datenbank).

ACHTUNG: Es sei an dieser Stelle bereits gesagt, dass nicht daran gedacht ist, eine Liste direkt eins zu eins in eine Datenbanktabelle abzubilden. Näheres siehe Teil 2 (Realisierung).

Jeder Listeneintrag enthält automatisch die folgenden Felder:

- Eindeutige ID (wird automatisch vom System gefüllt)
- User Ersterfassung (wird automatisch vom System gefüllt)
- Datum Ersterfassung (wird automatisch vom System gefüllt)
- User Letzte Änderung (wird automatisch vom System gefüllt)
- Datum Letzte Änderung (wird automatisch vom System gefüllt)
- Status (Einzelheiten hierzu später)
- Datum letzte Statusänderung (wird automatisch vom System gefüllt; eventuell kann hierauf verzichtet werden und eine Statusänderung wird wie eine normale Änderung behandelt)
- User Letzte Statusänderung (wird automatisch vom System gefüllt; eventuell kann hierauf verzichtet werden und eine Statusänderung wird wie eine normale Änderung behandelt)

Die Reihenfolge der Spalten definiert automatisch die Reihenfolge der Datenfelder in einer Erfassungsmaske (Felder dort vertikal angeordnet, von oben nach unten). Auf der Oberfläche zur Erstellung/Änderung der Listenstruktur muss auch eine Umsortierung möglich sein. Zusätzlich kann der Owner die Reihenfolge der Datenfelder in der Listendarstellung (für den Enduser, den eigentlicher Benutzer der Liste) definieren; dabei dürfen auch einzelne Spalten fehlen, die dann nur in einer Detailsicht (entspricht der Erfassungsmaske) sichtbar sind.

Schließlich kann für eine Liste noch definiert werden, ob in einer (oder mehreren) Spalten nur eindeutige Werte (also keine Dubletten) erlaubt sind. Dadurch kann man z.B. erreichen, dass jeder Enduser nur einen einzigen Eintrag in einer Liste machen kann oder dass ein Datum nicht mehrfach auftreten kann.

Beim Erstellen einer Liste wird eine eindeutige URL erzeugt, über die eine Liste als HTML-Seite oder Frame (innerhalb einer Webseite) aufgerufen werden kann (parametergesteuert). Eventuell kann über einen weiteren Parameter geregelt werden, ob die HTML-Seite beim Aufruf gleich die Listendarstellung enthält oder ein kleines Menü, in dem der Benutzer entscheiden kann, ob er die Liste sehen will oder eine Datenerfassung machen will.

Registrierung und Login

Für die meisten Anwendungsszenarien wird es nötig sein, dass sich ein Benutzer des Systems einmalig registriert (mit eindeutigem Benutzernamen, Emailadresse und Passwort) und dann bei jeder Nutzung ein Login macht (mit Benutzername/Passwort oder Emailadresse/Passwort). Selbstverständlich gibt es dabei auch den Turingtest/ die Roboterabfrage und das Speichern von Cookies, damit der Benutzer sich das wiederholte Einloggen erspart. Außerdem kann ein registrierter Benutzer sein Passwort ändern und seinen Account löschen.
Ein registrierter Benutzer kann dann natürlich (abhängig von den Benutzerrechten der jeweiligen Liste) mit einem Account die Listen verschiedener Owner benutzen. Man registriert sich also nicht für eine spezielle Liste, sondern ganz allgemein für eine Nutzung des Listenservers. Diese Nutzung kann das Erstellen von Listen beinhalten (für die man dann Owner ist), oder aber das passive (nur Lesen) oder aktive (auch Schreiben) Nutzen beliebiger Listen – natürlich abhängig von den weiter unten definierten Benutzerechten.

Zuordnung von Benutzerrechten

Per Default darf nur der Owner einer Liste Eingaben in diese machen und nur der Owner kann die Liste sehen (gut für Testzwecke und ganz spezielle Anwendungen).

Der Owner kann nun einstellen, dass

- jeder User die Liste sehen kann (also auch nicht registrierte User) oder nur registrierte User
- jeder registrierte User einen Listeneintrag machen kann (eventuell auch mehrere, abhängig von der Konfiguration der Liste)
- jeder Listeneintrag sofort in der Liste sichtbar wird (Status = Freigegeben) oder aber zunächst bis auf weiteres (siehe unten) unsichtbar bleibt (Status = Nicht Freigegeben)

Der Owner kann einer Liste verschiedene (registrierte) Benutzer (eventuell auch Benutzergruppen) mit unterschiedlichen Rechten zuordnen. Es können die folgenden Rechte vergeben werden:

- Moderatorenrecht: User, die das Moderatorenrecht an einer Liste haben, können alle Einträge lesen. Sie können Einträge löschen oder deren Status (aber keine anderen Daten) verändern – es gibt nur die zwei Stati: Freigegeben/Nicht Freigegeben.
- Leserecht: User, die das Leserecht an einer Liste haben, können alle Einträge mit dem Status ‘Freigegeben‘ lesen.
- Schreibrecht: User, die das Schreibrecht an einer Liste haben, können Einträge in diese Liste machen. Es kann für eine Liste konfiguriert werden, ob jeder Listeneintrag sofort in der Liste sichtbar wird (Status = Freigegeben) oder aber zunächst bis zur Freigabe durch die Moderatoren unsichtbar bleibt (Status = Nicht Freigegeben).

Noch offen: Sollen Benutzer auch (ihre eigenen) Einträge ändern oder löschen können. Dann müsste man dafür aber auch wieder eine Art Freigabe machen (eventuell den Status ‘gelöscht‘ einführen und die eigentliche Löschung den Moderatoren überlassen).

Listendarstellung und -bearbeitung

In der Listendarstellung werden die Spalten gemäß Strukturdefinition in einer Zeile dargestellt (Name der Spalte, Reihenfolge und eventuell Auslassung definierter Spalten).
Es sollte möglich sein

- den eventuell nicht sichtbaren Teil der Zeile durch scrollen zu erreichen – die zur Charakterisierung des Zeileninhalts wichtigen Felder sollten deshalb möglichst am Zeilenanfang stehen
- durch einen Button die Detailsicht auf den Datensatz aufzumachen – dort sind die einzelnen Datenfelder zeilenweise von oben nach unten angeordnet und es werden auch Felder dargestellt, die in der Listendarstellung nicht sichtbar sind. In dieser Darstellung können Listeneinträge (je nach den Rechten) auch bearbeitet werden
- durch Anclicken eines Buttons ‘Neuer Eintrag‘ (nur für User mit Schreibrechten) auch direkt in die (leere) Detailansicht zu kommen und dort Eingaben machen zu können
- durch Anclicken eines Spaltennamens die Sortierfolge der Liste zu ändern (alphabetisch nach den Einträgen der gewählten Spalte, abwechselnd aufsteigend oder absteigend). Die normale Sortierfolge ist in der Reihenfolge der Entstehung der Listeneinträge. Eventuell wird man auch eine spaltenbezogene Stringsuche anstoßen können
- In Spalten mit den Datenfeldern Email- bzw. Web-Adresse führt ein Anclicken natürlich in das Emailprogramm bzw. den Browser.

