# *mFrame*

***Eine Entwicklungsumgebung für datenabankgetriebene Webanwendungen***

## Kurzbeschreibung

ein Tool von Wilhelm Nagy

# Programmidee

Das Programmsystem mFrame ist ein Framework um Webapplikatinen möglichst effektiv zu erzeugen. Es basiert auf einer XML Steuerdatei. Das Framework ist an unterschiedliche Umgebunden leicht anzupassen. Eine Datenbankanbindung ist möglich aber nicht notwendig.

# Quickstart

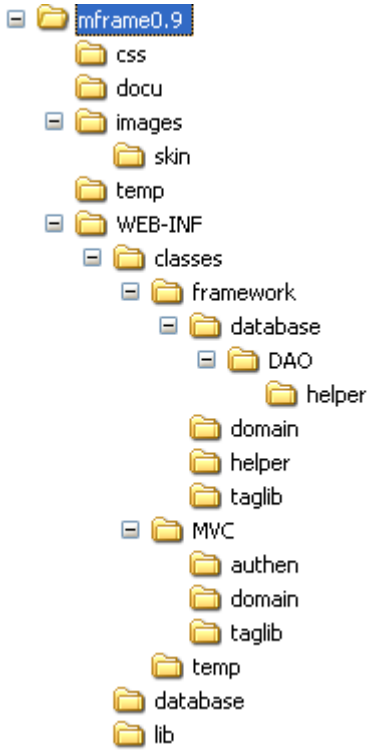
## *Hintergrund*

Schritt für Schrittanweisung für das Erstellen einer Applikation

## *Prerequisit*

Es wird davon ausgegangen, dass auf dem Zielrechner eine Tomcat oder Jetty Container lauffähit installiert ist.

Lieferumfang




**WEB-INF** (Wurzelverzeichnis der Applikation )

**classes**

**framework**

Enhält alle Datein des Frameworks

Kann bei einer neuen Version ausgetauscht werden

**MVC** (Wurzelverzeichnis der Anwendung)

**authen** (Authentifizierungsroutinen)

**domain** (Enthällt die Domainklassen)

**taglib** (Taglibklasse)

**lib** (Enhält alle Jar Files)

## *Installation*

Kopieren in ein Verzeichnis des Webcontainers.

Im Verzeichnis WEB-INF befindet sich die Datei mframe.xml. Dies ist die Zenrale Steuerdatei.

## Konfiguration der Appliaktionssteuerdatei (mframe.xml)

Die zu konfiguriernden Teile sind unterlegt

```
<server name="localhost">
  <approot value="mframe" />                          <-- Name der Appliaktion im WEB-INF
  <defaultstylesheet href="/mframe/css/xal.css" />    <-- Pfad auf die CSS Datei
  <urlencoding value="no" />
  <entrypoint value="start.groovy" />
     <mvcentrypoint value="MVC" />
  <logo  small=""/>
     <database>
       <environments>
        <produktion
             driverClassName = "org.hsqldb.jdbcDriver" <-- Datenbanktreiber
          username = "sa"                              <-- User
          password = ""                                <-- Passwort


        url = "jdbc:hsqldb:file:x:/MFrame/distro/mframe0.9/WEB-INF/database/mframe"
              dbtype="hsqldb">                         <-- Datenbank URL (Siehe JDBC)
```

Um das Seitenweise ausgeben bei der Liste zu ermöglichen muss eine Datenbankanweisung geschrieben welche ein Datenzahlenbereich definiert werden. Im Lieferumfang sind Beispiele für ORACLE oder hsqldb angegeben.

```
        <paginatesql_oracle>
                <![CDATA[
                        SELECT *
                     FROM (SELECT ROW_NUMBER() OVER(${orderby}) AS CNT_ROW_NUMBER, ${fields}
                     FROM ${table} ${fromextend} where ${where})
                    WHERE CNT_ROW_NUMBER between ${from} AND ${to}
                ]]>
       </paginatesql_oracle>
        <paginatesql>
                <![CDATA[
                select  limit ${from} ${pagesize} * from ${table} where ${where} ${orderby}
                ]]>
       </paginatesql>
             </produktion>
         </environments>
       </database>
 </server>
</config>
```

## *Menü*

Das Framwork ist mit einer Navigation ausgestattet. Für jede Funktion ist ein Eintrag

notweindig

```
<config>
 <login  path="/authen/login" />
 <logout path="/root/system/abmelden" />
 <start  path="/root" />
</config>
```

In dieser Sektion werden die Menüeinträge für die Navigation eingetragen.