Export

Es muss für den Owner einer Liste jederzeit möglich sein, die komplette Liste (oder auch Teile davon, eventuell auch verwendete abhängige Teillisten) zu exportieren in einem Format, das den Import in z.B. Excel erlaubt. Der Listenserver selber soll keine Auswertungen/Statistiken erstellen können.

Kopien von Listenstrukturen

Es müsste auch möglich sein die Struktur von Listen zu kopieren und dabei mit einem neuen Namen zu versehen. Auf diese Weise könnten bestimmte Listen immer wieder in verschiedenen Kontexten genutzt werden. Dies müssen nicht notwendig nur eigene Listen sein. Denkbar wäre auch ein Pool von häufig genutzten Musterlisten, die jeder Benutzer kopieren und dann eventuell noch für seine Zwecke anpassen kann.

Email-Benachrichtigungen

Gewisse Aktionen könnten auch sinnvollerweise zu Email-Benachrichtigungen führen.
Wenn etwa der Moderator einen Datensatz freigibt oder löscht, dann sollte eine entsprechende Benachrichtigung an den Ersteller des Datensatzes erfolgen.
Konfigurierbar sollte auch der Owner einer Liste über Änderungen an dieser Liste informiert werden.

Anwendungsszenarien

Single-User

In diesem Szenario vergibt der Owner keine weiteren Schreib- oder Moderationsrechte. Beim Leserecht kann er sich entscheiden, ob er keinem (nur sich selber), allen oder einem eingeschränkten Nutzerkreis Leserechte gibt. Der Owner ist der alleinige Autor seiner Liste.
Beispiele:

- Dies könnten etwa Linklisten sein, die er in seiner eigenen Webseite einbaut. Er kann dann diese Listen jederzeit von jedem mobilen Endgerät mit Internetzugang zu modifizieren bzw. ergänzen, ohne umständlich HTML-Seiten zu editieren und diese dann hochzuladen
- Dies können Blogeinträge (eventuell ergänzt um relevante Strukturdaten) oder Listen von Sammlungen jedweder Art sein, die er auf seiner Webseite einbauen will.
- Dies kann eine Sammlung strukturierter Daten sein, die zunächst für die Öffentlichkeit nicht sichtbar ist und über deren weitere Verwendung erst später entschieden wird.

Restricted-User

In diesem Szenario erlaubt der Owner einer begrenzten Anzahl von Usern den Input in eine Liste. Das Leserecht kann (muss aber nicht) auf denselben Benutzerkreis eingeschränkt werden.
Beispiele:

- Ein Sportverein (Schachverein) führt eine Liste über die auszutragenden Spiele eines Turniers. Bestimmte (oder alle) Mitglieder des Vereins erhalten das Recht, die Ergebnisse (z.B. der Spiele an denen sie selber teilgenommen haben) in die Liste einzutragen. Die Liste ist dann sofort online (falls keine Moderation konfiguriert ist) und sehr aktuell
- Eine Firma kann die Tagesberichte/Projektberichte/Stundenzettel ihrer Mitarbeiter (speziell auch im Außendienst) sehr komfortabel online auf jedem mobilen Endgerät mit Internetzugang erfassen lassen und bei Bedarf jederzeit runterladen und auswerten. Der Listenserver erlaubt (per Export) die Pflege eines Excel-Arbeitsblattes direkt und unabhängig voneinander durch verschiedene Mitarbeiter, die jeweils genau eine Zeile dazu beitragen (und die anderen Zeilen auch eventuell garnicht sehen sollen).

In beiden oben genannten Fällen könnte man auch sinnvoll abhängige Listen einsetzen:

- Der Sportverein pflegt einen Kalender (Liste 1) über alle Tage an denen z. B. Mannschaftskämpfe stattfinden. Davon abhängig ist eine Liste 2, in der die Resultate eines Mannschaftskampfes eingetragen werden.
- Die Firma pflegt eine Liste aller Arbeitstage (Liste 1) und pro Arbeitstag kann sich dann der Mitarbeiter genau einmal in eine abhängige Liste mit seinem Tagesbericht eintragen.

Unrestricted-User

In diesem Szenario erlaubt der Owner beliebigen (aber registrierten) Usern den Input in seine Liste (und meistens wohl auch das Leserecht). Sicher wird es hier immer eine Moderation geben müssen!

Beispiele:

- Eine Hobby-Webseite erlaubt damit Usern mit ähnlichen Interessen einen Link einzutragen, der auf die Webseite dieses fremden Users führt. Heute ist dazu die fehlerträchtige Kommunikation (Email) und aufwendige Übertragung des Links auf die eigene Webseite nötig.
- Ein private Webseite kann selber eine (allerdings im Vergleich zu Ebay primitive) Auktion für zu veräußernde Gegenstände durchführen. Am einzelnen Gegenstand hängt eine Liste mit den Geboten - nur sichtbar für den Owner, der auch sehen kann von wem (Email) das Gebot kommt.

Realisierung

Hier soll nur auf einige zentrale Aspekte der Realisierung eingegangen werden. Ein erstes grobes Datenmodell befindet sich im Anhang. Dort befinden sich auch erste Entwürfe für die Oberfläche.

Mastertabelle

Vorbemerkung: wir reden hier von sehr vielen Listen sehr vieler Benutzer, die aber alle nur wenige Einträge haben - die allermeisten Listen sicher unter 100 Zeilen.
Der zunächst naheliegende Gedanke, jede Liste über eine eigene Datenbanktabelle zu realisieren, führt zu erheblichen Problemen. Bei jeder Liste wird es vermutlich speziell in der Anfangsphase noch viele Änderungen geben (Namens- und Strukturänderungen). Dies würde dann zu ständigen Änderungen an der Datenbankstruktur im laufenden Betrieb (bei gleichzeitigen Eingaben in diese Listen) führen. Es ist kaum vorstellbar, dass dies in performanter und skalierbarer Weise geschehen kann.
Es wird deshalb vorgeschlagen, alle Listen in einer **Mastertabelle** abzulegen, deren Struktur sich nicht mehr ändert. In dieser Mastertabelle würden die von der Datenbank benötigten Datentypen mehrfach angelegt werden (z.B. 5 Datenfelder Text1, Text2,…, Text5) und dann über eine weitere Tabelle ein Mapping der verschieden Spalten auf diese Datenfelder verwaltet werden. In dieser **Mappingtabelle** würden auch der Spaltenname und der spezielle Spaltentyp (Email, Webadresse würden beide auf ein Textfeld abgebildet werden) stehen.
Natürlich wäre damit eine gewisse Platzverschwendung gegeben (es gibt dann sicher viele ungenutzte Felder in der Mastertabelle), doch dürfte sich dies kaum auf die Performance auswirken (der Zugriff wird ja über eine Indextabelle mit dem Id der Liste gesteuert). Auch gibt es dann natürlich ein Limit für die Anzahl Spalten eines Typs. Hier muss vielleicht in einer Anfangsphase noch nachjustiert werden, doch dann sollte es für die ganz große Masse aller Listen reichen.
Die Datenbank wäre über eine Mastertabelle auch sehr schön skalierbar. Falls die Mastertabelle nämlich eine kritische Größe erreicht, so legt man einfach eine neue Mastertabelle (mit identischer Struktur) an. Es sollte nur gewährleistet sein, dass sich alle Einträge einer Liste in derselben Mastertabelle befinden.

Strukturänderungen

In der Mappingtabelle würden sich natürlich die ständigen Namens- und Strukturänderungen an den Listen niederschlagen – allerdings wären damit keinerlei Strukturänderungen an der Datenbank verbunden, sondern es wären nur Änderungen an Datenbanksätzen. Strukturänderungen jeglicher Art sind bei noch leeren Listen natürlich völlig problemlos. Wenn jedoch bereits Einträge gemacht wurden, dann kann es Probleme geben. Hier sollten die folgenden Grundsätze gelten:

- Es sollten nur entweder neue Spalten hinzugefügt oder bestehende Spalten gelöscht werden können – neue Spalten werden dabei grundsätzlich in allen existierenden Datensätzen auf eine definierte NULL gesetzt (was immer das auch für die Datenbank heißt)
- Eine Änderung des Datentyps einer Spalte ist nicht erlaubt, sehr wohl aber die Änderung des Namens
- Der Benutzer kann eine Änderung des Datentyps mit Übernahme der existierenden Einträge indirekt erreichen, in dem er zunächst nur eine neue Spalte des gewünschten Datentyps einfügt und dann manuell die Einträge umkopiert (soweit diese kompatibel sind) und anschließend die ursprüngliche Spalte löscht. Später kann man dies natürlich auch automatisieren.

Abhängige Listen

Abhängige Listen lassen sich sehr einfach wie folgt realisieren: Der Id der Ausgangsliste wird einfach mit in die Listendefinition der abhängigen Liste übernommen. Ansonsten ist die abhängige Liste eine ganz normale Liste (mit eigenen Zugriffsrechten, die sich von den Zugriffrechten der Ausgangsliste unterscheiden können). Auf der Oberfläche wird immer zunächst die Ausgangsliste dargestellt. Der Benutzer kann dann ausgehend von einer Zeile (etwa über einen Button ‘+‘ an dieser Zeile) die Zeilen der abhängigen Liste einblenden, die zu der Ausgangszeile gehören (und denselben Id haben).

Kommerzialisierung

Eine Basisfunktionalität sollte kostenlos zur Verfügung stehen. Denkbar wären hier Einschränkungen bezüglich

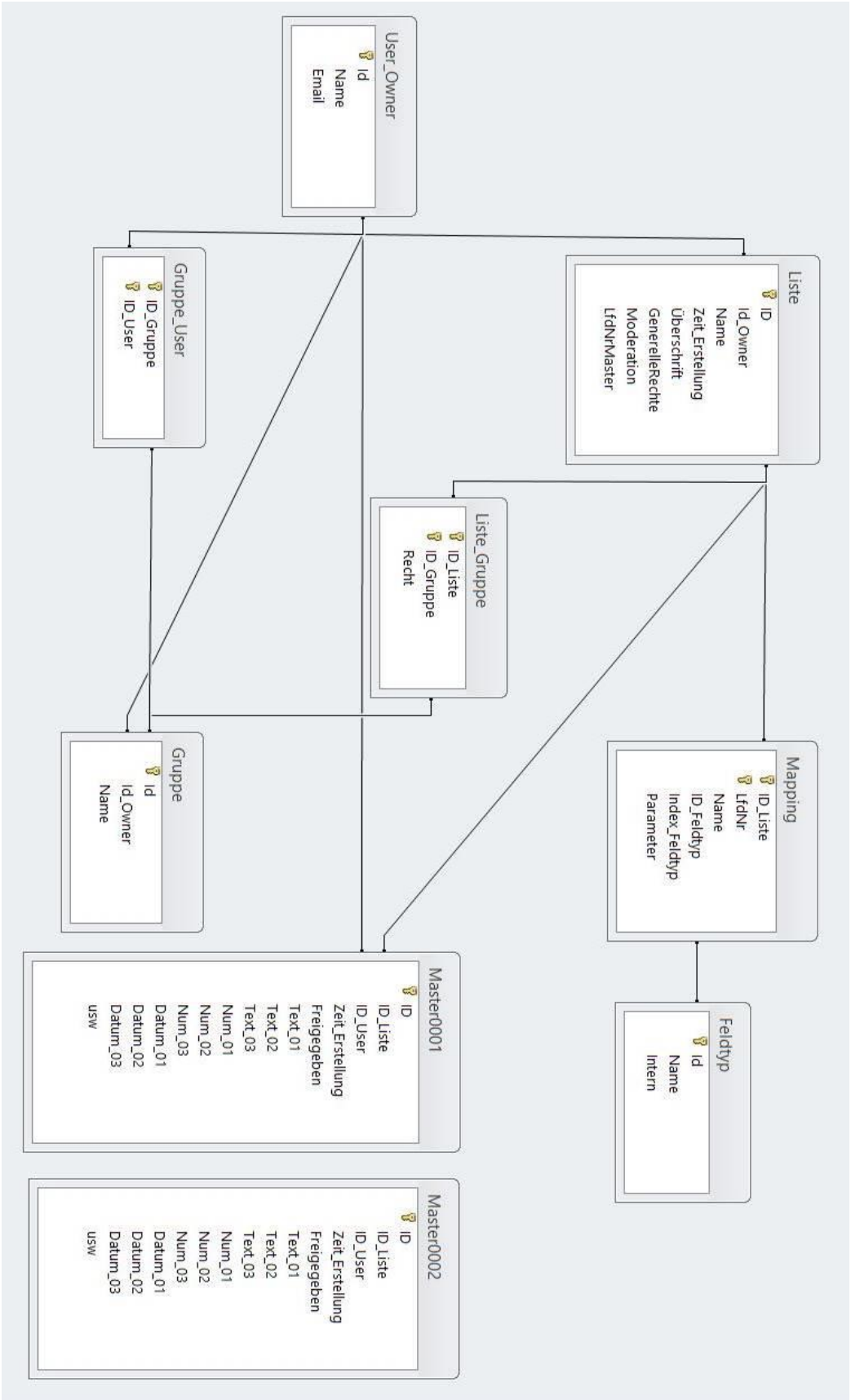
- Anzahl der Listen pro Owner
- maximale Größe der Listen (hinsichtlich Spalten und Zeilen)
- keine abhängigen Listen
- keine Anhänge