## *Menüeinträge*

Für jede Funkton muss ein Eintrag notiert werden.

Beispiel

```
    <entry path="/root">
     <text value="mfrage"/>
      <controller path="/mainController.groovy" />
     <title value="MFrame Test" />
    </entry>
```

Der path Eintrag im Menüeintrag
Jeder Menüeintrag muss eindeutig gekennzeichnet sein. Gleichzeitig wird die Aufrufhirrachie definiert.
**HINT:**
Die Naviation kann mit einen Rechtesystem versehen werden. Wird im Entryeintrag die Option `<right value="Rechteliste"/>` versehen so wird gegen diese Rechte geprüft.

# Controller

In den Controller wird die Applikationslogik geschrieben und Viewer aufgerufen.

## *Hello world controller*

`render „hello World“`

gibt den String „hello World“ aus.

# Domain

Domain stellen die Verbindung zur Datenbank her

```
package MVC.domain.jobs

// ---- Classe -------------------------------------------------------
class Jobs extends framework.database.DAO.DataAccessObject {
// -------------------------------------------------------------------

   String id
   String title
   Double minSalary
   Double maxSalary

   def meta = [
          idfield:[db:'JOB_ID'],
```

```
        tablename: 'JOBS',
        fields: [
                id:             [dbfield:'JOB_ID'],
                title:          [dbfield:'JOB_TITLE'],
                minSalary:      [dbfield:'MIN_SALARY'],
                maxSalary:      [dbfield:'MAX_SALARY']
                ]
        ]
  }
```

Für jedes in der Datenbank zu speichernde Feld wird ein Feld in der Domain ein Datenfeld angelegt. In der Datenstruktur meta werden die Datenbankdatails beschrieben.

## Viewer

Die Viewer sind Datein ähnlich JSP. Siehe groovy Spezifkation. Sie haben im einfachsten Fall nur HTML Kode können aber mit den Daten des Controller versehen werden.

```
<% import MVC.domain.jobs.Jobs    %>
<% Jobs jobs = new Jobs(db:db.dbh) %>
<h1>List der Jobs</h1>
<table>
<%jobs.eachDomain(){job->%>
   <tr><td><%=job.id%></td><td><%=job.title%><td></tr>
   <%}%>
<table>
```

## CRUD

**S**tandard **C**reate Read **U**pdate **D**elete Anwendung

Die Applikation enthällt eine Metoden zur einfachen Erstelleung von Standardeingabemasken.

```
import MVC.domain.employees.Employees
import framework.helper.CRUD

CRUD crud = new CRUD(
                    request:request,
                    domain:new Employees(db:db.dbh),
                    controllername:'EmployeesController.groovy',
                    // Funktionen aus dem Framework
                    addFunction:addFunction,
                    flash:flash,
                    view:view,
                    readTemplatefile:readTemplatefile,
                    render:render
                    )
crud.run()
```