Für kommerzielle Nutzer, die dann natürlich dafür zahlen müssen, sollten etwa folgende Features verfügbar sein:

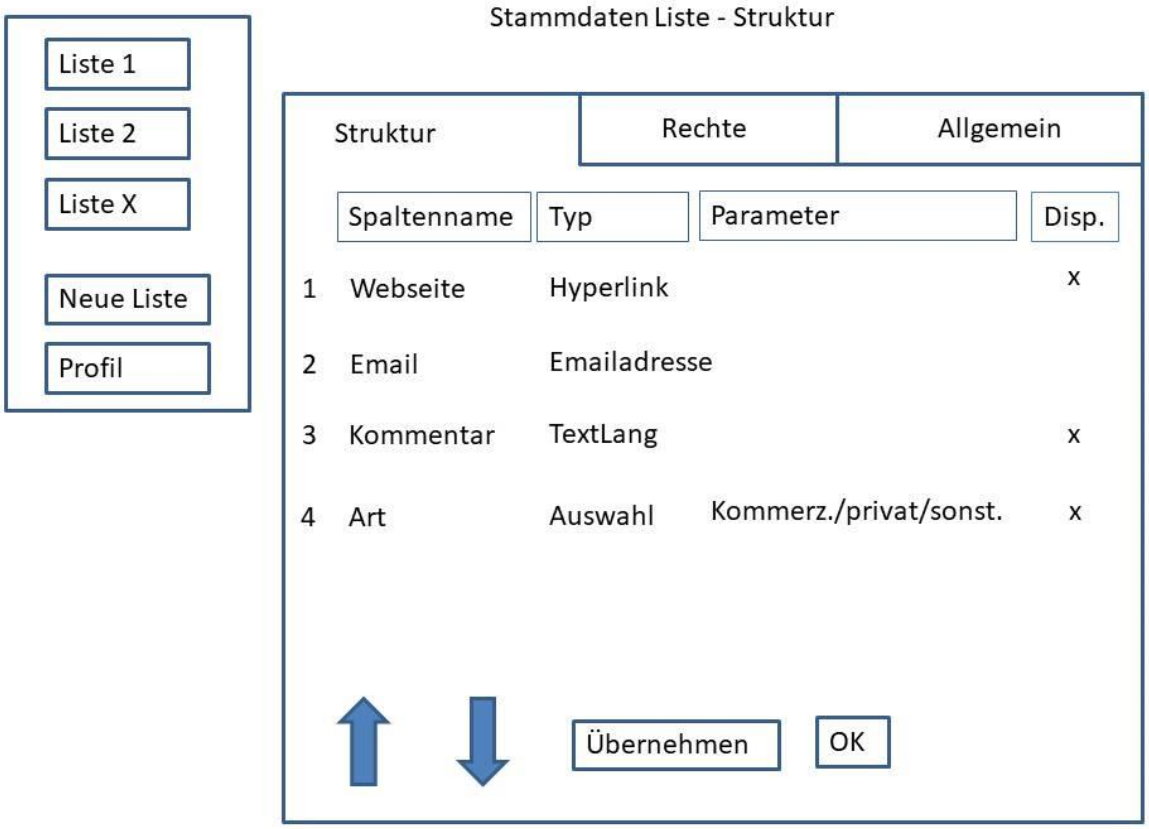
- verschlüsselte Übertragung und Verschlüsselung der Datenbank
- Garantierte Verfügbarkeit des Listservers
- Laufendes Backup
- Hotline

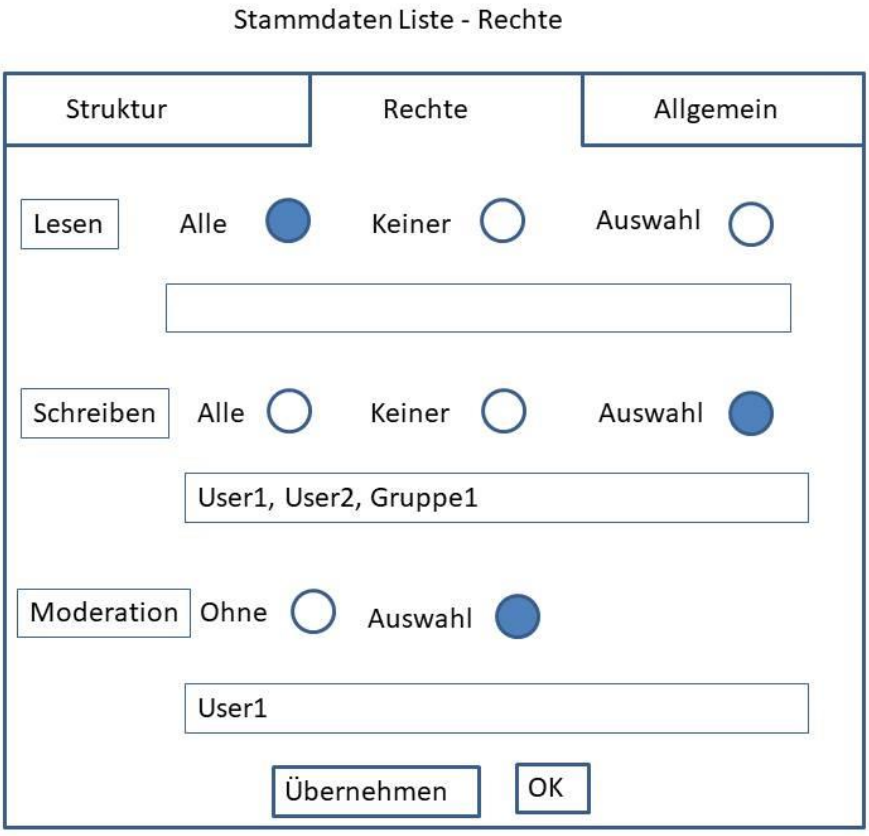
Anhang 1: Datenmodell

Ohne abhängige Listen. Tabelle Feldtyp eventuell überflüssig (wird im Code realisiert).

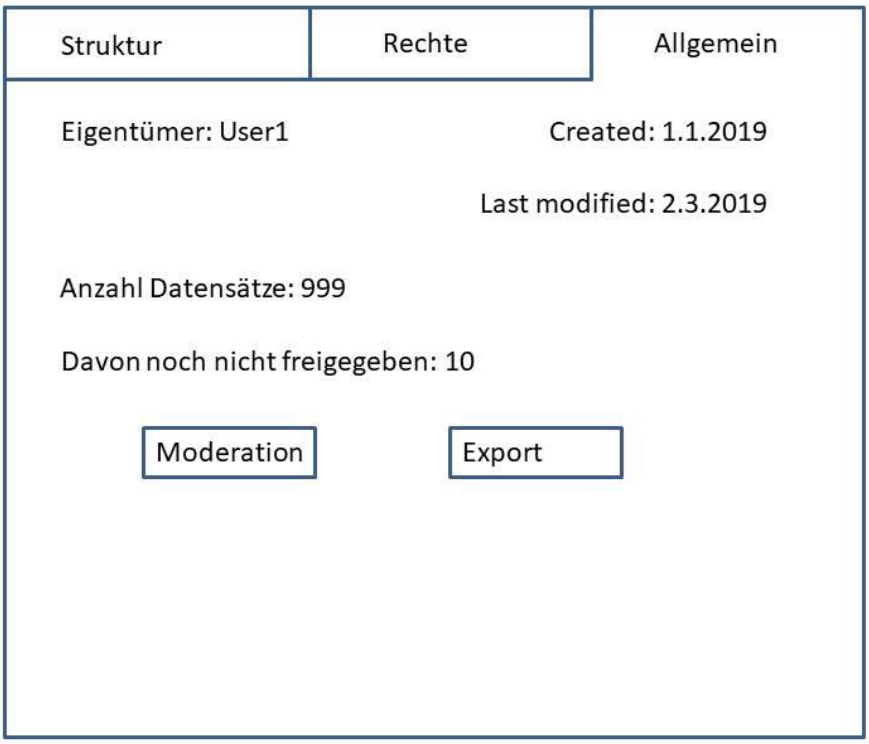


Anhang 2: Oberfläche

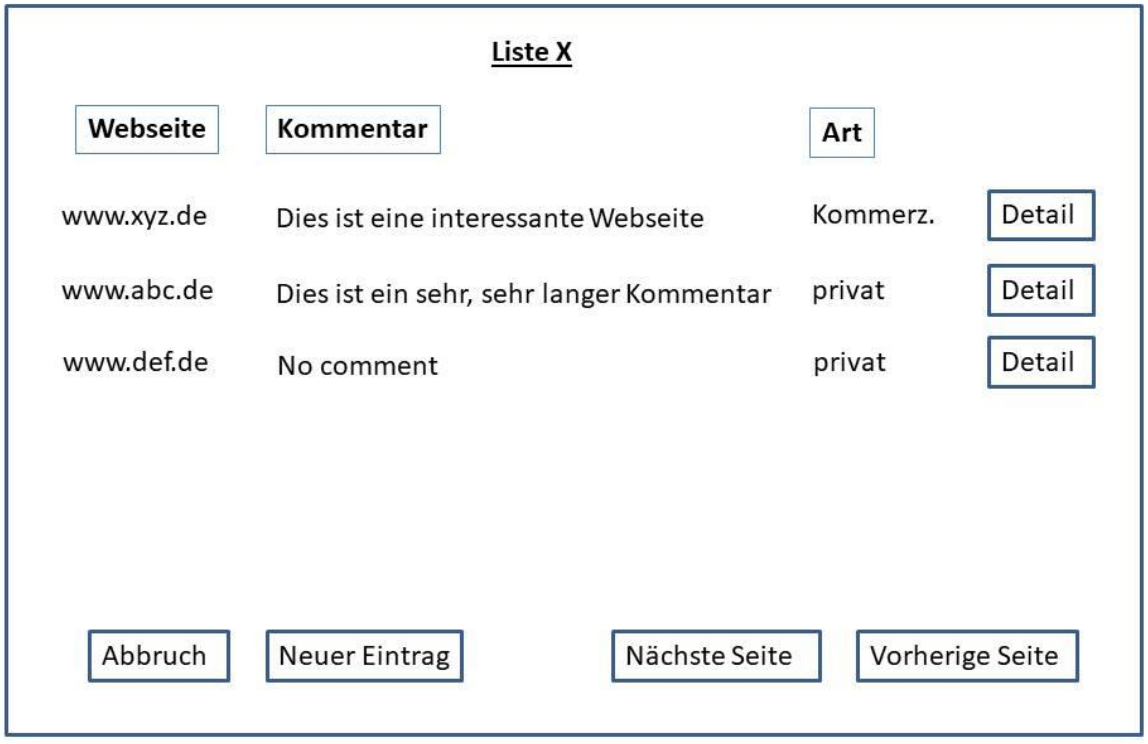




Stammdaten Liste - Allgemein



Listenansicht



Detailansicht für normalen User

Detailansicht für den Moderator

Anhang 3: Sourcecode in Perl der ersten Testversion

```
use strict;
use CGI qw(-nosticky :standard *table *Tr *p); # import
shortcuts
use CGI::Pretty qw( :html3 );
use CGI::Carp qw(fatalsToBrowser);
my (
     $ADMIN,        # Administrator 0 - false / 1 - true
     $PADMIN,
     $FILE,         # First part of filename
     $PFILE,
     $FFIELD,       # Filterfield
     $FVALUE,       # Filtervalue
     $SORDER,       # Sortierordnung 0 - descending / 1 -
ascending
     $SFIELD,       # Sortierfeld
     $CMD,              # Submit-Command
     @DATA,         # data-array
     @STRU,         # Structure-array
     $WRITE,        # "+" or empty
     %TYPE,         # Typenames
     %WORD,         # Special Words
     %COLUMN,       # Columnnames
     %MESSAGE, # Messages
     %TEXT,      # Textfile
     @ERROR,        # Errors
     $CSS           # css
);
$CSS =<<END;
     <!--
     INPUT,SELECT {font-weight:bold;}
     -->
END

%TYPE = (Text => "Text", Number => "Number", Date => "Date",
Email => "Email", URL => "URL",
               ModTime => "ModTime", Id => "Id", Selection =>
"Selection");
%WORD = (Yes => "Yes", No => "No");
%COLUMN = (Id => "Id", ModTime => "Modifizierungszeit", Type
=> "Datentyp", Columnname => "Feldname",
                    Position => "Position", Required =>
"Pflichtfeld", Parameter => "Parameter",
                    Comment => "Comment");
%MESSAGE = (del => "Datensatz wurde inzwischen von jemand
anders gelöscht - geben Sie einen neuen Datensatz ein",
                    mod => "Datensatz wurde inzwischen von
jemand anders verändert - versuchen Sie es noch einmal",
                    fil => "Pflichtfeld nicht ausgefüllt",
                    dat => "Ein gültiges Datum hat die Form
'JJJJ.MM.TT'",
```

```
                    num => "Keine gültige Nummer",
                    ema => "Keine gültige Email-Addresse",
                    sem => "Semicolon nicht erlaubt");
$FILE     = param(".FILE") || param(".PFILE");
Delete(".PFILE");
$ADMIN    = param(".ADMIN")  || param(".PADMIN") || 0;
Delete(".PADMIN");
$FVALUE   = param(".FVALUE") || "";
Delete(".FVALUE");
$FFIELD  = param(".FFIELD") || "*";
Delete(".FFIELD");
$SORDER   = param(".SORDER") || ">";
Delete(".SORDER");
$SFIELD   = param(".SFIELD") || "";
Delete(".SFIELD");
($CMD) = grep /^\.\./ , param();
my $WRITE = ($CMD eq "..UPDATE" || $CMD eq "..DELETE") ? "+" :
"";
if ($ADMIN) {
     @STRU = (
                [0,0,$TYPE{Id},$COLUMN{Id},0,$WORD{Yes},'','wird
vom System vergeben'],

     [1,0,$TYPE{ModTime},$COLUMN{ModTime},1,$WORD{Yes},'','wird
vom System vergeben'],

     [2,0,$TYPE{Selection},$COLUMN{Type},2,$WORD{Yes},join("/",
values(%TYPE)),'Datentyp aus Liste'],

     [3,0,$TYPE{Text},$COLUMN{Columnname},3,$WORD{Yes},'','eind
eutiger Feldname'],

     [4,0,$TYPE{Number},$COLUMN{Position},4,$WORD{Yes},'','Posi
tion in Liste / 0 - keine Anzeige'],

     [5,0,$TYPE{Selection},$COLUMN{Required},5,$WORD{Yes},$WORD
{Yes}."/".$WORD{No}, 'Pflichtfelder müssen ausgefüllt
werden'],

     [6,0,$TYPE{Text},$COLUMN{Parameter},6,$WORD{No},'','Parame
ter (abhängig vom Datentyp)'],

     [7,0,$TYPE{Text},$COLUMN{Comment},7,$WORD{No},'','beliebig
er Kommentar zur Anzeige beim Editieren']);
     %TEXT = (OBEN => "Tabellenstruktur", ADMIN =>
'help@cm.de');
} else { # open the strufile for read
     open(STRU, "< ../data/${FILE}.stru") || error("cannot open
${FILE}.stru: $!");
     my $i = 0;
     while (<STRU>) {chomp; $STRU[$i++] = [split(/;/)];}
     close(STRU);
     open(TEXT, "< ../data/${FILE}.text") || error("cannot open
${FILE}.text: $!");
```

```
     while (<TEXT>) {chomp; my($key,$value) = split(/;/);
$TEXT{$key} = $value;}
     close(TEXT);
}
if ($ADMIN) {
     open(DATA, $WRITE . "< ../data/${FILE}.stru") ||
error("cannot open ${FILE}.stru: $!");
} else {
     open(DATA, $WRITE . "< ../data/${FILE}.data") ||
error("cannot open ${FILE}.data: $!");
}
# get exclusive lock on the datafile
if ($WRITE eq "+") {
     flock(DATA, "2") || bail("cannot flock DATA: $!");
}
my $i = 0;
while (<DATA>) {chomp; $DATA[$i++] = [split(/;/)];}
if ($WRITE eq "+") {
     my $id = param($STRU[0][3]);
     @ERROR = doUpdate($id);
     if (defined(@ERROR) && @ERROR > 0) {
          $CMD = "..EDIT:" . $id;
     }
}
close(DATA);
if ($CMD eq "..FILTER") {
     $FVALUE = param(".VALUE");
     $FVALUE =~ s/\s+$//;