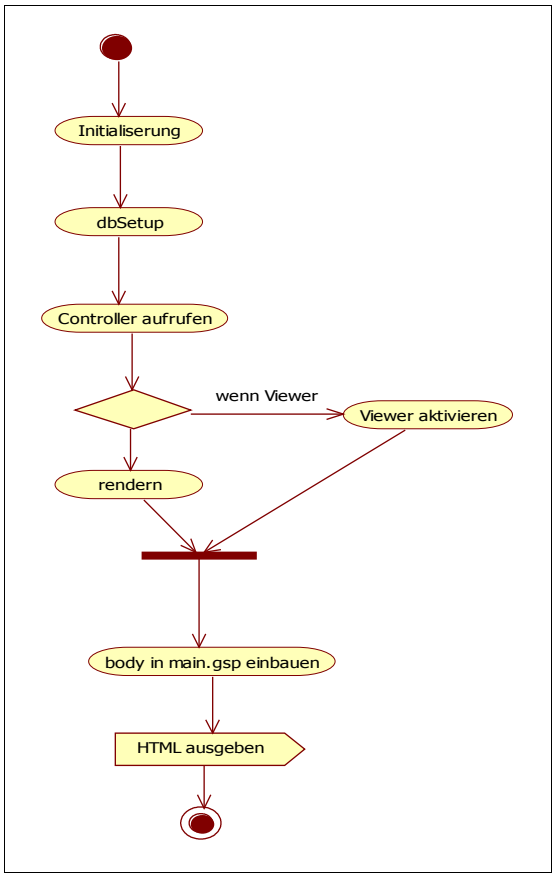
Diese stellt die Einfachste Form einen CRUD Controller dar.