     $FFIELD = param(".FIELD");
}
if (substr($CMD,0,3) eq "...")
{    # sort
     if ($SFIELD eq substr($CMD,3)){
          $SORDER = ($SORDER eq ">") ? "<" : ">";
     }
     else      {
          $SFIELD = substr($CMD,3);
          $SORDER = ">";
     }
}
if (substr($CMD,0,7) eq "..EDIT:"){
     printEdit(substr($CMD,7));
} else {
     printList();
}
#----------------- Subroutines ------------------------------
sub printAsHTML {
     print header,start_html("TEST");
     print h1(shift);
     print end_html();
     die ("SSS");
}
sub doUpdate
{    # returns @error - Liste von Fehlermeldungen
```

```
     my $id = shift;
     my ($row,@error);
     if ($id eq ""){
          $row = @DATA; $id = ($row == 0) ? 0 : $DATA[$row -
1][0]+1;
     } else {
          for ($row = 0;($row < @DATA) && ($DATA[$row][0] !=
$id); $row++) {;}
          if ($row == @DATA){
               Delete_all(); push(@error, $MESSAGE{del});
return @error;
          } else {
               if (param($STRU[1][3]) ne
printTime($DATA[$row][1])){
                    Delete_all(); push(@error, $MESSAGE{mod});
return @error;
               }
          }
     }
     my $str;
     if ($CMD eq "..UPDATE") {
          for my $i (2..$#STRU) {
               ($str = param($STRU[$i][3])) =~ s/\s+$//;
               param($STRU[$i][3] => $str);
               if ($str eq "") {
                    if ($STRU[$i][5] eq $WORD{Yes}) {
                         push(@error, "$STRU[$i][3] -
$MESSAGE{fil}");
                    }
               } else {
                    if ($str = check($str, $STRU[$i][2])){
                         push(@error, "$STRU[$i][3] - $str");
                    }
               }
          }
          if (defined(@error) && @error > 0){ return @error;}
          $DATA[$row][0] = $id;
          param($STRU[0][3] => $id);
          $DATA[$row][1] = time;
          param($STRU[1][3] => printTime($DATA[$row][1]));
          for my $i (2..$#STRU) {
               $DATA[$row][$i] = param($STRU[$i][3]);
          }
     } else { # ..DELETE
          splice(@DATA, $row, 1);
     }
     seek(DATA, 0, 0) or error ("can't seek to start of file:
$!");
     for $i (0..$#DATA) {
          if (!defined($DATA[$i])){
               print DATA "\n" or error ("can't print to file:
$!");
          } else {
```

```
               print DATA join(";",@{$DATA[$i]})."\n" or error
("can't print to file: $!");
          }
     }
     truncate(DATA, tell(DATA)) or error("can't truncate file:
$!");
     return;
}
#-------- print-Routinen -----------------------------------
-----------------
sub printList { # print formatted table
     my
($i,$j,$struSort,$dataSort,@colnames,$tmp,$col,$row,$type,$fou
nd);
     # get indexes in order of column 4 (position)
     $struSort = sort2dimByColumn (\@STRU, 4, 0, 1);
     $colnames[0] = "*";
     for $i (0..$#{$struSort}){
          if ($STRU[${$struSort}[$i]][4] == 0) { next;}
          push(@colnames, $STRU[${$struSort}[$i]][3]);
     }
     print header(), start_html({-TITLE => $TEXT{OBEN}, -
BGCOLOR=>'#CCFFFF',-style=>{-code=>$CSS}}),
                                   basefont({FACE =>
"ARIAL"}),a({-NAME => "Top"});
     print start_table({-BORDER => "0"}), start_Tr();
     print th({-width=>"1%"},a({-href => '/cm'},img({-
SRC=>"/cm/data/cm.jpg"})));
     print     th({-width=>"98%"},h2({ -ALIGN =>
"CENTER"},$TEXT{OBEN}));
     print     th({-width=>"1%"},h3({ -ALIGN =>
"RIGHT"},'Administrator: '.a({-href =>
'mailto:'.$TEXT{ADMIN}},$TEXT{ADMIN})));
     print end_table, end_Tr();
     print start_form({-NAME => "List", -METHOD => "POST"});
     print hidden({-NAME => ".PFILE", -VALUE => $FILE});
     print hidden({-NAME => ".PADMIN", -VALUE => $ADMIN});
     print hidden({-NAME => ".FFIELD", -VALUE => $FFIELD});
     print hidden({-NAME => ".FVALUE", -VALUE => $FVALUE});
     print hidden({-NAME => ".SORDER", -VALUE => $SORDER});
     print hidden({-NAME => ".SFIELD", -VALUE => $SFIELD});
     print p({ -ALIGN => "CENTER"},'<nobr>',
               scrolling_list({-NAME => ".FIELD", -VALUES =>
\@colnames,
                                    -SIZE => 1, -DEFAULT =>
$FFIELD}),
               textfield({-NAME => ".VALUE", -DEFAULT =>
$FVALUE, -OVERRIDE => 1,}),
               submit({-NAME => "..FILTER", -VALUE =>
"FILTER"}),
               submit({-NAME => "..EDIT:", -VALUE => "NEUER
EINTRAG"},'</nobr>'));
     print start_table({-BORDER => "0",-cellspacing => 1,-ALIGN
=> "CENTER"});
```

```
     print start_Tr({-BGCOLOR => "#6699FF"}),
               th(submit({-NAME => "...", -VALUE => ($SFIELD eq
"") ? $SORDER : " ",
                                        -style => "width:
100%;background-color:#6699FF;font-weight:bolder"}));
     $col = -1;  # sortcolumn
     for $i (0 .. $#{$struSort}) {
          $row = ${$struSort}[$i];
          if ($STRU[$row][4] == 0) { next;}
          if ($STRU[$row][3] eq $SFIELD){$col = $row;}
          print th(submit({   -NAME => "..." . $STRU[$row][3],
                                          -VALUE => (($col ==
$row) ? $SORDER : "") . $STRU[$row][3],
                                          -style => "width:
100%;background-color:#6699FF;font-weight:bolder" }));
     }
     print end_Tr();
   if ($col >= 0){
     $type = ($STRU[$col][2] =~
"$TYPE{Number}|$TYPE{Id}|$TYPE{ModTime}") ? 0 : 1;
   }
     $dataSort = sort2dimByColumn (\@DATA, $col, $type,
($SORDER eq ">") ? 1 : 0);
     my ($anf, $end);
     if ($FFIELD eq "*"){
          $anf = 0; $end = $#STRU;
     } else {
          for ($i = 0; $i < @STRU; $i++) {
          if ($FFIELD eq $STRU[$i][3]) {
               $anf = $end = $i;
               last;
          }
     }
   }
     for $i (0..$#{$dataSort}) {
          $row = ${$dataSort}[$i];
          if ($FVALUE ne ""){
               $found = 0;
               for $j ($anf..$end) {
                    if ($STRU[$j][4] == 0) { next;}
                    if ($DATA[$row][$j] =~ "$FVALUE"){$found =
1;last;}
               }
               if (!$found){next;}
          }
          print start_Tr({-BGCOLOR => ($i%2 == 0) ? "white" :
"silver"}),
                    td(submit({-NAME => sprintf
("..EDIT:%s",$DATA[$row][0]), -VALUE => ">"}));
          # print columns in sortorder
          for $j (0..$#{$struSort}) {
               $col = ${$struSort}[$j];
               if ($STRU[$col][4] == 0) { next;}
               $type = $STRU[$col][2];
```

```
               if ($type eq $TYPE{URL}) {
                    print td(a({-href =>
$DATA[$row][$col]}),$DATA[$row][$col]);
               } elsif ($type eq $TYPE{Email}) {
                    print td(a({-href =>
'mailto:'.$DATA[$row][$col]}),$DATA[$row][$col]);
               } elsif ($type eq $TYPE{ModTime}) {
                    print td({-ALIGN => "LEFT"},
printTime($DATA[$row][$col]));
               } else {
                    $tmp = ($type =~
"$TYPE{Number}|$TYPE{Id}|$TYPE{Date}") ? "RIGHT" : "LEFT";
                    print td({-ALIGN => $tmp},
$DATA[$row][$col]);
               }
          }
          print end_Tr();
     }
     print end_table();
   print p({-ALIGN => 'Center'}, a({-HREF => '#Top'},"nach
oben"));
     print end_form(), end_html();
}
sub printEdit {
     my $id = shift;
     my ($row, $type, $param);
     if ($id eq "") {
          $row = @DATA;
          for $i (0..$#STRU) { $DATA[$row][$i] = "";}
     } else {
          for ($row = 0;($row < @DATA) && ($DATA[$row][0] !=
$id); $row++) {;}
          if ($row == @DATA){
               error ("Id $id not found");
          }
     }
     print header(), start_html({-TITLE => $TEXT{OBEN}, -
BGCOLOR => '#CCFFFF',-style=>{-code=>$CSS}}),
               basefont({FACE => "ARIAL"}),h2({-ALIGN =>
"CENTER"}, "Datensatz eingeben/modifizieren");
     print start_form({-NAME => "Edit", -METHOD => "POST"});
     print hidden({-NAME => ".PFILE", -VALUE => $FILE});
     print hidden({-NAME => ".PADMIN", -VALUE => $ADMIN});
     print hidden({-NAME => ".FFIELD", -VALUE => $FFIELD});
     print hidden({-NAME => ".FVALUE", -VALUE => $FVALUE});
     print hidden({-NAME => ".SORDER", -VALUE => $SORDER});
     print hidden({-NAME => ".SFIELD", -VALUE => $SFIELD});
     print start_table({-BORDER => "0", -CELLSPACING => 1, -
ALIGN => "CENTER", -BGCOLOR => "#6699FF"});
     my ($i, $tmp);
     for $i (0..$#STRU) { # print field
          print start_Tr(), td({-style=>'background-
color:#6699FF; font-weight:bold'},$STRU[$i][3]);
          $type = $STRU[$i][2];$param = $STRU[$i][6];
```

```
          if ($type eq $TYPE{ModTime}) {
               print td(textfield({-NAME => $STRU[$i][3],-
style=>'background-color:#6699FF',
                                   -DEFAULT =>
printTime($DATA[$row][$i]),-READONLY => undef}));
          } elsif ($type eq $TYPE{Id}) {
               print td(textfield({-NAME => $STRU[$i][3],-
style=>'background-color:#6699FF',
                                     -DEFAULT => $DATA[$row][$i],
-READONLY => undef}));
          } elsif ($type eq $TYPE{Selection}) {
               my @list = split ("/",$STRU[$i][6]);
               print td({-align => 'right'},scrolling_list({-
NAME => $STRU[$i][3],-style=>'width:100%',