# Ausgabe aller Seiten im selben Layout

Alle Ausgaben werden in eine Template Injiziert.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
<title> inbox::start ${realpath} [${path}]</title>
<meta name="Author" content="W. Nagy">
<meta http-equiv="content-type" content="text/html; charset=ISO-8859-1">
<link rel="stylesheet" href="./css/xal.css" />
</head>
<body>
${menu}
${body}
<hr/><a href="start.groovy?path=$logoutpath">logout</a> User: ${user} V:
${mframeversion}  <% import MVC.helper.converter.DateConverter;out << new
DateConverter().giveAsASCIIDateTime(new Date()) %>
${flash}
</body>
</html>
```

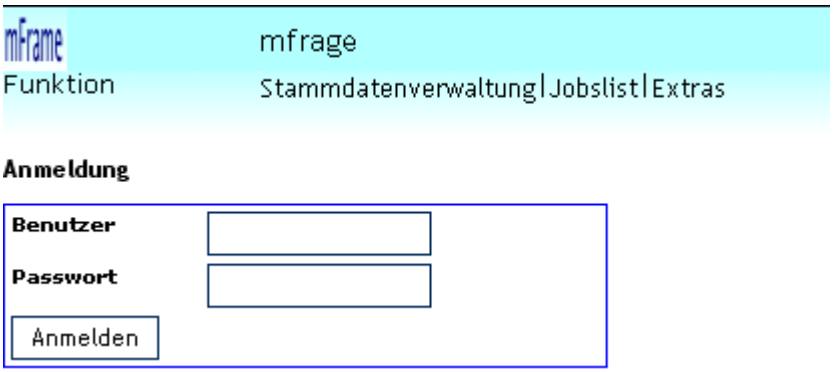
Platzhalter ermöglichen das einfügen von Controller („render“) und Viewerausgaben. Der Plazthalter ${menu} gibt die Stelle an welcher die Navigaton erscheinen soll. Eine Standardisierte ausgabe von Programmmeldungen wird im  Platzhalter ${flasch} ermöglicht.




Diese Graphik zeigt den logischen Ablauf des Injizieren.

# Screenshots

## *Einlogmaske*




### Codes dieser Maske (Viewer)

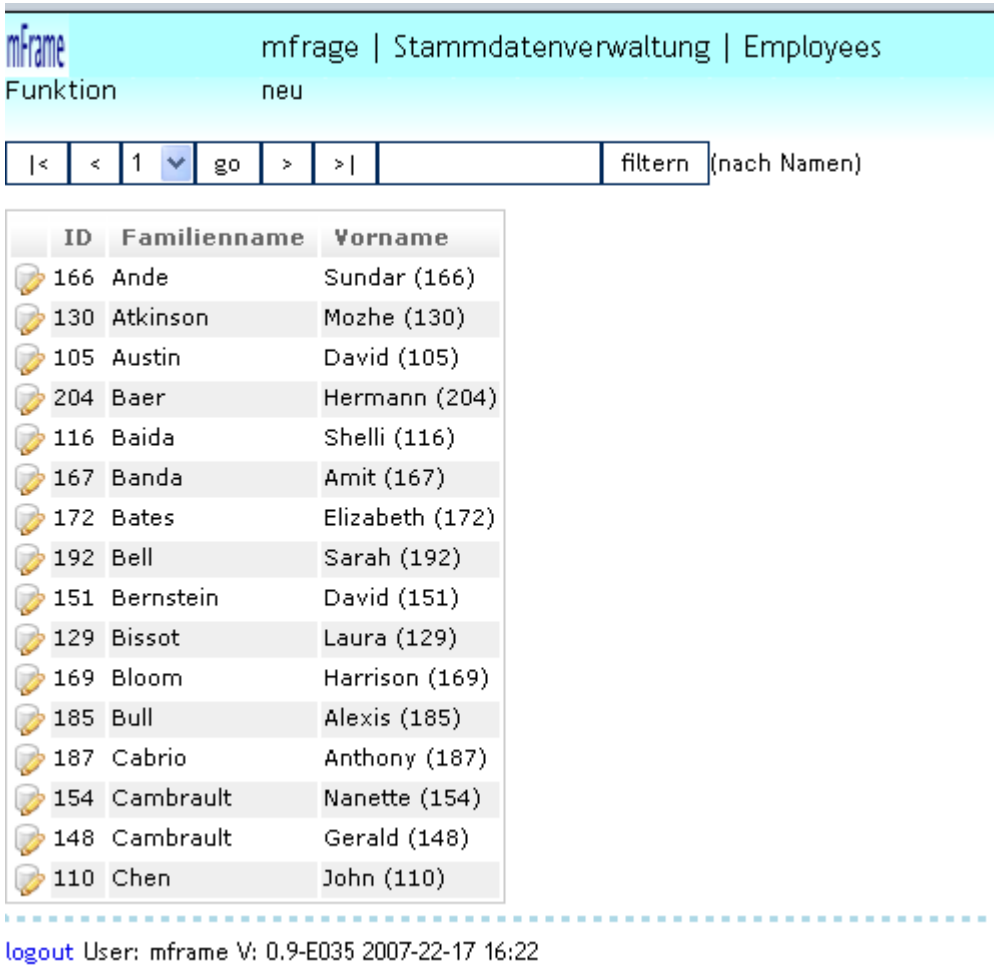
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
<title> inbox::login ${realpath}   [${path}]</title>
<meta name="Author" content="W. Nagy">
<link rel="stylesheet" href="./css/xal.css" />
</head>
<body>
<h1>Anmeldung</h1>
<form method="POST" action="start.groovy">
<input type="hidden" name="path" value="/authen/login"/>
<table border="0" width="1" class="msk">
 <tr><th class="msk">Benutzer</td><td class="msk"><input type="input" name="username"
value=""/></td></tr>
 <tr><th class="msk">Passwort</td><td class="msk"><input type="password" name="password"
value=""/></td></tr>
 <tr><td colspan="99"><input type="submit" value="Anmelden"/></td></tr>
</table>
</form>
${flash}
</body>
</html>
```