                                -VALUES => \@list,-SIZE => 1, -
DEFAULT => $DATA[$row][$i]}));
          } else {
               print td(textfield({-NAME => $STRU[$i][3], -
DEFAULT => $DATA[$row][$i]}));
          }
          print td(($STRU[$i][5] eq $WORD{Yes}) ? "*" : "
"),td($STRU[$i][7]);
          print end_Tr();
     }
     print end_table() ;
     print     p({ -ALIGN => "CENTER"},'<nobr>',
               submit({-NAME => "..UPDATE", -VALUE =>
"ÄNDERN"}),
                ($id eq "") ? "" : submit({-NAME => "..DELETE",
-VALUE => "LÖSCHEN"}),
               submit({-NAME => "..CANCEL", -VALUE =>
"ABBRECHEN"}),
               reset({-VALUE => "ZURÜCKSETZEN"}),'</nobr>');
     print end_form();
     if (defined(@ERROR) && @ERROR > 0){
          print start_p({ -ALIGN => "CENTER"}),
b("FEHLER:"),br;
          for $i (0..$#ERROR){
               print b($ERROR[$i]), br;
          }
          print end_p;
     }
     print end_html();
}
#------------- Hilfsroutinen ---------------------------------
------------
sub check {
     my $str = shift;
     my $type = shift;
     if ($str =~ /;/){
          return $MESSAGE{sem};
     }
     if ($type eq $TYPE{Number}){
          if (!($str =~ /^-?(?:\d+(?:\.\d*)?|\.\d+)$/)){
```

```
                return $MESSAGE{num};
           }
     } elsif ($type eq $TYPE{Date}){
           if (substr($str,-1,1) eq "."){
                return $MESSAGE{dat};
           }
           my @tmp = split(/\./,$str);
           for my $i (0..$#tmp){
                if (($tmp[$i] =~ /\D/) || (length($tmp[$i]) != 4
&& $i == 0) ||
                     (length($tmp[$i]) != 2 && $i > 0))
                     {return $MESSAGE{dat};}
           }
     } elsif ($type eq $TYPE{Email}){
           if ((substr($str,-1,1) eq "@") || (split(/\@/,$str)
!= 2)){
                return $MESSAGE{ema};
           }
     }
}
sub printTime {
     my $time = shift;
     if ($time eq "") { return ""};
     my ($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdst) =
localtime($time);
     return ($year+1900) . "/" . sprintf("%2.2d",$mon+1) . "/"
          . sprintf("%2.2d",$mday) . "-" .
sprintf("%2.2d",$hour) . ":"
          . sprintf("%2.2d",$min) . ":" .
sprintf("%2.2d",$sec);
}
sub sort2dimByColumn { # sort 2-dimensional array by specified
column
     # $_[0] - array to be sorted (reference)
     # $_[1] - index of column to sort on / -1 falls auf
arrayindex sortiert werden soll
     # $_[2] - type of comparison (0 - numeric / 1 - string)
     # $_[3] - descending (0) / ascending (1)
     # returns reference to a sorted array of indexes (of the
original)
     my (%tmp,$val,@tmp,@ret,$i);
     if ($_[1] < 0) {# Spezialfall
           for $i (0..$#{$_[0]}){
                if ($_[3]){
                     $ret[$i] = $i;
                } else {
                     $ret[$i] = $#{$_[0]} - $i;
                }
           }
           return \@ret;
     }
     # put values to be sorted in hash (as keys with their
corresponding indexes as values)
     for $i (0..$#{$_[0]}){
```

```
          if (!defined(${$_[0]}[$i])){
               $val = undef;
          } else {
               $val = ${$_[0]}[$i][$_[1]];
          }
          if (exists $tmp{$val}) {
               if (ref $tmp{$val})      {push @{$tmp{$val}},
$i;}
               else                                {$tmp{$val} =
[$tmp{$val},$i];
               }
          }    else                                {$tmp{$val} =
$i;}
     }
     # sort
     if ($_[2]) { # string-sort
          if ($_[3])     {@tmp = sort {$a cmp $b} keys %tmp; #
ascending
          } else         {@tmp = sort {$b cmp $a} keys %tmp; #
descending
          }
     } else { # numeric sort
          if ($_[3])     {@tmp = sort {$a <=> $b} keys %tmp; #
ascending
          } else         {@tmp = sort {$b <=> $a} keys %tmp; #
descending
          }
     }
     # put indexes in ret-array (in order of sorted keys)
     for $i (0..$#tmp) {
          $val = $tmp[$i];
          if (ref $tmp{$val})      {push @ret, @{$tmp{$val}};}
          else                              {push @ret,
$tmp{$val};}
     }
     return \@ret;
}
sub print2dim { # print 2-dimensional array
     my $row;
     print "Anzahl Zeilen = ", scalar(@{@_[0]});
     print start_table({-BORDER => "1"}) ;
     foreach $row (@{$_[0]}) {
          print Tr(td([@{$row}]));
     }
     print end_table();
}
sub error { # function to handle errors gracefully
     my $error = "@_";
     print header,start_html("Error");
     print h1("Unexpected Error"), p($error), end_html;
     print end_html;
     die ($error);
}
```