## *Eingabemake*

## Code der Maske (Viewer)

```
<%
import MVC.domain.employees.Employees
def taglib = new MVC.taglib.tlb(request:request)
%>

<form name="edit">

<%= taglib.getHiddenfieldsFromCGI(fields:['path','controller','filter'])%>
<%= taglib.addnew_save_action()%>

<table border="0">
 <
%=taglib.promptinput(prompt:'ID',name:'id',size:5,maxlength:5,type:'text',value:rec?.id,disabled:tru
e)%>
 <
%=taglib.promptinput(prompt:'Vorname',name:'firstName',size:32,maxlength:32,type:'text',value:rec?.f
irstName,disabled:false)%>
 <
%=taglib.promptinput(prompt:'Nachname',name:'lastName',size:32,maxlength:32,type:'text',value:rec?.l
astName,disabled:false)%>
 <
%=taglib.promptinput(prompt:'Email',name:'email',size:32,maxlength:32,type:'text',value:rec?.email,d
isabled:false)%>
  <tr>
   <th class="msk">Job ID</th>
   <td>
    <%=
       taglib.selectbox(name:'jobID',dbh:db.dbh,
          sql:'''select JOB_ID as "KEY",JOB_TITLE||' ('||JOB_ID||')' as "VALUE"
          from JOBS order by       JOB_ID''',
          selected:rec?.jobID
          )
    %>
   </td>
  </tr>
  <tr>
   <th class="msk">Manager ID</th>
   <td>
    <%=
       taglib.selectbox(
          name:'managerID',
          dbh:db.dbh,
          sql:'''select EMPLOYEE_ID as "KEY", LAST_NAME|| ' (' || EMPLOYEE_ID || ')'  as "VALUE"
                   from EMPLOYEES order by LAST_NAME''',
          selected:rec?.managerID, disabled:true
          )
    %>
   </td>
  </tr>
 <%=
   taglib.promptinput(prompt:'Gehalt',name:'salary',size:11,maxlength:11,type:'text',value:sprintf('
%.2f',(rec?.salary==null ? 0.0 : rec.salary)),disabled:false)%>

 <tr>
  <td colspan="99"><%=taglib.getSaveAndDeleteButton()%></td>
 </tr>
</table>
</form>
```

## *Seitenweise Auswahllist*




## Quellencode der Auswahlliste

```
<% def taglib = new MVC.taglib.tlb(request:request,db:db) %>
<%

String controller = taglib.cgiparam(name:'controller',nvl:'')
String path       = taglib.cgiparam(name:'path',nvl:'/root')
String filter     = taglib.cgiparam(name:'filter',nvl:'')

String where = taglib.cgiparam(name:'filter',nvl:' (1=1) ')
if (where != ' (1=1) ') {
   where = 'lower(LAST_NAME) like lower(\''+where+'%\')'
   }

String sql = db.getSelectLimit(table:'EMPLOYEES',
                               where:where,
                               orderby:'LAST_NAME',
                               listclause:['FIRST_NAME','LAST_NAME','EMPLOYEE_ID'],
                               pageno:taglib.cgiparam(name:'pageno',nvl:'1'),
                               server:server
                                      )

%>

<%= taglib.listwidget(table:'EMPLOYEES',
                      where:where,
                      filter:taglib.cgiparam(name:'filter',nvl:''),
                      title:'Angestellter',
                      filtertext:'(nach Namen)'
                      )%>
<table>
<%
out<<taglib.listheader(th:[' ','ID','Familienname','Vorname'])
```

```
Integer cntRow = 0
%>

<%db.dbh.eachRow(sql) {row->
  cntRow++
  String classname = ((cntRow % 2) == 0) ? 'listcellcolor1' : 'listcellcolor2'
  %>
 <tr>
  <td><%= taglib.getIcon(name:'database_edit.png',href:"start.groovy?path=$
{path}&controller=EmployeesController.groovy&id="+row.EMPLOYEE_ID+'&action=edit'+'&filter='+filter)
%></td>
  <%= taglib.listrow(td:[row.EMPLOYEE_ID,row.LAST_NAME,row.FIRST_NAME+'
('+row.EMPLOYEE_ID+')'],tr:false,class:classname)%>
  </tr>
<%}%>
</table>
```

## ToDo

Noch ist nicht alles geschafft. Obwohl das Framwork schon eingesetzt wird ist es noch nicht perfekt. Folgende Funktionen werden als nächstes realisiert.

- Vollständige Trennung vom den Datenbankfeldern
  Die Applikation soll kein Datenbankfeldnamen mehr wissen müssen.
- Einbau der Vollständigen Taglibunterstützung
  Die Syntys „`<%= taglib.selectbox .... %>`“ sollen durch HTML kompatible Ausdrücke wie:
  `<m:selectbox name=“lovID“ sql=“select...“ selected=“${lovID}“ />`
  ersetzt werden.
- Entfernen der SQL Anweisungen für Seitenweise Auflistung aus der Konfigurationsdatei.